CLPractice
Tools for Learning: Computation and Logic

Petr Manas

MInf Project (Part 1) Report
Master of Informatics
School of Informatics
University of Edinburgh

2020

Abstract

This report describes the design and development process of CLPractice, a tool made to aid the teaching of the Inf1A: Computation and Logic 1st-year Informatics course. The tool provides a framework for designing random question generators in Python as well as an interface for answering these questions. Student answers are stored in the database for the purpose of giving the course organizer 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 a number of Inf1A:CL students during the December revision week 2019 and the overall response suggested a significant usefulness of the tool for revision, both in its current state and upon the implementation of several planned changes. It is web application written in Python with the help of the Django framework and several other essential libraries.

Acknowledgements

The project report structure is loosely based on the structure from (Hepburn, 2016), especially chapters 2 and 3, but does diverge where appropriate.

I would also like to thank Professor Michael Fourman, my supervisor, for the helpful feedback throughout the development of this tool and also for the initial idea. During our interview for selecting a topic for my report, he gave me the idea of a tool like Duolingo for the purposes of teaching Inf1A:CL. Although the concept and implementation of the tool diverged from this initial idea quite a bit in the end, a ”Duolingo clone” was what sparked the idea for this project.

Table of Contents

Chapter 1

Introduction

This project focuses on creating a practice and revision tool for the Informatics 1A: Computation and Logic (Inf1A:CL) course.

The tool is a web application consisting of a number of questions and lessons the students can practice at their own pace. Every question implements a generator, which generates a new random instance of that question whenever a student chooses to practice it. This means that students can practice for hours and, potentially, never get the same exact question.

Data about student answers is collected for the purposes of progress statistics and further analysis. When the tool is used in a real-world setting by the students (discussed in section 5.1), this data can be used to identify which questions students struggle the most with, and adjust the teaching material, for instance.

Seeing as this is a two-year MInf project, the development of the tool will continue over the summer and next year. At this point, the tool is fully functional and implements several tested question generators. Over the next year, the collection of questions will be expanded significantly to meet the design of the curriculum, and a few more features will be implemented. All planned changes will be discussed in chapter 5.

1.1 Motivation

1.1.1 Efficient And Effective Revision

Students often use past papers and tutorials for revision, since these give examples of the types of questions that might appear on the exam. It is key that these particular questions are not only abstract descriptions of potential exam questions (e.g. ”give a regular expression for a finite state machine”) but specific instances of these questions. The answer is also often included so that the student can check whether they were correct or not, but sometimes the given answer might not be clear, there might a typo in it, or it might just not be there. With this tool, students can practice the same types of questions that would appear at the exam, but with several important benefits over using past papers and tutorials.

Number of unique questions Firstly, since every question is written as a generator, students can get potentially hundreds or thousands of different randomly generated question instances. As a comparison, if they went through 10 years of past papers and all the available tutorials, they might find a couple tens of question instances at best. Needless to say that it would take a significantly longer time to find these question instances and it would not be guaranteed they would be able to check their answers.

Immediate feedback On that note, being able to check if an answer is correct or not is incredibly important. Students cannot really learn how the arrow rule works from simply memorizing the answers to a few specific problems, largely because there are an infinite number of problems one can generate. Instead, they must learn the concept of the arrow rule and how to apply it. Once they think they understand it, however, the simplest way to check their understanding is to answer a problem and see if they were correct or not. This tool allows exactly that, for every single question instance that is generated, meaning each student can check their understanding on hundreds and thousands of different problems, if they desire.

Randomness Lastly, the generated questions are inherently random, which allows for a greater revision diversity. In past papers and tutorials, questions are often hand-picked to be representative and answerable in a set amount of time. When generating random questions, however, students have a chance to answer significantly harder and different questions from those that would appear on an exam, potentially better preparing them.

1.1.2 A Live View of Student Understanding

Although this tool is targeted to the students, the data that is collected could also be invaluable to the course organizer. The questions students raise in lectures, tutorials, or on forums like Piazza, are a great sign of specific lack of understanding, which the course organizer can act upon. The drawback is, however, that if a single student asks a question, we cannot know for sure if they are the only ones with the problem or if the entire class is struggling.

On the other hand, if the course organizer has access to anonymous statistics of how students answer various questions, they can make more founded conclusions. For instance, if 200 out of 300 students answer a specific question wrong several times in a row, it is likely there is either something wrong with that question of with the students' understanding of the concepts that are being tested by it.

It is important to note that anonymity and security must be preserved while analyzing student answers, or clear consent must be given if identify of the student is to be used. Since there are many ethical implications associated with using student data, for the purposes of this project, all data and statistics accessible to the course organizer must by fully anonymous.

As a final note, this particular motivation has not yet been implemented simply due to the lack of student answers, but it will be acted upon very early into the first semester

of next year, as discussed in sections 5.1 and 5.2.

1.2 Background

1.2.1 Previous Work

All of the currently available tools listed below have been an inspiration for this project in one way or another. While all of them focus on teaching and practicing, each one is focused on a different topic or uses a different means of teaching. Some are predominantly focused on teaching languages, while others are more open for custom-created courses, but do not offer any means of generating random questions. In one way or more, my tool differs substantially from each of the already available tools listed below. 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. They were the original inspiration for making a tool very similar to both, offering the students a platform for daily revision of important Inf1A:CL concepts. However, since neither tool offers random generation of questions, they both differ in a substantial manner from my tool. Although the concept of the tool has changed since the initial idea, Duolingo and Memrise remain a great source of inspiration for how to teach concepts efficiently and in an enjoyable way.

Anki is a tool designed for revision using flash cards with the help of spaced repetition (more on that in section 1.2.2). It is predominantly used for learning languages and similar heavily memory-based topics, such as definitions. However, it was one of the inspirations for this project, in terms of its simple aesthetics, the general design of questions, and the potential for using spaced repetition for revision even with randomly generated questions.

STACK is an online assessment package for mathematics developed at the University of Edinburgh. 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 key difference is that STACK is focused solely on questions for mathematics, whereas my tool can accommodate any sort of questions that can be generated using Python. In the context of mathematics alone, however, it is certainly more advanced than my tool is striving to be, hence for mathematical applications, it is a better solution.

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. The project is open source and accessible on (FSM Workbench

Github), and considering it is a great tool for specifically learning FSMs, I am planning to integrate it with my tool next year. Thanks to the integration, students could potentially design an FSM for a given regular expression or write down the regular expression satisfying a generated FSM right within this tool.

Karnaugh mAPP has also been an inspiration for this project, in terms of seeing what has already been developed and how it was useful for the purposes of teaching Inf1A:CL. Mikolajczak (2018b) created a tool for learning Karnaugh Maps, along with a number of questions designed for this purpose. This tool is also available on (Karnaugh mAPP Github) under a permissive licence and might be partially integrated in the future for the purposes of designing Karnaugh Map question generators.

HaSchool is the last "Tools for Learning: Computation and Logic" tool I have looked at, developed by Vazbyte (2019). It focuses on the Haskell aspect of Inf1A, hence providing a great inspiration for how my tool could potentially be used for other topics than logic.

1.2.2 Research

Generating questions using Context-Free Grammar

There has been number of 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, as will be discussed in section 2.3, it might not be possible to write certain types of questions as simple grammar rules. 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.

Effective learning through spaced repetition

Significant research has been done on using spaced repetition, a technique where material is revised at specifically spaced intervals, for more efficient and effective retention of information. This includes (Kang, 2016) and (Tabibian et al., 2019), both of whom found an improvement over traditional ways of learning and revision.

While spaced repetition techniques have not been incorporated into the tool as of now, they are something I am planning to incorporate into the tool in the future. The popular tool Anki makes use of spaced repetition in a very effective way and is one of the inspirations for this tool.

1.3 Project Goals

The goals for this project are the following:

- Develop a functional tool to aid CL students in practicing key concepts and revising for the exam.
- Ensure the tool is useful throughout the semester rather than only towards the end of it.
- Ensure the process of generating questions is secure and reliable.
- Prepare the tool for further development in the next year, ensuring code is readable, reliable, and easily extendable.

1.4 Summary of Work Done

1.4.1 CLPractice Tool

The majority of this report focuses on the design and development of this tool, which is the only real point of contact with the students. It is a web application consisting of only a few pages visible to the student. The tool generates random questions and allows the students to answer them on demand, and will be described in much greater detail in the upcoming chapters.

1.4.2 Generic Course Interface

I have written a set of Python classes that describe how a course, question, and lesson are defined and implemented. These generic parent classes include a lot of the code used to run the question generators, some of which will likely be moved elsewhere in the next year. Having a common interface, effectively an API (application programming interface), makes it possible and convenient to design many questions, lessons, and even courses in the future (more on that in section 5.3).

1.4.3 Set of Question Generators

The aforementioned course interface is implemented by the currently only course `cl2020`. It currently includes 9 question generators and a single example of a lesson. This number is quite low because these questions are largely proof-of-concept rather than a full syllabus and because of the time and stress constraints imposed on us by the COVID-19 global situation (further discussed in section 4.3). The generators cover a relatively small number of areas from Inf1A:CL but enough to showcase and test the tool. These areas include, for instance: the arrow rule, regular expressions, and truth valuations for And/Or Boolean expressions. One of the key future plans involves creating a much wider array of question generators and curated lessons in order to benefit the upcoming students of Inf1A:CL (section 5.1).

