Tools for Learning: Computation and Logic

Matthew Hepburn

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

2016

Abstract

This report details the design, implementation and testing of an educational tool to accompany the first year undergraduate course Informatics 1: Computation & Logic. The tool introduces students to finite state machines through an interactive question set. The tool also provides a general purpose finite state machine simulator. The tool is a HTML5 web application.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Matthew Hepburn*)

Table of Contents

Chapter 1

Introduction

The focus of this project has been the creation of a tool to accompany the teaching of Finite State Machines (FSMs) in the 1st year undergraduate course Informatics 1: Computation and Logic.

The tool provides a set of exercises for students to work though at their own pace, with the aim being to introduce the concepts they need to learn gradually. These exercises consist of both asking the student questions about a machine and of asking the student to create or modify a machine.

It is hoped that presenting the material in an interactive way will give students an immediate way to test their knowledge of the concepts and will help them to develop an intuitive understanding of FSMs.

An instance of this tool is available at `http://homepages.inf.ed.ac.uk/s1020995/stable/index.html`

1.1 Background

1.1.1 Existing Tools

There are a number of existing tools that allow the creation and simulation of Finite State Machines. However, they all have flaws that limit their effectiveness as tools for learning.

Chapman's *Finite State Machine Explorer* [**?**] and Faenov and Merz's *State Machine Simulator* [**?**] allow the creation and simulation of FSMs but their usability is impaired by the fact that they are Java applets, which are no longer widely supported. Running them required compilation from source and – in the case of *State Machine Simulator* – the creation of a stub Java method. This is likely to discourage most potential users. While both applications have educational value if a user is able to run them, their interfaces are dated (see Fig:1) and they are both limited to input alphabets consisting of single characters.

The much more recent *FSM Simulator* webapp by Zuzak and Jankovic [**?**] produces well laid out machines and provides a good visualisation of those machines processing input. This visualisation is done by highlighting the machine's current state(s) and the next input symbol to be processed – this is similar to the approach used in this project, although here the last transition used is also highlighted to make clear which paths were followed. *FSM Simulator's* use as a learning tool is limited however by the fact that machines can only be created using a text-based interface that assumes a high degree of familiarity with Finite State Machines.

SMCube [**?**] is a commercially available application that allows the creation and simulation of FSMs. Its focus is on the use of FSMs to model embedded systems. Despite there being a free demo of the application available, it is of limited use as a learning tool as it assumes that the user is already familiar with FSMs and provides the hurdle of having to install an application that is currently only available for Windows.

Evan Wallace's *Finite State Machine Designer* [**?**] HTML5 webapp provides a highly usable graphical way of creating FSMs but has no ability to simulate their behaviour. Its interface is also incompatible with touch devices, as it relies on keyboard shortcuts.

While not focused explicitly on FSMs, Google's Turing Machine Doodle [**?**] involves the modification of simple automata. It provides an excellent example of a method of introducing the mechanics of a system of automata in an intuitive way that does not require any written instruction. The Doodle does have educational merit but not to the extent that it makes a dedicated FSM tool unnecessary.

There are, then, no easily accessible FSM simulators that are suitable for use by students new to the topic.

1.1.2 Existing Research

The core of this project is the creation of an interactive tool to aid student learning. However, the majority of research evaluating the use of technology in education focuses on systems much larger in scope (e.g. intelligent tutoring systems [**?**] or MOOCs [**?**]) or featuring some novel element (e.g user affect sensing [**?**] or game-based learning [**?**]).

Many smaller tools have been created and written about, but there has been little research into their efficacy in improving learning outcomes.

Although there is a lack of research relating directly to systems like the one presented here, there are still useful insights to be gained from the literature.

1.1.2.1 Providing Feedback

A recent meta-analysis [**?**] considering computer based learning looked at the effect of item-based feedback (feedback relating to a particular item, such as a single question) on learning outcomes.

The meta-analysis considers three types of feedback –

- knowledge of result (KR), where feedback is limited to whether or not an answer is correct.
- knowledge of correct response (KCR), where KR feedback is supplemented by providing a correct answer.
- elaborative feedback (EF), where KR feedback is given in addition to an explanation.

The meta-analysis concludes that EF feedback is more effective at improving learning outcomes than KCR feedback, which itself is more effective than KR feedback. The benefits of EF feedback were found to be more pronounced when they are assessed using higher order learning outcomes, where knowledge must be applied to a problem rather than simply recalled.

Another finding of this meta-analysis related to the effects of a delay in providing feedback. Although students prefer immediate feedback [**?**] and spend more time reading immediate feedback [**?**], the results did not show any improvement in learning outcomes from immediate feedback. Indeed, although no statistically significant relationship was found, delayed feedback was associated with better learning outcomes.

Where possible, elaborative feedback has been used in this project. This has usually taken the form of allowing the user to step through a simulation of a machine's execution so that they can see how its behaviour differs from their answer. There are however cases were only KR or KCR feedback is offered. Only immediate feedback is used both due to the lack of strong evidence in favour of delayed feedback and due to the difficulty of implementing delayed feedback in a way that users would find acceptable.

1.1.2.2 Gamification

Gamification, the use of elements inspired by games to improve user engagement, is currently a popular topic with many claims made about the potential of the technique to harness the power that video games have to motivate players and hold their attention [**?**] [**?**]. There are also criticisms though, with some [**?**] pointing to a tendency of gamification to lead purely to extrinsic motivation in users, where they may be focused on the gamified elements but their engagement with the core subject is unchanged or even lowered [**?**]. Others, such as Bogost [**?**], argue that much of the current interest in gamification is the result of marketing rather than evidence.

A recent review of the literature around the use of gamification in education [**?**] considered a variety of design elements used in gamification (e.g. badges, levels, progress bars) and the evidence supporting their use. While the authors describe some improvement in measures of student motivation from the use of gamification, they conclude that there has not been enough research to assess the effect of gamification on learning outcomes.

Due to the lack of evidence of the effectiveness of gamification, it has not been used in this project. The addition of gamification features could be considered at a later date, with user testing used to assess its utility.

1.2 Project Goals

There are four key goals of this project:

- Provide an interactive set of exercises on FSMs that students can use alongside the Informatics 1: Computation and Logic course.
- Provide interactive FSM demonstrations to support the teaching of Inf1:CL.
- Provide an FSM creation tool that students can use to create and simulate FSMs.
- Ensure that the system is maintainable by following good coding practice as well as providing tools and documentation.

1.3 Summary of Work Done

1.3.1 FSM Exercise Set

A question set has been implemented consisting of 21 questions that start with simple deterministic machines and go on to introduce non-determinism, ε-transitions and transducers. The questions are varied and include constructing and modifying machines graphically as well as answering questions about machines.

Transducers are only covered briefly. There are currently no questions that cover the minimisation of machines or the conversion of machines to deterministic form.

The questions are not individually coded, but make use of a set of question types. This makes it significantly easier to add or modify questions. Question answers are not specified in advance – the correctness of a user's answer is determined on the client through simulation.

1.3.2 FSM Creation Tool

A tool has been implemented that allows the user to graphically create FSMs and simulate them on some input. It lacks more advanced features such as the ability to minimise machines, convert machines to deterministic form, or save machines.

1.3.3 Interactive Demonstrations

One interactive demo was produced to support the teaching of Inf1:CL. It was used in a lecture and as part of a tutorial exercise. It is a simple model of an ATM based on an example from the course lecture slides. The user can edit the machine and provide it with input interactively. The demo is available at `http://homepages.inf.ed.ac.uk/s1020995/stable/atm.html`.

1.3.4 Collection of Analytics Data

A system was implemented to gather information on how the tool is used. This data has been particularly useful in identifying questions causing students issues. Key metrics that the system is able to extract include the number of:

- unique users per day
- unique users per page
- users answering each question correctly
- users attempting each question but never entering a correct answer
- positive and negative ratings given to each question.

The system of requesting ratings for each question has not proved to be a success. The response rate has been too low for the data to be useful at the current level of usage.

