CLPractice 2.0
Tools for Learning: Computation and Logic

Petr Manas

MInf Project (Part 2) Report
Master of Informatics
School of Informatics
University of Edinburgh

2021

Abstract

This report describes the development and improvements made on CLPractice, a tool made to aid the teaching of the Inf1A: Computation and Logic 1st-year Informatics course, which was designed and partially implemented last year as a part of this two-year project. It is a web application written in Python with the help of the Django framework and several other essential libraries. The tool provides a framework for designing random question generators in Python as well as an interface for answering these questions. While these generators are showcased on the Inf1A course, the tool provides a generic Course module that can be used to design any number of entirely different Questions. Student answers are stored in the database to give the course organiser a set of statistics which may help in the teaching of the course, as well as designing tutorials and exams. The tool has been evaluated on around 180 Inf1A students during the December revision week 2020 and nearly 5,000 answers were recorded. The evaluation showed significant usefulness of the tool for practice and revision, and the students' eagerness to use it before their final quiz.

Acknowledgements

The project report structure is loosely based on the structure from (Hepburn, 2017b), but does largely diverge where appropriate.

I would like to thank Professor Michael Fourman, my supervisor, for the ”Duolingo clone” idea which sparked this project, and I am grateful for all the help along the way.

Table of Contents

Chapter 1

Introduction

This is the second part of a two-year project focusing on creating the CLPractice revision and practice tool for the Informatics 1A: Computation and Logic (Inf1A:CL) course. It is now publicly hosted on **`https://cltools.inf.ed.ac.uk/`** using a server provided by the School of Informatics of the University of Edinburgh.

The tool is a web application offering a number of questions and lessons the students can practice at their own pace or as a supplement to tutorials. Each question implements a generator written as a Python class, which generates a new random instance of that question every time the student chooses to practice it. This ensures the student gets a varied set of questions while potentially studying only a single key concept. To keep track of which instances have been answered before and to have the ability to re-generate a particular instance in the future, questions are generated using an integer seed value, which is stored in the database along with the provided answer. The student can select any number of questions or a single prepared lesson to practice.

Last year, the tool was developed to the point where it was functional and practically tested by several students taking the course. This year, I built upon the existing implementation, completed my 2nd-year goals, and extended the tool in terms of both in terms of features and functionality, and also increased the quantity and variety of question generators. The `generic` Course module has been refactored and cleaned up, and the Validators used for transforming and validating user input have been greatly improved, with the addition of using custom templates for the input elements. A detailed discussion of these improvements as well as specific examples of Question generators will appear in chapters 2 and 3.

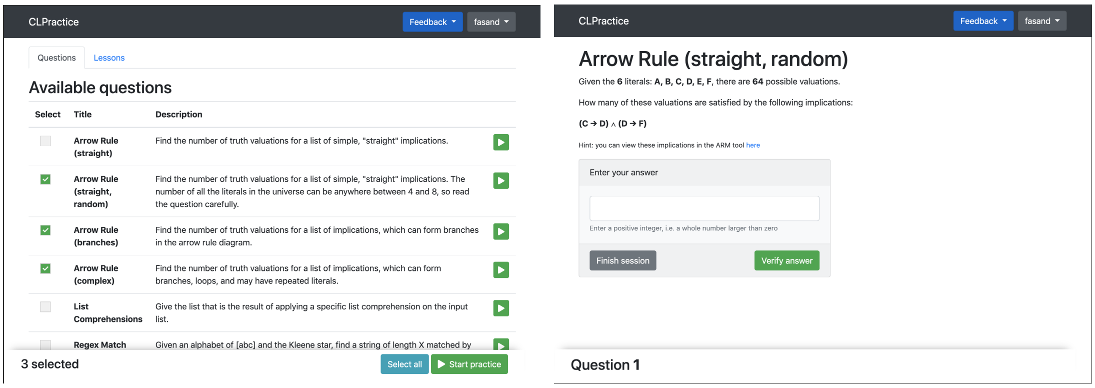
The tool was evaluated during the revision week, just before the final quiz for the course. Nearly 5,000 answers have been recorded from around 180 students and the gathered data has been analysed in chapter 4. From this data, a new module for the tool stemmed called `organise`, which displays a summary of all the recorded answers to the course organiser. This module and how it could be improved beyond the scope of the project is discussed in chapter 5. The first section will discuss the work that was done last year, and the following sections will discuss work done this year.

1.1 Previous work carried out (2019-2020)

The first part of this project (Manas, 2020a) concluded with having a functional tool to aid Inf1A CL (Computation & Logic) students in practicing key concepts and revising for exams. A relatively small but varied set of question generators was designed which could be useful to the students throughout the semester rather than only for exam revision. The way question generators were implemented allowed for simple extendability and further development in the future, as well as safe and reliable code execution. Several goals were set for this year of the project, which will be discussed in section 1.5 and reviewed in Conclusions. This section will discuss work done in the previous year on the tool itself, as well as the `generic` and `cl2020` modules.

CLPractice Tool

A web application was developed in Python using the Django framework consisting of only a few pages visible to the student. These were the pages that allowed the student to select questions or a lesson to practice (figure 1.1a), and the entire practice workflow. When a practice session was started, the user was guided through several views which would first prompt the student for an answer (figure 1.1b) and then evaluate their given answer. The state of the practice session was stored using Django's sessions middleware which stores the data in the database and provides the user with a session id cookie. At the end of each practice session, the student was shown a summary page showing which questions they got right or wrong.



Available questions

Select	Title	Description
□	Arrow Rule (straight)	Find the number of truth valuations for a list of simple, "straight" implications.
☑	Arrow Rule (straight, random)	Find the number of truth valuations for a list of simple, "straight" implications. The number of all the literals in the universe can be anywhere between 4 and 8, so read the question carefully.
☑	Arrow Rule (branches)	Find the number of truth valuations for a list of implications, which can form branches in the arrow rule diagram.
☑	Arrow Rule (complex)	Find the number of truth valuations for a list of implications, which can form branches, loops, and may have repeated literals.
□	List Comprehensions	Give the list that is the result of applying a specific list comprehension on the input list.
□	Regex Match	Given an alphabet of [abc] and the Kleene star, find a string of length X matched by

3 selected

Arrow Rule (straight, random)

Given the **6** literals: **A, B, C, D, E, F**, there are **64** possible valuations.

How many of these valuations are satisfied by the following implications:

(C → D) ∧ (D → F)

Hint: you can view these implications in the ARM tool here

Enter your answer

Enter a positive integer, i.e. a whole number larger than zero

Question 1

(a) Questions selected on the Dashboard

(b) An arrow rule question being answered

Figure 1.1: The Dashboard at the end of the first part of the project with a few questions selected for practice and one of these questions being answered.

Administrative Views

The administrative part of the tool was implemented largely using Django's generated admin interface. This interface was enough to look through question answers but not to test generators, so an additional view was designed specifically for this purpose. It allowed running questions with a random or a specified seed, essentially serving as a debugging version of the practice interface, which did not store the provided answers in the database.

Question Generator Set

Nine question generators and a single lesson were implemented within the `cl2020` course package, serving largely as a proof-of-concept rather than a full syllabus. The generators covered the arrow rule, regular expressions, and truth valuations for And/Or Boolean expressions, giving enough variety for a showcase of the tool and providing good grounds for further development this year.

Generic Course Interface

All implemented question generators and the entire `cl2020` package followed a generic course structure I designed last year after evaluating several different possibilities. This structure was provided by a set of Python classes that described how a course, question, and lesson should be defined and implemented. The generic interface or structure allowed for simple generator implementation with good reliability.

1.2 Background

The summary of existing tools is quite similar to the version in the year one report (Manas, 2020a), while the literature review is largely novel.

1.2.1 Existing Tools

Several tools have served as an inspiration to this project or have a similar purpose to my tool. Each tool, however, focuses on slightly different aspects of the problems I am solving and differs from my tool in one way or another. Where a citation is not explicitly given, a hyperlink to the tool is included in the tool's name.

Duolingo and Memrise are tools predominantly focused on learning languages but Memrise, specifically, offers courses on other topics as well. The tools offer countless questions and lessons for students to practice, but these are usually not randomly generated but manually defined. They were the original inspiration for making a similar tool, teaching concepts efficiently and in an enjoyable way, but the concept of CLPractice has taken several turns since then.

Anki is a tool designed for revision using flashcards with the help of spaced repetition. It is predominantly used for learning languages and similar heavily memory-based topics, such as definitions. It served as an inspiration for this project because of its simple aesthetics and the general design of questions by various Anki users.

FSM Workbench was designed and implemented by Hepburn for the ”Tools for Learning: Computation and Logic” project in the past, also spanning a two-year MInf project. In (Hepburn, 2016) and (Hepburn, 2017b), Hepburn created a tool for simulating finite state machines (FSMs) and teaching the associated concepts through a set of interactive questions. Although I ended up focusing on implementing Question generators for Karnaugh Maps and Gentzen rules instead of FSMs, this tool remains an option for future integration into CLPractice as it is open source and accessible on (FSM Workbench Github).

Karnaugh mAPP is another past project tool for learning Karnaugh Maps, along with several questions designed for this purpose Mikolajczak (2018b). It served as an inspiration in designing the Karnaugh Map question generator and the widget used to answer these questions, which are discussed in detail in section 3.4. This tool is also available on (Karnaugh mAPP Github).

HaSchool is one of the most recent tools to have come out of this project in the past, developed by Vazbyte (2019). It focuses on the Haskell aspect of Inf1A, hence providing great inspiration for how my tool could potentially be used for other topics than logic.

STACK is an online assessment package for mathematics developed by Chris Sangwin of the University of Edinburgh. It is a computer-aided assessment tool, similar to past tools like AiM, CalMath, or Wallis Sangwin and Grove (2006). It much more closely resembles CLPractice in the sense that its questions are also randomly generated and students, therefore, have a wide (infinite) array of possible exercises to practice on. The research behind this tool further informed the development and improvement of this tool, which is discussed in the following section. In the context of mathematics alone, STACK is certainly more advanced than my tool is striving to be, hence for mathematical applications, it is a better solution. However, the key difference is that STACK is focused primarily on questions for mathematics, whereas my tool can accommodate any sort of questions that can be generated using Python. This means that in terms of generality and extendability, CLPractice surpasses even STACK, and there are no other tools I know of that would allow this as well as mine.

Gentzen-specific Tools There are some available tools specifically designed for solving and teaching Gentzen (generally, inference) rule questions. Each of the ones I had considered had some strengths and weaknesses, which will be discussed in detail in section 3.5.1 rather than here so that it appears closer to my implementation.

1.2.2 Literature Review

Computers in Education & E-Learning

Although computers have been used in the classroom for several decades now, they have only moved towards the mainstream around the late 2000s, when the iPhone was launched, the first massive open online course (MOOC) was offered, and many new e-learning platforms were being created (Zawacki-Richter and Latchem, 2018). Over the coming years, these platforms like Udacity, Udemy, Coursera, Skillshare, and countless more became incredibly popular.

One of the aspects these MOOCs and other courses have in common with this tool is the freedom it offers the students. Kidd (2010) says that ”E-learning is free from limitations of space and time", and this is largely what makes it so accessible to students, especially when considering the recent sudden transition to remote learning due to the Covid-19 pandemic. Tools like CLPractice allow its users to take their own path through the subject material, do as much or as little as they want, and do so whenever and wherever they want.

Although most of the research on e-learning seems to relate to mobile readiness and MOOCs or similar platforms (Zawacki-Richter and Latchem, 2018), there have also been a number of studies on using intelligent tutoring systems, which are more relevant to this tool. Kulik and Fletcher have found in their review that the median effect of these systems was to raise test scores 0.66 standard deviations over conventional levels (Kulik and Fletcher, 2016). Regardless of the actual numeric improvement, these findings support the claim that a tool such as CLPractice could be a useful addition to the students' toolset.

Generating questions using Context-Free Grammar

There have been several articles and research papers written on generating simple sentences using context-free grammar (CFG). The grammar essentially describes a set of rules for a small language, which must be followed. This is particularly useful for checking that a given sentence follows the grammar of a language, but the problem can also be flipped and new sentences can be generated.

Both Purdom (1972) and Maurer (1990) found that this generation is possible for simple sentences. In the context of question generation, the question could therefore be written as a set of grammar rules, and an algorithm would generate random instances of each question.

Unfortunately, this approach does not allow a universal question generation but is largely limited to relatively simple problems. Moreover, then computing and verifying the answer to the generated question cannot be done without some amount of code, making this approach inappropriate for my application.

Computer Aided Assessment

One of the ways to overcome the limits of a simple CFG is to use a system that resembles grammar but is built with an emphasis on understanding and evaluating algebraic problems, like the ones often used in the computer-aided assessment. There has been significant research in computer-aided assessment (CAA), a lot of which has been done by Sangwin and Grove while developing the STACK system. Just as preceding tools such as AiM or CalMath, the system uses a computer algebra system (CAS), specifically Maxima, as the base for understanding student answers and generating questions.

Sangwin (2007) states that provided response questions (often multiple-choice questions) are almost always a constraint dictated by the CAA software rather than the preferred choice of the user. What STACK set out to do was evaluate student-provided answers rather than give multiple choices and have the student pick the correct one because the process of doing so and the experience for the student is significantly different in each case (Sangwin, 2015a). The system also provides the users with quite detailed feedback when they get the question wrong, which was reported to be one of the most liked things about STACK in Sangwin (2015b).

One of the main benefits of using computer-aided assessment is the economy of it: the more users use the system, the more economical it becomes because the development cost stays the same (Keady et al., 2012). This is particularly relevant when considering the Inf1A course with hundreds of students each year, all of whom can benefit from my tool.

The one important downside of using a computer algebra system, however, is within the name: *"algebra"*. These systems are focused on mathematical problems, and although they could be slightly skewed towards logic problems, a CAS alone would never be enough for the questions relevant to this course. Most significantly, this applies to question contexts requiring complex input or output, such as Karnaugh Maps, Finite State Machines, or Sequent Calculus.

1.3 System Overview

The tool is a web application written in Python using the Django 2.2 framework and a PostgreSQL database. The vast majority of user interfaces and question templates are designed using Django's powerful templating system, and a single dashboard view is designed with the help of the React JavaScript framework for better flexibility. A number of Python packages are also used for the purposes of random number generation (*NumPy*), passwordless login (*django-sesame*), Boolean logic (*PyEDA*), and more.