1.4.4 Generator Testing Views

For the purpose of testing the generators without storing answers in the database, as well as being able to view questions generated for a specific random seed, I have developed a simple view. This view is available for every configured question and is accessible at the relative URL:

`/practice/question/<str:question_id>/<int:seed>/`

Similar views have been implemented for examining the configured lessons and, eventually, courses. For now, the `/practice/course/` URL displays information about the only installed course, which is `cl2020` (class name `ComputationAndLogic`).

1.5 Roads Not Taken

As with any project, the specifications have changed many times during the development process. While some of my past decisions will be described in greater detail in chapters 2 and 3, there were a few notable ideas that did not get through to the final implementation, either because they were too ambitious, or simply because I eventually deemed them not very good.

1.5.1 Duolingo Clone

The original idea for the tool was to essentially clone the incredibly popular Duolingo[1] application, apply its best features to Computation and Logic, and improve upon certain parts. The main notable improvement that was apparent from the start was the presence of question generators, which Duolingo certainly uses to some extent as well. The distinction was supposed to be the fact that Duolingo is targeted at learning languages, and I wanted to make this tool topic-agnostic and available to all.

While there were other, more open Duolingo competitors at the time, such as Memrise, or even flashcard software such as Anki or Quizlet, none of them had question generators. These are absolutely key for concept-heavy topics such as logic, where each concept can be expressed through countless different exercises, which can often be easily generated.

The reason I strayed from this idea, in the end, was that Duolingo and flashcards heavily rely on small frequent practice sessions, ideally once every day. Another reason is that these solutions focus on memorizing rather than understanding, which is appropriate for languages but not so much for concept-heavy subjects. In my initial testing (section 4.1), I found that students preferred my tool for targeted revision, so I focused on that instead.

1.5.2 Question Set Structure

Initially, I had a very elaborate notion of how questions and lessons should be structured so that the tool could be used for any number of different subjects. In this struc-

[1] An application for learning languages, accessible on `https://www.duolingo.com/`

ture, a course designer would create a *Course*, within it several *Sections*, within each *Section* several *Lessons*, and then they would start writing the generators for individual *Questions*, grouping them within *Question Sets*. Each *Question Set* would then contain a selection of related *Questions*, and each *Lesson* would be constructed from one or more *Question Sets*.

While this structure seemed to make sense at first, I quickly realised that it was unnecessarily complicated and closed for this project's purposes. In my initial testing (section 4.1), students pointed out they would like to practice questions by themselves and not only as a part of a lesson. Making *Lessons* and *Questions* co-exist independently not only gave the students more freedom while practicing, but also made the structure significantly simpler. *Question Sets* and *Sections* also quickly stopped making sense, seeing as *Lessons* and *Questions* could simply be tagged and then filtered by these tags, giving a much looser structure and yet maintaining the original functionality.

1.5.3 Dynamic Difficulty

A part of the initial structure (subsection 1.5.2) was that questions could be labeled with a difficulty, be it "easy, medium, hard" or "1-10". An alternative was that each question would be able to generate a different instance of itself based on an input difficulty. After considering this for a while, I came up with the idea of using a *dynamic difficulty* for all questions. Either each question would receive a "difficulty" parameter in the range $[0, 1]$, or ideally, the tool itself would judge the difficulty of a question instance and return the one instance that would match the requested difficulty.

This turned out to be far more ambitious than I initially expected, since all the possible solutions I came up with had severe drawbacks. If the burden of generating the appropriate difficulty would be put on the question designer, the quality of the results could not be guaranteed and, in many cases, it would be very hard to find an appropriate heuristic to determine a question difficulty. If the tool were to do this itself, it would likely be impossible to find a general enough solution that would work for all questions, hence the result quality would also be low. Unfortunately, as interesting as dynamic difficulty might have been, the transformation of "seed $\rightarrow$ difficulty" is very abstract and cannot be clearly defined.

Machine learning on the completed answers, along with how long each answer took and whether the answer was correct or not, could potentially solve this issue. The problem with that solution, however, is that for this system to be reliable at all, many answers would need to be recorded first, they would need to be filtered for edge cases (such as someone leaving the practice window open for several hours), and then the results might still be far from reality. As a consequence of this, the system might start working towards the end of the semester, if ever, due to the requirement for a large number of answers, hence its usefulness would be little to none. Seeing as implementing something like this would be no small feat and the resulting usefulness would be very uncertain, I decided to let go of the idea for now, and potentially look back at it again next year (more on that in chapter 5).

1.5.4 Other Ideas

Some of the other ideas I had included a forum for student questions, additional resources such as reading accompanying lectures, and a tool for the course designers to test their questions. I have dismissed the idea of a forum quite early on, seeing as it is already effectively implemented for the course using Piazza. Additional resources would be a relatively simple addition but there is also a place for them already, which is Learn. Having resources at two different places would almost certainly not be any more convenient.

As for the tool for testing questions, this is still an idea I am thinking about but it would not be useful to my tool at its current state. While I will discuss this further in chapter 5, should my tool become more general and ready for arbitrary courses, a tool for testing and designing questions, as well as managing entire courses, would be a helpful addition. Now that the tool is focused only on Computation and Logic, however, there would be no use in writing an additional application for managing a single course.

Chapter 2

Design

2.1 System Design

Initially, the idea came from the 15 minute discussion with Professor Fourman we had before I had even chosen this project, rather than a formal design phase. Later on, this ad hoc brainstorming turned into a weekly meeting where we discussed what changes I had made that week and what was planned. Therefore, right from the beginning, I used an iterative approach both to the design of this tool and its consequent development.

In some ways, this resembled Agile development (Shore et al., 2007) and its many iterations and releases. The first few iterations revolved around putting a minimum viable product (MVP) together for personal testing and to gather feedback on. The target for the first properly tested release was the revision week in December 2019, when I had conducted initial student testing (section 4.1) and received invaluable feedback.

Advantages of an iterative approach The major advantage of using an iterative or agile approach was that changes could be pushed out and tested immediately. Even during the initial testing, one student notified me of a bug which I fixed in the next few minutes. This approach also meant that the relatively vague specifications could be made more specific throughout the development process as new knowledge came. After evaluating the initial testing, I had made significant changes to specifications of the system (section 4.1.4), which would not have been possible in a traditional linear approach.

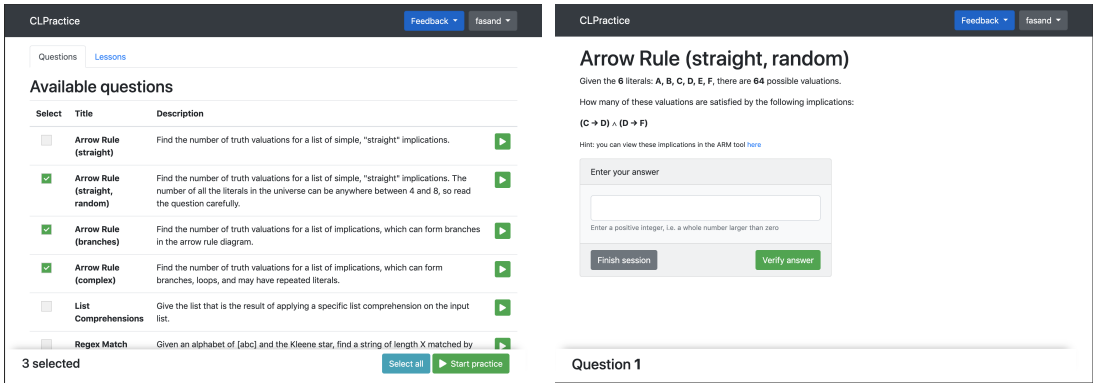
Reasons for using a different approach On the other hand, developing the tool iteratively also meant that the specifications kept changing constantly. Since there was no formal evaluation phase at the end of each iteration other than discussing the changes with Professor Fourman and the others doing this project, sometimes I spent too much time on things that did not make it into the final version. Using a more traditional approach of system requirements gathering, planning, development, and evaluation might have paid off. However, despite some of the downsides of my approach, I stuck to it throughout the development process because of all its advantages.

System Requirements As mentioned, the tool never had a formal set of system requirements and specifications but rather a developing set of goals and principles, with the final goals described in section 1.3. The functional description of the tool kept changing but the goals mostly stayed the same. The tool should be simple to use and understand, it should provide a clear benefit to the students in studying and understanding the materials from Inf1A:CL, and the tool should be easily extendable for future development, considering this is a two year project. These requirements and goals will be evaluated in chapter 6.

2.2 UI Design

Considering the tool is focused on generating questions and letting the students answer them, which is not particularly visual, the design of the UI was not paramount. However, the design of the user interface was nonetheless an important part of providing a good user experience, hence care was taken to make the UI clear and well arranged.

Since I am not a professional designer, I have resorted to using a UI library called Bootstrap. It is very popular for quick prototyping but also provides a clean interface which is immediately recognizable yet pleasant on the eyes. The library also offers many responsive layout features, making it great to write applications which can be viewed on various screen sizes. Along with its context-based color palette, it makes it very simple to design a good looking app with not too much effort.



(a) Questions selected on the Dashboard (b) An arrow rule question being answered

Figure 2.1: The Dashboard with a few questions selected for practice and one of these questions being answered.

