Refining the FSM Workbench

Matthew Hepburn

MInf5 Project
Master of Informatics
School of Informatics
University of Edinburgh

2017

Abstract

This report details the second year of the design, implementation and testing of the Finite State Machine Workbench, an educational tool to accompany the first year undergraduate course Informatics 1: Computation & Logic. The Workbench provides a guided introduction to the topic of finite state machines (FSMs) through a set of interactive exercises and allows students to experiment by creating and simulating their own machines. The Workbench is different from other FSM tools in its use of a real-time force-directed layout system – encouraging focus on FSM concepts, rather than on diagram arrangement – and in its novice-friendly design. The Workbench is a HTML5 web application, implemented using Scalable Vector Graphics (SVG) and the Data-Driven Documents (D3) JavaScript library.

Acknowledgements

With thanks to Professor Fourman, Thomas Cumming and Dagmara Niklasiewicz.

Table of Contents

Chapter 1

Introduction

1.1 Project Overview

The Problem – Supporting Inf1:CL Students

The aim of this project is to help students taking Informatics 1: Computation and Logic to improve their understanding of finite state machines (FSMs). The understanding that this project seeks to instil consists not just of knowledge of the principles of the topic, but also an intuitive understanding of finite state machines built through practical experience.

Finite state machines are particularly well suited for dynamic visual displays – static diagrams struggle to convey behaviour that is clear when seen in motion. However most of the existing resources for students, constrained to printable formats, are unable to make use of this fact.

Students can also find it difficult to check their understanding of FSMs using static resources – many problems have multiple equivalent solutions, and novices may struggle to determine whether their solution is equivalent to the one given. This difficulty in testing their understanding can deter students from the practice and experimentation that is vital to gaining an intuitive understanding of the topic.

Solving the Problem – the FSM Workbench

Through parts one and two of this project a web-based tool, the FSM Workbench, has been developed to support Inf1:CL students. An instance of this tool is available at `http://homepages.inf.ed.ac.uk/s1020995/fsmworkbench`. The tool's source code is available at `https://github.com/MatthewHepburn/FSM-Workbench`, under the GNU General Public Licence.

The Workbench allows students to create finite state machines, using a graphical interface which has been designed to be easy to learn to use even for users without

knowledge of the topic. Users can step through an animated simulation of their machine processing input. This gives students a way to test their understanding of the topic, and a view of the mechanics of finite state machines in action. The real-time force-directed layout system of the Workbench helps to keep students focused on the key concepts, rather than on tweaking the layout of their diagrams.

Building on this core functionality, a set of interactive exercises was developed to guide students through the topic. It makes use of several kinds of question, prompting the user to create machines as well as to answer questions about machines they are given. Starting with the basic mechanics of finite state machines, it goes on to introduce more advanced topics such as non-determinism, transducers, and machine minimisation. This provides students with a more structured learning experience.

Using either the free-form Create tool or the exercises will give students the chance to gain hands-on experience of working with finite state machines, which will improve their intuitive understanding of the topic.

As well as being of direct use to Inf1:CL students, the Workbench is also intended to support the delivery of the course. The system is suitable for giving demonstrations in lectures and can used to create diagrams, such as for use in tutorial sheets. Further, the interactive question system can be easily adapted to better suit the needs of the course lecturer, as questions can be added or modified without the need to edit the code of the application.

Project Goals

Beyond realising the functionality described above, there were other goals that this project sought to meet.

From a User's Perspective

Efficacy The Workbench must be an effective way for students to learn about FSMs. Showing unequivocally that the system is effective would require independently replicated randomised controlled trials – infeasible within the constraints of this project. Within those constraints, a case for the system's efficacy must be built from more informal evidence – although it is crucial that this includes evidence from Inf1:CL students. The groundwork for this case was laid in MIP1, where the design of the system was shaped by user testing and where giving students access to the tool led to a broadly positive response.

Usability To maximise the utility of the Workbench, it is important that it is easy to use – students should be focused on FSMs not on deciphering a complex interface. It is important to consider the particular needs of the target audience, especially when designing an application tailored to so narrow an audience as the students on a single course. One important aspect of this audience is that they are unlikely to spend many

hours with the Workbench – FSMs are a single aspect of one of several courses they will be taking. Few users will become experts with the system, so the interface should be optimised for learnability even if that comes at the expense of more experienced users. In addition to interface concerns, this focus on ease of learning should also be applied to the way that FSM concepts are presented in the Workbench.

Compatibility For the Workbench to be of real use to students, rather than simply being a proof of concept, it has to work well with most of the wide range of devices that students will use to access it. As a web application, the Workbench is well placed to meet this goal. However, even across modern browsers, inconsistencies are common and work must be done to ensure that the application operates correctly on all platforms. Further, the interface should be designed for compatibility with touch-based devices in addition to those with a keyboard and mouse.

Accessibility The most relevant accessibility concern for the Workbench is likely to be accommodating the needs of users with impaired vision. The system should work correctly with screen reading technology and with browser magnification settings. The interface of the system should be interpretable by colour-blind users.

From an Operational Perspective

Maintainability The Workbench should be sufficiently maintainable that it can continue to be useful after this project has concluded. To succeed in this aim, the system needs a logical structure explained with clear documentation, readable code, and a clear procedure to redeploy the system to a more permanent location.

Security As the system does not collect any personal data (see 2.5.3), the Workbench is not a security-critical application. As the system is hosted on a School server, many of the steps needed to ensure security are outwith the control of the developer. However, there are aspects of the system that could be vulnerable to exploitation if care were not taken to ensure security. For example, a cross-site scripting (XSS) vulnerability in the analytics system was noticed and addressed during MIP1[1].

The end result of this project is compared against these goals in 3.3.

1.2 Background

This summary of existing solutions and relevant research is largely unchanged from the version presented in the year one report [1], with the exception of the discussions of JFLAP and of interface design.

1.2.1 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 [2] and Faenov and Merz's State Machine Simulator [3] 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.

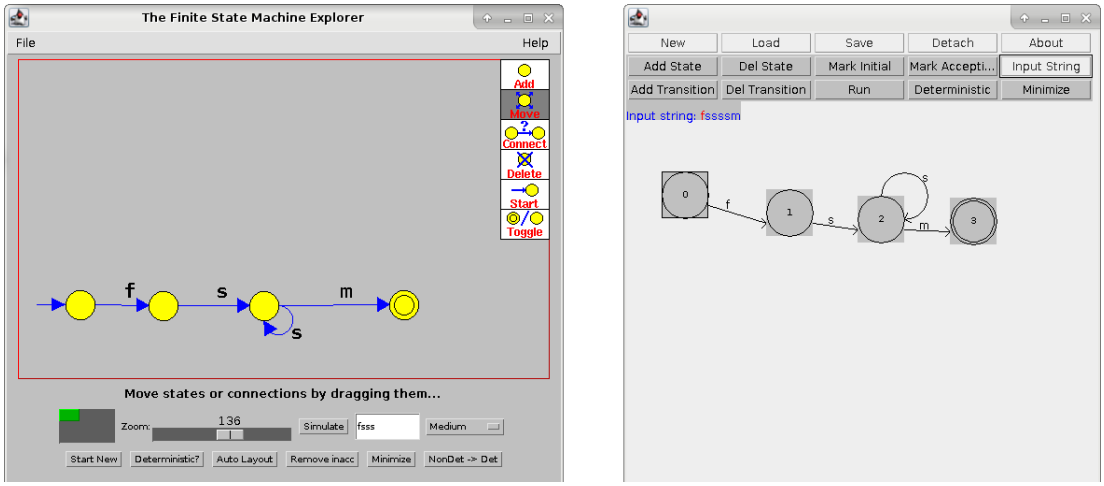

Figure 1: Chapman's *Finite State Machine Explorer* (L) and Faenov and Merz's *State Machine Simulator* (R)

The Java Formal Language and Automata Package (JFLAP) [4], developed by Susan Rodger and her students, is a freely available Java application designed to support the teaching of various topics including finite state machines. This summary focuses only on that finite state machine functionality. Unlike the Java applications of Chapman or Faenov and Merz, JFLAP is still under active development and does not pose any challenges to run. JFLAP is powerful, covering a far broader range of topics than would be possible in this project. This complexity is presented in a way that makes the tool difficult to learn to use however, even with an understanding of the topic. JFLAP does provide extensive documentation [5] as well as an accompanying book [6] but for students unfamiliar with the underlying material this may prove to be overwhelming. JFLAP, with its focus on education, is likely to be the best of the alternatives to the FSM Workbench. While some students would be likely to find it useful, JFLAP's usability issues limit its suitability for use by students new to the topic.