The application is split into several modules, which Django often refers to as *"apps"*:

- `core`: Responsible for user management, feedback and consent gathering, and providing the base models for other modules.
- `practice`: Implements the practice workflow, displays the Course contents, and stores user Answers.
- `organise`: Displays a summary of user Answers and gives insights to the course organiser; further discussed in chapter Course Administration.
- `generic`: Describes an interface for the Course, Question, and Lesson classes, while providing the code to load and execute these objects.
- `cl2020`: An implementation of the `generic` module for the purposes of Inf1A Computation and Logic. Both the specific and generic modules will be discussed in detail in chapter Question Generator Improvements.

1.4 Terminology Used

The reader may have noticed already that the terms Question, Lesson, and Course are sometimes being used capitalised and in other cases lower-cased. This is because I want to emphasize the nuanced distinction between the general meanings of these words and the representations I have designed in the context of this project. While

”question”, ”lesson”, and ”course” refer to their respective meanings in the English language, ”**Question**”, ”**Lesson**”, and ”**Course**” refer to the *Python classes or objects of these classes* which are used to describe these concepts in code. A Question is a class used to abstract the idea of a question (e.g., the Arrow Rule) using code that generates random instances of such a question. In this sense, every *Question* is an implementation of a *question*, but only a few types of *question* as *Questions*.

1.5 Project Goals

The project goals for this year were set at the end of last year:

- Semester-Wide Testing With New Inf1A Students
- Course Administration
- Making the Tool Course-Agnostic
- Other General Improvements

These goals will be discussed throughout the report and reviewed in Conclusions.

1.6 Summary of Work Done

System Improvements

Since last year’s implementation focused on the necessary functionality and having the tool tested, several areas have been purposely neglected. While last year, users ”logged in” using a participant ID, this year proper user management was implemented. Users can now log in using their email (or as a guest) without the need for a password. Improvements were also made to the `practice` module, along with better randomness for practicing Lessons. The tool has been prepared for Apache deployment on a University server and then deployed on `https://cltools.inf.ed.ac.uk/`.

Question Generator Improvements

The `generic` module has been refactored as well as improved towards customisability. Validators now support custom inputs other than a simple text field, and Questions can be grouped in folders while sharing common code with other parts of the Course.

New Questions

Several Question generators were implemented in order to showcase the new abilities of the `generic` module. The key ones include Karnaugh Maps (section 3.4) and Gentzen Rules (section 3.5). These generators accounted for a significant portion of this year’s efforts and will be discussed thoroughly in the aforementioned sections.

Automated Testing

A coherent suite of tests was written to provide code consistency and assurance for future development. These include unit tests for specific modules and classes, as well as integration tests for the key workflows a user encounters while using the tool. A statement coverage of 93% and branch coverage 91% was achieved using this test suite while covering all the important parts of the application. The reasons for a below-100% coverage and a further discussion of what was tested appear in section 4.1.

Real Scenario Evaluation

Once the tool was deployed, it was advertised to the current students of Inf1A for their use in practice during revision week. During this single week, nearly 5,000 answers were recorded from around 180 unique students. This data was then analysed in chapter Evaluation and used to guide the implementation of a new `organise` module.

Course Administration

This new `organise` module now serves as a dashboard for course administration. The gathered Answer data is visualised in tables and charts, giving the course organiser greater insight into how their students are doing, which Questions are the most popular or difficult, and how they could tailor their teaching to the students' needs.

1.7 Changes Made Due to COVID-19

Since this project required no physical access to specific hardware or lab spaces, the development of the tool was hardly affected by the pandemic. That being said, it had some clear effects on the deployment and evaluation of the project.

Although communication with the IT support has been established early last year, it took a significant amount of *back-and-forth* to get the University server up and running and deploy the tool. A significant portion of the time spent on this was caused by waiting for responses to queries due to the staff being overwhelmed. Since I lived abroad this entire year, solving these problems in person was not an option. Similarly, it took much longer to get an ethics approval for evaluation than it did last year, which meant I had to accommodate for a shorter time frame available for user testing.

More importantly, this testing then had to be done completely remotely and individually, with virtually no interaction between me and the students. In contrast to last year's evaluation, where I had the opportunity to be in a room with tens of students, demonstrate the tool to them, and gather valuable feedback in person, this experience was rather impersonal and I did not gather nearly as much qualitative feedback. Thankfully, as described in chapter Evaluation, the data and feedback I had gathered this year turned out to be quite useful nonetheless.

Chapter 2

System Improvements

2.1 User Login & Consent Gathering

Since last year the tool's primary purpose was to test it in an organized study, users could only log in using their assigned participant number or as a guest. This year, the tool needed to be ready for a much larger number of users (all of Inf1A), and participant number distribution would be unfeasible. For that reason, a proper login workflow was implemented so that each user could operate as a singular entity as well as give consent for anonymously using their answering data.

2.1.1 Passwordless Login

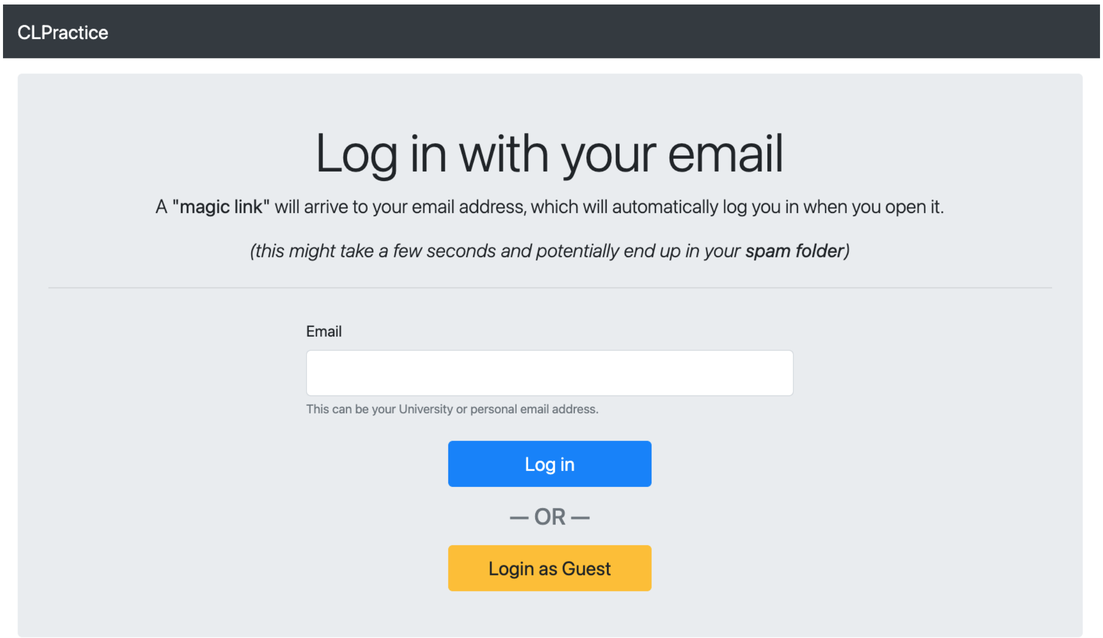
In order to make the login workflow smoother for students, a passwordless login was implemented using the *django-sesame*[1] package. Once the user enters their email address in Figure 2.1, an email with a ”magic link” is sent to them by email. When they click this link, they are automatically logged in without having to enter a password.

This means that a user is authenticated purely based on access to their email address, which would be required for registration either way. It also means users do not have to remember or store another one of their passwords, which greatly simplifies the login workflow. Since the magic link can log anybody in as the user who originally received it, to improve security, these links expire after 10 minutes. If a user only wants to test the application without using their email, they can also log in to a shared ”guest” account with a single click, which is then excluded from data statistics because consent cannot be correctly stored for multiple users on a single account.

2.1.2 Consent Gathering

In order to ethically store and analyze user data, an application has been submitted to the Ethics Board with RT 5333 and an ethics approval has been granted. Since paper consent forms could not be safely distributed to and collected from students, consent had to be given and stored electronically. To make the process as hassle-free for the

[1] Available on: https://github.com/aaugustin/django-sesame



CLPractice

Log in with your email

A "**magic link**" will arrive to your email address, which will automatically log you in when you open it.

(this might take a few seconds and potentially end up in your ***spam folder****)*

Email

This can be your University or personal email address.

Figure 2.1: User can choose to log in using their email address or temporarily as a guest.

users as possible, a consent form (Appendix A.1) was displayed on their first login, which linked to a participant information sheet (Appendix A.2) with all the necessary information.

The HTML form is a copy of the paper template along with an extra field asking if the user 18 years or older. Although each user is asked for consent at least once, they are free to select ”No” for all questions and their data will be excluded from the analysis. The answer to the form is stored in the database on each User object, one Boolean field per question. The date time of when the user last submitted the form is also stored, and users are allowed to change their answers if they decide they want to withdraw their consent.

2.2 Practice Module

The way a practice session is run is relatively the same as last year and can be summarized into the flow diagram in Figure 2.2. A user selects a number of questions or a lesson to practice at the Dashboard and sends a `POST` request to the Start view (using a hidden-form button). They are then redirected to the Session view which both displays the question and accepts a student answer upon form submission. When a user has answered that question and wants to move on, they send a `POST` request to the Next view which retrieves a new question and redirects back to Session. When they are finished with their session, they move on to the Summary view where they can evaluate their session and proceed back to the Dashboard.

As it was last year, the state management for this practice workflow is implemented using Django’s session storage. This ensures the state is not directly accessible to

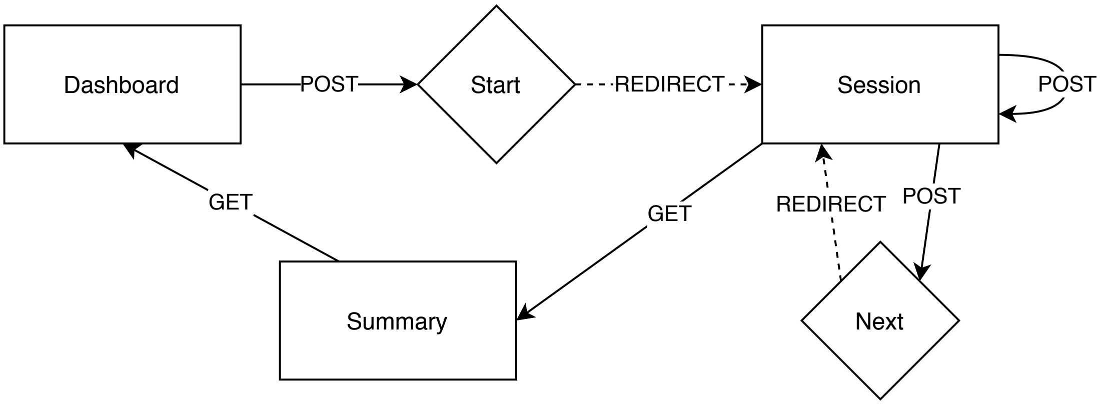


Figure 2.2: Diagram of the HTTP requests while running a practice session. Rectangles show views which show a template when loaded through a `GET` request. Diamonds show action views which can only be access using a `POST` request and then redirect to a different view. (Manas, 2020a)

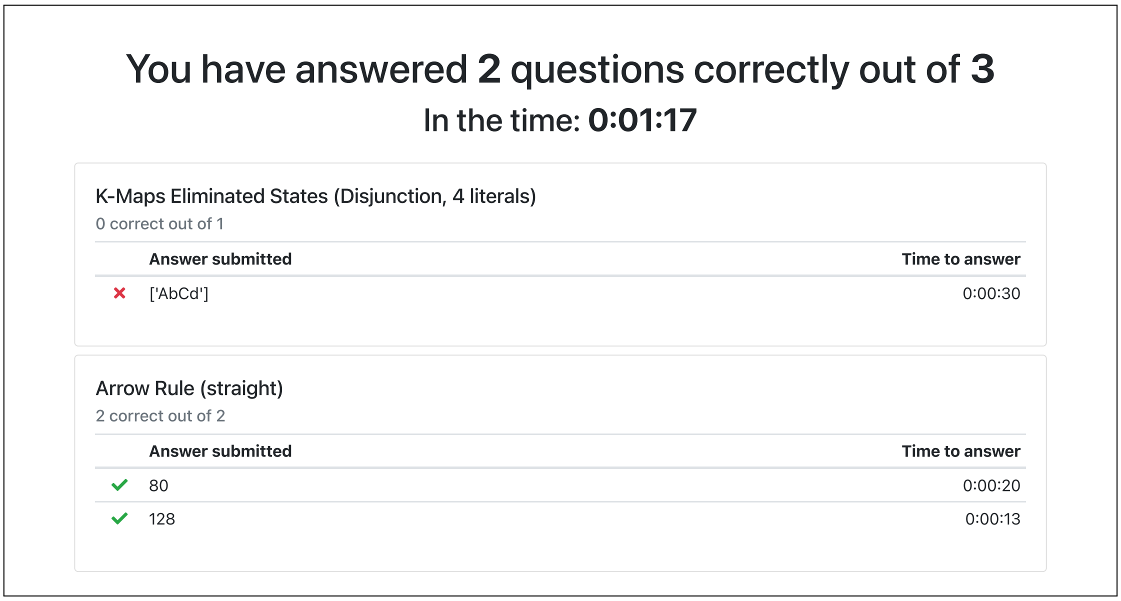
the user and persists across browser restarts and page reloads. Another significant benefit to this approach that became more relevant this year is its ease of testing. With the addition of an automated test suite 4.1 I can simply check the system is running correctly whenever I make a change. Since state is managed purely by Django sessions, the unit tests have direct access to its values before and after performing any action, which makes it easy to check for consistency and correct assignment. If state was managed by React and Redux instead, for instance, a separate testing suite would be necessary for that specific application, which is certainly feasible but also unnecessary within the current setup.

2.2.1 Visual Improvements

Although the visuals of this workflow have not changed significantly since last year, one notable improvement is the inclusion of ”Time to answer” in the Summary view (Figure 2.3). This small addition now allows the user to see not only how they are performing but also how long it takes them to answer each question and all questions combined, which can serve as a useful metric when preparing for a timed quiz.

2.2.2 Better Randomness for Lessons

To ensure the user does not get the same question multiple times in a single practice session, every time a question is generated, a checksum of its *context* is stored in the session. The *context* is what is generated by a question generator and then used to display that question and verify the user's answer. Therefore, if two different seeds generate the same context, the resulting questions will be identical, which is what we want to avoid. A `sha256` checksum is hence stored for every question generated within a practice session and before a new question instance is given to the user, the system checks it has a unique checksum, otherwise it keeps trying with new random seeds until that happens.