A visualisation of recent analytics data is available at `http://homepages.inf.ed.ac.uk/s1020995/stable/stats.html`. This only shows data collected after mid December 2015 as the analytics system was significantly modified at that point to improve its robustness.

1.3.5 Support Tools

A tool was implemented to aid in the creation of questions. The tool allows the creation of a JSON object that can be inserted into the collection of questions. The tool is functional but crude – it has been helpful in the creation of the current question set but it is unintuitive and it will require additional work to be of significant use to any other developer. An instance of this tool is available at `http://homepages.inf.ed.ac.uk/s1020995/stable/edit.html`.

Chapter 2

Design

2.1 System Design

2.1.1 Design Methodology

There was no formal design phase prior to the commencement of implementation. Instead an iterative approach to design was used. Starting with a prototype derived from Ross Kirsling's *Directed Graph Editor* [**?**], the design of the system changed gradually as features were added and previous decisions were revised.

This approach meant that there was always a recent working version of the system throughout development. This provided some protection against unforeseen setbacks so that, if development was cut short at any point, there would be a working system to show as evidence of progress.

Another advantage of this approach was that it allowed knowledge gained during development to be used to guide the evolution of the system's design. This included not just the technical experience gained, but also the insight gained from user testing.

This is not to suggest that formal design phases are not valuable however. This approach, despite its benefits, did involve taking on technical debt in the form of a lower quality codebase in version 1.0 of the system than if it had been built from scratch to meet a specific design. Work to pay off this debt was then done as part of the development of version 2.0.

Overall though, this approach to design has suited the requirements of the project well.

2.1.2 System Requirements

As the design process was informal, so too was the process of requirements capture. The requirements and their priorities shifted over time based on user testing and discussion with Professor Fourman.

Due to the open-ended nature of this project, the requirements were primarily non-functional in nature.

Usability For the system to be useful, it must be possible for the majority of Inf1:CL students to be able to use the tool without instruction. The tool itself should be as easy to use as possible, so that it does not distract students from the material it is presenting.

User Experience It is important that students do not find the tool excessively tedious to use – if most would prefer to study the material from a textbook, then the tool is unlikely to see much use. The material should be presented clearly and concisely, without requiring excessive reading. The progression of difficulty should also be steep enough that students do not become bored by multiple questions labouring the same point. It is also important however for the material not to be presented in a way that is so challenging as to demotivate a significant number of students. A balance must therefore be reached in the way material is presented.

Compatibility To ensure there are as few barriers as possible to students accessing the tool, it is desirable for it to be usable on as many machines as possible. In particular, the tool must be compatible with the browsers available on University and School computers so that students without access to their own machine are not excluded. The popularity of touchscreen devices means that the tool should be usable on these devices as well as those with a keyboard and mouse.

Maintainability For the tool to be useful beyond the end of this project, it should not be too difficult for another developer to make changes to it. Ideally, it should be possible to alter the question sets without any familiarity with the underlying code so that the tool can easily be easily kept up to date with any changes in the way the course is delivered. A testable architecture and a suite of tests would help to fulfil this requirement by making it easier to alter the system without breaking it.

Security Security is not a critical concern here as no sensitive data is collected and the downsides of the system being disabled are minor. However, it is still important to avoid the possibility of cross-site scripting (XSS) attacks that could be used to phish for other credentials such as Informatics login passwords.

Accessibility The system should be usable by people with impaired vision, for example by interfacing with screen reading technology. Those accessing the system via a screen reader should be considered when designing and implementing the user interface. Colourblind users should also be accommodated.

2.2 UI Design

The user interface of the system was designed to be easy for users to familiarise themselves with. It had the additional constraint that it had to be usable both on touch devices and with a mouse and keyboard. The design of the interface was altered over time to take into account feedback from user testing. This iterative approach is important as the way users interact with a system is often very different from the way envisaged by the designer, necessitating revisions to make the behaviour of the system better match user expectations.

This section describes some of the design decisions made and how those decisions were affected by feedback from user testing.

2.2.1 Overall Aesthetic

A simple visual style was used for the tool, with little use of colour or decorative elements. The goal of using this style was to keep the tool free of visual clutter to allow the user to focus on the material being presented. This style also has the advantage that there are fewer opportunities for an artistically challenged designer to go astray.

2.2.2 The Control Palette

One area where there was conflict between usability and touch support was in choosing a primary method of interaction with the FSM.

Although the initial mock-up of the interface (Fig:2) used a control palette to select interaction modes in a way similar to that commonly found in image editors, the initial prototype of the system based on Ross Kirsling's *Directed Graph Editor* [**?**] instead used keyboard shortcuts for that purpose. Although this interface is efficient and easy to use with a keyboard and mouse, it is unusable on devices without a keyboard.

While it would perhaps be possible to provide a different interface to users who are not using a physical keyboard, this would increase the complexity of the system significantly and would likely not be completely accurate in its identification of input method. Instead, one interface was chosen with the aim of it being appropriate in either situation. This interface was the control palette (see Fig:6), using the visual language of image editors as this is likely to be familiar to the target audience.

While providing users with a control palette does not preclude also providing keyboard shortcuts, having to introduce both the palette and keyboard interfaces would risk harming usability by overloading the user with information.

The design does include some duplication for the sake of efficiency however – most functions can be performed through context menus opened by right-clicking or long-pressing on states and transitions. The reasoning for this is that context menus self document through the name of each choice, so they do not need to be explained in

the same way as keyboard shortcuts. It is not known whether many users make use of the context menu (no data is collected on it, though it may be a useful addition to the analytics system), but direct observations of user testing suggest that usage is limited.

2.2.3 Issues Selecting Links

Adapting Kirsling's design involved making graphical changes to bring the representation of FSMs closer to the way they are drawn in the course materials. This required drawing the links between states using a much thinner line. User testing made it clear that this had a negative effect on usability, with users finding the thinner line difficult to click on.

The first attempted solution was to make links expand when hovered over with the pointer. This helped somewhat when users were overshooting the link, but did nothing for undershooting which did not trigger the expansion. This solution also did nothing to help users interacting through touch devices although, as touch input is designed to compensate for some imprecision, it seems to have been less of an issue there.

As this was still an issue, the design was again revised. The new solution was to treat clicks *near* each link as clicks *on* that link. This regained the usability of Kirsling's thicker links, while retaining the thinner links needed to match the target style.

2.2.4 Open-ended Transition Conditions

The method by which the user specifies the conditions for a particular transition also had to be revised based on user testing. Initially conditions were specified by typing them, comma separated, into a text box. This worked for an input alphabet of single characters, where there is limited room for misinterpretation if case is ignored, but led to problems when dealing with longer symbols. Typos and spelling mistakes led to machine errors which were difficult to diagnose, harming student motivation.

The design was then revised (Fig:2) to explicitly record the input alphabet of each machine so that users could select transition conditions using a list of checkboxes, one for each allowed symbol. This restriction of input possibilities eliminates the issue of typos and spelling errors in this context.

2.2.5 Representing Machine Execution

The problem of presenting the execution of machines in a clear way was more difficult to solve through user testing due to the difficulty of assessing whether a subject was understanding what they were presented with and whether they were finding it useful.

Expected Input Representing DFAs processing expected input (i.e. input for which there exists a transition) was relatively simple – the current state is highlighted and

the transition followed to get to that state is highlighted. This representation becomes less clear for NFAs in multiple states however, with multiple highlighted states and links being difficult to interpret. The current approach works but it could be refined to be clearer, for example, by making use of animation to join execution steps or by colour-coding different decision paths.

Unexpected Input Finding a clear representation of a machine processing unexpected input was more difficult. The first problem was to determine how that input should be handled. The system uses the model of an implicit sink state that the machine moves to when there is not a specified transition for a particular input symbol. This model is used as this is how FSMs are most commonly presented in INF1:CL. An alternative approach would be to have the machine ignore any input which does not have a specified transition, as is commonly done when modelling system controllers as FSMs. Another option would be to treat any machine without a specified transition from every state for every symbol in the input alphabet as invalid. Providing the option of using the alternative approaches would be useful in giving students a deeper understanding of the assumptions made when working with FSMs but that has not yet been possible due to time constraints.