A web version of the JFAP Finite Automaton Editor was produced using JavaScript and SVG for inclusion in the OpenDSA project [7], [8]. It supports the creation and

simulation of finite state machines It also provides procedures to step through the processes of conversion to DFA and machine minimization. The tool has the potential to be useful and its availability as a web application makes it more easily accessible for students. Its main limitations are its user interface – which is not compatible with touch devices and which shares some of JFLAP's usability issues – and its lack of polish.

Both JFLAP and JFLAP Web are discussed in more detail in 3.2, where they are compared with the Workbench.

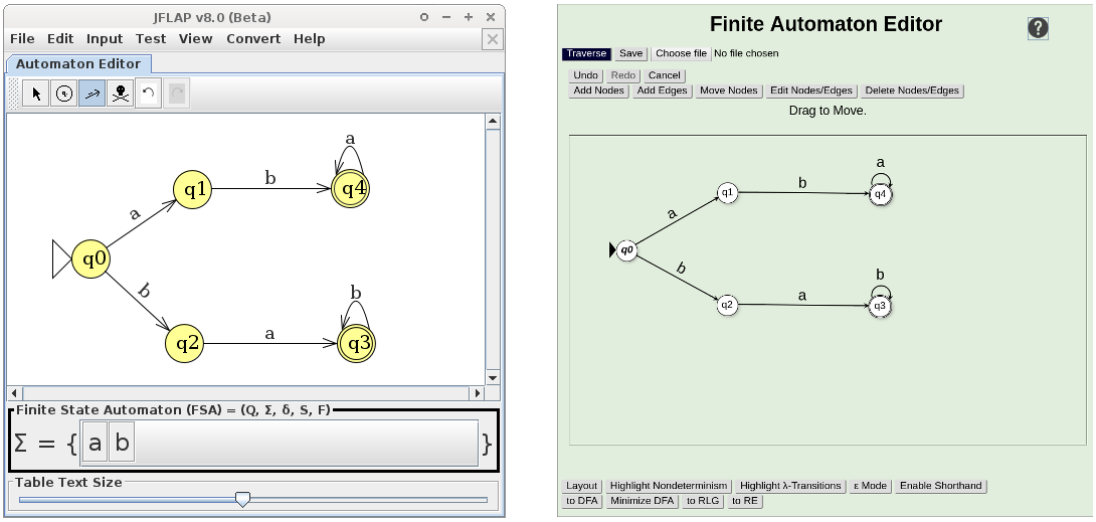

Figure 2: JFLAP (L) and JFLAP Web (R)

The FSM Simulator webapp by Zuzak and Jankovic [9] 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 the FSM Workbench, although in the Workbench 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 some conceptual understanding of finite state machines.

SMCube [10] 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 [11] 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 [12] 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.

Even the existing solutions best suited to student use suffer from interface design that is intimidating and difficult for novices to grasp. Further, these solutions require their users to manually position every element of their FSMs. This can be tedious and time-consuming, discouraging experimentation and distracting students from the underlying concepts. There is, therefore, a need for a usable and novice-friendly FSM simulator which avoids this issue.

1.2.2 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 [13] or MOOCs [14]) or featuring some novel element (e.g user affect sensing [15] or game-based learning [16]).

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.

1.2.2.1 Providing Feedback

A recent meta-analysis [17] 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 considered 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 a correct answer is given in addition to KR feedback.
- elaborative feedback (EF), where an explanation is given in addition to KR feedback.

The meta-analysis concluded 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 were assessed using higher order learning outcomes, i.e. 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 [18] and spend more time

reading immediate feedback [19], the results did not show any improvement in learning outcomes from immediate feedback over delayed 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.

1.2.2.2 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 motivational power of video games [20], [21]. There are also criticisms though, with some [22] 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 [23]. Others, such as Bogost [24], 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 [25] 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 in education, it has not been a focus of this project. The addition of a minor gamified feature was trialled, but it was not found to be effective (see 2.5.2).

1.2.2.3 Interface Design

While the literature on interface design is extensive, it was difficult to find actionable design insights. Simple design rules usually fail to capture the nuances of human behaviour, and so the field is more focused on refining designs through iteration guided by user testing than on formulating general purpose guidelines for interface design.

The notion of consistency provides an illustrative example of the problem of simple design rules. It is a common (and unsurprising) recommendation that user interfaces should be consistent with themselves and with other interfaces that users are likely to be familiar with [26], [27], [28]. However, even advice as simple as this has its

dissenters. Grundin and others argue [29], [30] that, while interfaces should not be inconsistent without good reason, too great a focus on achieving consistency can lead to problematic designs. While consistency is an important part of ensuring that software behaves in the way that users expect, user expectations are not always consistent and certainly are not always consistent in the way that designers expect. So, design rules must be applied with care and they cannot be used as a substitute for user testing.

This is not to suggest that general design principles should not be considered during the design process. Nielsen's frequently cited work on usability heuristics [31], [32] was useful in identifying and prioritising areas where the usability of the Workbench could be improved. For example, the addition of undo/redo functionality was initially judged to be of low priority as most errors can be quickly corrected using the other editing tools. Further, adding functionality that is not strictly necessary ran counter to the goal of keeping the interface as simple as possible. However, when judged using Nielsen's heuristics, this was a clear usability failure of the system and so the functionality was added.

1.3 System Overview

This section exists to give a broad overview of the FSM Workbench.

Structure

The system has two primary student-facing modes, both available from the Workbench's homepage at `http://homepages.inf.ed.ac.uk/s1020995/fsmworkbench`. First, the Create mode is a general purpose FSM editor and simulator. The second mode is the interactive question set, which includes 9 distinct question types. The system also includes two sets of questions, which accompany tutorial sheets (see 2.8).

Other subsystems exist to aid development and maintenance. One is the question creation system, which provides an interface to specify questions using the 9 implemented question types, available at `http://homepages.inf.ed.ac.uk/s1020995/fsmworkbench/questionCreator.html`. The other is the analytics system, which collects data on usage of the system. A live summary of some of the collected metrics is available at `http://homepages.inf.ed.ac.uk/s1020995/fsmworkbench/stats.html`.

Technologies

On the client side, the Workbench is implemented using Scalable Vector Graphics (SVG) and JavaScript with the Data-Driven Documents library (D3). The JavaScript code uses ECMAScript 6 syntax but it is currently transpiled to ECMAScript 5 for compatibility as browser support for ECMAScript 6 remains inconsistent.