You have answered **2** questions correctly out of **3**

In the time: **0:01:17**

K-Maps Eliminated States (Disjunction, 4 literals)

0 correct out of 1

	Answer submitted	Time to answer
✗	['AbCd']	0:00:30

Arrow Rule (straight)

2 correct out of 2

	Answer submitted	Time to answer
✓	80	0:00:20
✓	128	0:00:13

Figure 2.3: The Summary page displayed when a user decides to finish their practice session. (In)correct answers are shown, as well as the time it took to answer each question.

While this was already the case last year for *questions* mode, it has now also been implemented for *lesson* mode. I originally expected it would be nearly impossible for a lesson to display the same question instance twice because lessons usually contain a variety set of questions, each of which only appears a few times, but some interactive testing proved me wrong. This ensures that randomness is now provided for both question and lesson practice.

How to make this even better One issue that was briefly discussed last year was the fact that checksums of Question context are not always enough. The Arrow Rule Question instances $A \rightarrow B$ and $B \rightarrow C$ are essentially identical but inevitably have different checksums because different literals are used. Since every Question generator is different and often two different instances can have the same answer, it would be nearly impossible to get a generic measure of the *”distance”* between two Question instances. Instead, a method such as `compute_distance(context1, context2)` or `equals(instance1, instance2)` would need to be implemented for every Question generator. Since this would need to be done manually and the method implementation might not always be entirely clear, I decided against this solution and instead rely purely on the fact that two instances are not identical. If this proved to be a greater issue in the future, this solution could be easily implemented, but all Question generators would need to be updated.

2.3 Deployment

After a few months of back-and-forth with the University's IT support (further discussed in section 4.2), my tool is currently hosted on a University server at the address:

https://cltools.inf.ed.ac.uk/. The server has 2 cores and 4 GB of RAM, which is more than sufficient enough for the current user load, although, at times when many students access the tool at once, the 2 cores get fully utilized.

Since I could not get root access to the machine, my options for deployment were strictly limited to running an Apache server with the mod_wsgi[2] module, which enables Apache to run a WSGI Python application such as a Django server. This restriction also means the application is limited to use Python 3.6.8, which is the latest version installed on the server. For that reason, Django 2.2 is used instead of the latest 3.1 release, and other minor issues with Python coverage had to be resolved. The tool is then connected to the pgresearch PostgreSQL server, which provides a secure and stable database connection.

Better deployment for the future Although the current deployment setup has proven to work sufficiently, it is far from ideal for frequent updates. The server does not have git installed, hence the current update process is to pull the repository on the ssh gateway and then copy the files over to the Apache directory. Having the latest Python and Django versions would also be beneficial for future-proofing and new tools like Memcached[3] could be integrated to offer better speeds and higher reliability. Therefore, should the project continue beyond my last year of study, it should ideally be *dockerized*[4] and deployed on a server with Docker installed. Updating the tool would then be significantly smoother and more consistent, as well as independent of the physical server it would run on.

[2] https://modwsgi.readthedocs.io/en/master/

[3] https://memcached.org/

[4] https://www.docker.com/

Chapter 3

Question Generator Improvements

Although the practice module is what the user interacts with when using the tool, question generators are the heart of the tool which provides the substance of user interaction. Every question instance presented to the user is generated using Python code which can use arbitrary installed packages and displayed using Django's templating system. To make the interaction between the practice module and question generators possible, each generator must implement a generic interface, or rather extend a generic class, which provides additional functionality to make development as simple as possible. This interface is encompassed within the generic module and the cl2020 module is an implementation of this interface for the purposes of presenting this tool with questions for Inf1A: CL.

While the practice module has only been improved upon (discussed in section 2.2), a large chunk of the question generator code this year has been rewritten or developed from scratch. Firstly, the general structure of the generator interface will be discussed, followed by specific improvements such as Validators, and finally, examples of specific generators implemented this year will be presented as a showcase of what can be accomplished with the generic interface.

3.1 The generic Module Structure

The generic module is a simple Python package that serves as the basis for writing question generators. It provides base classes for the Course, Lessons, and Questions, as well as the Validators that can be used within each Question or extended. It also provides a few utility functions useful for any generator, such as a ”coin toss” with variable probability or a de-duplication method for a list. Lastly, it contains the base Django HTML templates for the question form which is displayed for the user when answering a question, and for the basic <input> tag method of answer entry, both of which can be used as-is or extended or completely replaced in an implemented course.

This extendability provides the course designer with virtually limitless options – they can leave everything as is and focus on implementing generators, or if they have more specific needs (such as will be discussed in section 3.4), they can implement custom solutions. Consequently, the tool is not limited only to Inf1A but can accommodate a variety of different courses, only limited by the designer's imagination.

3.1.1 `class Course():`

At the very top of the hierarchy, we have the `Course` class, which defines the name and other meta-information about a course, as well as the list of implemented Question and Lesson generators. The course designer only needs to fill in this essential information and the base class takes care of the rest. It provides methods to load (import) Questions and Lessons based on their ID and a prescribed folder structure. It also provides methods to get a new unique Question instance based on stored checksums, as described in section 2.2.2.

Whenever a Course class is initialized, it loads all installed Questions and Lessons in order to verify they load correctly and without errors. Although this could be seen as a potential waste of computation when only a single Question is needed, it ensures consistency and only takes a fraction of a second, hence it has very little effect on performance.

3.1.2 `class Lesson():`

A `Lesson` class is as simple to implement as `Course`, since it is merely an aggregate of Questions. Apart from the basic meta-information, Lessons contain a `questions` list of dictionaries, which specifies a Question ID its number of occurrences in a row. The Questions are loaded in the order they are set, which enables the course designer to design Lessons that have the same format for all students, yet each Question instance is still randomly generated. The generic class only implements a method to obtain a Question at a certain index, so that the `practice` module needs only worry about how many total Questions appear in a Lesson and how many have been answered so far.

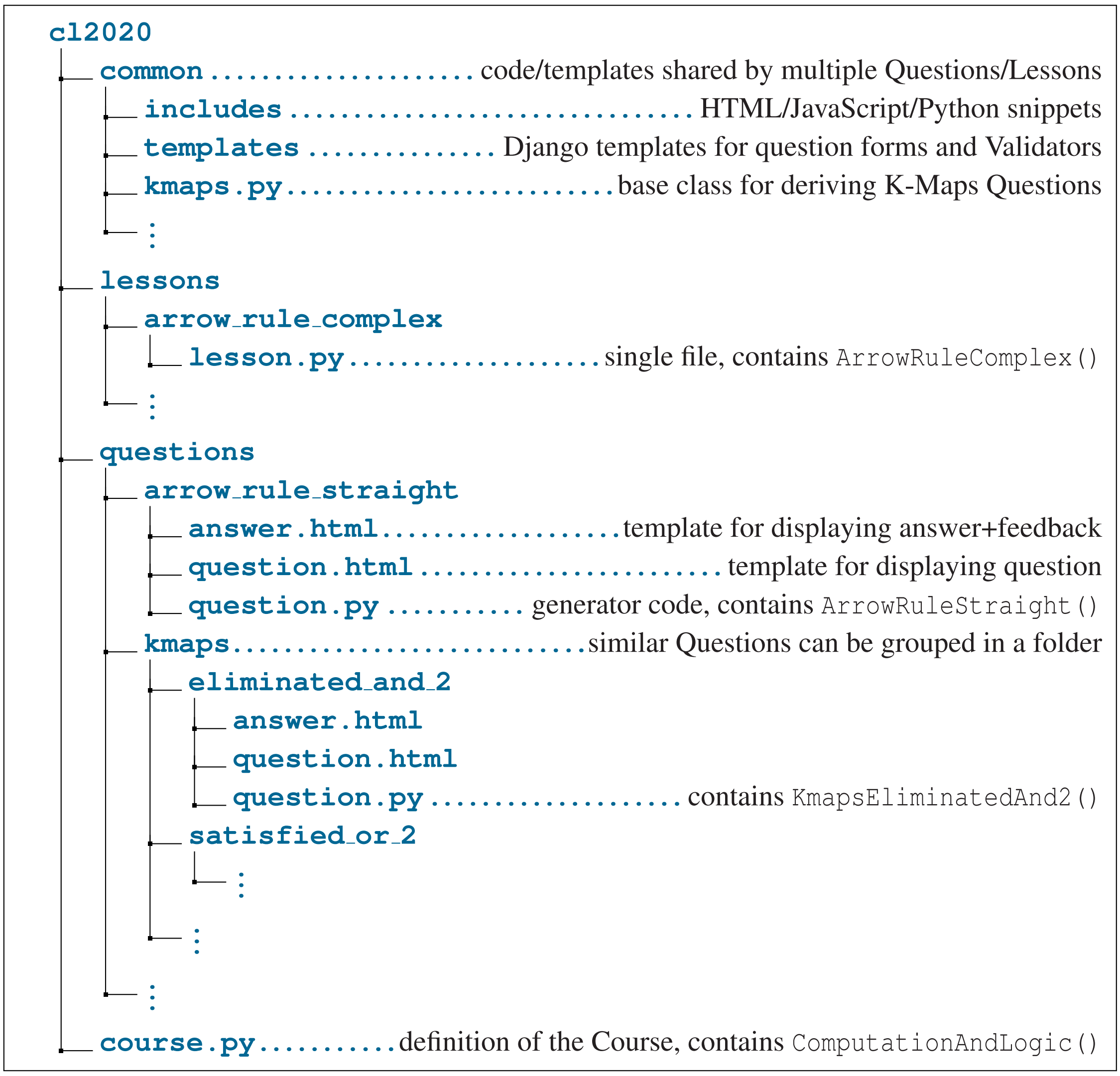
3.1.3 `class Question():`

Question generators are the vital building blocks of this hierarchy since they are the one thing that actually gets randomly generated. As with the other classes, it needs to define information like its title and description, which are then displayed to the user, and a correctly formatted ID as well as a version. The version is key due to the random nature of question generators since even a single line edit can cause all previously used seeds to generate completely different question instances. For this reason, it is stored in the `Answer` database model when a user answer is recorded in the format ”`id@version`”.

The Question ID, class name, and folder location must also be consistent for the Question to be dynamically loadable. For instance, a Question with the ID ”`arrow_rule_straight`” must appear in a folder with the same name and the class name must be ”ArrowRuleStraight”. In case multiple questions are grouped in a folder, both the ID and the class name must contain the group folder's name. For instance, the ”`eliminated_and_2`” question within the ”kmaps” group folder has ID ”`kmaps.eliminated_and_2`” and class name ”KmapsEliminatedAnd2”. When this structure is followed, it enables the Course to load all of its installed questions simply based on their IDs using the `importlib` package.

3.1.4 The Implemented `cl2020` Structure

To get a better sense of what a fully implemented course looks like, the folder structure of the `cl2020` course is illustrated in figure 3.1.



```
cl2020
├── common ..................... code/templates shared by multiple Questions/Lessons
│   ├── includes ................................ HTML/JavaScript/Python snippets
│   ├── templates .............. Django templates for question forms and Validators
│   ├── kmaps.py .......................... base class for deriving K-Maps Questions
│   └── ⋮
├── lessons
│   ├── arrow_rule_complex
│   │   └── lesson.py ................... single file, contains ArrowRuleComplex()
│   └── ⋮
├── questions
│   ├── arrow_rule_straight
│   │   ├── answer.html .................. template for displaying answer+feedback
│   │   ├── question.html ........................ template for displaying question
│   │   └── question.py ........... generator code, contains ArrowRuleStraight()
│   ├── kmaps.......................... similar Questions can be grouped in a folder
│   │   ├── eliminated_and_2
│   │   │   ├── answer.html
│   │   │   ├── question.html
│   │   │   └── question.py ................... contains KmapsEliminatedAnd2()
│   │   ├── satisfied_or_2
│   │   │   └── ⋮
│   │   └── ⋮
│   └── ⋮
└── course.py.......... definition of the Course, contains ComputationAndLogic()
```

Figure 3.1: Folder structure of the `cl2020` module including comments about important files.

By default, only the `course.py` file and `lessons` and `questions` folders are required for a functioning course, but often several different Questions or Lessons will use common features. These live in the `common` folder and include files such as HTML/-JavaScript/Python code snippets, templates used to generate question views and Validator inputs, or fully implemented question classes and methods that can be used to write many unique questions with a few lines of code.

While lessons are defined by a single `lesson.py` file within a folder corresponding to the lesson's ID, questions need two more template files and can be collected in folders. The sub-folders allow for similar questions to be collected close to each other, as well as potentially share code and templates specific only to this small selection of questions. The template files are then used to make the HTML displayed to the user using context values provided by the generator.

3.2 How To Implement a Question Generator

Now that we have a better idea of the file structure and class hierarchy of a Course, we can look at how the actual question generator is implemented and used within the application. While this was touched upon in last year's report, generators have been refined this year and no further changes will be made, so we can go into more detail now.

The key concept for talking about generators is the notion of a *"context"*. Since the generators are random by nature, the way they are discretized is using an integer seed value for a Numpy random number generator (RNG). Then, the generator code can use this seeded RNG to generate a unique instance of the question, which can be described by a single number, a list of items, or any other object. What describes a particular instance is therefore called the question *"context"* and it is used throughout the Question instance to generate the question text displayed to the user, the answer and feedback, as well as to verify a user answer against a ground truth.

Every question generator must implement the following methods:

`generate_context(rng):` The method receives a seeded RNG and uses it to perform random actions which lead to a unique instance of the question. For instance, in the case of the "Regex Match (fixed length)" question, this function returns a random regular expression and the length of the string a user needs to submit which satisfies the regex. The return of this function is stored on Question initialization and can later be accessed using `self.get_context()`.

`get_question_context():` Given the pre-generated context of the Question, this method can either simply pass it to the question template or pre-format it for easier display. For instance, if the context contains a list of tuples of literals, they can be formatted into a string showing they, in fact, represent a conjunction of implications. This is necessary because, even though Django's templating system is incredibly powerful, it is still limited in what it can provide.

`get_answer_context():` In order to be able to judge a user's answer to the question, we need to know what the correct answer should be, which is what this method provides. Given the Question's context, it computes the correct answer as well as preformats what gets passed down to the template, similar to `get_question_context`.

verify_answer(answer): This method receives the user's answer already cleaned by the Question's Validator (for instance, as an integer or a list) and checks whether it is correct or not. In most cases, this is as simple as checking equality between the answer and what `get_answer_context` returns, although more computation can be done here if needed.

3.2.1 Abstracting Questions Into Code

Although the generic course structure has been written with simplicity, clear structure, and rapid generator development in mind, it may not be clear just how a tutorial or exam question can be abstracted into Python code. This is one of the major conceptual problems of this project, and it had to be solved individually for each question simply because every question is a little different.