When the machine enters the implicit sink state (in all decision paths in the case of NFAs) no state is highlighted. It is possible that making the sink state explicit at this point would improve student understanding. More work is needed here to represent what is happening in a way that is both clear and theoretically appropriate.

2.2.6 Simplifying Machine Layout

The tool uses a simple physics simulation to help users to lay out their machines.

Motivation One of the goals when designing the UI was to avoid distracting the user from the key aspects of the material. Simplifying the process of laying out a machine was one of the ways this was done. A balance was struck between denying the user any control over the layout (as is done by Zuzak and Jankovic [**?**], which works because machines are not specified in a graphical way in their system) and requiring the user to specify every detail of the layout (as is the case in Wallace's *Finite State Machine Designer* [**?**]). The aim then is to reduce the number of layout decisions the user needs to make, while still giving them the ability to adjust the layout when necessary.

Force Layout The process was simplified by using a 2D physics simulation where each state is modelled as having a charge, causing states to repel each other. To counteract this repulsion, each transition imposes a constraint on the distance between the states that it connects. This is implemented using D3's Force Layout [**?**], which is efficient enough to run smoothly even on low-end mobile devices. Force Layout only provides the physics simulation – the display of machines had to be implemented separately.

Effectiveness From the user's perspective, the use of Force Layout results in machines moving to an equilibrium where states are closer to other states that they are connected to but states are not too close together. The user can then adjust the layout if they wish by dragging states around. This works well for relatively simple machines, producing a reasonable layout without the need for the user to intervene. For more complex machines, particularly machines which are highly interconnected, Force Layout is not always able to help as these machines can become tangled. In these cases the system can hinder the user's efforts to lay out their machine and even undo their work.

In recognition of the potential for the addition of Force Layout to be a distraction or an annoyance, constants such as the strength of the repulsive force were tuned over many iterations in an effort to make interacting with the system as natural and predictable as possible. Nonetheless, Force Layout has prompted some minor user criticism and would benefit from some additional polish.

Refinements For the machines users work with in the question set the current Force Layout system is generally sufficient – which is why further refinements have not been a high priority. Some possible improvements have been considered but have not been evaluated with user testing. To allow the system to cope with more complex machines, the user needs some method of disabling the physics simulation for at least some states. This could take the form of pinning in place any states that have their position manually adjusted (as demonstrated by Bostock [**?**]); providing an additional tool to allow states to be pinned in place; or providing a switch to completely turn off Force Layout.

2.3 Question Design

Questions were designed iteratively, with the process drawing on user feedback and analytics data to revise and refine the set of questions.

2.3.1 Using Instruction

The goal of the question set was to help students learn about FSMs, not just by practising what they know but also by providing them with instruction alongside the questions. Users are introduced to concepts gradually and are given the opportunity to use each concept before being introduced to the next one. Although this tool is designed to support not supplant other instruction, including instruction in the tool has the potential benefit of refreshing the memory of students and helping students who have missed or misunderstood other instruction. It also gives a greater opportunity for the tool to be useful outside of the scope of Inf1:CL.

2.3.2 Identifying Concepts

Part of the challenge of designing a question set was the identification of individual concepts to be taught – while some were obvious (such as the concept of an accepting state or an ε-transition), others were more subtle and required user testing to identify. Working with people who were unfamiliar with FSMs was valuable, as it allowed the identification of gaps in the instruction more effectively than working with users who could draw on prior experience. There were two notable examples of concepts that this testing identified as not being explicitly introduced – the fact that states can be visited more than once and the fact that states can have transitions to themselves. Identifying these issues allowed the question set to be adjusted accordingly.

2.3.3 Choosing Question Types

Another conceptual challenge faced when designing questions was in choosing a set of question types. While it would have been possible to implement the logic for each question separately, this would have been an inefficient use of development time and would have reduced the system's flexibility by increasing the work needed to add or modify a question. Instead, questions are built around seven distinct question types (discussed individually in 3.3.1). When choosing these question types, there were several important considerations.

Variety A varied question set helps to hold the student's interest. Providing problems which require different approaches to solve also has benefits beyond student motivation – prompting students to consider the material from different perspectives promotes a deeper understanding of that material than a more repetitive approach. As part of providing this variety of problems, different question types require both the design of a machine (with specifications provided in a variety of ways and with differing levels of exactness) and the prediction of a machine's behaviour.

Feedback It was important that the chosen question types could be implemented in a way that was able to provide a student with useful feedback. This goal was not completely achieved, with some of the question types only able to provide KR feedback (i.e. a correct/incorrect response). Most of the question types however are able to provide more useful explanatory feedback, usually by allowing the user to step through a simulation of a machine's execution. All of the implemented question types use only automatically generated feedback, but for some question types it is possible that more detailed pre-written feedback would also be useful if common errors were identified.

Exam-like Having question types that allow problems similar to those found in previous Inf1:CL exams was desirable for two reasons – doing so provides students with an opportunity to practice questions of this type and it is assumed that students are more likely to invest time in the tool if doing so is clearly going to help them with exams. To

achieve this goal, it is not necessary for all question types to allow exam-like questions – that would be an unnecessary constraint. The question type most influenced by this goal was "satisfy-definition", where the user must construct a machine from its formal definition but other question types (such as designing a machine to satisfy a regular expression) were also shaped by this goal.

2.3.4 Identifying Problems

In addition to insight gained from in-person testing, the analytics system was also useful in identifying issues with the question set. For example, figure 4 shows how problematic questions could be identified by the disproportionate drop in users continuing to the next question. In this example 30% of users who reached the second question did not advance further.

One of the possible causes for this drop was that the second question was too similar to the first – they were both introducing the basic mechanics of machines changing state based on input and they both used the select-states question type. This repetition may have been off-putting to users familiar with the material. The repeated use of the select-states question type (one of the less interactive question types) may also have created a negative first impression of an uninteresting and homogeneous question set that would discourage students from using it further.

To address the initial repetition of question types the first question was replaced with one using the demo question type, which also has the advantage of being more interactive. The issue of basic material being uninteresting to students who are familiar with it was one of the reasons why a sidebar was added that allows students to navigate to any question. This gives students the option of jumping ahead to material they are interested in and also gives students more information about what later questions will cover.

Identifying issues from analytics data requires more intuition and guesswork than using in-person testing to identify the source of problems and so may lead to inaccurate conclusions. When trying to identify sources of boredom, it does however have the advantage over in-person testing that users do not feel any social pressure to continue using the tool if they lose interest in it. The conclusions that could be drawn from the data were limited by the relatively small number of users however, especially when considering questions later in the sequence.

2.4 Gamification

As discussed previously, there is some evidence that gamification can lead to improved user motivation but no real evidence of its effects on learning outcomes. Nonetheless, student motivation is an important concern. While the need to pass exams (complete with a financial penalty for failure [**?**]) does provide some extrinsic motivation to stu-

dents, gamification may be able to provide more positive and more immediate incentives to learn.

The addition of some game-like features was considered. For example, a progress bar of questions completed could encourage users to complete more questions or a timer on questions could increase users' focus. However, it is also possible that they could simply be a distraction or an annoyance. To successfully gamify the tool the potential downsides would have to be balanced against the potential gains, with user testing used to ensure that the result was a net improvement in the user experience.

Because of this uncertainty over the usefulness of gamification, it was a lower priority in the development process than work with a clearer benefit. No overt gamification features have been implemented at this time but they may form a part of further work on this project.

Chapter 3

Implementation

3.1 System Overview

The FSM editor and exercise set are web-based. They use JavaScript, with the machines being displayed using the SVG features of HTML5. The D3.js JavaScript library is used to manipulate and bind data to SVG elements.

Each question in the problem set is specified using JSON. A Python build script uses these specifications to create a HTML file for each question, making use of the Jinja2 templating engine.

Analytics data is saved to the server using Common Gateway Interface(CGI) scripts written in Python.