Analytics data is collected and aggregated with Python scripts, called through the Common Gateway Interface (CGI). Python is also used for the build process, which includes building question pages using templates, as well as transpilation and minification steps for compatibility and efficiency.

1.4 Summary of Work Done

1.4.1 System Rewrite

A large part (>100 hours) of the MIP2 development work consisted of completing the rewrite of the system that began towards the end of the MIP1 phase. The design and testing done in MIP1, along with the JavaScript experience gained, enabled the creation of a system that is more reliable and more extensible than the previous system. While much of the work was focused on re-engineering existing functionality in a more logical and more reliable way, the rewrite also included some significant functional changes such as the ability to display multiple machines at once and the addition of support for machines with multiple initial states.

1.4.2 Interactive Question Set

The interactive question set developed in MIP1 was ported to the rewritten system. In addition, two new question types were developed covering the conversion of machines to DFA form and the minimisation of machines.

1.4.3 Display and Animation

As part of the process of refining the FSM Workbench, changes were made to improve the visual clarity of the tool. One way this was done was by implementing collision detection for some transition and label elements to reduce the frequency with which elements overlap. The addition of animation when showing machine execution also helped to improve visual clarity.

1.4.4 Tutorials

Two tutorials for Inf1:CL students covering finite state machines and regular expressions were co-written with Prof. Fourman and Dagmara Niklasiewicz. Students were encouraged to make use of the FSM Workbench when completing the tutorial and the Workbench was modified to support students with this. This process allowed for the collection of analytics data and informal feedback, both of which provided confirmation that students were able to learn how to use the tool and found it to be helpful.

The process of developing the tutorials also led to useful insights, discussed in 2.8.2.

1.4.5 Visualising the Subset Procedure

One of the FSM topics in Inf1:CL that the Workbench did not support in MIP1 was the conversion of non-deterministic finite automata (NFAs) to deterministic finite automata (DFA) form. An initial attempt at addressing this took the form of a new question type and some questions in the exercise set. This approach lacked clarity however, so another attempt at showing the process was made, this time with the result incorporated into the Create tool (see 2.9).

1.4.6 Other Quality of Life Improvements

Both user feedback and direct use of the tool led to the identification of some minor features that could improve the usability of the tool. The improvements that were implemented included a system for saving and loading machines to and from the browser's local storage, the ability to export machine diagrams as SVG files, and undo/redo functionality.

1.4.7 Analytics and A|B testing

The analytics system developed during MIP1 was refined to collect more granular data on how users interact with the system, for example determining that <10% of users made regular use of the context menu system. The changes to the analytics system were done with a focus on ease of extensibility, which reduced the work required to instrument later additions. Support was also added for A|B tests, although the single test conducted did not lead to statistally significant results (see 2.5).

Chapter 2

Refining the FSM Workbench

2.1 Strategy

Work in the MIP1 phase of the project followed a development strategy which emphasised frequent iteration, allowing for experimentation and giving an opportunity for alternative approaches to be evaluated. This approach came at the expense of code quality, which suffered both from the focus on the speed of development as well as from the inability to foresee and plan for changes brought about through experimentation. Technical debt was accrued, with development rushed to allow multiple iterations within the single semester of Inf1:CL's delivery.

With the design groundwork done and the results validated by user testing in MIP1, the MIP2 phase of the project focused more on re-engineering the Workbench than on extensive redesigning. To that end, the strategy of this phase relied on slower, more carefully considered development since the design of the Workbench could be relied upon to change much less than it did during the earlier stages of development.

2.2 Estimating Development Time – Rewriting and the Pareto Principle

Rewriting the MIP1 prototype, to bring the codebase up to a good standard, required far more work than was initially expected. This section discusses why this was the case.

Both rewriting an application and advancing an application from prototype to production quality are tasks whose complexity is easy to underestimate.

The underestimation of the work needed to rewrite an application to improve code quality is a common psychological trap for developers [33], [34] , that goes beyond the usual difficulties of estimating development time. When attempting to estimate the work required, it is easy to overlook the necessary nuances of the codebase. The exist-

ing code may seem overly complicated to achieve functionality that seems straightforward from a high level but even if some simplification is possible, it is likely that much of the complexity is there for good reasons. Rewriting can also uncover aspects of the problem that were not fully addressed in the original code, for example edge cases that were only imperfectly worked around.

It can also be easy to believe that having a working prototype means most of the development work is done, but in reality the Pareto principle – that 20% of the cause leads to 80% of the outcome – is a more accurate guide. As an application goes from prototype to completion, improvements can require greater and greater amounts of work as that final 20% of effect is achieved. Partly this results from the way that applications can appear more complete than they really are – the obvious cases may work but problems lurk in more complex cases. Another contributing factor may be that easily implemented improvements may be made early in the process, as the developer seeks to maximise the return on their time investment – this selective effect results in the pool of remaining problems increasing in average difficulty. Also, as the complexity of an application increases, every new addition has more potential to interact adversely with another part of the system – as development progresses each new change is more likely to require changing some other aspect of the code, increasing the time needed.

So, there are good reasons why the task of rewriting the code of the MIP1 prototype into production quality code took more work than expected. The process could have been sped up somewhat if the focus had simply been on implementing functionality, but the extra work needed to achieve this using high quality code was worthwhile, resulting in a more maintainable and extensible application.

2.3 Rewrite

The first task in this phase of the project was to complete the system rewrite begun in MIP1.

2.3.1 Motivation

Towards the end of the MIP1 development phase, it was decided that it was necessary to rewrite large parts of the application. The reasoning for this decision is laid out in detail in the MIP1 report, but essentially it was driven by two reasons. First was a desire to change some of the underlying assumptions in the code of the MIP1 prototype, namely that only one FSM would be displayed at a time and that each finite state machine could have only a single initial state. Second was the low code quality of the prototype, manifesting both in severe architectural flaws and in more general deficits in readability and robustness.

2.3.2 MIP1 Work

MIP1 identified the need for re-engineering and established the basic framework on which MIP2 is based. At the end of MIP1, the rewritten version of the system had the basic display and editing logic in place in a rough form. Some of the logic required to simulate finite state machines had been reimplemented, but no visualisation had yet been written. None of the question types from the initial version had been reimplemented. A demonstration of the partially rewritten system as it existed at the end of MIP1 is available at `http://homepages.inf.ed.ac.uk/s1020995/v2-mip1/`.

2.3.3 Architectural Changes

The most significant architectural shortcoming of the prototype was the unnecessarily tight coupling between the display logic and the finite state machine logic, with display logic modifying the state of the model directly. It is necessary for the display logic to rely on the model it is displaying to some extent but the dependence of the model on the display logic was unnecessary and made the software more difficult to test or extend. This interdependence is part of the reason why altering an assumption in the model required so significant an amount of work as to justify a rewrite.

The rewritten version rectified this deficiency, separating out the finite state machine logic into its own file and improving encapsulation by providing functions to modify the model rather than allowing other parts of the program to directly modify the model's internal state. This separation allowed for the creation of an automated test suite for the model, independent of any browser, which was not possible before.

In addition to being separated from the display logic, the FSM logic was rewritten to use an object-oriented representation of machines. This was done primarily to allow the system to model more than one machine at a time. Modelling additional machines was difficult under the previous architecture, where all of the FSM functions operated only on the single machine being displayed. Beyond the immediate advantage of being able to display more than one machine, modelling additional machines greatly simplified the process of programming more complex FSM operations. For example, the algorithm used to find an input sequence to distinguish two machines requires modelling the two input machines and both of their inverses. This would have been very difficult to implement in a readable way under the old architecture.