An example of two of the primary screens the user interacts with can be seen in Figure 2.1. The first (2.1a) shows the Dashboard where the user can select questions to practice. Using checkboxes suggests the user can select one or more questions and a green button implies performing a positive action. The second screen (2.1b) shows a generated question being answered. At the top of the page, we see the description of the question, and below that is a card box with the answer input. The user can therefore quickly distinguish the box for answer from the rest of the page and understand that the green button will be used to verify their answer. The navigation bar is present in

all user screens, giving the user an option to return back to the Dashboard at any time, leave feedback, or to log out of the tool.

Future improvements The UI will most likely stay the same even after next year's wave of development, seeing as it is clear and serves its purpose. Once more questions with custom types of answer input appear, however, there might be some changes to layout necessary. At the moment, the question generators implement a template for the question description, which can use any available HTML elements. With custom input, some of these generators would also implement a template for the answer card body, presenting the user with different input elements other than a text or number input field. Various improvements can also be made to the templating tools available to question generators, such as including the ability to display LaTeXequations and process markdown into HTML.

2.3 Question Design

Generating random questions from a template is not an unfeasible task when using context-free grammars, for example (Purdom, 1972) (Maurer, 1990). The generator can be written as a set of grammar rules, some of which generate numbers, Boolean literals and expressions, and the final the question itself. However, it is much harder to then compute and verify the answer using grammars alone, hence some code is necessary.

After considering whether questions should be generated on the server side or the client side (section 3.7.1), I chose to use Python for this task. Seeing as the answer computing portion of the generator code would need to be written in Python, it only made sense to allow for the question itself to be generated using Python as well. This approach gives each question generator incredible flexibility and, with enough care, essentially any conceivable question can be written as a random generator.

Implementation Each question generator implements the API discussed in section 3.3 along with providing two template files `question.html` and `answer.html`. Each of these files receives a dictionary with the context generated for the specific instance of a question and using the Django templating language provide HTML code which is displayed to the user before and after answering a question. This supplied context can be generated through arbitrary Python code, using any number of imported libraries, giving endless possibilities.

2.3.1 Example: Arrow Rule for Straight Implications

One of the key concepts in Inf1A:CL is finding the number of truth valuations for a Boolean expression using the arrow rule. The student is given a number of literals (generally A through H) within a universe and a set of implications using these literals. They are then asked to find the number of valuations where all these implications are true by drawing an arrow diagram and performing legal cuts on the connecting arrows, each symbolising a valuation.

```
# self: mandatory argument because the code is part of a class
# rng: a seeded Numpy random number generator
def generate_context(self, rng):
    literals = list("ABCDEFGH")
    num_literals = len(literals)
    implications = []
    lit1, lit2 = None, None
    # Loop between 1 and "number of literals" times
    num_loops = rng.randint(1, num_literals)
    i = 0
    while i < num_loops:
        # Pop a random literal for the first iteration
        if lit2 is None:
            idx = rng.randint(0, num_literals)
            # Removes lit1 from literals
            lit1 = literals.pop(idx)
        # Then reuse previous 2nd literal
        else:
            lit1 = lit2
        idx = rng.randint(0, num_literals)
        # Removes lit2 from literals
        lit2 = literals.pop(idx)
        # Append the created implication
        implications.append((lit1, lit2))
        i += 1
    return implications
```

Figure 2.2: Code used to generate the implications for the straight-implication arrow rule question.

For this question, the student is given the simplest form of this question, where all the implications are in a "straight line". For example: $(A \to B) \wedge (B \to C)$. Figure 2.2 shows the algorithm used to generate this set of implications in pure Python with the use of the Numpy's random features. Literals are iteratively popped from the initial list of all literals in the universe and implications formed from a set of two literals are added to a list which is returned as the question's context.

In figure 2.3 we see the algorithm which computes the number of truth valuations for a given instance of the question, from the generated context. Since this is a simple example of an arrow rule question, the number of truth valuations can be computed using a simple equation rather than by evaluating every possible valuation. In more complex scenarios, such as when branches and loops are introduced in the implications, all possible valuations are considered. The return of the function is a dictionary with a single "answer" key because this is then used to generate the template for the answer which is displayed to the user in case they answer the question incorrectly.

```
def get_answer_context(self):
    implications = self.get_context()
    n = len(implications)
    N = len("ABCDEFGH")
    return {
        'answer': (n + 2) * 2**(N - n - 1),
    }
```

Figure 2.3: Code used to compute the correct number of truth valuations.

Integration with other tools Seeing as one of the other tools for Inf1A:CL learning this year focuses on arrow rule in particular, the students can use this tool in conjunction with my tool to verify their answer visually. Harrison Coneboy, the creator of the Arrow Rule Mastery tool, provided a URL endpoint[1] which can be used to display the question generated by my tool in his. In the future, with Harrison's permission, it could be possible to even run a simplified instance of his tool within CLPractice to let the user verify they had drawn the correct arrow diagram in the first place, for example.

[1]The mentioned example can be tested in the ARM tool using `https://harrisonconeboy.github.io/Arrow-Rule-Mastery/arm.html?implications=[(a,b),(b,c)]` (accessed 02/04/2020)

Chapter 3

Implementation

3.1 System Overview

The tool is a web application written in Python using the Django framework. Currently, only a single client page is written using the React framework, while the rest uses Django's powerful templating system. In the future, more pages will likely use some of the advantages of client-side JavaScript frameworks like React. The various languages, frameworks, and libraries will be described in more detail in section 3.2.

For the purposes of making the tool easily extendable in the future, a generic course structure is provided as a Python package, which will be discussed in section 3.3. The only currently available course `cl2020` implements this structure – i.e. its course class and all implemented lessons and questions extend the generic classes.

Although Django refers to its modules as "apps", I will use the term "module" in order to avoid confusion with the tool as a whole. The `core` module provides the functionality for user and study participant login, as well as some base classes for database models, which are then extended in their specific applications. It also implements a simple feedback gathering mechanism which stores submitted user feedback in the database. This will be discussed in section 3.4.

Most of the tool's functionality, such as question generating, displaying, and answering, happens in the `practice` module. Once the user is logged in, this is the only module they interact with. This will be discussed in section 3.5.

During the development of the tool, I have encountered several conceptual problems. Section 3.6 will discuss how questions are generated within each practice session so that they are as random as possible and the user does not get the same question twice. The issue of how questions should be generated in the first place, whether server-side or client-side, and how this generator core should be store, is discussed in detail in section 3.7.

3.2 Technologies Used

Although most of the project was developed from the ground up, several libraries and frameworks were used so as not to re-invent the wheel. The majority of these 3rd party solutions were used on the back end (i.e. server-side), where nearly all functionality of the tool is concentrated.

3.2.1 Server-Side

The server is written in Python 3.7 and the majority of the code is written within the Django framework (version 2.2). Since Django is focused on creating fully fledged web applications, it makes working with views, templates, and databases quite simple and the resulting product is robust. Since this tool is relatively broad and complex, Django is a more appropriate choice than, for instance, a framework like Flask, which is targeted more towards smaller applications and having more freedom when it comes to architecture design decisions. The database used is PostgreSQL for its simple integration with Django, reliability, and the fact it is open-source. Should a need to change the database arise in the future, the migration would be made very simple by Django's database-agnostic model interface, where each database table is effectively implemented as a Python class.

PyEDA Apart from the essential building block frameworks and libraries, there are also a number of key libraries used in question generators. Since the majority of the current questions revolve around logic, PyEDA is incredibly useful. While the library is targeted at electronic design automation, it also offers implementations of Boolean algebra and expressions, as well as simple SAT solvers.

EXREX Another very useful tool is EXREX, a library for generating random (or even all possible) matching strings for given regular expressions. While this library is obviously very useful specifically for questions on regular expressions, it is invaluable for all sorts of other questions, where a part of the question generator can be written as a complex regular expression and EXREX can generate individual random instances of that expression to produce a random question context.

NumPy Last but not least, a generally useful library is NumPy, which offers an immense array of mathematical functions, which can be used in question generators, for instance. It is also used for its random number generator (RNG), which can be stored in a variable as an object and individually seeded, as compared to Python's default `random` module, which can only be seeded globally. Thereby every question generator receives a NumPy RNG as one of its context function's arguments, so that it can be pre-seeded and the specific question context can be re-generated later with the associated seed.

3.2.2 Client-Side

At the moment, all client pages except for one are only using Django templates and very little JavaScript or other dynamic elements. The dashboard page for selecting questions to practice uses the React framework to make simultaneously selecting a number of questions and displaying this number in a readable format very simple. Since only one page uses React, no JavaScript pre-processing build process has been set up yet, and instead the HTM package is used for designing the HTML view and React is simply imported using a `<script>` HTML tag. If at any point in the future development process it becomes clear that a JavaScript framework would be useful in other parts of the tool, a robust build process would be implemented.

The state of an instantiated practice session is not stored locally at all but instead is managed through Django's sessions. Although it could also be stored in the browser's LocalStorage, for instance, all client-side solutions can be modified by the user, whereas server sessions do not suffer from this issue. This will be discussed in more detail in section 3.5.2.

3.3 A Generic Course Structure

In order to make course creating and question and lesson implementation easier, I have created a simple class structure which all installed courses must follow. In programming languages like Java, this would effectively be a set of interfaces and abstract classes, where some methods are provided but private and others are public and must be implemented by each inherited course, lesson, or question. Unfortunately, while Python is object-oriented to an extent, it does not have private methods and variables or abstract classes and interfaces. Because of this, the current implementation of the course structure interface is not appropriate for 3rd party courses since any class extending from `Course`, for instance, can override any of its methods and variables.