3.2 Technologies Used

3.2.1 Clientside

D3.js The D3 JavaScript library is designed to aid in data visualisation and it works well with SVG. Because of this, it was useful in displaying machines. Its data-binding features were also used to keep the visualisation and internal representation of a machine consistent.

Other Libraries No other JavaScript libraries were used on the client. The addition of the ubiquitous [**?**] jQuery library would have made development easier – in large part because jQuery's widespread usage ensures a plentiful supply of documentation, examples and support – but the addition of a large library would have had a detrimental impact on performance. Using jQuery for DOM manipulation may have also led to greater compatibility with older Internet Explorer versions, but the tool would probably still not have worked correctly on those platforms without significant extra work.

ES6 The initial version of the tool was written using only ECMAScript (ES) 5 features as later features are not yet well supported in all popular browsers. However, while working on the second version of the tool, the decision was made to include some ES6 syntax to improve code readability. To avoid compromising compatibility the build process was modified to include the Babel transpiler, which converts ES6 code into ES5 code. While this does increase complexity and could make problems more difficult to debug, in practice it has not caused any issues. The use of ES6 features in the tool has so far been limited to minor syntax changes such as arrow functions to improve code clarity but may later include more significant language additions such as generator functions.

3.2.2 Serverside & Build Process

Homepages Server The tool is hosted on the School homepages server. This has the significant advantages of not requiring payment and of physical proximity to the target users which results in a low latency connection. However, if the tool is to be used beyond the end of this project, it will have to be migrated to another location. As the server uses an almost standard Apache configuration [**?**] this should not pose any significant challenge.

Python Python is used in this project as part of the build process and as part of the analytics system on the server. The initial intention was to use Python 3 for both of these roles, but as Python 3 is not available on the homepages server, Python 2 had to be used there. This inconsistency is not ideal, and action will need to be taken to update the serverside scripts if the default version of Python is changed to 3.

The primary reason for using Python in both of these roles was the familiarity of the author with the language – many other languages would have also been suitable but their use would have required more time to be spent on familiarisation.

As the build process has grown more complex and required the use of additional tools for minification and transpilation the limitations of Python when considering dependency management have become clear. Incorporating the existing build logic into a system such as Node Package Manager (NPM) would make it easier to perform the initial setup of the build process by allowing dependencies to be managed automatically. A solution using NPM would be a good choice for this as the minification and transpilation tools used in the build process are available through NPM.

3.3 Core Code – `fsm.js`

The component requiring the largest amount of development time has been `fsm.js`. This JavaScript component deals with the simulation, display and editing of Finite State Machines as well the question logic used in the exercise set. It also holds code used in the analytics system, which will be discussed in section 3.4. It could be argued

that fsm.js should be split into several smaller files to make the structure of the code clearer. However to avoid increasing the number of HTTP requests (and so increasing loading times) it would have been necessary to combine JS files as part of the build process, introducing additional complexity.

3.3.1 Question Types

fsm.js implements 7 distinct question types, designated as follows. None of the question types make used of hard-coded answers – the correctness of answers is always determined by simulation on the client. The question types are:

does-accept The user is presented with a machine and a list of input sequences and asked to select those input sequences that the machine accepts. This question type as it is currently implemented has a significant flaw however – the way that feedback is provided makes it easy for the user to deduce the correct answer without considering the problem. This is because there are only two possibilities for each sequence (it either is accepted or is not), so informing the user which answers were incorrect reveals the correct answer.

give-list The user is given a machine and asked to provide a number of input sequences of given length which are accepted by the machine. If the user enters a sequence of the correct length that is not accepted by the machine, they are given the option to step through a simulation of the machine processing the input to help them to understand why their answer was incorrect.

satisfy-definition The user is given the formal definition of a FSM in the form of a set of states, accepting states and transitions as well as the initial state and the alphabet of the machine. The user must then construct the machine. Feedback such as " there should be a link from '1' to '2' for input 'a"' or "the FSM should have a state labelled '3"' is given if required. While questions of this type may not be particularly engaging, previous exams for Inf1:CL have required students to work with formal FSM definitions [**?**][**?**] so it is something they may wish to practice.

satisfy-list The user is told to construct a machine that accepts some list of input sequences but does not accept some other list of input sequences. Correct/incorrect feedback is given for each input sequence, augmented by the ability to step through a simulation of the machine processing any of the input sequences.

satisfy-regex The user is given the definition of a regular language, possibly specified using a regular expression or possibly specified in natural language (but still with a regex as part of the question definition). The user must construct a machine which accepts this language. Feedback is given to the user in the form "the machine accepts

the string ‘a’ which it should reject” or “the machine rejects the string ‘bba’ which it should accept”. At this time, the check for correctness uses an inefficient algorithm which means that the number of number of links in the machine must be kept low to maintain acceptable performance.

select-states The question specifies a machine, the set of states it is currently in and a sequence of input and asks the user to select the state(s) the machine would be in after processing that sequence. This question type only provides Knowledge of Result (KR) feedback which, as discussed previously, is the least effective type of feedback. The question type could be further refined to allow users to step through a simulation as a way of giving more detailed feedback – although this would reveal the answer, it may be more educationally valuable.

demo The naming of this question type is somewhat misleading and will be revised as part of the version 2.0 rewrite. The mode allows input to be provided to a machine directly using a button for each symbol in the machine’s alphabet. This can be purely as a demo, with no defined objective. It can also be used in two distinct question types – one where the user must supply a given machine with input it will accept and one where a transducer must be provided with input that will result in a particular output.

3.3.2 Version 2.0

Motivation

The initial version of `fsm.js` started as an adaptation of Ross Kirsling’s *Directed Graph Editor* [**?**]. It grew from an initial ~350 lines to ~3300 lines of code. Features were added in an ad hoc way, with a focus on creating working code quickly rather than on creating a high quality, readable codebase. Some improvements such as increasing the division between the model and the display logic were possible through refactoring but other issues would require very significant changes to address. The structure of Kirsling’s app worked for an application of that size but it did not scale well.

In addition to issues of code quality, two of the assumptions that the code was built on were incorrect.

First, the assumption that only FSMs with a single initial state would be considered. FSMs are often presented in this way [**?**][**?**] but sometimes a set of initial states is permitted. Allowing a machine to have a set of initial states would bring the tool more closely into line with the way that Inf1:CL is taught.

Secondly, an unchanged assumption of Kirsling’s application was that it would only be dealing with a single graph. However, there are situations were it would be useful to display two distinct FSMs side-by-side e.g. to show the same input being processed by a deterministic machine and a non-deterministic machine.

Significant modification of `fsm.js` was needed to address these issues. Because of the extent of the required modifications, a substantial portion of the codebase had to be rewritten. A rewrite also provided an opportunity for experience gained from the initial implementation to be incorporated, leading to an increase in code quality.

Changes

In addition to correcting the assumptions discussed above, the rewrite also provided the opportunity to incorporate ES6 (also known as ECMAScript 6, ES2015 and Harmony) features into the application. The features used were primarily syntactic sugar to improve readability. As discussed previously, this ES6 code can be transpiled to ES5 to maintain compatibility.

The largest single change was to the way that the machine model is stored and modified. To accommodate the requirement of maintaining multiple machines, object prototypes were introduced for machines, states and transitions. These prototypes contain the functions used to manipulate machine, state and transition objects, whereas the initial implementation allowed the single machine that it supported to be altered directly from various points in the code. This change better separates the FSM logic from the UI logic, making both parts easier to test and understand.

There were other more minor changes, aimed at improving code quality. For example, more use is made of D3's `append` function when adding elements to the DOM as this is generally more readable than directly creating HTML strings.

Status

At time of writing, version 2.0 is still incomplete. The core functionality of creating and simulating a FSM is operational, but there is not yet any way of visualising that simulation. Only one question type has been implemented at this time. Because of the significant changes made, questions written for the initial version will need to be converted or recreated.

Completing the rewrite will be the first objective of the second half of this project.