For some of the regular expression questions, a regex is used to generate a random regex, which in turn is used to generate a random string, for which the user is meant to find a regex that satisfies it. As convoluted as it may sound, this is one of the simpler ways a question instance is generated and the actual code is quite straightforward. Another less complex regex question picks a random length for the final expression, fills it with random letters from an alphabet of ”a,b,c”, and occasionally tosses a coin for whether or not a Kleene star (`*`) should be added.

Arrow rule questions, on the other hand, usually start with a set of literals that can be used (mostly 8 literals ”ABCDEFGH” for 256 possible valuations) and then need to generate a list of tuples of literals, which finally corresponds to a conjunction of implications for the question. Depending on how complex the question should be, e.g., whether or not implication loops and branches can occur, the generator needs to preserve the desired properties while still generating something random. This is done by generating the implications sequentially in a `for` loop and keeping track of which literals have been used already and which not.

The key takeaway from this section should be that writing a random question generator highly depends on the designer's imagination and abstraction ability and that many different ways exist to abstract the same problem. Two of the most complex question generators on Karnaugh maps and Gentzen rules will be described in further sections, but there is no strict limit as to just how complex a generator can be.

3.3 Validators

Similar to section 3.2, Validators have appeared in my last year's implementation but in a much more basic state. While Validators for the basic forms of input such as numbers or strings of text were provided, it was unclear how a custom Validator could be written, and more importantly, how it could be used to display a completely different form of input to the user. Examples of such custom Validators will be discussed in sections 3.4 and 3.5, while this section will focus on the general improvements of Validators and how their extendability was made possible.

The purpose of a Validator is to abstract out the validation and parsing of a user's input for an answer. Since all answers initially arrive as HTTP POST data, they need to be converted to their appropriate data type (e.g.: an integer) and check that they make sense in the Question's constraints. Validators have a `clean(data)` method to perform any necessary value parsing and to check constraints, which also calls `clean(data)` recursively on the extended class, finally getting to a base `Validator()`. The base class can then be extended and the designer can write a custom Validator, or they can use one of the pre-defined Validators for common cases such as constrained numeric input.

3.3.1 Improving Existing Validators

The main improvement was the streamlining of Validator extension and making it possible to use custom templates for new Validators. This was achieved by adding the `form` class variable and the `get_form_context()` method.

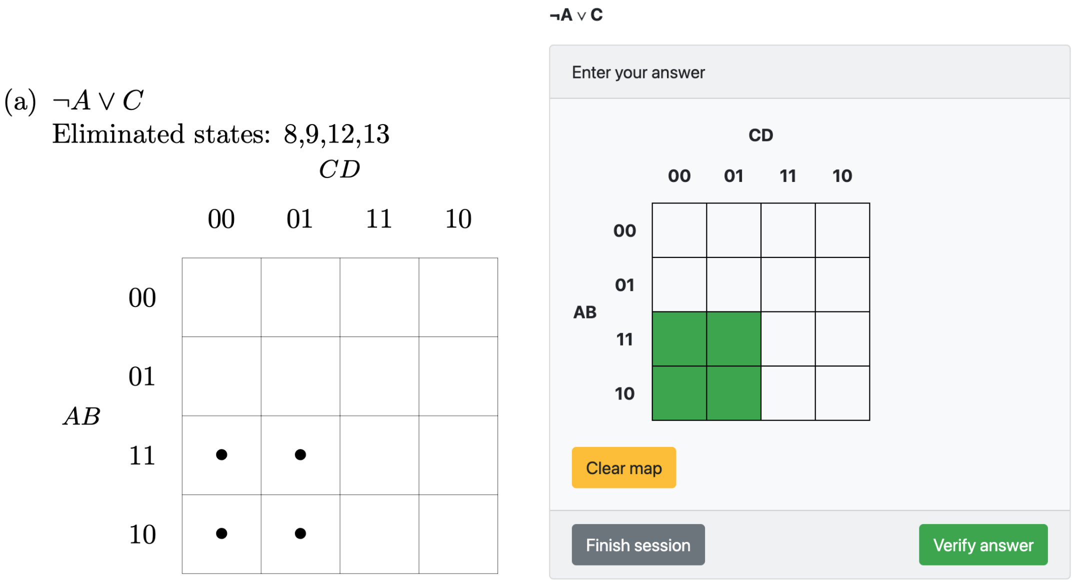
Every Validator needs to have `form` set to the template file with the HTML elements taking in user input. For classes inheriting from the `InputValidator`, this template contains an `<input>` tag with most of its attributes (e.g.: type, pattern, name) set to context variables provided by the Validator class. The `get_form_context()` method provides the values for this context depending on the nature of the Validator, i.e., if it is expecting an integer or a string satisfying a regex, and the initialized parameters for a particular Question. One of the improvements this year is that thanks to this method, when an `IntegerValidator` is used with minimum and/or maximum value set, the input help text contains this information, e.g., *"Enter an integer lower than 256"*. Similarly, the `RegexValidator` can also display the regex pattern it uses for validation or have it replaced with a manually-entered `help_text`.

3.3.2 Custom Input Form

Loading the Validator input form template and context from a variable made it possible to write entirely custom Validators which did not rely on a simple `<input>` tag. Since proper Django templates are used for this, the template can be as simple as plain HTML and as complex as a combination of back-end Python pre-formatting, native and custom template functions, template inheritance, and interactive elements such as images, generated LaTeX, and JavaScript. This means, again, that the Question designer's imagination and needs are the only limits when it comes to tackling user input.

3.4 Karnaugh Maps Generator

The first Question generator to take full advantage of these improvements was one for Karnaugh Maps (K-Maps). While questions on the Arrow Rule or Regular Expressions are more than happy with a simple number/text input field, K-Maps require a 4x4 input grid (for four literals) to be comprehensible. Figure 3.2 shows how K-Map questions typically look in tutorials and exams and how they have been closely recreated as a part of a custom Validator.



(a) $\neg A \vee C$
Eliminated states: 8,9,12,13

AB \ CD	00	01	11	10
00				
01				
11	●	●		
10	●	●		

¬A ∨ C

Enter your answer

AB \ CD	00	01	11	10
00				
01				
11	🟩	🟩		
10	🟩	🟩		

(a) Tutorial Solution

(b) CLPractice Question

Figure 3.2: Comparison of the question *"mark the states eliminated by $\neg A \vee C$"* in tutorial solutions and in CLPractice using a custom Validator input.

3.4.1 `KmapsValidator`

The input 4x4 grid is an HTML table with simple CSS styling and it is controlled by JavaScript; when a user clicks a table cell, the `selected` attribute is added and these cells are displayed with a green background. Each cell has a data attribute describing what literal valuation it corresponds to. For instance, `<td data-expr='abCD'></td>` corresponds to A and B being false and C and D being true. Although this is certainly not the only possible solution, it is one that worked very well for me, since it made it easy to compare the user's answer to the generated correct answer.

Since the user's answer is essentially encoded in a table rather than a form input, it needs to be mirrored in a hidden input field containing the selected cell data attributes separated by commas. When an answer is submitted, the Validator takes this hidden string and splits it on the commas to get a list of selected cells again. Thanks to template inheritance and extension, the same K-Maps table that is used for user input is used in the next step for displaying the correct answer as a part of the feedback.

3.4.2 A Generic K-Maps Question

Tutorials and exams most frequently ask for K-Map states **eliminated** by a **disjunction** of **two** literals. However, there are instances where the question asks for satisfied states instead, or the expression is a conjunction of more than two literals. In essence, it does not make much sense to design a Question generator for a single case, but rather it should be able to handle all cases.

The `KmapsBaseQuestion` abstract class does exactly this: it sets basic constraints which all extended Questions must adhere to, such as the literals available, and leaves three variables to be filled in: `connect`, `num_literals`, and `question_type`. The `connect` variable is set to a ”disjunction” or a ”conjunction”, i.e., whether the literals are connected with an $\vee$ or an $\wedge$. We can have 1-4 `num_literals` generated as part of the question, which affects how many cells will need to be selected due to the nature of the expression. And lastly, `question_type` is either ”eliminated” or ”satisfied”, in order to be able to cater to both types of questions.

The initialized preferences are then passed to the `Kmaps` class which generated random Question instances and produces correct solutions for them. Thanks to this abstraction, creating a specific type of such a Question is then a matter of 6 lines, hence many such Questions can be created. Figure 3.3 shows the implementation code for a K-Maps Question asking for states eliminated by a disjunction of two literals. Since the question and answer templates are also generated automatically, this is all that is needed. Of course, it must be clear that this simple implementation is enabled by the 230 lines of code for the generic implementation in `cl2020.common.kmaps`.

```
from cl2020.common.kmaps import (Kmaps, KmapsBaseQuestion)

class KmapsEliminatedOr2(KmapsBaseQuestion):
    _id = 'kmaps.eliminated_or_2'
    question_type = KmapsBaseQuestion.QUESTION_ELIMINATED
    connect = Kmaps.CONNECT_OR
    num_literals = 2
```

Figure 3.3: Code for the Question generator for K-Maps: ”mark the states eliminated by a disjunction of two literals”.

3.5 Gentzen Generator

Another clear candidate for a customised Question and Validator were Gentzen Rules, which require a Boolean expression tree structure to be rendered, ideally using LaTeX. Since the typical tutorial/exam question on this topic gives an entailment and asks the student to say whether it is universally valid or give a counterexample, we also need to consider two different types of possible inputs, not only a single custom input.

3.5.1 Existing Gentzen Tools

While this Question generator was built from the ground up, it should be noted there are several existing tools for Gentzen rules (or sequent calculus) available. The implementation approach I took is different from these tools in many ways, but they served as an initial inspiration and a reference of what has been done already.

UMSU Tree Proof Generator

The Tree Proof Generator by Schwarz (2021) takes an entailment or another Boolean expression and computes whether it is universally valid or not. If it is, it also generates a proof tree with all necessary steps; however, this is in a form significantly different from the Gentzen rules used in the course. The resulting tree does not use LaTeXbut rather HTML with additional CSS-transformed borders to display branches. Although for other courses, this might be a useful tool, its notation is not standard within Inf1A.

Seqcalc

The Seqcalc tool by Mehta (2019) takes a very different approach by suggesting the available rules within a language and letting the users instantiate them and then drag them into the proof tree. However, these rules are again more general than Gentzen rules, and the tool likely cannot be used for our purposes at all. On top of that, it has a long list of available shorthands for symbols such as implies, but these are hidden out of sight by default. All entered expressions also must use capital letters for literals, otherwise, it is not correctly parsed, which made the tool quite confusing to use.

Logitext

Logitext by Yang (2012), on the other hand, despite being the oldest tool on the list, is by far the best I have found. After entering an entailment using keyboard characters such as `->` for $\rightarrow$, the entailment is parsed and displayed in an interactive form. The user can then select the operator they want to apply a Gentzen rule to, and the tree grows upwards. This tool is certainly a great example of what an interactive version of my Question generator could look like, even if that is not its primary purpose at the moment. Because of this interactivity, the tool generates styled HTML operated with JavaScript instead of LaTeX, which seems to be an efficient approach. Since the tool is quite old and almost entirely written in Haskell, it would be quite challenging to integrate it into CLPractice despite it being open-source.

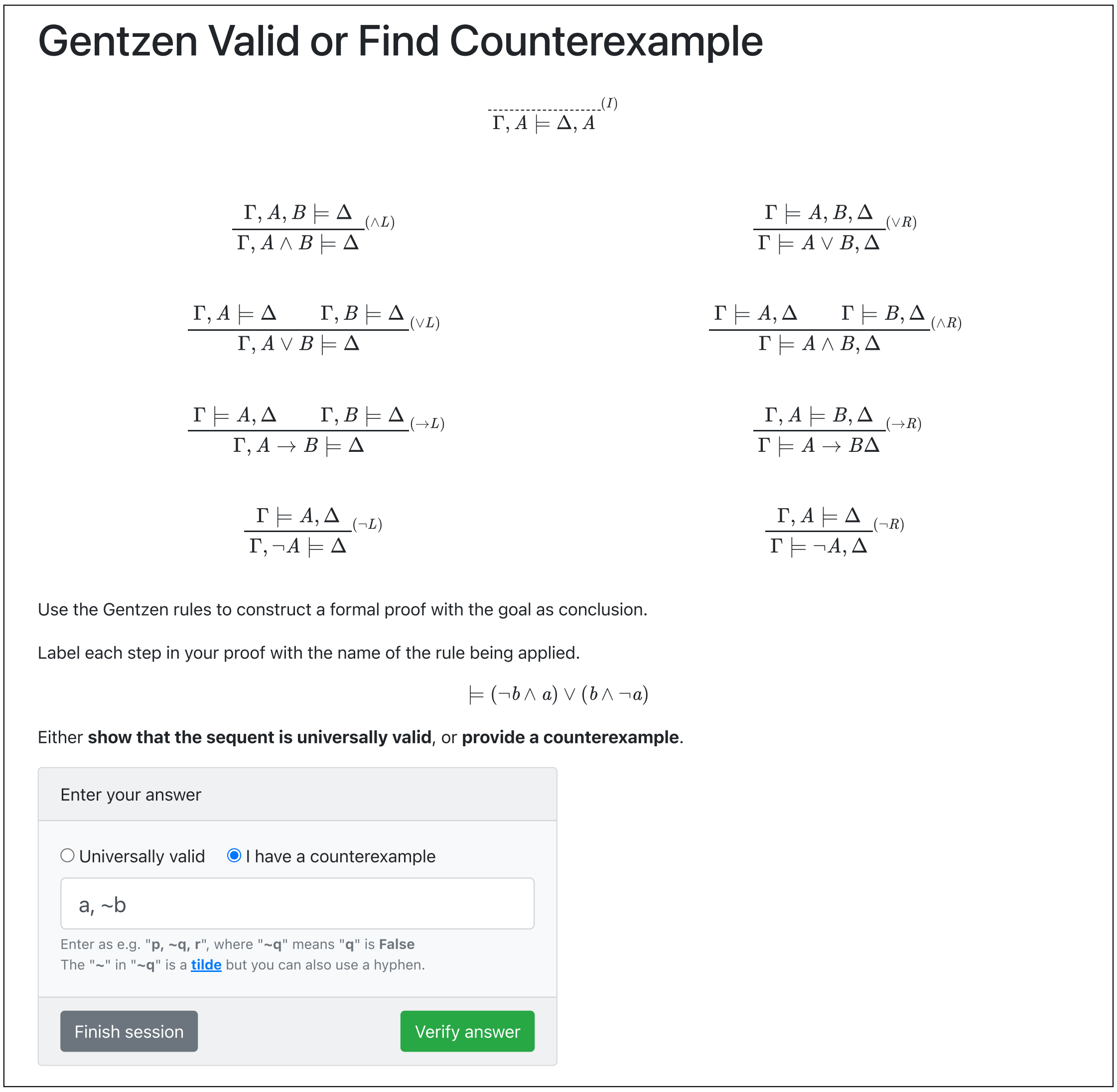
Wolfram MathWorld Sequent Calculus

As a notable mention, Wolfram MathWorld also seems to have support for sequent calculus (Sakharov), but it was unclear to me how to use it. This may be only offered as an explanation of the concept rather than an actual tool, or potentially this is only available in Wolfram's Mathematica system, which is not easily accessible to students.