To combat this, I will likely simplify the interface in the future and move the system-used code which is not a part of the Course API into a different part of the tool, such as the `practice` module (section 3.5). For now, however, since the only course written for the tool has been written by myself, the utmost robustness of the Course API structure is not of essence.

Figure 3.1 shows a modified UML diagram of the course structure and the relationships between each class. Since, as already mentioned, Python is not a purely object-oriented language and it is not possible to specify the types of variables and method arguments and returns, the diagram only gives an idea of what the structure *should* look like. Because variables can be of any type and methods can be overwritten, a system will need to be put in place which verifies the integrity and correctness of course implementation. This system will be implemented next year when the planned changes from chapter 5 are implemented.

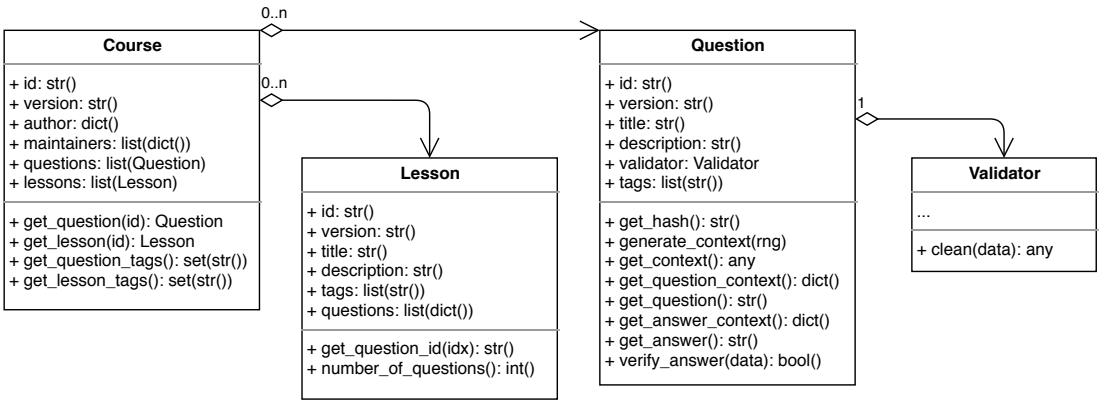


Figure 3.1: Course UML diagram. Python type functions are used to indicate types, e.g. `str()` represents a string and `list(dict())` represents a list of dictionaries.

3.3.1 Validators

To indicate what type of input is expected for a question and to validate this input, Validators are used. This is an idea I had borrowed from Django's Validators which are often used for forms and models.

Each Validator is a Python class which implements a `clean(data)` method. This method is, initially, given all the data from a `POST` request used to submit the answer to a question, and "cleans" that data into the expected output or raises a `ValidationError`. While the base `Validator` may be used as well, most questions will use one of its child classes, such as a `NumberValidator`, which inherits from an `InputValidator`, which ten inherits from the base `Validator`. When `NumberValidator.clean(data)` is called, this method first cleans the supplied data using the parent `InputValidator.clean(data)` method, which calls the base classes `Validator.clean(data)`, which simply validates that some input was given. Thanks to this approach, new Validators can easily be written and most of the code does not have to be repeated.

How Validators are used in Questions

From the question design point of view, if I am designing a question which expects a positive integer smaller than or equal to 256 as an answer, I simply set that question's `validator` variable to be equal to:

```
PositiveIntegerValidator(max_value=256)
```

When this question is loaded for the user, they are presented with an HTML input field with the properties:

```
type="number" min_value="0" max_value="256" step="1"
```

meaning that, for instance 256.1 is an invalid value and the user's input field will catch that even before an answer is submitted. This client-side validation is invaluable because it gives the user immediate feedback if they enter an invalid value.

Validating answers server-side

On the server side, when an answer is submitted, the value is validated again. Firstly, because while client-side validation is convenient, it does not guarantee that the user does not change the input field or crafts their own `POST` request. Secondly, this is because the `clean(data)` method returns a cleaned version of the submitted answer, which is then used in the question's `verify_answer(data)` method. In the case of the positive integer validator, for instance, this method returns an integer number in the range or raises a `ValidationError`, meaning the answer will not be verified and the user will be notified of an error.

Generating appropriate HTML input fields

As briefly mentioned, if a question uses a number validator, the user is presented with a number input field with minimum, maximum, and step values specified. Currently, something similar is done for the other implemented Validators, such as the `StringValidator`, showing a text input field. To allow for custom user input, such as inputting several numbers rather than just one, the option of writing a custom Validator will be implemented. This Validator will then need to specify the HTML code to be generated as well as the mandatory `clean(data)` method. Although the generic Validators will serve most questions well, some questions, such as drawing a Karnaugh map for a Boolean expression, will greatly benefit from advanced custom input elements.

3.4 The `core` Module

User authentication, feedback gathering, and base database models all lie in the `core` module.

User authentication Seeing as the tool is still in development, users can primarily log in using a chosen identifier or their study participant number. They can also log in as a "guest" by leaving the input field blank. While this means that anyone can log in using any identifier, without the need for any form of verification, this is not a concern at the moment as no sensitive data can be accessed this way. Once the tool goes into student testing (section 5.1), students will be able to register an account and log in with a password. This functionality is already partially implemented using Django's authentication system but registration is not yet possible.

Feedback gathering Because feedback is important, it should be as simple as possible for the users of the tool to voice their opinions and notify me of bugs. For this reason, the `core` module includes a view "mixin" (a class which is meant to be inherited by many other classes and provides only very specific functionality) with the feedback form attached. In the `practice` module, this mixin is imported and used in all appropriate views, which enables a dropdown form in the application navigation bar. This submitted feedback is then store in the database as text, with an identifier of the user who submitted that feedback.

Base database models and other functionality In order to streamline the development of the database, I have written a few base classes which all modules can inherit from. Namely, the `BaseModel` provides, for instance, timestamps for the creation and modification of database entries, and other convenience fields, and all implemented database models inherit from this base model. I have written a small number of template tags as well, which enable running Python code from the templates themselves using a tag shorthand.

3.5 The `practice` Module

The primary module of the tool is the `practice` module. It houses the database model for storing answers and all the functionality necessary for running a practice session.

3.5.1 Storing Answers

All submitted user answers are stored in the database, identified by the user and several other properties.

Submitted answer and correctness are the essential properties of any answer. Marking if the question was answered correctly or not allows for very quick and simple generation of statistics, such as what percentage of questions has the user answered correctly. The actual answer is also important for analysis purposes. It is possible that the submitted answer was marked as incorrect when it was, in fact, correct, due to a bug in the question implementation. Thanks to storing the submitted answer, the course designer can notice these bugs and prevent them from happening again.

Session id is used to identify which exact practice session the answer was answered in. This is particularly useful when showing a summary at the end of the session and for detailed progress statistics.

Question seed represents the specific instance of the question that was generated and answered. This number is used to seed the random number generator used for all random actions within question generation. By storing the seed, we can re-generate the identical question later, in order to analyse where the user made a mistake, for instance.

Course, lesson, and question ids are stored to identify exactly which question was answered. The version of each is stored as well, so that if any changes are made to the question generator, for instance, it is clear that the stored question seed might not work for the new version. In the future, older versions of courses, lessons, and questions could be loaded in order to re-generate the answered question even after a new version has been published.

Timestamps of when the question was started and when the user finished answering are also stored. Although the time is not necessarily a reliable heuristic for difficulty since the user could be, for instance, away from the keyboard for a prolonged period

of time, it can be useful nonetheless. When unrealistic outliers are eliminated, the resulting times can be useful to give an idea of how long it takes students to answer each question on average. The course organizer could, potentially, then use this information to design tutorials and even exams with these times in mind, allowing the students enough time to answer each question.

3.5.2 Using Session Storage for State Management

When a session is initiated and while it is running, it is paramount to keep track of what question the user is currently answering, how many questions have been answered so far, and what questions will come next. For this purpose, some sort of a state must be managed throughout the duration of the session. It is key that this state cannot be modified by the user and hence is always reliable and true, otherwise undefined behaviour could occur. It is also necessary that the state persists over page loads and, ideally, does not get deleted even on restarting the browser.

For this reason, state management is implemented using Django's session storage, which fits all these criteria. It can hold any serializable data, such as text or even lists and dictionaries. The contents of the session are stored in the database by default, meaning the user does not have access to the data and cannot read it or modify it, keeping the session secure. This data also persists over any number of page loads and browser restarts since it is tied to the user's login session. It is, therefore, deleted only when the user logs out or on demand – for instance, when a user finishes a practice session.

3.5.3 Running a Practice Session

Upon logging in, the user is presented with a dashboard which contains the questions and lessons available for practice. When they select the questions or a lesson to practice, they submit a form which sends a `POST` request to the Start view, which initiates the practice session. To get a better idea of what the entire process looks like, figure 3.2 shows the relationships between individual views and which requests are sent to move between views. If a bad request is sent, the user is presented either with Session view, if the error is small, or back to Dashboard, if the error is larger, such as when a `POST` request is sent to the Verify view without a practice session running at the moment.

Start The Start view receives either a list of a question ids or a single lesson id through a `POST` request. Based on which it is, it initializes the session state slightly differently, sets the type of the practice session appropriately (”questions” or ”lesson”), and generates a random seed for one of the selected questions or the first question in the lesson. Once the state is initialized, the user is redirected to the Session view.