The implementation of this object-oriented representation was achieved by using three constructors to create objects representing machines, states and transitions. When these constructors are invoked using the `new` keyword, a new object is created containing both data and copies of the functions to manipulate it.

A more efficient way to achieve the same effect would have been to add functions to the prototype of each constructor (as shown in listing 2), rather than declaring them within the constructor. This is a more efficient method as only a single instance of each function is created, rather than one instance per object [35]. The less efficient method was used as the possibility that otherwise equivalent methods of creating objects would

```
const Model = {
    Machine: function(id) {
        this.id = id;
        this.nodes = {};
        this.links = {};
        ...

        this.isEquivalentTo = function(machine){...};
        ...
    },
    Node: function(machine, nodeID, ...){
        this.machine = machine;
        this.id = nodeID;
        ...

        this.toggleAccepting = function(){...};
        ...
    },
    Link: function(machine, linkID, ...){
        this.machine = machine;
        this.id = linkID;
        ...

        this.setInput = function(inputList, hasEpsilon){...};
        ...
    },
    ...
};
```

Listing 1: The method used to implement an object-oriented machine representation

have significantly differing efficiency was not initially considered. The use of ES6 in the rewritten version would also have allowed the use of the `class` keyword [36], which is syntactic sugar for the more efficient declaration. It may be worthwhile to refactor the code to use the more efficient `class` declaration, although the constructors are called infrequently enough that the efficiency gains would be very minor.

These architectural changes were all present in the rewrite to some extent by the end of MIP1, with their implementation being completed and extended in MIP2.

```
const Model = {
    Machine: function(id) {
        this.id = id;
        this.nodes = {};
        this.links = {};
    },
    ...
};
Model.Machine.prototype.isEquivalentTo = function(machine){...};
Model.Machine.prototype.minimize = function(){...};
```

Listing 2: A more efficient way of implementing an object-oriented representation, which was not used

2.3.4 ECMAScript 6

The process of rewriting provided an opportunity to incorporate features from ECMAScript 6 into the codebase, with transpilation to ECMAScript 5 performed to maintain compatibility.

One of the most notable of the new language features that was used in the rewrite is the addition of block level scoping through the `let` and `const` keywords. This makes it possible to avoid one of the more subtle idiosyncrasies of JavaScript, leading to more readable code.

Arrow functions – which provide a concise, readable way of declaring anonymous functions – were also used. However, the behaviour of arrow functions subtly differs from that of standard function declarations in that the invocation of an arrow function does not rebind the `this` keyword, which risks causing confusion in some instances. In practice, the benefits have outweighed the potential downside.

Finally, generator functions (functions whose execution can be resumed after returning) found some minor usage, with their ability to encapsulate state making them well suited to generating a series of names or identifiers.,

2.4 Graphical Improvements

The clear display of finite state machines is a core part of the functionality of the Workbench. The MIP1 prototype had some failings in this area – complex machines could become too tangled to easily interpret, and no use was made of animation to make transitions easier to follow when visualising execution.

2.4.1 Smarter Element Placement

To improve visual clarity, a system was added to place node labels and loops in such a way as to reduce overlapping. This placement is achieved by regularly testing for collisions at points around each node with a loop or a label until a point with no collision is found (or until all candidates have been ruled out and the default position is used).

The need to work in real-time even on less powerful devices imposed additional constraints on the system.

The real-time nature of the system meant that it had to return a consistent placement of elements to avoid those elements moving erratically. Inconsistent placement would be more acceptable in a system that only does layout once (such as Zuzak and Jankovic's FSM Simulator) or that does layout on demand (such as JFLAP). Consistency was accomplished by using a simple system where elements are positioned in a fixed order in such a way that no pair of elements is mutually dependent.

Performing positioning in real time also limits the amount of computation that can be done. The initial implementation of this placement system, despite its relative simplicity, had a significant impact on performance, with noticeable stutter even on desktop machines. This necessitated some optimisation work, as that level of performance degradation was too severe for the benefits gained. By using memoisation to reduce the number of expensive operations and by replacing expensive bounding box queries with direct calculations or close approximations the impact on performance was reduced to an acceptable level.

The efficacy of these optimisations was determined by using the Chrome JavaScript CPU profiler while perturbing the machine to cause the system to reposition elements. One example of the improvements made is the `getTextLength` function, which determines the exact length in SVG coordinates that a string will be when rendered with a specified class and font size. This is frequently called when determining the layout and it is an expensive operation as the only exact method is to add the string to the DOM, query its length, then remove it. Memoising the results of this operation led to it accounting for ~0.5% of CPU usage time rather than ~10%.

The expensive inbuilt function to get the exact bounding box of an element (`getBBox`) was using ~30% of CPU time prior to optimisation. The frequency with which it is called was reduced through the memoisation of `getTextLength` and by calculating the bounding box directly where possible. This led to `getBBox` accounting for ~0% of CPU time after optimisation.

Although significant performance gains were made, less powerful devices would still benefit from further optimisation to this system.

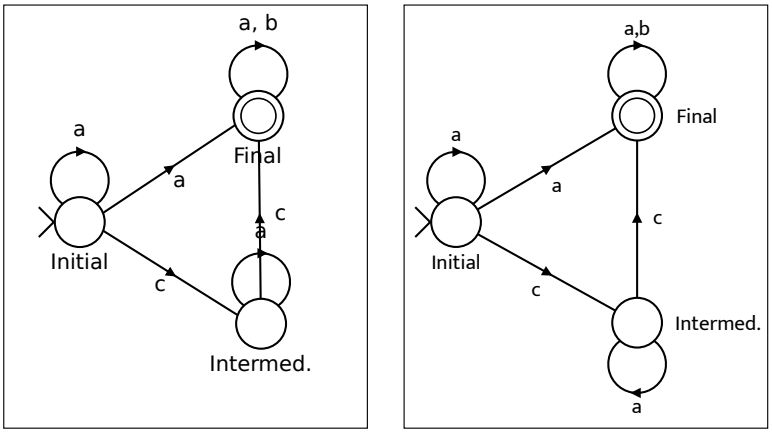

Figure 3: Naive element placement (L), smart element placement (R)

2.4.2 Animation

Animation can make it easier to follow changes in state than if abrupt transitions are used. This was the goal when incorporating animation into the visualisation of machine execution.

Animation was present in this system in the MIP1 prototype – input symbols fall away as they are consumed and transition lines swell as they are highlighted. However, this failed to capture the key movement of the machine changing state.

In MIP2, the animation of input being consumed was removed – it was excessive and drew attention away from the machine itself – and replaced by a visually cleaner system of greying out and striking through consumed input and underlining the next symbol to be processed. The transition between states was visualised in a way that emphasised movement. The green colour used to signify the current state travels along transition lines as a green arrow before appearing in the new state.