3.5.2 `GentzenValidator`

The custom Validator class extends the `RegexValidator` and simply builds upon it. For the counterexample user input, the default regex pattern validation is used, while treating the unique case of a *universally valid* entailment differently through the use of a string constant *"VALID"* which can be entered instead of the counterexample.

To let the user simply select the entailment is universally valid rather than type *"VALID"*, the input form template is also extended to contain a set o two radio buttons and only displaying the `<input>` field when "I have a counterexample" is selected. In that case, the original regex input field is displayed since it fits the requirements well enough. This scenario is visible at the bottom of figure 3.4, where the HTML-formatted help text guides the user towards what input is expected from them.



Gentzen Valid or Find Counterexample

$$\frac{}{\Gamma, A \models \Delta, A}(I)$$

$$\frac{\Gamma, A, B \models \Delta}{\Gamma, A \wedge B \models \Delta}(\wedge L) \qquad \frac{\Gamma \models A, B, \Delta}{\Gamma \models A \vee B, \Delta}(\vee R)$$

$$\frac{\Gamma, A \models \Delta \qquad \Gamma, B \models \Delta}{\Gamma, A \vee B \models \Delta}(\vee L) \qquad \frac{\Gamma \models A, \Delta \qquad \Gamma \models B, \Delta}{\Gamma \models A \wedge B, \Delta}(\wedge R)$$

$$\frac{\Gamma \models A, \Delta \qquad \Gamma, B \models \Delta}{\Gamma, A \rightarrow B \models \Delta}(\rightarrow L) \qquad \frac{\Gamma, A \models B, \Delta}{\Gamma \models A \rightarrow B\Delta}(\rightarrow R)$$

$$\frac{\Gamma \models A, \Delta}{\Gamma, \neg A \models \Delta}(\neg L) \qquad \frac{\Gamma, A \models \Delta}{\Gamma \models \neg A, \Delta}(\neg R)$$

Use the Gentzen rules to construct a formal proof with the goal as conclusion.

Label each step in your proof with the name of the rule being applied.

$$\models (\neg b \wedge a) \vee (b \wedge \neg a)$$

Either **show that the sequent is universally valid**, or **provide a counterexample**.

Enter your answer

○ Universally valid ● I have a counterexample

a, ~b

Enter as e.g. "**p, ~q, r**", where "**~q**" means "**q**" is **False**
The "**~**" in "**~q**" is a **tilde** but you can also use a hyphen.

Figure 3.4: An example of a ”Gentzen Valid or Find Counterexample” Question, as displayed to the user.

3.5.3 Solving Step-By-Step

Gentzen inference rules, more than any other implemented Question generators, are about the journey rather than the destination. In many ways, these questions are about a few simple principles and then applying the inference rules in a sensible order. Because of this, it is often more important for the student to see how these rules should be applied and what the inference tree should look like instead of simply getting a correct answer to the question. For this reason, this Question generator has been designed from the ground up to solve the question just as a student would, while also keeping track of the steps performed before a result is reached.

Rules And Data Types There are currently 10 possible rules that can be applied to any given entailment: left and right hand variants of *Not*, *Or*, *And*, and *Implies*, the *Identity* and an *End* rule for when no more rules can be applied. An `Entails` object contains a *left* and *right* side, each of which is a list of expressions from the `pyeda` package. A `TreeNode` object is used to store the entailment at any given point, the rule that is being applied to it, and the result of that rule being applied. That result can be another `TreeNode`, a `TreeBranch`, or `None` if it is a final step. A `TreeBranch` is simply a `TreeNode` which has two separate results instead of one, hence is branched.

Recursive Reduction The production of the reduction tree (see figure 3.6 for example) is done using a single function `reduce_gentzen` which is called recursively on an entailment and uses 10 inner functions, one for each rule. These rules are applied in the order that minimizes unnecessary branches, i.e.: *Identity*, $\wedge L$, $\vee R$, $\neg L$, $\neg R$, $\rightarrow R$, followed by rules that produce branches: $\rightarrow L$, $\vee L$, $\wedge R$, and lastly the *End* rule which means no other rule can be applied. The first rule to return a non-empty result is marked as the correct/best one and is returned. Since at the end an *Identity* or *End* must be returned, all other rules call `reduce_gentzen` recursively on the result of the applied rule.

```python
def l_or(ent: Entails) -> Optional[TreeBranch]:
    left, right = ent
    for i, x in enumerate(left):
        if type(x) == OrOp:
            # Split into a and b
            a = x.xs[0]
            b = Or(*x.xs[1:])
            new_a_left = left[:i] + [a] + left[i + 1:]
            new_b_left = left[:i] + [b] + left[i + 1:]
            return TreeBranch(
                Rules.L_OR, ent,
                reduce_gentzen(Entails(new_a_left, right)),
                reduce_gentzen(Entails(new_b_left, right))
            )
```

(a) Code performing the $\vee L$ rule within `reduce_gentzen()`

$$\frac{\Gamma, A \models \Delta \qquad \Gamma, B \models \Delta}{\Gamma, A \vee B \models \Delta}\ (\vee L)$$

(b) The $\vee L$ rule definition, as given in an exam

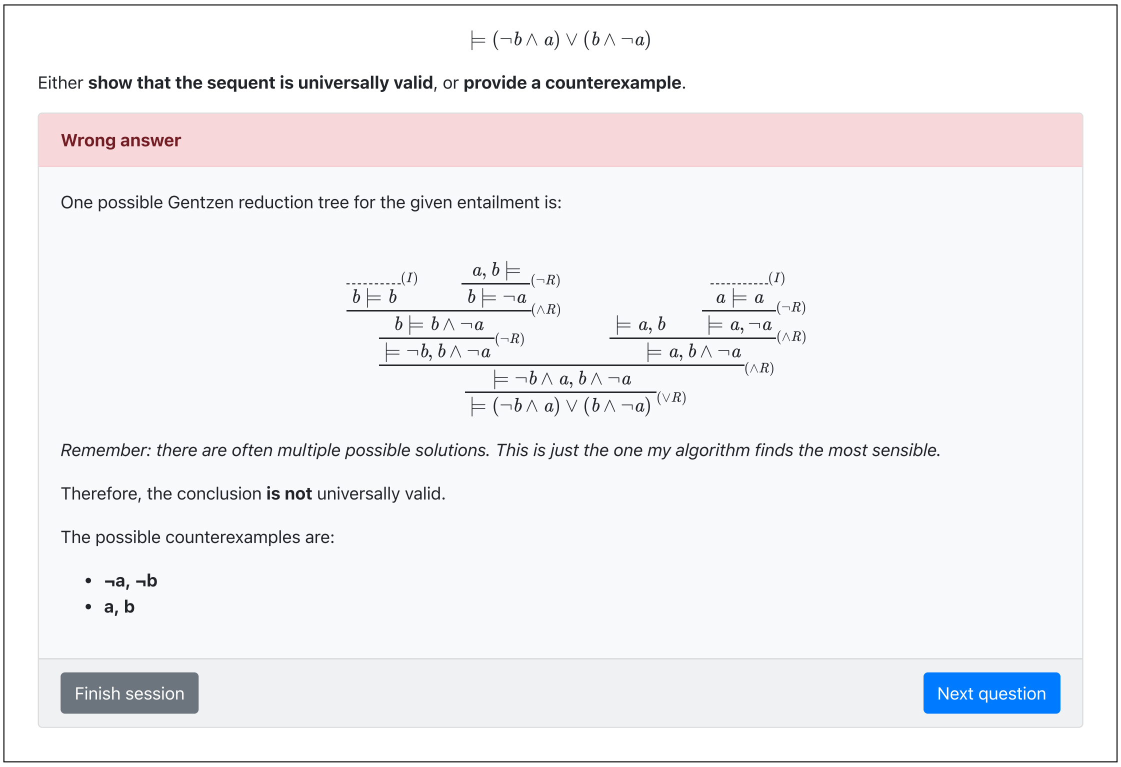
Figure 3.5: Example of a function within `reduce_gentzen()` for the $\vee L$ rule, which produces a branched result. After the two new branches are computed, the method is recursively applied to keep reducing until the end.

Figure 3.5 shows one of the rule function's code as well as what the common definition of the rule looks like. The function definition includes types thanks to Python's `typing` package, to make its input and output clear. It is not necessary to understand exactly what each line does but rather see the bigger picture of what such a rule function looks like since the functions for other rules look very similar and follow similar principles.

3.5.4 Rendering LaTeX

Producing a reduction tree in Python is one thing, but then displaying it to the user in a clear manner is another. Although some existing tools produce their own styled HTML to mimic LaTeX, in my opinion using native LaTeXwas the best option for my needs. The tree structure, branches, and rule annotations are generated using the `bussproofs` package, which was chosen partly because it closely mimics the style used in tutorials and exams. More importantly, though, it was available for import on MathJax, a JavaScript display engine for LaTeX, which enabled the code to be rendered on the client-side as well as allow the users to look at the source code itself if they desired.

MathJax JavaScript import with `bussproofs` installed is loaded from a CDN and a loading screen is displayed until the code is fully rendered. This is done using the plug-and-play `cl2020/common/includes/mathjax.html` file which can be included anywhere LaTeXneeds to be used. The result can be seen in figure 3.6 which shows feedback given to a wrong answer in the form of what a possible correct reduction tree could look like and what the resulting counterexamples are.



$\models (\neg b \wedge a) \vee (b \wedge \neg a)$

Either **show that the sequent is universally valid**, or **provide a counterexample**.

Wrong answer

One possible Gentzen reduction tree for the given entailment is:

$$\begin{prooftree}
\AxiomC{}\RightLabel{(I)}\UnaryInfC{$b \models b$}
\AxiomC{$a, b \models$}\RightLabel{$(\neg R)$}\UnaryInfC{$b \models \neg a$}
\RightLabel{$(\wedge R)$}\BinaryInfC{$b \models b \wedge \neg a$}
\RightLabel{$(\neg R)$}\UnaryInfC{$\models \neg b, b \wedge \neg a$}
\AxiomC{$\models a, b$}
\AxiomC{}\RightLabel{(I)}\UnaryInfC{$a \models a$}
\RightLabel{$(\neg R)$}\UnaryInfC{$\models a, \neg a$}
\RightLabel{$(\wedge R)$}\BinaryInfC{$\models a, b \wedge \neg a$}
\RightLabel{$(\wedge R)$}\BinaryInfC{$\models \neg b \wedge a, b \wedge \neg a$}
\RightLabel{$(\vee R)$}\UnaryInfC{$\models (\neg b \wedge a) \vee (b \wedge \neg a)$}
\end{prooftree}$$

Remember: there are often multiple possible solutions. This is just the one my algorithm finds the most sensible.

Therefore, the conclusion **is not** universally valid.

The possible counterexamples are:

- **¬a, ¬b**
- **a, b**

Figure 3.6: Feedback on a wrong answer to the ”Gentzen Valid or Find Counterexample” Question.

The LaTeXcode itself is generated recursively from the beginning `TreeNode` (the assigned entailment) to the final *Indentity* and *End* leaves. Since the initial entailment is at the bottom of the reduction tree which grows up, the returns from recursive calls are added on top of what has been gathered so far. While the code to perform this conversion is not particularly complex, it is also relatively long and not very informative. It and the rest of the Gentzen implementation are located in `cl2020/common/gentzen.py`.

3.5.5 Algorithm Imperfections

Even though the Gentzen algorithm has been tested on a large number of different entailments, including randomly generated ones provided by the generator, it has room for improvement. The most obvious issue that sometimes manifests itself is that the algorithm is smart, but not smart enough. In some cases, especially when one rule can be applied to multiple different elements, one of these elements is a better choice than the rest, while the algorithm simply selects the first one.

To give a specific example, take the entailment: $x, z \models x \land y, x \land z$. If the $\land R$ rule is applied on $x \land z$, it produces a single branch, where both sides immediately produce the *Identity*, hence the final reduction tree is produced in a single step. On the other hand, since $x \land y$ appears first on the right-hand side of the entailment, the algorithm decides to apply the rule to that instead, which means an additional $\land R$ rule must be applied and the resulting reduction tree is not ideal. While the effect of this imperfection is subtle in this example, on larger entailments, the algorithm can produce much wider reduction trees than are necessary, which start to overflow the bounding feedback container.

This could be solved, for instance, by trying every combination of the possible rules to apply and picking the one with the shallowest reduction tree (fewest branches and rules applied). While it would result in additional computation, which could prove to be an issue for more complex entailments, it is a realistic solution for this problem. On the other hand, showing the user a potentially imperfect but correct solution can be a part of the learning process since in most cases, several different reduction trees can be produced for the same entailment, all of which are correct.

Chapter 4

Evaluation

4.1 Automated Testing

Several unit and integration tests were written to guarantee consistency and reliability across code changes. The application is tested as a whole using the *URLAccessor* class I have written in the past for a work-related project (Manas, 2020b). The class, given a Django application name (e.g.: ”practice” or ”core” in our case), fetches all its registered URL routes, and given additional URL parameters accesses each of these URLs as a virtual user. Since Django runs tests on a fresh empty database, a new user must be created and logged-in in order to access all parts of the application. This part of the testing suite does not necessarily guarantee the code is running as expected but rather that no errors are preventing a user from simply loading a page.

The second, more important part of the test suite is located in the `tests.py` of each of the modules (core, practice, organise), and checks if each module is working correctly. This includes loading the installed Course and running its automatic checks, starting a new practice session and checking that the browser session is correctly updated, as well as running a full practice session including answering the generated question and viewing the Summary page. All question generators are tested by loading them through the organise module and answering them as a virtual user. Some of the tests also check for occasions when a bad POST request is sent and the application is expected to return an error message and redirect the user. The Course loader itself loads all installed Questions and Lessons when initialized to get the necessary meta information, hence the Course code is tested on nearly every request by default.

The statement coverage of these tests is 93% and branch coverage 91%. Although these numbers do not guarantee the code is tested well, they serve as a good measure of which parts of the code were given thought while testing and which could still be improved upon. Most of the code that is not yet tested is either exception handling for unlikely cases (e.g.: where the user manipulates a POST request or tries to otherwise break the application on purpose) and for unused code. This, for instance, includes some code for Validators that are not necessary for my generator purposes but could be used for new generators, or error handling for when a question generator is incorrectly defined and cannot be loaded, which is a burden placed on the course designer (me).