Session Here, the selected question is generated using the randomly selected seed and displayed to the user along with an input field for the answer. To answer the question, the user submits a form which sends a `POST` request to the Session view itself. This answer is then cleaned using the question's validator, verified, stored in the

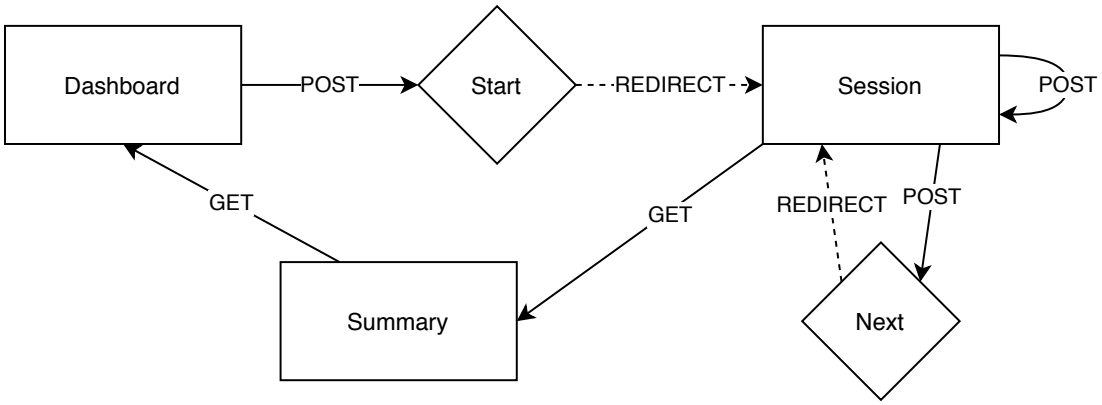


Figure 3.2: Diagram of the HTTP requests while running a practice session. Rectangles show views which show a template when loaded through a `GET` request. Diamonds show action views which can only be access using a `POST` request and then redirect to a different view.

database, and the user is notified if they had answered the question correctly or shown the correct answer, in case their answer was wrong. In the questions mode, the user also has an option to skip the question at any point, in case they think the question is too easy or too hard, for instance.

Next After an answer has been submitted and the user is happy to continue, the user presses a button which submits a hidden form and sends a `POST` request to the Next view. This view is responsible for getting the *next* question for the user to answer. In the case of practicing a lesson, the next question is simply the next question defined in the lesson definition. When in questions mode, however, the next question is any random question which has the least number of answers for that specific practice session. This means that if, for instance, the user selects 10 different questions to practice and answers in total 100 times, each question will have been answered exactly 10 times but in random order. This view also handles the ”skip” action in questions mode. Once a new question id has been picked, a new seed is generated, which will be further discussed in section 3.6.

Summary When the user finishes a lesson or wants to stop answering questions (which otherwise keep going forever), they can click a button link to go to the Summary view. Here, a list of the questions answered within the active session is displayed, along with a summary of the number of correct answers. Upon a second click on a button link, the active session is deleted (all answers have already been stored) and the user is redirected back to the Dashboard.

3.6 Randomness of the Generated Questions

A fundamental problem of writing question generators is that there might simply not be that many possible random states. For some of the simple logic questions, there might be as few as 256 or even 16 possible states, meaning that most random number

generator seeds will give the same exact instance of a question. Although this can be partially remedied by designing better question generators which have more possible random states, this number will almost always be tiny compared to the number of possible seeds [1].

For this reason, I simply assume there is a reasonable chance that an identical question is generated twice in a row and I try to prevent it by other means. Whenever a question is answered, a `sha256` hash (also known as a checksum) for that instance of a question is stored in the session under that question's id. When a new instance for that same question is being generated, all the previously generated checksums are first checked. If the newly generated checksum has appeared in the past, it means that identical question instance has appeared in the past as well, therefore a new question seed should be generated. This process is then repeated until a unique question instance is generated or until 10 trials are reached, in order to prevent an infinite loop caused by exhausting the number of all possible question states.

Are these guaranteed to be truly unique? Unfortunately not. Using checksums guarantees that the absolute identical question is not generated again but it does not guarantee that the newly generated question is not essentially the same question as the previous one, just with one tiny change. While a very slightly different question is still a different question, this is not ideal because the user can likely immediately identify that the new question is essentially the same as the previous one and enter the same answer without thinking about it. This problem will therefore be further investigated in future development. One possible solution is to not only check the checksum of the generated question context but also check that the answer is different and, perhaps, that specific parts of the context are different. The reason this has not yet been implemented is because it is easier said than done when each question can be implemented in a slightly different way, therefore any heuristic would inevitably fail in some edge cases.

3.7 Reasoning Behind Generator Implementation

3.7.1 Generating Questions Client-Side vs. Server-Side

The first conceptual problem I had to solve, even before proper generator design, was whether questions should be generated on the server or in the client's browser. Both approaches make sense and fall short for different reasons.

Downsides of generating client-side Client-side question generation has one obvious downside, which is the fact that if the generator code is running in the client browser, the client (the user) has full access to that code. While it can be minimized and obfuscated in various ways, in the end, the user will always eventually be able to understand how each question is generated and, more importantly, how the answer for

[1] There can theoretically be an infinite number of different seeds but I always generate a random number in the range $[10^8, 10^9]$. This is so that the number is guaranteed to be lower than 2^{32}, which is a limit for some random number generators, and so that it is not too small.

that question is computed, because that code needs to be sent over as well. Why would this matter if the tool is meant for practice and revision, though? Surely, the students would have no incentive to ”crack the code” and get answers for free. Simply put, it matters because the idea of the tool has developed significantly from the beginning of the year, and it might develop more next year. My hope is to have the students use it next year as a part of their tutorials, which already suggests the tool should be a little cheat-proof. If at any point the tool is used to determine even a tiny part of the students' marks, it needs to be secure a properly cheat-proof, which means client-side question generation is out of the picture.

A better but problematic solution The reason why server-side generation was not the immediate choice, then, was because it also comes with a lot of issues, some potentially greater than the possibility of cheating. The main issue comes from the fact that the code is executed on the server, in a language which allows access to the file system, the database, the Internet, and so on. While likely the greatest threat posed by JavaScript code injection is cookie stealing (Ofuonye and Miller, 2008) through attacks like cross-site scripting, being able to execute arbitrary Python code could allow the attacker to download the contents of the database, delete everything, and take over the server. This is why I did not make this decision lightly and why it was crucial that the server-side code execution was safe and secure. This was yet another significant conceptual problem, which will be discussed in greater detail in the following few subsections.

3.7.2 Initial Approach: Storing Code in The Database

At the beginning of the development process, it seemed crucial to get a minimum viable product ready for testing, hence I opted for using Django's powerful model-based database system for organizing and managing essentially all the data within the tool, including question generator code.

This approach proved to be quite effective at first. I was able to make quick changes to the generator code on the fly while the tool was being tested, stored answers had both-way relations with their respective questions and lessons, and everything was in one place. Lesson design was also incredibly simple thanks to Django's auto-generated administration where questions could be selected from a dropdown list and added or removed on demand, straight from the browser. Generator code could easily be edited from the browser rather than having to upload updated files directly to the server running the tool or pushing to the PyPI package index and then updating the package on the server.

Risks and inconveniences However, it also meant that anyone with access to the database could effectively run arbitrary code on the server running the tool, which was a severe security risk I tolerated only for the means of testing the tool at first. It was also relatively uncomfortable to write or even just update the code from the database administration since it was presented in a `<textarea>` tag with no syntax highlighting or shortcuts. For efficient generator development, the code would have needed to be

stored and tracked in two places anyway: one where it would be developed and tested, and the other would have been the database.

For all these reasons, once the tool went through the initial testing (section 4.1), I started thinking of better solutions for this problem.

3.7.3 How to Store Generator Code

Even though I had already decided that the question generation should happen at the server side (section 3.7.1), and that just storing the code in the database was not a great idea (section 3.7.2), it was not entirely clear what the correct solution was. Since every solution I came up with had considerable pros and cons, this was another conceptual problem I had to deal with in order to have a good quality tool, and this happened quite late at the development stage.

Code cleaning with the database The simplest solution was to keep storing code in the database and maintain all the benefits associated with it but somehow make sure that the code was safe. This is possible through code cleaning, whereby potentially problematic parts of the code, such as `import` statements, would be simply removed. However, while this approach can significantly reduce the risk of malicious code, it cannot guarantee that all malicious code will be removed. For instance, running an infinite while loop and adding a random element to a list each iteration will quickly eat up resources and, well, run forever if a timeout is not set. This code will, however, be almost impossible to identify as malicious, because there are many cases where a while loop is indispensable.

Sandboxed environment A slightly different approach is to execute the code even if it might be malicious but do it in a sandboxed environment such as Pybox by Engelberth et al. or similar closed-source implementations. In this case, the code would be executed in an isolated process which would not have access to the rest of the tool, therefore even if the code was malicious, it could theoretically do no harm. Some of the downsides of this approach include a significantly more involved setup necessary even just to test the tool and greater resource usage due to the extra overhead. In this sense, even though this approach would almost certainly be the most secure, it would not be the most efficient. Therefore, sandboxing might be partially used in the future but currently is not implemented.