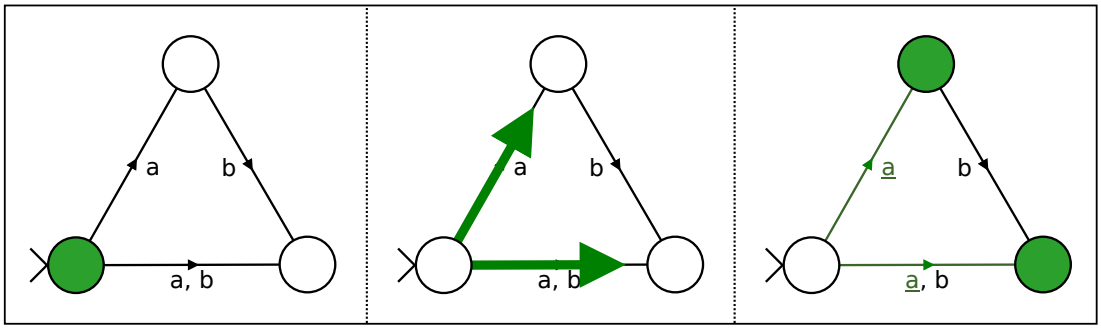

Figure 4: Animation of transitions in MIP2

While this system is effective at making the change of state clear, there is a flaw in the implementation which manifests in more complex examples. When chains of transitions are activated, i.e. where the first step enables an ε-transition, all of the transitions are activated at once. A clearer visualisation would have multiple waves of animation in this case, with the first transitions completing before any dependent transitions are

shown. This is not an insurmountable problem, but fixing it would require a moderate amount of development time.

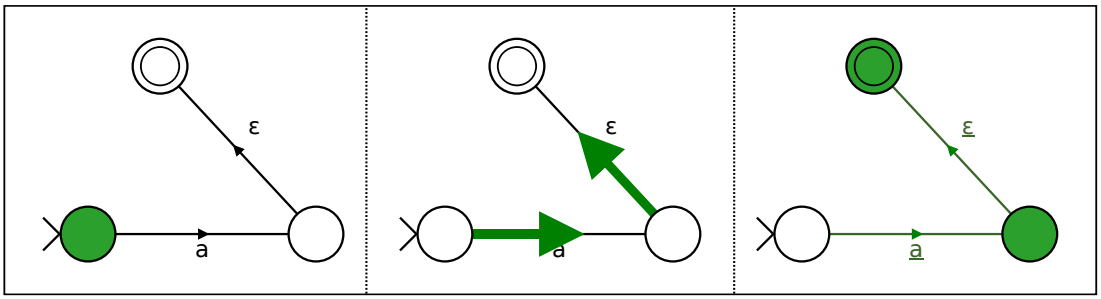

Figure 5: Flaw when animating chains of transitions

2.5 Analytics System

The analytics system was overhauled towards the end of MIP1, prior to the decision to rewrite, so it did not require the same extensive re-engineering as the rest of the client side code. The modular nature of the analytics code meant that it was one of the few parts of the prototype client that could be reused in the rewritten version. New functionality was added however, to make it easier to integrate data collection into other code and to add support for A|B testing.

2.5.1 Improving Extensibility

Part of the rationale for the MIP1 overhaul of the analytics system was to make it more flexible. One way this was done was by switching the server side data storage format from tab delimited values to a JSON based format [37], allowing for changes to be made to the types of data being recorded without the need to keep track of the fields present in each version of the system.

This change was leveraged in MIP2, with the addition of functions to record any data and send it with the base analytics data on page unload. This greatly simplifies the process of capturing new kinds of data, as no modification of the analytics code is required.

```
setSessionVar: function(varName, value){...},
incrementSessionCounter: function(counterName){...},
pushToSessionArray: function(arrayName, value){...},
```

Listing 3: Functions to record arbitrary analytics data

This mechanism was of use when instrumenting the subset tool (see 2.9.2) and could be used to increase the granularity of instrumentation in other areas of the Workbench.

This serves as an illustrative example of how even relatively simple design changes led to notable improvements in the maintainability of the Workbench.

2.5.2 A|B Testing

This improved analytics mechanism was also used to implement an A|B test. As discussed in 1.2.2.2, there is a lack of strong evidence to support the use of gamification features in educational applications. Therefore, it was desirable to evaluate any gamification features before making them available to all users. A|B testing provides a good way to evaluate the impact of small changes, without the distortions that research conducted in person can introduce.

The trial that was conducted sought to determine whether adding a progress bar to the interactive question set would lead to users completing more questions. Visitors to the Workbench were randomly divided into two groups, with one seeing the progress bar and the other acting as a control. Only users who visited at least one page in the question set were included. It is likely that some double counting of users occurred, e.g. users who used the question set from both a personal computer and a School computer would be counted twice.

No statistically significant effect was found however, with the control group answering slightly more questions than those shown the progress bar. If an effect exists, it is too small to be detected using the 138 users who met the inclusion criteria. While further trials of this modification may lead to a more conclusive answer, it is likely that efforts would be better spent investigating more promising modifications.

While this trial did not lead to a statistically significant outcome, the implementation of the procedure was sound and could be used as the basis for further investigations.

2.5.3 Ethical & Legal Concerns

Collecting analytics data brings both legal and ethical issues.

As described in the MIP1 report [1], the Data Protection Act 1998 imposes restrictions on the collection and processing of personal data. As of the end of MIP1, the only data items collected by the Workbench that could potentially be considered personal data for the purposes of the Act were user IP addresses. For the purposes of the Act, the determination of whether data is personal data (i.e. can be used to identify an individual) depends not just on the data itself but on any other information "which is in the possession of, or is likely to come into the possession of, the data controller" [38]. An IP address and access time can – under certain circumstances – be used to identify an individual. However this identification would require information (such as ISP records) that are not likely to be made available to the Workbench's data controller. Based on this, the Act does not apply to the data collected by the Workbench.

However, in line with guidance from the Information Commissioner's Office (ICO) [39], the analytics system was altered in MIP2 to discard the final octet of user IP

addresses. This does not significantly reduce the utility of the data collected while providing greater certainty that personal data is not being collected.

The analytics system makes use of browser local storage to track some aspects of usage, including issuing each user with a unique, randomly generated identifier for analytics purposes. The use of local storage is governed by the Privacy and Electronic Communications (EC Directive) Regulations 2003 (PECR) as amended by the Privacy and Electronic Communications (EC Directive) (Amendment) Regulations 2011. PECR requires that users give specific and informed (although not necessarily explicit) consent to data being set on their device [40]. To comply with this requirement a notice was added to homepage of the Workbench informing users that data was being stored on their device. Further, a prominent link was included in the header of all Workbench pages to a detailed privacy policy. This policy describes the data collection process, covering local storage usage as well as data processed on the server side. This policy is short (~550 words) and designed to be easily understood by the Workbench's target audience. In addition, a facility was added to the policy page to allow users to view and/or clear all data stored in local storage or as cookies from the Homepages domain.

The privacy policy page registered ~50 visitors but had a median visit length of only 4 seconds, so few users are getting much information from the policy. It may be the case that a more invasive method may be needed to ensure compliance with PECR, such as a banner which partially obscures the screen until its contents are acknowledged by the user.

New legislation, in the form of the General Data Protection Regulation (GDPR) [41], will come into force in 2018 and may require changes to the way that consent is gained to store data on users' devices. Final guidance on the impact of the GDPR has not yet been issued by ICO and there is uncertainty over how the relevant sections will be interpreted [42], with some readings (including the ICO draft guidance) requiring that users will need to give explicit, opt-in consent [43]. This reading would necessitate changes to the Workbench to ensure continued compliance with the law.

2.6 Automated Testing

2.6.1 Automated Testing in MIP1

In the first year of this project, two attempts were made to produce an automated test suite for the Workbench. Due to the architectural issues described in 2.3.3, it was not possible to isolate the FSM logic for testing, so all testing had to be done through browser automation. These test suites were created using Python and Selenium Webdriver, both with manually written code and with code generated using Selenium IDE.

These tests were time consuming to create, slow to execute, prone to timing-related failure, and brittle – being broken frequently by minor changes. Ultimately, both attempts were abandoned as it was judged that the tests did not provide enough value to justify the time needed to maintain them.