3.3.3 Automated Testing

The limited automated testing that has been implemented has focused on `fsm.js`. A small suite of integration tests were written early in development using Python and Selenium WebDriver to issue commands to a browser. However, these tests proved to be of limited use for several reasons. They were time-consuming to create and maintain. They were also brittle, being broken by minor changes in the system. Had they been updated after every change, they would have diverted a significant amount of time from work on core functionality. The test suite also had reliability issues, with tests frequently having different results on different executions on the same build. This

was likely mostly due to difficult to debug issues around timing but as the tests were not providing enough utility to justify their cost in development, this issue was never investigated fully.

A second small suite of integration tests was developed later, this time using Python generated by Selenium IDE rather than hand coded Python. Creating tests this way was significantly faster than with the previous method, although it still involved a non-trivial amount of work for each test. These tests also suffered less from the timing issues that had rendered the previous suite unreliable. They had one significant issue in common with the previous suite though – they took a long time to run, ~1 minute for a very small test suite. Again, their utility was ultimately not worth the time needed to maintain them while the system was under development.

Future testing work will make more use of unit testing, which the architectural changes of version 2.0 will make easier.

3.4 Analytics System

3.4.1 Clientside

The client side of the analytics system is implemented in `fsm.js`, with some functionality duplicated in `analytics.js` for pages such as the index that require analytics but not FSM functionality. Whether this is the best approach will need to be considered during the move to version 2.0 of `fsm.js`.

The clientside code assigns each user a unique identifier, stored in the client's local storage. The identifier is a version 4 Universally Unique Identifier (UUID), generated using JavaScript's `Math.random()` function. Although some issues have been observed with the randomness of this function on some systems [**?**], this is unlikely to be a problem in this context due to the low number of users and the low impact of a collision in the analytics system.

The clientside code tracks three distinct event types. For each event, the user identifier, page identifier and current URL are logged.

Usage logging records the amount of time a user has spent on each page, by logging data when the `beforeunload` event is fired.

Answer logging is done when the user submits an answer (except for demo mode questions where there is no notion of submitting an answer). The data includes the answer, whether the answer was correct and the time elapsed since the page was loaded.

Ratings are logged when the user rates a question.

Data is sent to the server using a POST request containing a Uniform Resource Identifier (URI) encoded JSON object.

3.4.2 Serverside

The post request from the client is handled on the server by a CGI script written in Python. There are three of these scripts, one for each type of event. However, as the functionality of each script is very similar, each scripts imports `logger.py` which contains the functionality to extract and store the expected data from the POST request. In addition to the data in the URI encoded JSON object, the remote IP address and the User-Agent string are extracted from the POST request and stored along with the current system time.

Data is written to disk using a variant of JSON, where each line is a JSON object. This is done as storing the data in legal JSON as an array would require reading from the file to determine if an array was present and replacing the terminating "]" with a comma rather than simply appending to the file. This method avoids this inefficiency. The saved file is still easily parsed using standard JSON tools by reading line-by-line.

Initially a less verbose log format of space separated values was used. This format however is inflexible, with it being necessary to never remove a field in order to keep newer formats compatible with older data. Using JSON makes it easy to add or remove fields as the demands on the analytics system change – this would be particularly useful when making temporary changes such as when conducting an A|B test. Although JSON is significantly more verbose, its repetitiveness means that compression can avoid a significant increase in file size.

3.4.3 Visualisation

To make it easier to extract actionable information from the analytics data, a log parser and a simple data visualiser were created. The visualisation system runs in the browser and uses D3.js to create bar charts from the parsed data. Some data (such as usage by day) would be more appropriately displayed using line graphs, but the implementation of a second chart type was not judged a high priority.

Six measures are extracted from the data:

- the number of daily unique visitors by day
- the number of overall unique visitors by page
- the mean time spent on each page
- the number of correct and incorrect answers by question
- the number of unique users who have attempted each question, split into those that at some point gave a correct answer and those who did not
- the number of ratings by question, split into positive and negative ratings

The statistics page fetches data from the server on load via a CGI script. If the logs were last parsed more than five minutes ago then the parser is run on any new log

entries and up to date data returned as a JSON object, otherwise the existing data object is sent. This caching is done to limit the load on the server.

3.4.4 Ethical & Legal Concerns

When collecting user data, it is important to ensure compliance with relevant legislation such as the Data Protection Act 1998. The Act allows for the processing of non-sensitive personal data for research or other 'legitimate interests' of the data controller without the need to obtain user consent [**?**].

User IP addresses are the only items of data that the system collects that could be considered personal data. The Act defines personal data as being data relating to an individual who can be identified using that data.

The Information Commissioner's Office (ICO) states that the decision as to whether data is personal depends not just on the data itself but on the intentions and likely intentions of the data controller [**?**], therefore there can be no universal answer as to whether IP addresses should be considered personal data. ICO provides very little guidance relating to IP addresses, but they do state in their Code of Practice relating to personal information online that an IP address could be personal data if the address relates to a single device with a single user [**?**] – therefore any IP address is potentially personal information and should be treated as such.

IP addresses are clearly not sensitive data as they give less information than a user's name which is not (generally) considered sensitive [**?**] so their processing for analytical purposes is lawful. However, it may be the case that it would be better to avoid the issue of storing personal data entirely, such as by anonymising IP addresses through the removal of their final octet before storing them, which is one approach suggested by ICO [**?**].

3.5 Question Creator

The functionality of the question creation system is located in `questionCreator.js`. This tool allows the user to specify a question using the FSM editor in addition to drop-down menus and text fields. The user can then obtain the JSON object that specifies the question. An instance of this tool is available at `http://homepages.inf.ed.ac.uk/s1020995/stable/edit.html`.

Due to the security restrictions on file system access from browser JavaScript, it is not possible to modify a local question database directly. An effort was made to allow file modification via the "load" and "save as" dialogues but this would have been a difficult to use and difficult to implement solution. To allow easy editing of existing questions, a standalone application could be created, but this has not been a development priority. As a workaround, questions can be sent to the server to be saved to a file there, but this is not a good solution. The user may also simply copy and paste the data into a question database.

The tool has been useful during the development process but it will require work to improve its usability for it to be useful to anyone not familiar with the details of the system.

The tool will also need to be updated to reflect the changes in question format that will result from the completion of version 2.0 of `fsm.js`.

Chapter 4

Evaluation

4.1 User Testing

Input from users of the system was gathered from early on in the development process and used to guide development. This feedback was gathered in a number of ways.

In-person Testing

The system was tested by watching users interact with the system to determine what problems they had with the system and where the behaviour of the system differed from their expectations. No formal protocol was used, as the goal was to gain general insight rather than to answer a particular question in a rigorous manner.

This testing provided useful insight but the sourcing of volunteers was difficult. Family and friends volunteered as test subjects and were useful, but the effort to attract volunteers from the target demographic of Inf1:CL students was unsuccessful and attracted only one volunteer whose time was shared with two other developers.

Feedback via Analytics

The analytics system provided useful insight into the way the tool was being used. This was particularly useful in comparing user response to different questions, as was discussed in the section on question design. While the system draws on data from a large number of users compared to the other methods, drawing useful conclusions from the data is difficult. The power of the analytics system could be increased through the use of A|B testing, but that was not done as part of this project.

Feedback via Survey

A link to a survey was included in the tool in an effort to gather additional feedback. However, feedback from the survey was of limited utility, combining the low response rate of in-person testing (12 survey responses between August 2015 and January 2016) and the difficulty of extracting insight of the analytics approach.

No particularly useful insights were gained from the use of the survey. The design of the survey was flawed in that it was seeking to evaluate the tool generally, rather than answer some particular question about the user experience. With a better designed, more focused survey it is possible some insight into a particular question could be gained. However, as it takes so long to gather a useful number of responses, most questions could be better answered using other methods.

Feedback via Question Ratings

As noted previously, the system of soliciting a rating for each question was not successful as the response rate was too low to draw any conclusions from the results. In addition to the low response rate, users who did rate questions tended only to rate the initial questions. When soliciting feedback in this way, a balance must be struck between maximising the response rate and minimising the annoyance to users. Given the small number of users of the tool, some significant annoyance would have to be imposed to obtain a useful number of responses, which may not be desirable.