Drag & drop interface Another extreme but relatively safe solution would be to only enable generator input through a drag & drop interface, where the course designer would not actually write any code but instead combine various code blocks to reach the desired outcome. While this solution would make the tool accessible to a significantly wider population of course organizers who are not programmers, it would come with too many drawbacks. The first obvious drawback is that it would require an immense amount of additional work even just to be able to create very simple generators. A related issue, and a more important one, in my opinion, is that the generators implemented through this approach would always be limited by the quality of the input in-

terface. There would need to be an incredible number of various code blocks designed for all sorts of generator applications and there would inevitably always be some edge cases, making the tool significantly less useful. Lastly, this approach would still not guarantee absolute security because it would essentially suffer from the same issue as code cleaning: even seemingly clean code can drain resources and act maliciously.

Python package The solution I have decided to use in the end is to load the generator coded and associated meta information from a Python package, which is both secure and relatively convenient. The course and all its questions and lessons are stored in Python files, all encompassed in a a full course package. Although this package is currently hard-coded, it can later be shared on PyPI and installed in other instances of the tool. This approach has been mostly described in chapter 3 since this is how the tool works right now, but it is not currently fully implemented because there was simply no need for it just yet. As will be discussed in section 5.3, however, when the tool becomes course-agnostic (usable for courses other than Computation and Logic), this will likely be implemented through PyPI packages.

Benefits of using Python packages Overall, this approach won because of the consistency and transparency of the used code. While code can be changed very easily when stored in the database, this is not possible when it is a part of a published and installed package. The code only changes when the tool administrator intentionally and explicitly updates the course package. Since the code should be publicly available, it can be examined and reviewed by the tool administrator and anyone else. This means that technically there can be malicious code within the packages, but this would have to be intentionally put there by the course designer, which is highly unlikely. And even if that did happen, the tool administrator has a chance to read the code and decide if it is safe or not.

Chapter 4

Evaluation

4.1 Initial Prototype Testing – Revision Week

During the December revision week, Professor Fourman – my supervisor and the lecturer of the Computation and Logic course – organised six revision sessions spanning three days (2/12/2019 - 4/12/2019). The students working on honours projects related to the course, including myself, were invited to present their tools and let the students taking the course test them. I was able to attend three of these sessions over the span of the first two days as a tutor, and in each session, I let the students use my tool for revision.

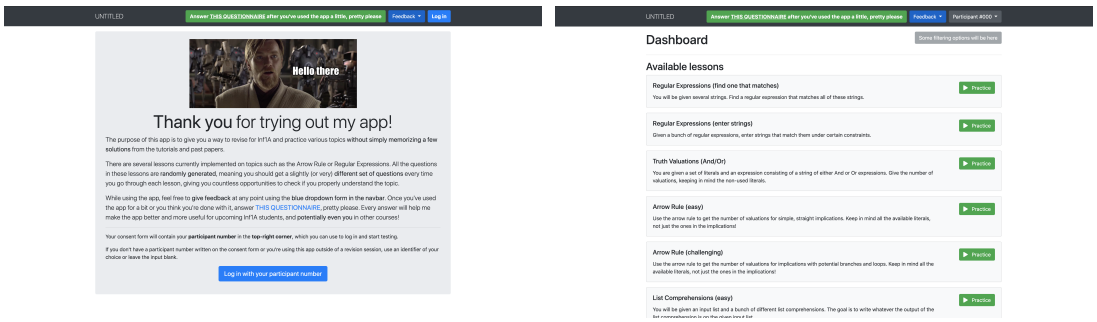
Each participant received a Participant Information Sheet (A.2) and signed a Participant Consent Form (A.1) before testing the prototype. Each participant was given a unique participant number, which they then used as identification in the tool and in a subsequent questionnaire (A.3). Both the participant information sheet and the consent form contained a link to the tool in their headers. The link pointed to a server running on my laptop, which was made publicly available using a port tunneling service[1]. This approach was not ideal since the server tunnel occasionally crashed for unknown reasons, my laptop had to be running for anyone to be able to access the tool, and the connection was rather slow. However, there were not many other freely available solutions that would be able to run a Python server, so this approach prevailed and the associated costs were relatively small.

Upon accessing the link, the participants were greeted with a landing page (figure 4.1a) which contained more information and guided them towards the tool itself (figure 4.1b). The participants were occasionally briefed on what questions were available to revise, but other than that, they were not guided on how to use the tool.

4.1.1 Day 1

During the first day of testing, several issues have surfaced throughout the session. The first immediate issue was that the link to access the tool was not visible and legible

[1] LocalTunnel, available on: `https://localtunnel.github.io/www/`



(a) Landing page (b) Dashboard

Figure 4.1: Landing page and question dashboard at the end of day 1 of testing; initial design was missing questionnaire button in navigation bar and landing page was significantly simpler.

enough on the participant information and the consent form as it was handwritten (this was before both documents were updated with the header links, as seen in A.1 and A.2). Some people also wanted to try the tool later at home, which was not possible to guarantee at the time due to using the port tunnelling service described above. Since many people were focused on revising for the exam, testing the tool was not their highest priority, and therefore only a relatively small percentage of the total number of people tested the tool.

A few people gave feedback using the feedback form in the navigation bar but fewer than was expected. The primary issue with feedback, however, was the fact that only a single participant answered the subsequent questionnaire, and this was only because he was reminded of it. The origin of this issue was the fact that the only link to the questionnaire appeared on the logout screen but no one logged out. This issue was fixed for the next session by including a big green button link in the navigation bar pointing to the questionnaire, as well as adding a link to the questionnaire to most of the other pages, as shown on figure 4.2.

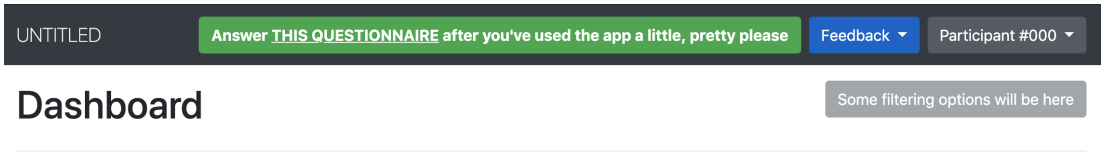


Figure 4.2: Added large green button link to a questionnaire

The landing page and login pages were also improved after the first session to give the users more information and guide them better (figure 4.1 shows the pages after the update). One of the participants encountered a bug while answering a question which they pointed out to me and I was able to then fix it within a few minutes. This was one of the main advantages of using a port tunnelling service and running the server on my laptop – the server was incredibly easy to reload and changes were available immediately.

4.1.2 Day 2

For the second day, I decided to wait about an hour before giving out consent forms because my tool was not very relevant to the first few questions of the mock exam. In hindsight, this was probably a bad decision because halfway through the session, people were not likely to start using the tool. In the end, almost every student in the room took a consent form and signet it but not a single person used the tool during the first session of the day.

During the second session, the participant information sheets and consent forms were still publicly available on several desks but only a few select people were explicitly given a consent form and suggested to try the tool. This was done on a basis of the tool's relevance to what they were currently struggling with. This turned out to be a better strategy as some of these participants used the tool immediately after signing the consent form.

After the sessions were over, one of the participants used the tool from 10:57 to 11:47 pm and managed to go through every single available lesson. A few other participants also used the tool in the following days, whenever it was running on my laptop. In total three participants from the second day's sessions answered the questionnaire and two more participants from the first day also answered it. This was very likely due to the very prominent green button in the navigation bar.

4.1.3 Results

Over the span of the two days of testing, a total of 201 question answers from at least 16 unique participants (some were answered as a *guest*) have been recorded, including: the submitted answer, the correct answer, time taken to answer the question, and the exact time at which the question was answered. After the testing was over, 68 more answers were submitted, the vast majority of which came from 3 users one day before the Computation & Logic exam. Although the size of this data is relatively small, it showed that students were interested in using the tool. In particular, it suggested the tool might be the most useful for revision.

Overall, 32 signed consent forms were collected, out if which a total of at least 19 participants used the tool and 7 answered the questionnaire. While the answers to generated questions within the tool are a useful metric of how users will use the tool and what questions they might struggle with, the most valuable information came from the few questionnaire answers (full list available in the appendix A.4).

Although the current usefulness of the tool was assessed at a mean of 3.86/5, the participants assessed the potential usefulness at 4.86/5, with all answers except for one being 5/5. Out of the suggested improvements, the most popular improvement was to have more questions from past papers and tutorials, followed by more randomly generated questions. A mobile interface/app and an overall nicer interface was nearly uniformly ranked at a range of slightly useful to extremely useful. The most neutrally assessed improvement was progress statistics, which ranked predominantly at moderately useful.

The participants enjoyed in particular the randomness of questions, the ability to ”just practice”, and the potential of breaking down major concepts of Computation & Logic for revision. Some participants also mentioned that the questions were sometimes quite straightforward or a bit removed from the CL concepts themselves. The improvements they suggested predominantly focused on better (and more) questions, a more apparent connection between the questions and the course material, and working steps for questions as well as better assessment of why an answer might be wrong. The topics to cover included essentially all topics of Computation & Logic, with emphasis on NFA to DFA, regular expressions, and Haskell implementations of these concepts.

4.1.4 Impact on Project Description

The answers from the questionnaire and a few informal discussions with students had a big impact on the project description. Initially, the idea of the tool was to be something the students would use on a regular basis, maybe for 10 minutes every day. However, the questionnaire answers made it clear that students preferred the tool for practice and revision, as well as clearing up certain concepts, rather than as a tool they would use often. In particular, it became clear the tool would be especially useful for exam revision and, potentially, for practicing certain concepts ahead of or during tutorials.