2.6.2 Unit Testing in MIP2

With the architectural issues that prevented unit testing in MIP1 addressed, it was possible to create a suite of unit tests for the rewritten system. These tests, written in JavaScript using the Chai assertion library with the Mocha test framework, covered the finite state machine logic, including the logic used to mark question answers. The test suite is not comprehensive (an attempt was made to measure coverage using Nyc but the results have not been reliable) but its coverage of the core functionality was useful as it provided assurance that changes did not have unintended effects.

The ability to isolate the FSM logic led to a much more stable test suite, as the requirements and implementation of this functionality changed far less frequently than the code handling user interaction.

2.6.3 Integration Testing in MIP2

A suite of integration tests was also produced for the rewritten Workbench, again using Selenium Webdriver for browser automation, but this time using the JavaScript API and the same assertion library and test framework as the unit tests.

JavaScript was used this time, as opposed to Python in MIP1, as it simplified development to use a single testing framework for both unit and integration testing. Further, one of the reasons for using Python previously was the author's greater familiarity with the language, but the experience gained through extensive use of JavaScript in MIP1 negated that reason. Finally, using JavaScript had the advantage of simplifying the deployment process (see 2.7.1) as JavaScript dependencies can be easily managed by Node Package Manager, which was already in use.

This suite of integration tests was somewhat more successful than the suites created during MIP1, primarily as it did not suffer from the same timing issues that caused intermittent failure in those suites. It is not completely clear why this should be the case as the Python Webdriver API is mostly synchronous [44] which should have made Python-based code less prone to these errors but the relatively low quality of the Webdriver documentation makes it difficult to determine the issue. Regardless, the asynchronous JavaScript API, with actions coordinated using Promises, provided more consistent test results.

Another improvement was the introduction of cross-browser testing, with tests automatically run using both Chrome and Firefox. This identified some graphical issues in the tool resulting from different interpretations of the SVG specification by the authors of those browsers. One example of this is that Chrome places midpoint markers in the middle of arc sections while Firefox does not. Another is that Chrome allows SVG definitions to be referenced from CSS files, while Firefox does not [45].

The process of creating tests also helped to uncover issues, for example it uncovered errors that had been made in the process of porting exercises to the rewritten version of the tool.

However, the integration test suite suffered from some of the same issues that led to the abandonment of integration testing in MIP1 – tests were still slow to run, time consuming to write, and prone to breaking unexpectedly. While the more stable design of the rewritten application made unexpected breakage less common, there were still issues caused by browser updates altering the behaviour of Selenium Webdriver. For example, one such change completely broke all of the Chrome tests. Web browsers change rapidly, so it may be inevitable that tests targeting the latest browsers require substantial maintainance to remain useful.

2.7 Deployment Considerations

As part of meeting the goal of delivering a system useful beyond the scope of this project, it was important to consider the process of deploying the system. Even for a relatively small application such as the Workbench, if care is not taken, building and deploying the system can be very difficult for anyone other than the original developer. This is particularly true for systems with a single developer, as there is no check that the process is logical and no pressure to write or update documentation.

The need for development to better support deployment and other operational concerns has become more widely recognised of late, in particular as part of the DevOps movement [46].

2.7.1 Dependency Management

One barrier to smooth deployment is a complex set of dependencies requiring manual set-up. This can also make it more difficult for other developers to contribute to a project.

Building the Workbench from source requires a range of software packages including a templating engine, a transpiler, a minifier, and a vector mathematics JavaScript library. To simplify deployment, almost all dependencies are managed using the Node Package Manager (NPM). This provides a way for the necessary packages to be downloaded and installed using a single command. The other major dependency, the Python templating engine Jinja2, is available through the pip package manager.

NPM was used as the primary dependency manager, rather than other JavaScript dependency managers such as Yarn or Bower, as NPM's position as the default package manager for Node.js has resulted in strong support from package developers. The unrestricted nature of the NPM system for publishing packages has led to widespread problems in the past [47], but it is still generally reliable.

2.7.2 Other Work

In addition to the use of dependency management software, other steps were taken to simplify the deployment process.

As part of the MIP2 re-engineering work, the minification process was modified to remove the need for the UglifyJS and UglifyCSS packages to be globally installed. This avoids the need for root permissions when installing the dependencies and avoids possible dependency conflicts with other software.

A script to build and deploy the Workbench with a single command was created. While this is not a general solution, as it would need to be modified to be used with another DICE account or for a deployment destination other than the Homepages server, it would provide a useful starting point for a new deployment.

Assumptions based on the structure of the Homepages server were hard-coded into the MIP1 build process, for example the relative location of the CGI directory. These assumptions were identified and removed, to simplify deployment to new locations.

Finally, the build process was tested by downloading and building the project from a clean live image. This ensured not only that the project's dependencies were correctly specified but also allowed issues with the process's documentation to be identified and fixed.

2.8 Tutorials

Two tutorials for Inf1:CL students were co-written with Prof. Fourman and Dagmara Niklasiewicz, covering finite state machines and regular expressions. Students were encouraged to use the FSM Workbench while working on the exercises but it was not required.

Rather than simply directing students to try out the exercises using the Create tool, a set of pages was created for each tutorial. These sets can be found at `homepages.inf.ed.ac.uk/s1020995/fsmworkbench/tut5/q1.html` and `http://homepages.inf.ed.ac.uk/s1020995/fsmworkbench/tut6/q1.html`

Data from the analytics system suggests a reasonable uptake of the system with 152 and 163 unique users viewing at least one question of the first and second tutorials in the Workbench and 29 and 31 visitors viewing all of the questions in the Workbench. Across both tutorials, every question received at least 60 unique visitors.

2.8.1 Design Considerations

A question set had already been implemented as part of the FSM Workbench, but that was not used when creating the tutorials. This was primarily because a good set of questions for independent study is not necessarily going to provoke the sort of discussion that gives tutorials their value.

Writing tutorial questions necessitated a different approach to feedback than when creating the original question set. In the original question set, the goal was to offer immediate feedback to the student. It was judged that providing immediate feedback could remove an important aspect of students' motivation for engaging with tutorials – the desire to verify their understanding of the material. For this reason, none of the tutorial questions made use of the previously developed feedback systems. Students were still able to use the simulation functionality of the Workbench to test their understanding, but the Workbench did not attempt to assess the correctness of their answers. As discussed in 1.2.2.1, delayed feedback may be more effective than immediate feedback, despite a preference amongst students for immediate feedback.

Removing the constraint of being able to provide immediate feedback made it easier to pose the more open-ended and conceptual questions that tutorials are well suited to. This extra freedom was used to prompt students to think about finite state machines beyond their basic mechanics in the hope of cultivating a deeper understanding of the topic – for example by prompting them to think about what a particular state of a machine means, or to think about the limits of the languages that FSMs can accept.

2.8.2 Insight Gained

While it is hoped that the production of the tutorial sheets was a useful exercise in its own right, the process also led to some useful insights into ways the Workbench could better support users in the instructor role and into the limitations of the interactive question set.

Firstly, instructors have different priorities when using the Workbench. Through use of the tool in that role it became clear that some missing features, that had been judged to be low-priority when considering the requirements of students, would be of great use to instructors. One such feature was the ability to locally save and load machines, which would have been of particular use when making revisions to tutorial questions. Another feature – the ability to export machines in SVG format – was of such clear utility in this role that it was implemented before commencing question writing.

Secondly, the process highlighted some of the limitations of the existing question set, both in terms of its content and in terms of student engagement.