Indirect Feedback

User feedback was also gathered in less direct ways, such as by talking with the teaching assistant for Inf1:CL about students' experiences and by reading posts on the Inf1:CL Piazza forum discussing the tool. These methods did not lead to actionable changes, but did provide confirmation that students were generally able to make use of the tool.

Future Evaluation

The evaluation strategy of the second half of the project will be adjusted based on what has been successful in the first half.

In-person testing will be largely unchanged – efforts will be made to recruit participants for in-person testing both from students taking the resit exam in August, and from the 2016/17 Inf1:CL class during the first semester. More usage data will be collected through the analytics system, which may be expanded to measure additional variables (for example recording the number of users utilising the ability to view an execution trace for each question). The analytics system may also be expanded to include A|B testing functionality that could be used to evaluate potential changes.

A different approach will be taken with feedback from surveys and question ratings. The use of surveys will be de-emphasised – if a survey is used at all it will be to answer a specific question rather than as a method of obtaining general feedback. Requests for question ratings could be made more obtrusive in an effort to achieve a useful response rate; or the system of question ratings may simply be discontinued.

4.2 Requirements

Another way of assessing the success of the project is to compare it to the project requirements outlined above. As this is the first part of a two year project, this comparison will also highlight areas that will require further work over the next year.

Usability As discussed above, the feedback from users has provided a good indication that students are generally able to make use of the tool without significant difficulty. Further usability improvements could be identified through further testing, particularly by making use of A|B testing.

User Experience The quality of user experience is difficult to measure directly. 100% of the 12 survey respondents agreed or strongly agreed that the questions presented are interesting, but this is not a particularly meaningful result. As with usability, changes made to improve the user experience will be based on evidence from user testing.

Compatibility The tool was developed using the Chrome browser with additional testing done using Firefox. As can be seen in figure 5, these two browsers account for 75% of users. As Safari (which, as the third most observed browser, was used by 12% of site visitors) is only available for Apple devices, there were limited opportunities to conduct tests using it. While Safari and Chrome are similar in that they both use the WebKit layout engine, they use different JavaScript engines and so there is the possibility of compatibility issues. Testing was also done using the browsers available on University Windows machines and School DICE machines to confirm compatibility.

This compatibility testing did uncover some significant issues, with important elements such as arrowheads or check-boxes not appearing in Firefox. These issues have been fixed but there are still minor visual differences in the tool across browsers.

Maintainability Maintainability is still a problem with the project. Completing version 2.0 of the project should address some of the issues of code quality. The failure to create a useful automated test suite also lowers maintainability, and will have to be rectified in the second half of the project.

The presence of the question creator tool does in theory provide a way of modifying the question set without knowledge of the system, but in practice it is probably too unintuitive in its current state.

Security Due to the the limited amount of work done on the server-side, the attack surface of the system is limited. A flaw in the analytics system that made it vulnerable to stored cross-site scripting (XSS) attacks was identified and fixed. The issue was that the page names displayed on the statistics page were taken directly from analytics responses, which could have been crafted to contain JavaScript.

An attack on the analytics system could disable it relatively easily, by causing the system to produce enough logs to exceed the 50MB disk quota, but this would not prevent access to the tool and would not result in the disclosure of any of the log data.

Accessibility The accessibility of the tool is currently lacking. No accessibility testing was carried out. While it may not have been possible to secure testing assistance from a person with visual impairments from the small target audience of students of Inf1:CL, other approaches to evaluate and improve accessibility could have been used. For example, the system could have been tested by accessing it via a screen reader or by using software to simulate visual impairments such as colour-blindness.

While solving accessibility challenges can be a project in itself [**?**], it is an important consideration that will be addressed in the second half of this project.

Chapter 5

Further Work – Year Two

The plan is for work in year two to be done in a similar way as the work discussed in this report – using an iterative approach to development, incorporating regular user feedback. For this approach, making detailed up-front plans is not necessarily appropriate as information and experience gained at each stage will shape the future path of the project. Nonetheless, a rough timeline has been included (Fig:7).

However, there are some things that will definitely need to be completed. These include completing the version 2.0 rewrite; addressing the lack of accessibility features; and introducing ways for users to work through NFA to DFA conversion and the machine minimisation process.

Other avenues that could be explored, depending on how the project progresses, are discussed below.

5.1 Integration with Other Tools

Integrating with other tools targeting Inf1:CL would provide a more cohesive experience for students on the course.

This integration could take the form of simpler changes, such as agreeing on visual design elements to give the applications a unified look, through to more complex integration, such as introducing questions to tie together concepts from different tools. It could even take the form of having some interoperability, such as by allowing elements to be dragged from one tool into another.

5.2 Using FSM tool for Other Courses

The tool could be adapted to also serve the requirements of other courses that make use of FSMs such as Inf2A. The tool has already been provided to Inf2A students as

a resource to revise the basics of FSMs, but its utility could be further enhanced by adding a question set covering the use of FSMs in language processing.

5.3 Harnessing a Larger Audience with A|B Testing

A|B testing is widely used to optimise the conversion rate on commercial sites. If a sufficiently large audience for the tool could be found, such as by providing it for use in other universities, it is possible that A|B testing could be used to optimise the user experience.

Chapter 6

Conclusion

This project resulted in the creation of a tool to support the teaching of finite state machines to students taking the Informatics 1: Computation and Logic course.

This tool provides a way for students to create FSMs and simulate their execution on some input. The tool also provides an interactive question set utilising this simulator. The question set incorporates instruction, with the aim of introducing concepts gradually to guide students through the learning process.

The tool includes an analytics system that collects usage data. This data has already been useful in improving the tool, as it helped to identify user stumbling blocks. This system could be further expanded in the project's second year to include an A|B testing system.

Feedback from users has been gathered throughout the development process, with the tool being distributed to Inf1:CL and Inf2A students. While no formal trial of the effectiveness of the tool was conducted, anecdotally the reception was positive.

To improve maintainability and aid development, a companion tool that facilitates the creation of questions was implemented. While it has succeeded in aiding development, further work will be needed for it to be useful to other developers.

Work on a second version of the system, which addresses some failings in the original codebase has begun. Completing this work constitutes the initial goal in year two of the project, laying the ground for further work.

Work on the second half of the project will continue to take an iterative approach, with the path of the project guided by the insight gained from each iteration, rather than from a detailed up-front plan.

References

[1] Matt Chapman. The Finite State Machine Explorer. http://www.mathcs.bethel.edu/~gossett/DiscreteMathWithProof/fsme/fsme.html.

[2] Kyril Faenov and Stephan Merz. State Machine Simulator. http://www.pst.ifi.lmu.de/personen/merz/software/fsm/Simulator.html.

[3] Ivan Zuzak and Vedrana Jankovic. FSM Simulator. http://ivanzuzak.info/noam/webapps/fsm_simulator/.

[4] E4Coder. SMCube. http://www.e4coder.com/content/smcube-state-machine-editor.

[5] Evan Wallace. Finite State Machine Designer. http://madebyevan.com/fsm/.

[6] Google. Alan turing's 100th birthday. http://www.google.com/doodles/alan-turings-100th-birthday.

[7] James A Kulik and JD Fletcher. Effectiveness of intelligent tutoring systems a meta-analytic review. *Review of Educational Research*, pages 42–78, 2015.

[8] Russell Boyatt, Mike Joy, Claire Rocks, and Jane Sinclair. What (use) is a mooc? In *The 2nd international workshop on learning technology for education in cloud*, pages 133–145. Springer, 2014.

[9] Sidney D'Mello, Blair Lehman, Jeremiah Sullins, Rosaire Daigle, Rebekah Combs, Kimberly Vogt, Lydia Perkins, and Art Graesser. A time for emoting: When affect-sensitivity is and isn't effective at promoting deep learning. In *Intelligent tutoring systems*, pages 245–254. Springer, 2010.