4.2 ”Semester-wide” Testing

Last year, I set one of my goals for this year to be a ”Semester-wide testing with new Inf1A students” in order to evaluate the tool in a real setting. At the beginning of the first semester of this year, the tool was ready to be deployed but the situation at that time made this harder than expected.

Although I have been in contact with IT support about hosting my tool on a University server since January 2020, communication came to a halt around March due to COVID-19. After reopening the ticket, a significant amount of ”back-and-forth” was necessary before the tool was finally deployed towards the end of November. Student testing has then been further delayed by the ethics approval for this study taking significantly longer than usual since the ethics board was overwhelmed.

For these reasons, the evaluation finally took place during the revision week, specifically from **04. 12. 2020** to **10. 12. 2020**. This was still favourable since the students had their final Inf1A: CL test on Thursday 10. 12., meaning they had a full week to use my tool for practice. Students logged in with their email of choice, upon which they were given a web consent form (Appendix A.1), which linked to a PDF of the Participant Information Sheet (Appendix A.2).

Results at a glance

During the 7 days, **4935 question answers** have been recorded from **179 distinct users** and 4 pieces of general text feedback were given. Students started logging in from the first day the tool was available but the vast majority of answers were recorded in the last two days of the evaluation period.

4.2.1 Quantitative Analysis

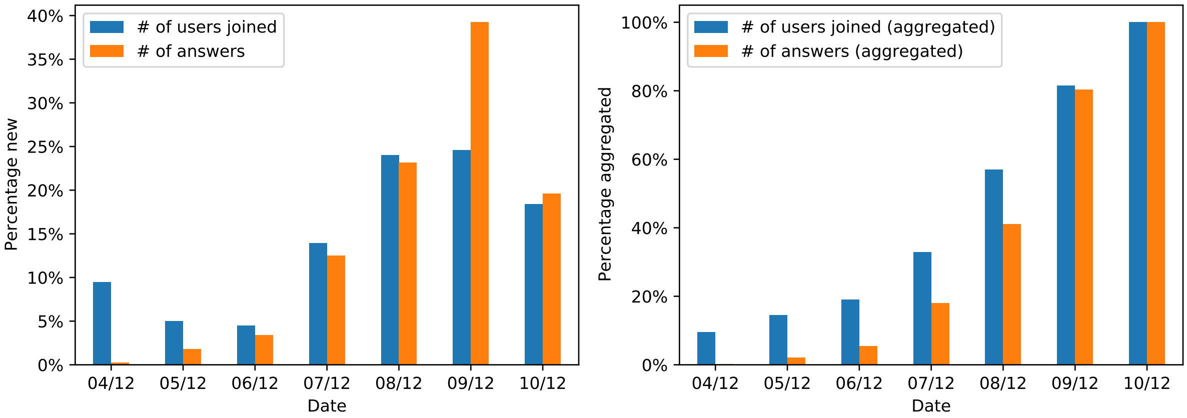


Figure 4.1: Percentage of students joining vs. percentage of answers by date, and the same data aggregated up to 100%.

Figure 4.1 shows what percentage of the final number of users joined on each day as well as what percentage of total answers were recorded. The right subfigure shows this data aggregated from start to finish.

Key observations from this figure are:

- Although by 08/12 over 50% of users have joined and logged into the tool, only around 40% of the total answers have been recorded.
- On 09/12, there was a leap in the use of the tool: nearly 40% of all answers (specifically, 1944), submitted by 67 unique users, were recorded on that day.
- This leap in the number of answers shows that the tool met its greatest success just before the final quiz on 10/12.
- Nearly 20% of users joined even on the final day of evaluation and used the tool to revise, and the percentage of answers is lower than usual since most users stopped revising around 16:00, with the last answer being around 20:00.

It is also notable that between 07/12 and 09/12, the last submitted answer of the day was within 15 minutes of midnight. In particular, the time of the last submission on 09/12 was 23:59:59, suggesting users were keen on using the tool as much as they could before the quiz.

4.2.2 Qualitative Analysis

Performing qualitative analysis was difficult since I couldn't interact with students in person. I had hoped that more students would use the feedback form in the tool and was a little surprised to see that did not happen. Due to this, frankly, bad assumption and planning on my part, other means of measuring the tool's effect on students' performance had to be found. Comparing this year's test results with last year's was not an option since the course was run very differently from last year, hence there were too many factors to consider. Instead, I opted to measure the students' change in performance while using the tool itself over time.

For each user, the *times-to-answer* (TTAs) of each Question were aggregated, grouped by Question ID, and ordered by the date time of when each answer was submitted. Since many Answers showed absurdly small or large TTAs due to the user being *away-from-keyboard* or submitting almost instantly, the 10th and 90th percentiles of TTAs for each Question across all users were computed, and only the TTAs falling in that range were considered. For each user separately, a linear regression model was fit to the progression of their TTAs for each Question and the linear coefficient extracted. These coefficients were then averaged across each Question and are displayed in table 4.1.

To my pleasant surprise, the linear coefficients for every Question except for two were negative, meaning **students got faster at answering the questions as they used the tool more**. Using an average as a measure means that the final results include instances where users got slower, yet despite this, the majority of the questions have seen an average increase in speed. The coefficient can be interpreted as an **average** change in seconds for TTA with each subsequent Question instance. When that coefficient is negative, it means that the average linear slope of that Question's TTAs is decreasing, hence users are getting faster by that number of seconds. It should be noted that this data is not rigorous statistics but rather an interesting metric for the success of the

Question ID	Linear coefficient
arrow_rule_branches	-28.51
arrow_rule_complex	-8.80
arrow_rule_straight	-14.48
arrow_rule_straight_random	-4.21
gentzen.counterexamples	-55.03
gentzen.random_counterexamples	0.09
kmaps.eliminated_and_2	0.81
kmaps.eliminated_or_1	-2.43
kmaps.eliminated_or_2	-1.16
kmaps.eliminated_or_3	-5.72
kmaps.eliminated_or_4	-1.39
kmaps.satisfied_and_2	-2.47
kmaps.satisfied_and_3	-2.43
kmaps.satisfied_or_2	-2.21
regex_any_length	-1.33
regex_find_expression	-3.98
regex_fixed_length	-0.89
truth_valuations_andor	-2.90

Table 4.1: The average coefficients from linear regression on the progression of time it took users to answer each question. The linear coefficient can be interpreted as the change in number of seconds for each new answer.

tool. For instance, it does not include whether or not the answers were more frequently correct with decreased TTA.

4.2.3 Evaluation Conclusions

The data collected over the revision week led to a very important conclusion: students find the tool useful and are keen to use it for revision before quizzes or exams. This is supported both by the sheer number of submitted answers and signed-up users, as well as by the spike in answering on the day before the final quiz. Table 4.1 also suggests that users' performance increases on average after using the tool. Since students were joining up to the very end of the evaluation period, this suggests that perhaps not all Inf1A students knew about the existence of the tool and would have benefited from it being available sooner. Having the students use this tool in their tutorials could have perhaps helped this onboarding process and given them time to get acquainted with it.

It also became clear that the data shown in figure 4.1 and table 4.1 was only a part of the story and other useful information could be gathered. In particular: what questions users struggled with the most, how long it took them to answer each question, and what was the success rate for each question. These questions could be answered in the Course Administration, and the course organiser could draw their own conclusions. The data collected through this evaluation made developing such a module possible.

Chapter 5

Course Administration

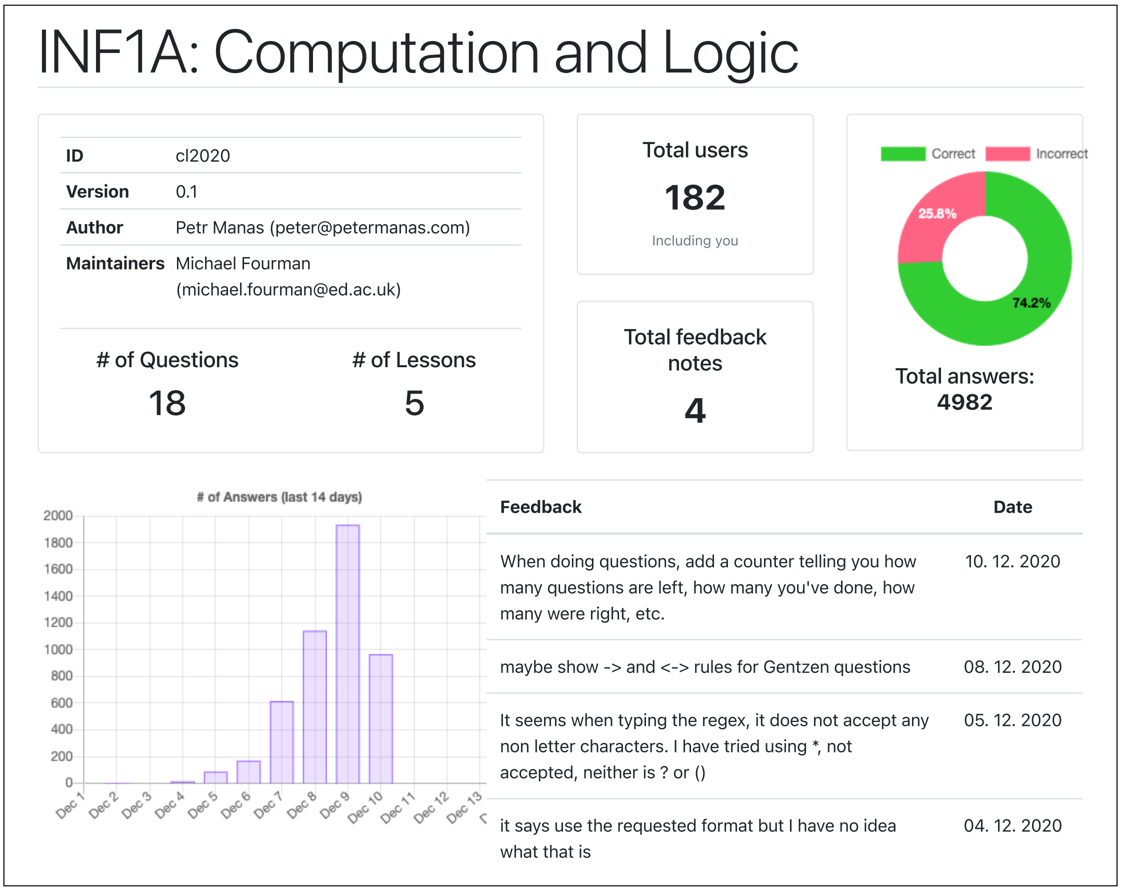
As discussed in section 4.2, the data collected through evaluation needed to be visualized somehow. More importantly, this should not be a one-time action but rather something the course organiser can do continually whenever they want. The Course Administration module aims to address exactly that: give the course organiser a brief analysis of how popular and difficult various questions are, and simply give an overview of how students are doing. Since the options for data analysis are nearly endless, I will be showcasing the few metrics I have found important and realistic to implement within the scope of this project, and I will discuss further potential improvements in section 5.3.

5.1 Organise Dashboard

Most of the Organise module can be seen on the dashboard which is shown split up in figures 5.1 and 5.2. The purpose of the dashboard is to display all the relevant information to the course organiser while remaining coherent. At some point, if the amount of information grew beyond what could comfortably be displayed on a single page, the dashboard could either be scaled vertically, i.e., simply more sections would be added to the bottom, or horizontally, i.e., by splitting sections into separate tabs and adding additional navigation elements.

The top of the page (figure 5.1) contains an *"at a glance"* overview of the installed course and how it is doing. We can notice the number of Questions and Lessons in the course, but also how many users have registered so far, and how many answers have been recorded, including the percentage of correct and incorrect answers as a pie chart.

Further down, we see a 14-day plot of the number of answers on each day to gauge user interest and the increasing or stagnating trend of using the tool. Since this example uses data from the evaluation in section 4.2, the peak number of answers appears on December 9 and far surpasses all other days. This section of the page also includes any feedback that users had given recently. During the evaluation, the feedback was very helpful, so it is useful to have it quite visible on the dashboard.



INF1A: Computation and Logic

ID	cl2020
Version	0.1
Author	Petr Manas (peter@petermanas.com)
Maintainers	Michael Fourman (michael.fourman@ed.ac.uk)

# of Questions	# of Lessons
18	5

Total users

182

Including you

Total feedback notes

4

Total answers: **4982**

Feedback	**Date**
When doing questions, add a counter telling you how many questions are left, how many you've done, how many were right, etc.	10. 12. 2020
maybe show -> and <-> rules for Gentzen questions	08. 12. 2020
It seems when typing the regex, it does not accept any non letter characters. I have tried using *, not accepted, neither is ? or ()	05. 12. 2020
it says use the requested format but I have no idea what that is	04. 12. 2020

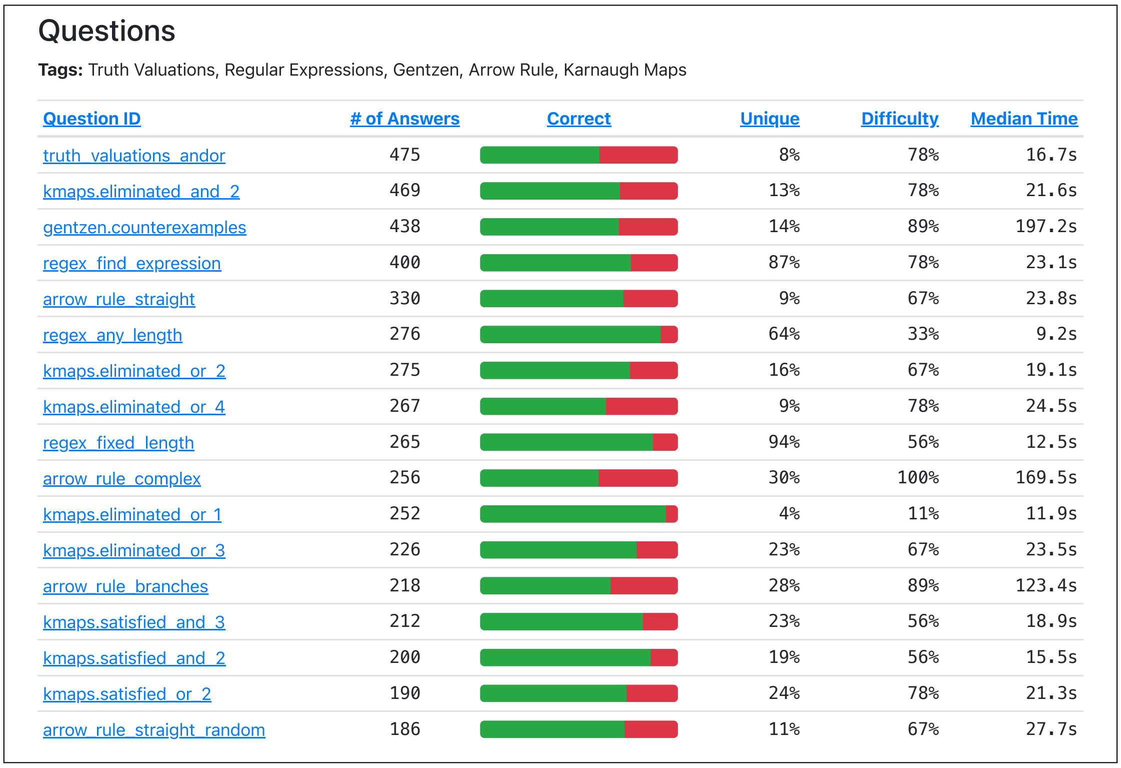
Figure 5.1: The first half of the Organise module dashboard, showing data from the semester evaluation described in section 4.2. Information displayed includes total number of users and answers, as well as recent feedback.