Writing questions without the constraints of the interactive question set – that questions be in a form that the system can mark – made clear the extent to which these limitations were preventing the interactive question set from addressing important aspects of the topic. For example, when attempting to understand a FSM it is useful to intuit what it means for the machine to be in a particular state (e.g. "the input seen so far is of even length"). However, posing questions that encourage students to consider this issue is difficult when trying to provide good automated feedback. These constraints also prevented the posing of questions exploring the limits on the languages that FSMs can recognise, on the exact relation between NFAs and DFAs, and on the differences between the graphical representation of machines and the formalism behind them.

It is possible that the question set could be refined to address some of these concerns

within the current constraints. However, a more direct solution may be to accept the limitations of the system with regard to more conceptual questions and deal with those in a different way that complements the existing system. Tutorials provide a good mechanism for this, but it may be desirable to have a self-contained solution as well. The simplest answer, used to sidestep the limitations of machines since the earliest automated teaching tools [48], [49], would be to provide multiple choice questions to allow students to verify their understanding of the topic. This approach was not considered during this project due to the desire to present work that is novel and of technical interest. However, it may be the most appropriate way to cover those aspects of the topic that are unsuited to the existing question framework. Multiple choice questions are already used in at least one course within the School of Informatics to give students a quick way of testing their understanding [50].

The number of students engaging with the tutorial questions was large compared to the number of students persisting with the question set beyond the initial questions. It is not clear however how much of this effect is due to differences in the content of these sets of questions and how much is due to tutorials being mandatory. It does mean that tutorials can be used to make a large proportion of students engage with the Workbench, so this may be a good area to focus further work on to maximise its effect.

2.9 Subset Procedure

As part of Inf1:CL students learn how to use the subset procedure to convert NFAs into DFAs, but the work done in MIP1 did nothing to support this topic. Over several iterations, different approaches to the problem were developed.

2.9.1 Initial Attempt – the Subset Question Type

The first attempt at teaching the subset procedure took the form of a question type in the exercise set (see Fig.6 or `homepages.inf.ed.ac.uk/s1020995/fsmworkbench/inf1/dfa-convert-1.html`). Students are presented with two panes – one containing the NFA they are converting and the other the DFA they are constructing. Starting from the set of initially reachable states, they are prompted to identify additional sets of reachable states for a particular input symbol by selecting them on the NFA. When they do, these sets (and the appropriate transitions) are added to the DFA.

This approach has the advantage that the connection between the states in the original machine and the states in the DFA is made clear.

However, there are reasons to suspect that it may be of limited value in teaching students how to perform the conversion on their own. The system does not include anything analogous to the table that would be drawn when performing the operation by hand, leaving students to learn a good pen and paper approach from other sources. The system does all of the work of keeping track of which combinations of sets of states and input symbols have already been examined and does not communicate how this

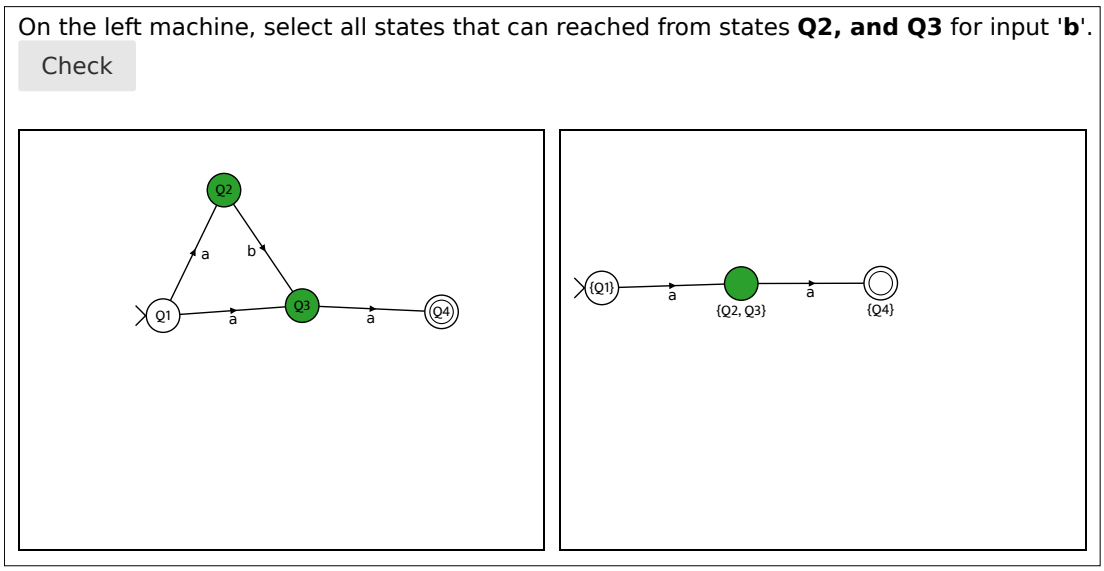

Figure 6: Subset procedure as implemented in the question set

is done. This stops students from having to consider an important aspect of the process. Further, because there is no visible table or other visualisation of this process, the prompts can seem arbitrary which may lead to students simply following the instructions without quite understanding why. States and transitions are also added to the DFA automatically, again removing the need for students to consider an important aspect of the process.

These issues stem in part from the difficulty of presenting the user with full control of the process and a full view of the system state while keeping the interface simple enough to be easily understood. Screen space can also become a significant limitation when attempting to display two machines as well as a visualisation of the state of the process.

While this approach may help students to understand the specific point of the relation between sets of states in the NFA and states in the DFA, it would need significant refinement to be of more general help to students seeking to understand the whole conversion process.

2.9.2 Second Attempt – Subset Procedure as Create Option

A somewhat different approach was used in the subset procedure tool that was implemented in the Create mode (see Fig.7). This was designed to more closely mimic the approach that a student would use when performing the process on paper. The partially constructed DFA is not shown graphically, with the user instead working to construct its transition table. Users are also given more control over the process – they are given the responsibility for choosing the order in which to fill table cells and add rows, as well as identifying new reachable sets of states as before.

This version of the procedure was added to the Create mode rather than replacing the

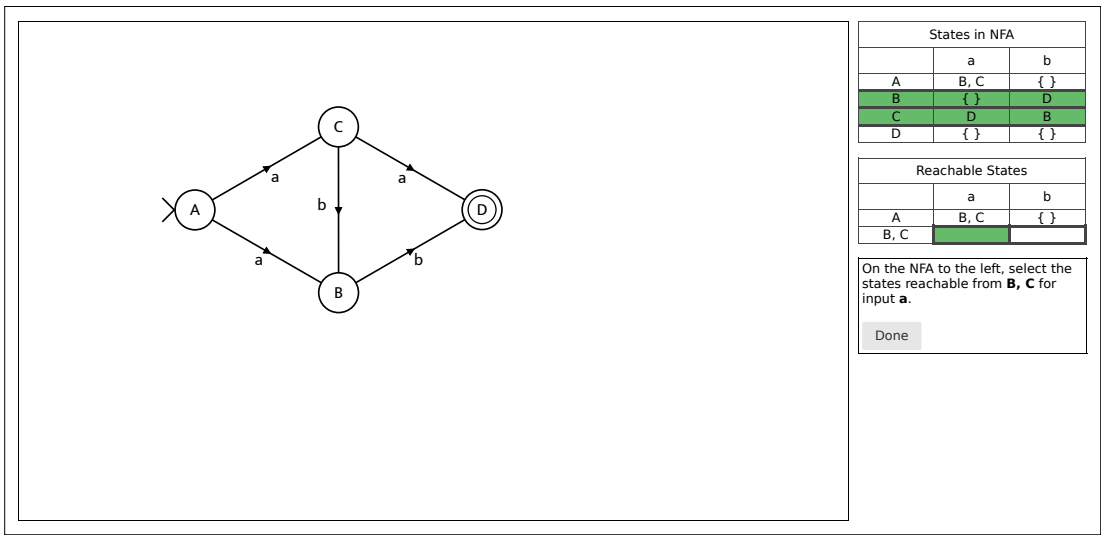

Figure 7: Subset procedure as implemented in Create mode

previous question type as relatively few students were making it far enough into the question set to make use of it.