[10] Thomas M Connolly, Elizabeth A Boyle, Ewan MacArthur, Thomas Hainey, and James M Boyle. A systematic literature review of empirical evidence on computer games and serious games. *Computers & Education*, 59(2):661–686, 2012.

[11] Fabienne M Van der Kleij, Remco CW Feskens, and Theo JHM Eggen. Effects of feedback in a computer-based learning environment on students' learning outcomes a meta-analysis. *Review of educational research*, 85(4):475–511, 2015.

[12] Sandra Miller. Formative computer-based assessments: The potentials and pitfalls of two formative computer-based assessments used in professional learning programs. 2009.

[13] Fabienne M van der Kleij, Theo JHM Eggen, Caroline F Timmers, and Bernard P Veldkamp. Effects of feedback in a computer-based assessment for learning. *Computers & Education*, 58(1):263–272, 2012.

[14] Tim Sullivan. How games could save the world. *Harvard Business Review*, 89(6):134 – 135, 2011.

[15] Extra Credits. Extra credits - gamification - how the principles of play apply to real life. `https://www.youtube.com/watch?v=1dLK9MW-9sY`, 2012.

[16] Chris Franklin. Errant Signal - Gamification. `https://www.youtube.com/watch?v=pWfMjQKXZXk`, 2012.

[17] Chris Hecker. Achievements considered harmful? `http://chrishecker.com/Achievements_Considered_Harmful`.

[18] Ian Bogost. Why gamification is bullshit 2. *The gameful world: Approaches, issues, applications*, 65, 2015.

[19] Fiona Fui-Hoon Nah, Qing Zeng, Venkata Rajasekhar Telaprolu, Abhishek Padmanabhuni Ayyappa, and Brenda Eschenbrenner. Gamification of education: A review of literature. In *HCI in Business*, pages 401–409. Springer, 2014.

[20] Ross Kirsling. Directed Graph Editor. `http://bl.ocks.org/rkirsling/5001347`.

[21] Force layout. `https://github.com/mbostock/d3/wiki/Force-Layout`.

[22] Mike Bostock. Sticky force layout. `http://bl.ocks.org/mbostock/3750558`.

[23] The University of Edinburgh. Additional fees. `http://www.ed.ac.uk/student-funding/tuition-fees/undergraduate/additional-fees` (accessed 2016/02/21), 2015.

[24] W3Techs. Web technologies of the year 2015. `http://w3techs.com/blog/entry/web_technologies_of_the_year_2015`.

[25] School of Informatics Computing Support. Informatics homepages web service. `http://computing.help.inf.ed.ac.uk/homepages` (accessed 2016/03/04).

[26] Informatics 1 - Computation and Logic resit exam. `https://exampapers.ed.ac.uk/record/49243/1/20141344.pdf`, August 2014.

[27] Informatics 1 - Computation and Logic exam. `https://exampapers.ed.ac.uk/record/39966/1/2011162.pdf`, December 2010.

[28] Michael Sipser. *Introduction to the Theory of Computation*. Cengage Learning.

[29] Rana Barua and Kishan Chand Gupta. *Finite Automata*. CRC Press, 2012.

[30] Mike Malone. TIFU by using Math.random(). `https://medium.com/@betable/tifu-by-using-math-random-f1c308c4fd9d`.

[31] Administrative Data Liaison Service. Data Protection Act – legal guidance. `http://www.adls.ac.uk/adls-resources/guidance/legal-guidance/data-protection-act/` (accessed 2016/02/24).

[32] Information Commissioner's Office. Determining what is personal data – v1.1. `http://ico.org.uk/media/for-organisations/documents/1554/determining-what-is-personal-data.pdf` (accessed 2016/02/24).

[33] Information Commissioner's Office. Personal information online code of practice. `http://ico.org.uk/media/for-organisations/documents/1591/personal_information_online_cop.pdf` (accessed 2016/02/24).

[34] Information Commissioner's Office. Guide to data protection. `http://ico.org.uk/for-organisations/guide-to-data-protection/` (accessed 2016/02/24).

[35] Chris Wait. Talking Buses: A navigation-assistance app for blind and partially-sighted users. Master's thesis, University of Edinburgh, 2014.

Appendix A

Figures

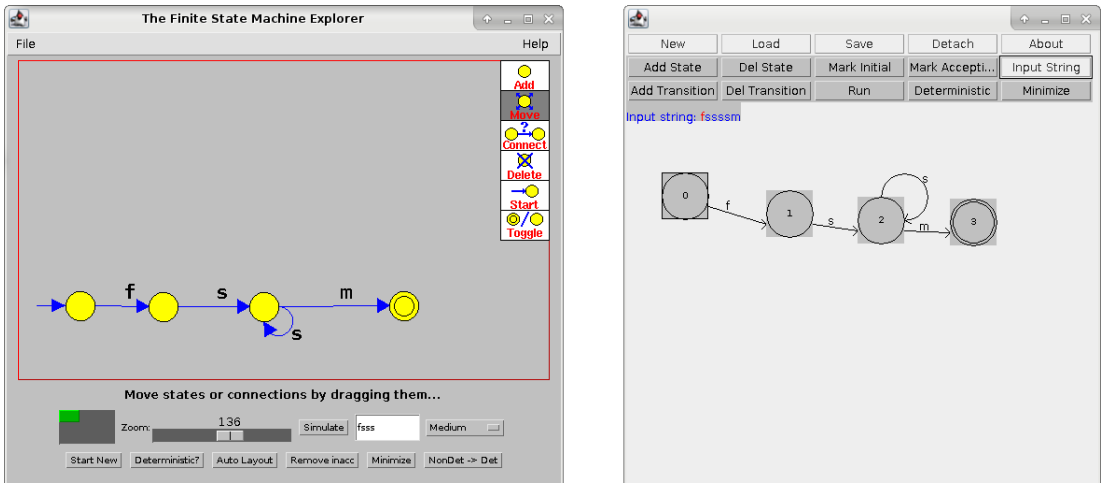

Figure 1: Chapman's *Finite State Machine Explorer* (L) and Faenov and Merz's *State Machine Simulator* (R)