5.2 Questions Table

The Questions section contains a table that is meant to summarize the state of all installed Questions and how they are being answered. Some of the simplest metrics we can find are the number of answers for a Question and what proportion of these answers were correct and incorrect. This measure alone can give the course organiser an idea of which questions (or topics) are the most popular, and which Questions students often answer correctly or make mistakes in. Based on this knowledge, they can then pick specific topics for the next tutorial in order to challenge or test their students.

The *"Unique"* column measures what percentage of all the answers for a Question was unique. A high number such as 94% means that nearly all submitted answers were different from each other, as is the case for `regex_fixed_length`, where users have countless possible correct answers. On the other hand, 4% in `kmaps.eliminated_or_1` means that only extremely few sensible answers exist[1], hence the bulk of the submitted answers are not unique.

[1] In fact, there are only 6 possible correct answers for this particular Question, hence nearly half of the unique answers were incorrect.



Questions

Tags: Truth Valuations, Regular Expressions, Gentzen, Arrow Rule, Karnaugh Maps

Question ID	# of Answers	Correct	Unique	Difficulty	Median Time
truth_valuations_andor	475	🟩🟥	8%	78%	16.7s
kmaps.eliminated_and_2	469	🟩🟥	13%	78%	21.6s
gentzen.counterexamples	438	🟩🟥	14%	89%	197.2s
regex_find_expression	400	🟩🟥	87%	78%	23.1s
arrow_rule_straight	330	🟩🟥	9%	67%	23.8s
regex_any_length	276	🟩🟥	64%	33%	9.2s
kmaps.eliminated_or_2	275	🟩🟥	16%	67%	19.1s
kmaps.eliminated_or_4	267	🟩🟥	9%	78%	24.5s
regex_fixed_length	265	🟩🟥	94%	56%	12.5s
arrow_rule_complex	256	🟩🟥	30%	100%	169.5s
kmaps.eliminated_or_1	252	🟩🟥	4%	11%	11.9s
kmaps.eliminated_or_3	226	🟩🟥	23%	67%	23.5s
arrow_rule_branches	218	🟩🟥	28%	89%	123.4s
kmaps.satisfied_and_3	212	🟩🟥	23%	56%	18.9s
kmaps.satisfied_and_2	200	🟩🟥	19%	56%	15.5s
kmaps.satisfied_or_2	190	🟩🟥	24%	78%	21.3s
arrow_rule_straight_random	186	🟩🟥	11%	67%	27.7s

Figure 5.2: The Questions table at the bottom of the Organise module dashboard showing data from the semester evaluation described in section 4.2.

The following column *"Difficulty"* attempts to measure exactly what it sounds like and will be discussed further in section 5.2.1. The last column shows the median time to answer the question (regardless of whether or not the answer was correct). Since there were many *away-from-keyboard* instances in the data, mean and maximum measures are useless without at least limiting the data up to the 90th percentile, whereas median shows representative values without pre-processing. The timing in particular shows which Questions take the most effort and can often be seen as the most difficult. It could also be used to estimate how long each question will take students on a tutorial/exam and plan them accordingly so that they are accomplishable within a time limit.

It should be noted that the page also contains a similar table for Lessons but this is not shown because with the current data it is simply not as interesting – Lessons were not very popular during the evaluation, so very few conclusions can be drawn.

5.2.1 Measuring Difficulty

How to measure a Question's difficulty solely based on the gathered data was a conceptual problem that was largely solved heuristically. Looking at the Questions table, there are three primary metrics related to how difficult a Question is: percentage of incorrect answers, percentage of unique answers, and median time to answer.

- Higher % of incorrect answers → higher chance to answer incorrectly → higher difficulty

- Higher % of unique answers → more potential answers to choose from (less straightforward question) → higher difficulty
- Higher median time to answer → more time required to think about the answer → higher difficulty

With that being said, we cannot say that each of these metrics has the same effect on the difficulty, let alone that they are linearly proportional to difficulty. For instance, if the median time to answer is 200s for Question A and 20s for Question B, it does not mean that Question A is 10x as difficult. Similarly, the percentage of unique answers can be correlated with difficulty but a Question like `regex_fixed_length` is quite simple despite its 94% unique answers. For this reason, many different approaches with scaling and modifying the individual metrics were tested, which finally led to the formula in figure 5.3.

$$log_2(\frac{I^2}{100} \times log_2(U) \times log_2(T))$$

$$\text{WHERE } I \textit{ (Incorrect)} \in [0,100], U \textit{ (Unique answers)} \in [0,100], T \textit{ (Time median)} > 0$$

Figure 5.3: Difficulty formula displayed in the Questions table. Resulting values are scaled to $[0,100]\%$ by dividing by the maximum.

Since % of incorrect answers proved to be the most correlated with how difficult a Question was, it was squared before scaling down again. This meant that if 20% of answers were incorrect, this factor would equate to 4, while if 40% were incorrect, which is much more significant despite being only double, the factor would be 16. Unique answers and time medians, on the other hand, were found to be relevant but not intensely important, hence both are scaled down logarithmically. The product of these factors is then again scaled logarithmically so that the difficulty curve is more linear than exponential, and the results for all Questions are scaled to the range $[0,100]\%$.

Clearly, this formula is not perfect and only serves as a heuristic for the course organiser rather than ground truth, while still being a useful metric. One potential way to improve our gauge of how difficult Questions are is to simply ask the users. If users are asked this right after they had answered a Question or at the final summary page, we might get a better idea of the perceived difficulty of these Questions. Unfortunately, these answers could also be inflated or deflated based on whether the user answered correctly or not. Ultimately, however, this would likely be a more relevant and useful metric than the heuristic that is in place now. The simple reason why this has not been implemented yet is that it has not occurred to me as a possibility until all data was already collected and it was too late.

5.2.2 Question Testing

Each of the Question ID links within the table directly to a simple Question testing interface. This is the same view that has been implemented last year but now has been integrated into the Organise module. The course organiser can view and answer the Question as a student would, as well as see its full context and the seed that was used to generate it. They can also generate the Question for a particular seed, for instance, one that a student has answered incorrectly in the past, and see the exact Question instance. Alternatively, they can simply keep reloading and see the various possible Question instances and get a feel for them before releasing the Question to the students.

I have personally used this mini-tool while designing nearly every single Question generator because it made my work significantly easier than manually running code in a Python shell. It still has room for improvements, however. The mini-tool could, for instance, run the generator thousands of times and measure the uniqueness of the generated instances. This, however, re-opens an issue of what makes an instance unique that was discussed in section 2.2.2. Alternatively, it could help the course designer run unit tests on individual Questions which could ensure consistency among implementation changes. Unfortunately, as sensible and straightforward as this sounds, the randomness of each generator means that changing a single line of code can change **all** the results for given seeds. Therefore, simply storing seed-result pairs would only make sense if the Question was never updated, which in turn would significantly reduce the usefulness of unit testing in the first place.

5.3 Possible Improvements

When this system would truly be put to regular use, many potential usability improvements would certainly become apparent as with any tool. One clear improvement would be greater control over the filtering of visible data. A straightforward example of this is being able to view a summary of data from a specific time period, such as the past week, month, or the entire semester, as well as manually inputting start and end dates to observe the effects of a specific change. More than that, the course organiser could pick a specific question or lesson and combine it with the date selection, to get even more specific data.

Another obvious improvement is better-computed metrics, for instance: average improvement of success rate for students over time, development of time to answer for each question over time, or finding correlations between success at multiple questions. The latter could be used to check if questions are similar or not, or rather if there are any correlations between being good at one question and the rest. Many more metrics could be implemented based on the preferences of the course organiser when the tool would be used for a certain amount of time.

Although many of these improvements would be implemented when this tool would be put to real use, they were simply not a priority for the purposes of this project. Despite having a significant amount of data from user testing (Section 4.2), all this data falls within a single week and a single context which is the revision week. This means that it would not be possible to meaningfully showcase any of these features, yet implementing them would take a significant amount of time.

Chapter 6

Further Work

With this project coming to an end, there are several possible paths it can take in the future, each of which would involve further improvements outside of the scope of this report. The source code for this project could be made open-source for others to use it at their own will, it could be extended for public use and hosted at my expense, or it could be kept private and used solely by the Inf1A course. Each of these scenarios or a combination of them is possible but they all come with different problems to consider.

Open-sourcing the Project

This would be by far the simplest and most likely solution since it involves the fewest changes. After adding instructions for how to deploy the tool, as well as potentially dockerizing it for simple deployment, the GitHub repository on which the project is hosted could be made public and potentially lightly advertised through University services.

The tool is usable as-is with very little setup, but since it is currently targeted at a selected portion of the Inf1A curriculum, anyone interested in using it would likely need to write several of their own Question generators. Because of this, adding more detailed documentation for the `generic` module would prove invaluable for potential users. Since effort was already put into writing simple documentation for the module in section 3.1, this would not be a significant amount of extra work.

Hosting It Publicly

Although the tool is currently hosted publicly at `https://cltools.inf.ed.ac.uk`, making this public version of the tool any useful for users outside of the University would again involve extending the tool. Either simply a greater variety of Question generators could be implemented, which would be time-consuming in its own right, or the tool could be rewritten to support Course and Question design from the browser.

Since the security and problematics of that approach have already been discussed in last year's report, this would understandably be a large conceptual problem. On the other hand, giving this ability to the users could move the tool closer to the likes of Memrise, and potentially further thanks to the ability to write generators in native Python. However, due to the outlined potential issues with this approach and a significant time effort, this is likely not going to be a likely scenario.

Keeping It Within the University

Similar to the first scenario, keeping this tool hosted on a University server and simply extending it or educating someone else on how to extend it would be a viable option. Since the Question set is designed for Inf1A and has proven to be useful and favored by the students in section 4.2, the tool would be nearly ready-to-go with the initial work to implement a few more Question generators. Alternatively, open-sourcing the project and this scenario could be combined, and other Universities could be contacted about also hosting this tool.

At the very least, I would certainly like to discuss the possibility of using this tool within the Inf1A course in the coming years with the new course organiser. The necessary changes and improvements for using the tool actively in-class could be done over the summer and I would be happy to guide the course organiser on how to implement new Questions, as well as implement a few of them myself. Especially after seeing the positive responses from the students this year, I believe a tool like CLPractice could be incredibly useful for 1st-year students struggling with Inf1A or simply wanting to practice outside of the tutorials.

Chapter 7

Conclusions

7.1 Review of Project Goals

Semester-Wide Testing With New Inf1A Students

This project goal was set last year with very different expectations of what this year would look like. Due to the difficult continuing COVID-related situation, it was impossible to achieve this goal as planned for reasons outlined in section 4.2, but a smaller and nonetheless useful evaluation took place instead. Having students use this tool during the revision week made it potentially even clearer just how useful it could be for revision since the number of answers grew as the final quiz grew closer. Hence, despite having to change this goal from ”semester-wide” to ”revision week” still proved to be a success and provided me with sufficient data for constructing the `organise` module.

Course Administration

This goal stems from the previous one and a Course Administration has indeed been successfully implemented this year. Thanks to the revision week evaluation, there was enough data gathered for display in the new administrative module. This module now serves as a summary of the students’ performance and has great potential to be extended by additional metrics and data analyses.

The sub-goal *”Testing Question Generators”* has also been partially achieved by integrating the generator tester from last year into the `organise` module. Since it was found that more a complex testing suite was not necessary for this project’s purposes, additional improvements such as observing patterns from generating large numbers of random instances have not been implemented. These still remain as potential further work if they are found to be useful when deployed as a part of the course but at the moment fall outside of the scope of this project.

Making the Tool Course-Agnostic

As the `generic` module was refactored and improved upon this year, a part of these improvements was to make it easy for whoever deploys this tool to use a different Course from the provided `ComputationAndLogic`. All direct mentions of the installed Course have been replaced by a loading function that uses the Course specified in the project settings. This means that to use a different Course, it simply needs to be present in the project folder and a single line in the settings must be updated.

A potential further improvement to this could be to enable the simultaneous installation of several different courses and letting the user chose which one they want to practice. While this would not require significant effort, it was deemed unnecessary for the purposes of this project since only the single Course is being showcased.

Other General Improvements

While this was set as a rather vague goal last year, some of the specifics outlined then were indeed achieved this year and more. General system improvements such as passwordless user login using an email address and several improvements to the `practice` module were outlined in chapter 2.

More improvements came specifically for the Question generator portion of this tool, which was discussed in detail in chapter 3. The `generic` Course module has been refactored and Validators were extended to support using custom inputs by allowing for writing new templates for them. This made it possible to achieve a completely custom interface and experience for questions that could truly benefit from it, such as Karnaugh Maps.

7.2 Addressing Exceptional Criteria

While I hope that the achievement of the basic and additional criteria has been clear throughout the report, I believe separate attention should be given to the exceptional criteria.

Outstanding scholarship or engineering, and/or publishable research

Since this was not a research project but rather a software engineering one, no publishable research has been conducted within the project's scope. Beyond the scope, however, I believe there is potential for publishable research on the effect of using this tool for revision as compared to relying solely on tutorials and past papers. Seeing as this year's teaching circumstances were far from the norm and the tool was made available to the students only at the end of the course, not enough qualitative data could have been gathered. Despite that, reasonable conclusions about the usefulness of the tool were achieved in chapter 4, and with a more directed study in the coming year, this could potentially be extended into findings relevant to a small journal article.