Another improvement mentioned by several participants was the focus on giving more guidance while answering each question and more feedback after the question has been answered. This could include, for instance, splitting some of the questions into smaller concepts which the students could more easily practice before getting the full picture. A big part of this improvement was also including steps required to reach an answer and generally more feedback on the answer, for instance showing where the student likely made a mistake.

All these answers and suggestions were instrumental for shaping the final state of the project, and since this testing was done relatively early on in the development process, it allowed me to make these suggested changes and test them again.

4.2 Heuristic Evaluation

After most of the improvements from the initial evaluation were implemented, a secondary evaluation of how usable the tool is was necessary. Since Computation and Logic is a 1st-semester course and my first evaluation happened a few days before the exams, it was significantly harder to find student participants for a secondary evaluation. That is why I chose to do a simple heuristic evaluation instead, and look at how my tool abides by the 10 Nielsen’s Heuristics (Nielsen, 1994).

The evaluation was conducted by myself and another fellow student doing the Tools for CL honours project. We each evaluated the other’s tool. Although our level of expertise was rather limited, we have both taken the Human-Computer Interaction course the past semester, so we each had some experience with heuristic evaluation. The scope of this evaluation was also rather small, using only a single other evaluator. However,

some of the issues raised were instrumental in making the tool more usable and I had missed them myself.

A table of the full comments given by the evaluator can be found in appendix B. One of the key issues found was a bug where once a practice session was started and then ”back” was pressed in the browser window, some of the questions remained visually selected, although they were actually not – the checkboxes were green but the tool was not treating them as checked. Another significant issue uncovered was the fact that users had the freedom to finish a question practice session at any time, but only after a question was answered, when the user might not notice the ”Finish practice” button. Both of these issues were addressed and the usability of the tool improved.

We have also conducted a cognitive walkthrough of several key tasks, which is meant to measure if each task is simple to complete and if it is clear how to complete it. During the walkthrough, the evaluator found only a single issue with not being sure how to log into the tool, which was promptly fixed.

4.3 Changes Made as a Response to COVID-19

I have included this section only in this chapter since I was fortunate enough to not be affected by the pandemic in a significant way with respect to my honours project. I could continue developing the tool from my computer at home and did not need access to any specific hardware. Although the current situation certainly did impact the scope of this project – I had planned to write a few more question generators and polish the tool a bit, for instance – this could be moved to the next year of my project or finished over the summer.

The primary effect of these circumstances was that I was unable to conduct a secondary in-person evaluation with the students who would be taking resits in August. I was considering conducting a talk-aloud study with some of these students and gathering more feedback on the updates I have implemented after the two evaluations mentioned in sections 4.1 and 4.2. Seeing as Inf1A resits have been cancelled and I had to leave the country, this study became fairly impossible to conduct and will also have to be moved to the next year.

Chapter 5

Further Work – Year Two

5.1 Semester-Wide Testing With New Inf1a Students

Seeing as the tool is currently fully functional and students find it useful (section 4.1), it can be tested on a larger scale in a real-world setting. This testing will be conducted with the upcoming students taking Inf1A in the first semester of 2020/21 and will most likely subsume the testing mentioned in section 4.3. The tool will ideally be hosted through the university and students will be given access to it. There will be new questions and lectures designed by myself with the help of the course organizer, either for each tutorial, a selection of tutorials, or for other opportunities.

While the curriculum will be decided and question generators written over the summer, the purpose of this testing will remain the same: to see how useful this tool can be in-class and as an addition to tutorials and revision. I have chosen this relatively vague research question since a full semester is a long time to test a single thing, and the focus of the testing will likely change over time. However, during the entire time, student answers will be recorded, feedback collected, and the tool updated. I should therefore be able to evaluate how useful this tool can be not only for the students but also for the course organizer, who will be kept in the loop throughout the testing and will have access to anonymous student progress statistics, described in further detail in section 5.2.

5.2 Course Administration

Currently, the recorded answers are accessible through Django's semi-automatically administration page. While this can be extensively customized and works very well for certain types of data, it does not do a good job at showing summaries of data, such as *"what percentage of students answered this question wrong?"*. For this purpose, I will develop a course administration page where the course organizer will be able to access useful statistics rather than the raw data.

These statistics can include the aforementioned percentages of students answering a question one way or another, or any number of other queries. This data could then be

presented in the form of searchable tables or graphs, making it much easier to understand how students are really doing. While my current idea of this page is relatively vague, system requirements will be gathered at the beginning of the semester and updated throughout the semester testing (section 5.1).

5.2.1 Testing Question Generators

A potential feature of this administration could be the ability to test the designed question generators in an efficient manner. Generators are designed to be pseudo-random (because their state depends on the given seed), which means it is quite hard to guess exactly what questions they will generate, how often they will repeat, etc.

This tool would allow the generator designer to run the generator for a large number of times (at least in the thousands) and collect data from each run, such as how long it takes and what context is generated. The number of unique contexts generated would then suggest how many truly unique questions can be generated, how often these repeat, and maybe even how they are distributed seed-wise. If, for instance, most seeds generate an identical context, and a select few seeds each generate a unique context, it is possible the generator is not random enough or there is a similar issue present, which would otherwise be almost impossible to identify.

The need for this tool and the appropriate scope of the administration will be fully determined throughout the semester and constantly re-evaluated. However, it is likely that similar administration features will find their users once the tool can be used for any number of different courses, which will be discussed in 5.3.

5.3 Making the Tool Course-Agnostic

While the focus of the project will remain on Computation and Logic, this tool would undoubtedly be useful in many other courses for which questions can be conceivably be generated, such as Data and Analysis, any Mathematics course, and many more. Making the tool course-agnostic is also a clear move because a lot of the necessary groundwork has been implemented already. When I was rewriting the tool from loading the code from the database (section 3.7.2) to loading it from a Python package, I have made sure to do it in an easily extendable way. All questions and lessons already implement a class ”interface”, and the entire `cl2020` package is put together through a `ComputationAndLogic` course class, which can easily be switched for a different course.

At this moment, however, these ”interfaces” can be overwritten because Python does not really have interfaces in the same way for instance Java does. The `cl2020` package is also hard-coded as the only available course and imported from a folder rather than an installed package. These issues will therefore have to be resolved to make the tool fully course-agnostic. The packages will most likely be distributed and installed through Python’s pip package installer and registered in the tools configuration file.

5.4 Other General Improvements

Apart from the specific planned improvements mentioned in the above sections, there are a few other plans I have for next year which do not warrant standalone sections. General improvements will be made on an iterative basis while testing with the students, making the tool more secure and sturdy, and improving user experience. Following up on testing from section 4.1, if next year's students show interest in using the tool on a phone, the mobile interface might deserve an improvement.

While many questions have simple number-based answers, there is a place and use for questions with a more custom interface, such as where students could design a finite state machine from a regular expression. For some of these questions, I would like to use more of the already available tools such as the FSM Workbench (Hepburn, 2016) or Karnaugh mAPP (Mikolajczak, 2018b) to bring the quality of the question generators to a new level.

Using spaced repetition Lastly, if time allows it and depending on the preliminary outcomes of the full-semester testing (section 5.1), I would like to experiment with using spaced repetition principles within the tool. Spaced repetition essentially suggests that by increasing the space between initially learning a concept and reviewing it, and by learning concepts in short bursts over a long time rather than drilling, one can retain information much better than if learning completely ad hoc. The research surrounding this is vast, and many sources such as Kang (2016) and Tabibian et al. (2019) suggest significant improvements in learning efficiency when using spaced repetition. These concepts could be implemented in the tool, for instance, through correctly spaced reminders to practice specific questions and lessons, or by proving each student with an automatically curated practice session every day, with the questions that have the highest priority according to spaced repetition.

Chapter 6

Conclusions

This first part of the project was an overall success in my eyes. The tool I set out to develop functions properly and was shown to have clear current and potential usefulness for the purposes of teaching Inf1A:CL. Furthermore, the project has clear goals set for next year and is primed for the completion of these goals. As for all of the current project goals for the project, a brief discussion of how each one was achieved follows.

6.1 Review of Project Goals

Develop a functional tool to aid CL students in practicing key concepts and revising for the exam. The tool is fully operational and several question generators have been implemented and tested by myself and the students. Already during the initial testing, the students found the tool to be useful at the moment and potentially even more in the future (section 4.1). It is not yet publicly available or otherwise accessible to the students due to delays caused by the COVID-19 pandemic. It is, however, prepared to be deployed, which it will be at the start of next year.

Ensure the tool is useful throughout the semester rather than only towards the end of it. By making it very easy to write new question generators, the tool is incredibly extendable and can accommodate various uses throughout the year. While it will undoubtedly be the most useful for exam revision, it can just as well be used throughout the semester, either through designed tutorial lessons (section 5.1), or through deliberate practice of selected questions.

Ensure the process of generating questions is secure and reliable. As discussed in section 3.7, a lot of thought has gone into how question generators should be implemented in order to make them secure and cheat-proof, as well as convenient and reliable. The final implementation satisfies this goal and will be further developed and improved, if necessary.

Prepare the tool for further development in the next year, ensuring code is readable, reliable, and easily extendable. Code readability and reliability was one of

the key focuses throughout the development process, and was successful through a clear code structure and numerous helpful comments. The tool has proven to be quite reliable but no formal test suite has been developed yet due to the ever-changing nature of this project this year. Now that the project specification is clearer, the lack of tests can be remedied next year, in order to provide a truly reliable system. Thanks to the necessary groundwork described in section 3.3, the tool is ready for further development and can be easily extendable by designing more questions or even full courses (section 5.3), for instance.