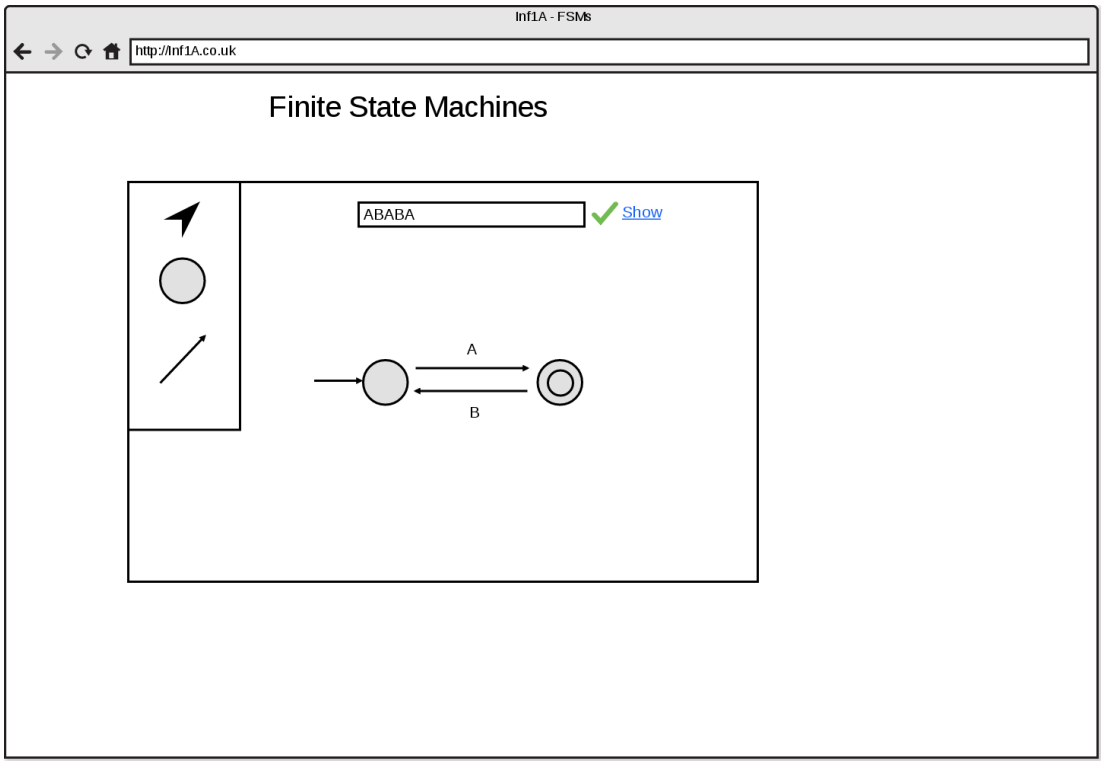

Figure 2: Initial mockup of FSM editor interface

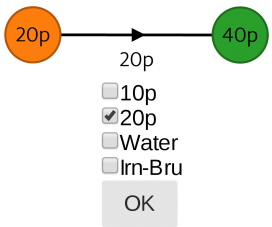

Figure 3: Revised method of specifying transition conditions

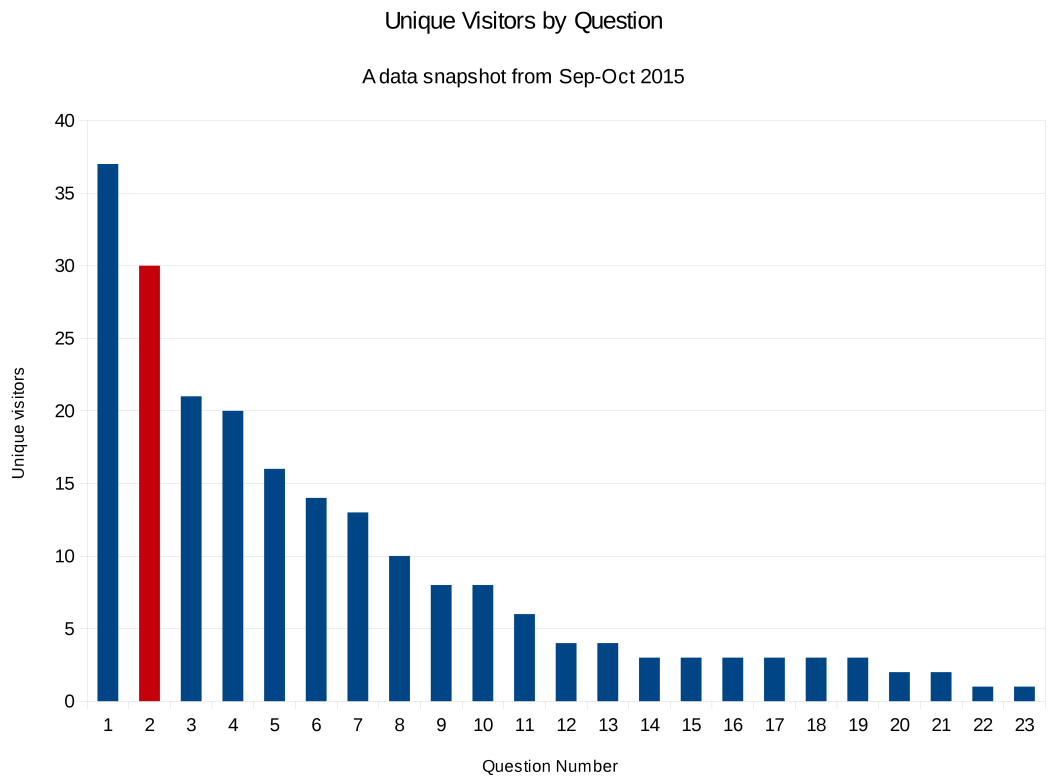

Figure 4: Usage data aids in the identification of questions that cause users to quit

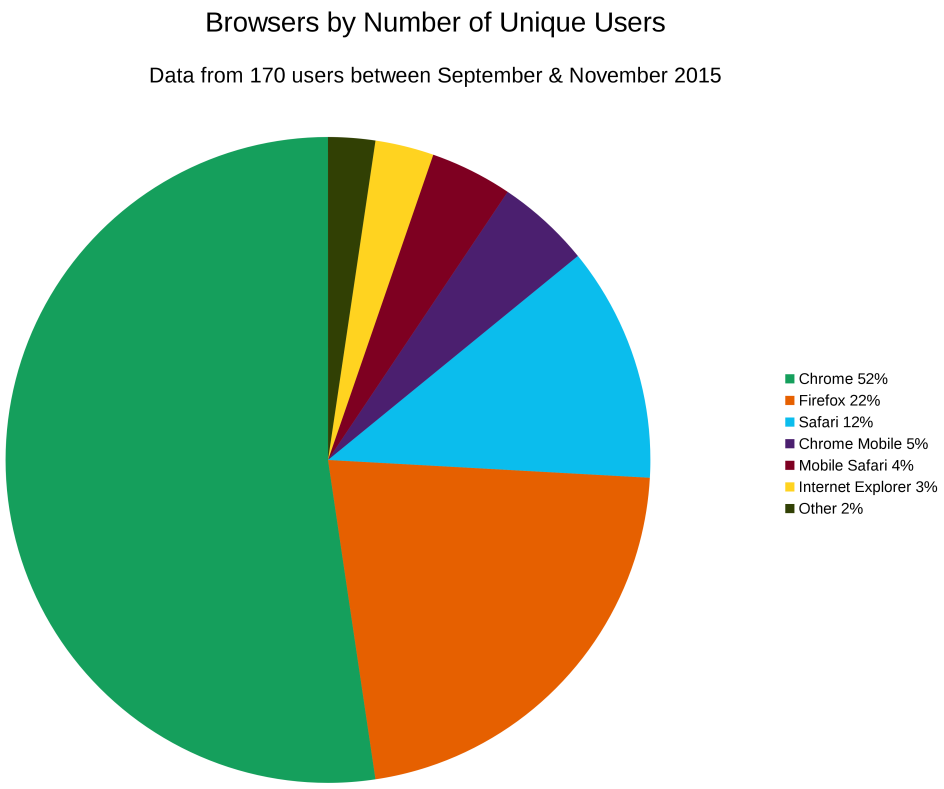

Figure 5: Browsers used to access the tool between September and November 2015

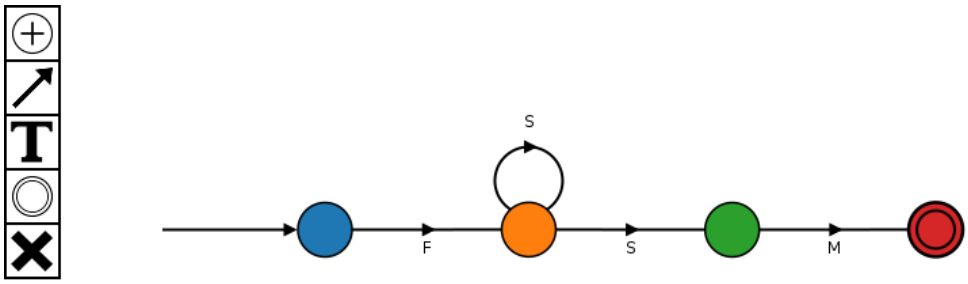

Figure 6: The editor interface

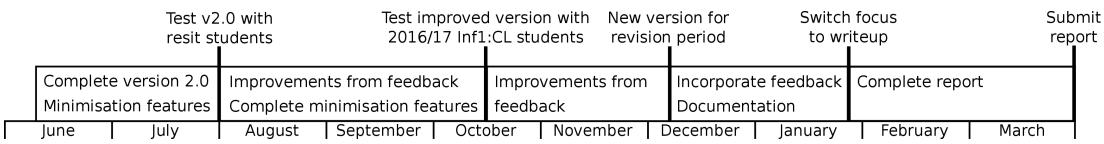

Figure 7: Timeline for the second half of the project