Moreover, the software itself has already been published on `https://cltools.inf.ed.ac.uk` and used by a significant number of target users – around 180. For that reason, I believe this project has achieved this criterion at the very least partially.

Evidence of originality

Although there have been many Inf1A CL tools in the past, and more tools are available outside of the course but relevant to it, most of them generally focus on a single area. Even the mathematically-focused tool STACK can be used for many areas within Mathematics but is still limited to that domain. I believe that CLPractice, despite being currently showcased for Inf1A, could be used for any number of different areas and domains, which makes it unique. The flexibility of being able to design custom Question generators using Python and all its available packages, while these generators are encompassed by the tool which takes care of executing them, cannot be seen even in the most general tools I have discussed in section 1.2.1 such as Memrise. For this reason, I believe the project demonstrates sufficient evidence of originality.

7.3 Lessons Learned

This project taught me many valuable lessons about software development as well as evaluating my software. While this was not my first interaction with the toolset I had used, the project served as a very good practice of what I already knew and a permissive environment for learning new things. I got to deploy my first Django web application using Apache's `mod_wsgi` and I now know I would rather never do that again. I also had a good reason for dynamically importing Python packages for the first time, which I am sure will be useful in the future.

Another lesson was that the tool's deployment and evaluation should have been started much sooner, especially given that communication with the University's IT support and Ethics board took well over a month alone. It was, however, also a valuable lesson in making this delay work in my favour by evaluating in a different time frame from what I had planned and yet getting potentially more useful results. Overall, this project certainly improved my abilities as a software engineer and will serve as a good foundation for my future projects.

7.4 Project Outcome

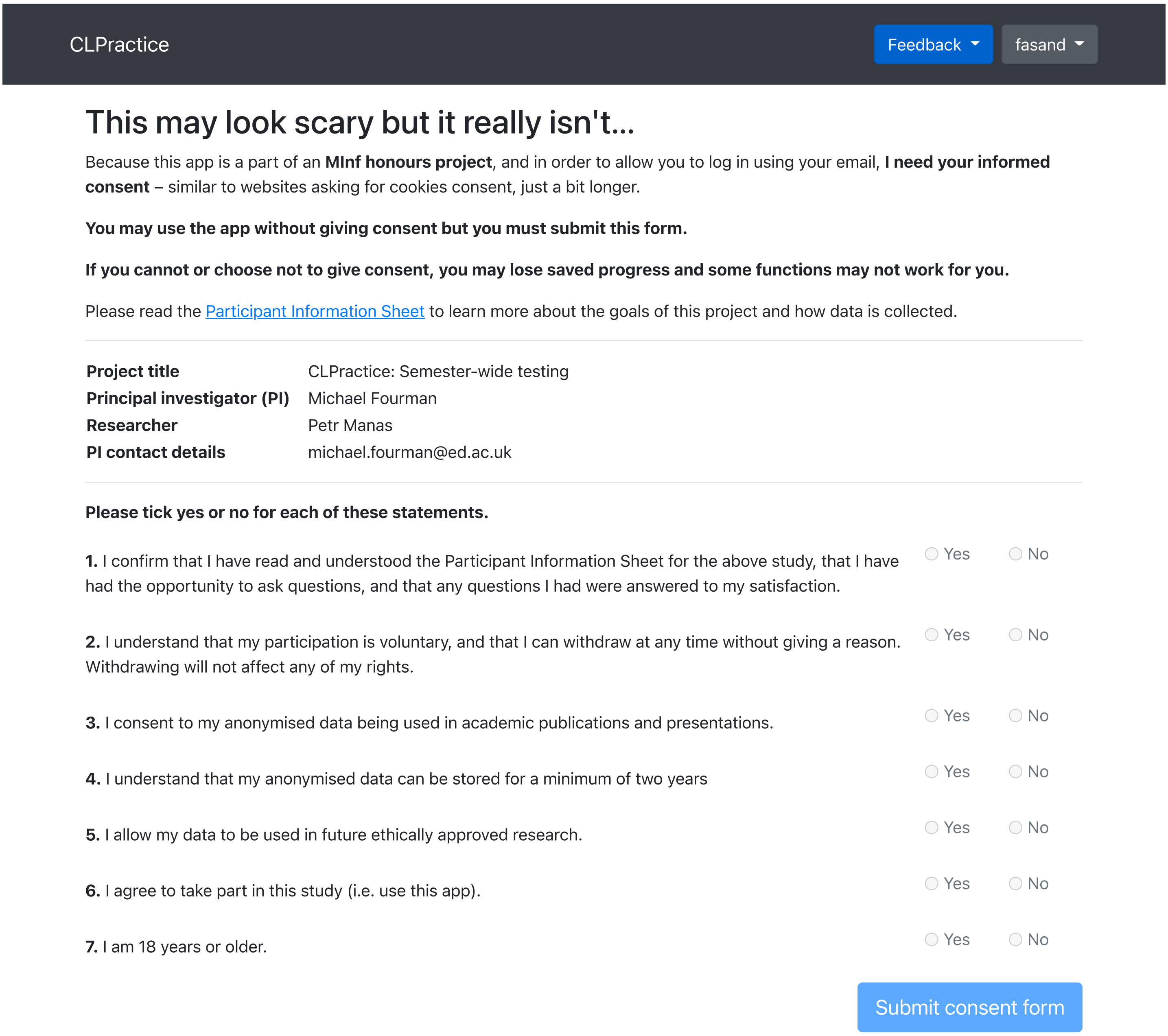
The project as a whole was a success. The second year of the MInf project seamlessly continued from the foundation I had laid last year, and I was able to get the tool properly tested by nearly 180 students of the course. Since the students seem to have found the tool useful and there is always more work to do, I am open to continuing the development of this tool in the future and potentially having it integrated more closely with the Inf1A course. In conclusion, I am more than happy with the outcome of this project and I am looking forward to seeing how its future unfolds.

Appendix A

Semester-wide Testing

A.1 Participant Consent Form

A screenshot of the web participant consent form from the perspective of a logged in user (*”fasand”* = me) who has not yet submitted the form.



This may look scary but it really isn't...

Because this app is a part of an **MInf honours project**, and in order to allow you to log in using your email, **I need your informed consent** – similar to websites asking for cookies consent, just a bit longer.

You may use the app without giving consent but you must submit this form.

If you cannot or choose not to give consent, you may lose saved progress and some functions may not work for you.

Please read the Participant Information Sheet to learn more about the goals of this project and how data is collected.

Project title	CLPractice: Semester-wide testing
Principal investigator (PI)	Michael Fourman
Researcher	Petr Manas
PI contact details	michael.fourman@ed.ac.uk

Please tick yes or no for each of these statements.

1. I confirm that I have read and understood the Participant Information Sheet for the above study, that I have had the opportunity to ask questions, and that any questions I had were answered to my satisfaction. ○ Yes ○ No

2. I understand that my participation is voluntary, and that I can withdraw at any time without giving a reason. Withdrawing will not affect any of my rights. ○ Yes ○ No

3. I consent to my anonymised data being used in academic publications and presentations. ○ Yes ○ No

4. I understand that my anonymised data can be stored for a minimum of two years ○ Yes ○ No

5. I allow my data to be used in future ethically approved research. ○ Yes ○ No

6. I agree to take part in this study (i.e. use this app). ○ Yes ○ No

7. I am 18 years or older. ○ Yes ○ No

A.2 Participant Information Sheet

Participant Information Sheet

Project title:	CLPractice: Semester-wide testing
Principal investigator:	Michael Fourman
Researcher collecting data:	Petr Manas
Funder (if applicable):	N/A

This study was certified according to the Informatics Research Ethics Process, RT number 5333. Please take time to read the following information carefully. You should keep this page for your records.

Who are the researchers?

Petr Manas, Michael Fourman (supervisor).

What is the purpose of the study?

To allow INF1A students to use the CLPractice web app along with progress tracking and statistics. To identify which generated questions are the most popular and assess each question’s difficulty based on in/correct student answers. To evaluate how often students will use the app for tutorial and exam practice.

Why have I been asked to take part?

This app is targeted towards 1st year informatics students taking the INF1A: Introduction to Computation course. The app aims to aid revision and practice of key concepts. It can be used in combination with tutorials in order to practice for small tests, which is why it will be running throughout the semester.

Do I have to take part?

No – participation in this study is entirely up to you. You can withdraw from the study at any time, without giving a reason. Your rights will not be affected. If you wish to withdraw, contact the PI. We will stop using your data in any publications or presentations submitted after you have withdrawn consent. However, we will keep copies of your original consent, and of your withdrawal request.

What will happen if I decide to take part?

You will be given access to an app using your login email of choice, which will be able to track your progress as you answer randomly generated questions related to the INF1A course, particularly Computation and Logic.

You may use the app as often as you want, either as a helper tool for tutorial tests or for exam practice. Since every answer you give is stored in the database (only accessible to you and the researchers), you will be able to track your progress and see improvement or which areas you might want to focus on. The data collected includes which questions you have chosen to answer and the answer you provided, how long it took you to answer each question, and how long you spent using the app as a whole.

At any point while using the app, you can submit feedback concerning your experience with the app and generated questions through a form within the app itself or through an external short questionnaire.

Are there any risks associated with taking part?

There are no significant risks associated with participation.

Are there any benefits associated with taking part?

You will have a chance to use a unique app targeted at helping students with studying and question practice for INF1A. The anonymous data and feedback collected during this study will help with further development of the app in order to better help students in the future.

What will happen to the results of this study?

The results of this study may be summarised in published articles, reports and presentations. Quotes or key findings will be anonymized: We will remove any information that could, in our assessment, allow anyone to identify you. With your consent, information can also be used for future research. Your data may be archived for a minimum of 2 years.

Data protection and confidentiality.

Your data will be processed in accordance with Data Protection Law. All information collected about you will be kept strictly confidential. Your generated question answers (un-identifiable data) will be tied to the email you used for login. Any potentially identifiable data will be referred to by a unique participant number rather than by name. Your data will only be viewed by the researcher/research team.
All electronic data will be stored on a password-protected encrypted computer, on the School of Informatics' secure file servers, or in a School of Informatics' database. No paper records will be stored. Your consent information will be kept separately from your responses in order to minimise risk.

What are my data protection rights?

The University of Edinburgh is a Data Controller for the information you provide. You have the right to access information held about you. Your right of access can be exercised in accordance Data Protection Law. You also have other rights including rights of correction, erasure and objection. This does not include anonymous usage data which is not associated with your participant number. For more details, including the right to lodge a complaint with the Information Commissioner's Office, please visit www.ico.org.uk. Questions, comments and requests about your personal data can also be sent to the University Data Protection Officer at dpo@ed.ac.uk.

Who can I contact?

If you have any further questions about the study, please contact the lead researcher, Petr Manas (s1652610@sms.ed.ac.uk).
If you wish to make a complaint about the study, please contact inf-ethics@inf.ed.ac.uk. When you contact us, please provide the study title and detail the nature of your complaint.

Updated information.

If the research project changes in any way, an updated Participant Information Sheet will be made available on https://web.inf.ed.ac.uk/infweb/research/study-updates.

Alternative formats.

To request this document in an alternative format, such as large print or on coloured paper, please contact Petr Manas (s1652610@sms.ed.ac.uk).

General information.

For general information about how we use your data, go to: edin.ac/privacy-research

Bibliography

M. Hepburn. Tools for learning: Computation and logic, 2016. MInf Project (Part 1) Report.

M. Hepburn. Fsm-workbench, 2017a. URL https://github.com/MatthewHepburn/FSM-Workbench.

M. Hepburn. Refining the fsm workbench, 2017b. MInf Project (Part 2) Report.

G. Keady, G. Fitz-Gerald, G. Gamble, and C. Sangwin. Computer-aided assessment in mathematical sciences. In *Proceedings of The Australian Conference on Science and Mathematics Education (formerly UniServe Science Conference)*, 2012.

T. T. Kidd. A brief history of elearning. In *Web-based education: Concepts, methodologies, tools and applications*, pages 1–8. IGI Global, 2010.

J. A. Kulik and J. D. Fletcher. Effectiveness of intelligent tutoring systems: A meta-analytic review. *Review of Educational Research*, 86(1):42–78, 2016. doi: 10.3102/0034654315581420. URL https://doi.org/10.3102/0034654315581420.

P. Manas. Clpractice – tools for learning: Computation and logic, 2020a. URL https://project-archive.inf.ed.ac.uk/all/ug4/20201668/ug4_proj.pdf. MInf Project (Part 1) Report.

P. Manas. Urlaccessor test class for webarchivcz/seeder, 2020b. URL https://github.com/WebarchivCZ/Seeder/blob/3c37eb581d66bd5d31ac653e58a4ec767a75e66e/Seeder/www/tests.py. part of my work project, commit 3c37eb5.

P. M. Maurer. Generating test data with enhanced context-free grammars. *IEEE Software*, 7(4):50–55, 1990. doi: 10.1109/52.56422. URL https://doi.org/10.1109/52.56422.

F. Mehta. The sequent calculus calculator, 2019. URL https://seqcalc.io/.

A. Mikolajczak. Karnaugh-mapp, 2018a. URL https://github.com/Arcadius19/Karnaugh-mAPP.

A. Mikolajczak. Tools for learning: Computation and logic: Karnaugh mapp, 2018b. 4th Year Project Report.

P. Purdom. A sentence generator for testing parsers. volume 12, pages 366–375, 1972. doi: 10.1007/BF01932308. URL https://doi.org/10.1007/BF01932308.

A. Sakharov. ”sequent calculus.” from mathworld–a wolfram web resource. URL `https://mathworld.wolfram.com/SequentCalculus.html`. created by Eric W. Weisstein.

C. Sangwin. Computer aided assessment of mathematics using stack. In *Selected regular lectures from the 12th international congress on mathematical education*, pages 695–713. Springer, 2015a.

C. J. Sangwin. Assessing elementary algebra with stack. *International journal of mathematical education in science and technology*, 38(8):987–1002, 2007.

C. J. Sangwin. Who uses stack? a survey of users of the stack caa system, may 2015. 2015b.

C. J. Sangwin and M. Grove. Stack: addressing the needs of the neglected learners. In *Proceedings of the Web Advanced Learning Conference and Exhibition, WebALT*, pages 81–96, 2006.

W. Schwarz. Tree proof generator, 2021. URL `https://www.umsu.de/trees/`.

N. Vazbyte. Haschool: an online tool for accelerating haskell learning, 2019. Fourth Year Project Report.

E. Z. Yang. Logitext, 2012. URL `http://logitext.mit.edu/main`.

O. Zawacki-Richter and C. Latchem. Exploring four decades of research in computers & education. *Computers & Education*, 122:136–152, 2018.