Appendix A

Initial Testing

A.1 Participant Consent Form

Go to **inf1a.localtunnel.me** Participant number:_____

Participant Consent Form

Project title:	Tools for CL: evaluating alpha prototype
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.

		Yes	No
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.		
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.		
3.	I consent to my anonymised data being used in academic publications and presentations.		
4.	I understand that my anonymised data can be stored for a minimum of two years		
5.	I allow my data to be used in future ethically approved research.		
6.	I agree to take part in this study.		

Name of person giving consent	Date dd/mm/yy	Signature

Name of person taking consent	Date dd/mm/yy	Signature
Petr Manas		

THE UNIVERSITY of EDINBURGH informatics

A.2 Participant Information Sheet

Participant Information Sheet

Project title:	Tools for CL: evaluating alpha prototype
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 4293. 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 evaluate the first prototype of a web app and gather feedback from students currently taking this course and preparing for the exam. To identify design issues and other places for improvement in the app. To evaluate if the current prototype is something the students would find helpful, and if not, what they are looking for.

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 concepts for the course, which is why it is being presented at the exam revision sessions.

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 a prototype of an app, either on a provided computer or through a URL address, and you will then be asked to simply use it and take mental note of your experience – for instance, what you liked about the app, what was frustrating, or what you would improve. In this app, you will go through a set of available lectures, where you will answer questions related to INF1A.

You may use the app for as long as you want, limited by the duration of the revision session and the discretion of the researcher.

While using the app, anonymous usage data will be automatically collected; this 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.

After you have finished using the app, you will be given a short questionnaire about your experience with the app.

Are there any risks associated with taking part?

There are no significant risks associated with participation.

Are there any benefits associated with taking part?

The outcomes of this study will influence the development of an app targeted at helping 1st year students with studying for INF1A, and possibly for other courses as well. Any feedback you give will allow the researcher to develop an app that better helps the students.

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 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 on the University's secure encrypted cloud storage services (DataShare, ownCloud, or Sharepoint) and all paper records will be stored in a locked filing cabinet in the PI's office. 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

A.3 Questionnaire

1. **What is your participant number?** (This is the number shown in the top-right corner of the app, in the navbar. If you are unsure, ask whoever gave you the consent form. If you have not been given a consent form, please ask for one or enter ”guest”.)
   - Enter your answer
2. **How would you rate this app based on its current usefulness for CL revision?**
   - Select 0 - 5
3. **How would you rate this app based on its *potential* usefulness for CL revision once some improvements are implemented?**
   - Select 0 - 5
4. **How useful would be the following improvements:**

   - More questions from past papers and tutorials
   - More randomly generated questions
   - A nicer interface
   - A mobile interface/app
   - Progress statistics

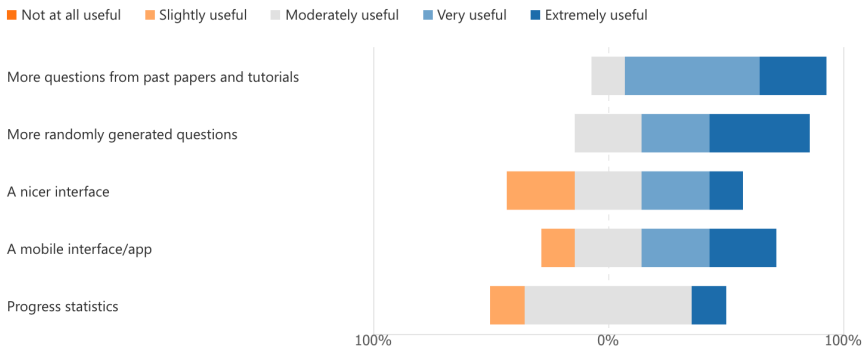
   (a) Not at all useful
   (b) Slightly useful
   (c) Moderately useful
   (d) Very useful
   (e) Extremely useful

5. **What did you like in particular about the app/idea?**
   - Enter your answer
6. **What were some of the things you didn’t like about the app/idea?**
   - Enter your answer
7. **What improvements would you make to the app to make it more useful?**
   - Enter your answer
8. **What specific topics would you like to be covered in the app? (e.g. arrow rule, NFA to DFA, ...)**
   - Enter your answer
9. **Any other general feedback?**
   - Enter your answer

A.4 Questionnaire Results

Answers to text-based questions are often abbreviated or paraphrased.

1. **What is your participant number?**
   - Answers omitted, 7 different participant numbers in total
2. **How would you rate this app based on its current usefulness for CL revision?**
   - 3,3,4,4,4,4,5, mean: 3.86, median: 4, out of 5
3. **How would you rate this app based on its *potential* usefulness for CL revision once some improvements are implemented?**
   - 4,5,5,5,5,5,5, mean: 4.86, median: 5, out of 5
4. **How useful would be the following improvements:**



5. **What did you like in particular about the app/idea?**
   - Randomness of the questions
   - Ability to practice whatever topic one struggles with
   - Potential to break down some of the major concepts to make CL easier to understand
   - Instant answers to questions, good and fast practice
   - Useful in terms of clearing concepts
   - Useful for CL revision
   - Provides more questions to practice on, great since we aren't given much material otherwise
6. **What were some of the things you didn't like about the app/idea?**
   - No working steps on how to get a solution
   - The questions are removed from the concepts and a lot of the CL concepts are quite visual (e.g. regex, arrow rules)

- Took a while to load answers, for regex it should show which ones your input fails at
- Some topics like valuations have very straight forward questions
- Did not show solutions that have all the explanations and computations

7. **What improvements would you make to the app to make it more useful?**
   - A bit more UX
   - Steps on how to get the final answer, skip questions, more topics, more variety of difficulties and questions, especially from past papers with worked solutions
   - It should be developed towards the Haskell implementation of Logic as opposed to becoming more visual
   - Optimise and explain why answers are wrong
   - Harder questions
   - Cover more topics and more questions from tutorials
   - Make the questions follow the exact same patterns as there are in the mock or past papers, just random
8. **What specific topics would you like to be covered in the app?**
   - All the ones that are examinable, arrow rule, NFA to DFA, regex, syllogisms, boolean algebra
   - Haskell implementation of these concepts
   - I can't think of a part of CL I wouldn't want to be included
   - NFA to DFA and Tseytien rule
   - NFA to DFA, CNF with Karnaugh Map, boolean algebra
   - Gentzen rules
9. **Any other general feedback?**
   - Great concept, design and implementation
   - Good progress

Appendix B

Heuristic Evaluation

Heuristic principle	**Used correctly?**	**Comments**
Visibility of system status	Yes	Users know what page/question number they are on, as well as which account they are logged into. One issue, when pressing back on the browser, previously selected questions appear selected but aren't.
Match between system and real world	N/A	N/A
User control and freedom	Yes/No	Users cannot finish exercise set [practice session] if they haven't completed the current question, forcing them into completing it.
Consistency and standards	Yes	Bottoms [bottom navigation in a practice session] are of consistent colour and placement
Error prevention	N/A	No possible errors
Recognition rather than recall	Yes	All options are shown
Flexibility and efficiency of use	N/A	N/A
Aesthetics and minimalist design	Yes	Easy to read, aesthetically pleasing, and minimalistic
Help users recognize, diagnose, and recover from errors	N/A	N/A
Help and documentation	Yes	Example inputs given to guide users

Table B.1: Heuristic evaluation of my tool conducted by a fellow Human-Computer Interaction student, also working on the Tools for CL project, based on the 10 Usability Heuristics for User Interface Design by Jakob Nielsen.

Bibliography

M. Engelberth, J. Göbel, C. Schönbein, and F. C. Freiling. Pybox - a python sandbox. In N. Suri and M. Waidner, editors, *SICHERHEIT 2012 – Sicherheit, Schutz und Zuverlässigkeit*, pages 137–148, Bonn, 2012. Gesellschaft für Informatik e.V. URL https://dl.gi.de/handle/20.500.12116/18269.

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.

S. H. K. Kang. Spaced repetition promotes efficient and effective learning: Policy implications for instruction. *Policy Insights from the Behavioral and Brain Sciences*, 3(1):12–19, 2016. doi: 10.1177/2372732215624708. URL https://doi.org/10.1177/2372732215624708.

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.

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.

J. Nielsen. Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '94, page 152–158, New York, NY, USA, 1994. Association for Computing Machinery. ISBN 0897916506. doi: 10.1145/191666.191729. URL https://doi.org/10.1145/191666.191729.

E. Ofuonye and J. Miller. Resolving javascript vulnerabilities in the browser runtime. In *2008 19th International Symposium on Software Reliability Engineering (ISSRE)*, pages 57–66, 2008.

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.

J. Shore et al. *The Art of Agile Development: Pragmatic guide to agile software development*. "O'Reilly Media, Inc.", 2007.

B. Tabibian, U. Upadhyay, A. De, A. Zarezade, B. Schölkopf, and M. Gomez-Rodriguez. Enhancing human learning via spaced repetition optimization. *Proceedings of the National Academy of Sciences*, 116(10):3988–3993, 2019. ISSN 0027-8424. doi: 10.1073/pnas.1815156116. URL https://www.pnas.org/content/116/10/3988.

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