Including it in the Create mode meant students had the freedom to use the subset tool on arbitrary machines, which led to some additional design challenges. While catching input that would not lead to meaningful results, such as machines without initial states, was straightforward, state names were a more interesting problem. For the table of states to make sense to the user, each state in the machine must have a name that can be referred to by the table. Rather than simply rejecting machines with unnamed states, names are issued automatically with an attempt made to follow any naming convention the user has partially applied.

While the first subset tool was designed to emphasise the connection between the original machine and the resulting DFA, in this version the goal is to make the relation between the original machine and the table of reachable states clear. To encourage users to make this connection, their attention is directed back and forth between the two elements. In particular, it was important to prevent the user from completing the task using only the table as this could lead to them developing only an understanding of the mechanics of the process without understanding why it works. The most significant way in which this attention switching is achieved is by making the user interact with the machine to identify sets of states and with the table to advance the process by adding rows and selecting new cells to fill. The connection is further strengthened by highlighting states in the NFA when the user mouses over the corresponding row in the table.

As with the previous implementation, a prompt is used to tell the user what they are supposed to be doing. This is significantly more important with this design as the user has more than one kind of task they must perform. Context-sensitive hints are given as to what the user's next action should be, rather than relying on a static set of instructions. The prompt is also used to describe the action that the user has just made to provide them with additional insight into the meaning of the process they are

following.

While this implementation is an improvement in many areas – it gives the user more control, a clearer idea of why they are doing what they are doing, and it maps more closely to the pen and paper technique students need to learn – there are aspects that would benefit from further work.

One refinement would be to give the user control over constructing the DFA from the completed table, this is currently done automatically but it is an important part of the process that students must be comfortable with. If the user were required to give the states in the machine the suggested names (as is required in the satisfy-definition question type, described in more detail in the MIP1 report) it would be possible to map their work onto the completed table in a way that would allow meaningful feedback to be given if they make an error.

Other refinements might require more user testing to identify. One area of concern is the amount of highlighting and animation that is used – it is not clear if the correct balance has been struck between the desire to reinforce the connection of the NFA to the table and the need to avoid presenting too much information to the user at once.

Instrumenting for Analytics

The second version of the subset tool was instrumented in a particularly granular way. As this was a completely new interface with a high degree of complexity, there was a good chance that it would have some usability problems. By collecting granular data, it was hoped that problem areas could be identified. For example, the system records whether a user has successfully filled in a cell in the table and whether they have successfully added a row. If the number of users adding rows was found to be significantly lower than those adding cells, this would have suggested that the row-filling operation was unclear. As noted in 2.5, the functions to record and transmit arbitrary data greatly simplified the process of performing this instrumentation.

Unfortunately, there was less than a week between the revised subset tool being ready to test with students and the Inf1:CL exam so only a small amount of data was collected. 10 users opened the subset tool and 6 of those completed the conversion process. These numbers are too low to draw any meaningful conclusion from. However, as the instrumentation work is already done, obtaining more useful results would simply be a matter of getting the tool in front of more students.

Chapter 3

Evaluation

3.1 User Testing

3.1.1 Lessons from MIP1

User testing was of particular importance in MIP1, being key to that phase's focus on iterative design. To this end, several methods of assessing users' responses to the system were attempted.

Observation of volunteers from outwith the target demographic interacting with the system was used initially to identify significant usability failings. Beyond these basic usability concerns however, their usage of the system is not necessarily predictive of how Inf1:CL students will use it.

Once the system had been refined, another round of observations was attempted using Inf1:CL volunteers. Too few volunteers were found to reach any meaningful conclusions, so other, less direct, methods were attempted.

The Workbench was promoted to Inf1:CL students, including through lectures and tutorials, and many students made use of it. Several ways of leveraging this usage to obtain actionable feedback were attempted, with mixed results.

A survey was produced to assess the usability and utility of the system and students were encouraged to complete it. However, the response rate was low and it was difficult to find meaning in the results.

A facility was added to the question set for users to rate each question, in an attempt to identify problem areas in need of additional work. The response rate was far too low for the results to be meaningful. While it is likely that the response rate could have been improved through the use of more obtrusive prompts, this would have harmed the usability of the system.

Data from the analytics system was more useful, although a degree of speculation was required to come to actionable conclusions. The analytics data did show that students

were indeed making use of the system, which matched the broadly positive informal feedback gathered from students and tutors.

Given these difficulties and given that most of the interface design had already been tested in MIP1, a more focused approach to gathering user feedback was used in MIP2, informed by the success and failures of user testing in MIP1.

3.1.2 Gathering Feedback in MIP2

There were two goals in gathering feedback from users in MIP2. First, was to confirm that the findings from MIP1 – that Inf1:CL students were generally comfortable using the system – still held. The second was to better understand how students use the system, to identify how best to support that usage.

3.1.2.1 Confirming Usability

The first goal was met primarily using analytics data, backed by informal feedback relayed through the course tutors.

The analytics data showed a good uptake of the tool, with >100 unique visitors to the question set and to both tutorial sets as well as >150 unique visitors to the Create tool. It should be noted that these figures are likely to be an overestimate to some extent, as users will be counted more than once if they access the Workbench from more than one browser. This may be a common occurrence, for example students may access the Workbench from their personal machine as well as from a School or University device.

The data also provided some evidence that students were spending time with the tool, although this data had some issues which compromised its accuracy. Median total times of >5 minutes on the Create tool and >10 minutes on each of 6 tutorial questions were recorded . Across all users, the median time spent with the Workbench was ~13 minutes. The analytics system measures the time spent on a page simply as the time between the page being loaded and being unloaded – it does not distinguish between time where the Workbench is in use and time where it is idle. This issue means the figures given may not accurately represent actual usage of the tool. A median measure was used, as a mean could be significantly distorted by a small number of users leaving the tool open on their machine for a long period of time.

The informal feedback provided additional evidence that students were using the tool, and did not identify any problems

3.1.2.2 Understanding Usage

The second goal, of better understanding usage of the system to guide development, also relied heavily on analytics data. In MIP1 data-gathering was focused on the smaller details of the way that users interacted with the system, as the details of the interface were being tested and refined. While further refinements of this type were still

sought in MIP2, more focus was placed on understanding usage at a more macro level. For example, in MIP1 the usage statistics for individual questions were examined to identify issues with particular questions, whereas in MIP2 the usage statistics for the question set as a whole were considered to compare the set's effectiveness against other aspects of the Workbench.

Analysis of analytics data is a technique particularly well suited to understanding user behaviour at this wider level. More invasive techniques, such as a think-aloud protocol or direct observation of usage, can alter the behaviour of users enough to make it impossible to determine how they would prefer to interact with the system outwith the experimental setting. This is particularly an issue when considering questions as subjective as whether a user prefers to study in a more structured way through the problem set or a less structured way through experimentation with the Create tool. More invasive techniques may have helped to understand the reasons behind user preferences, but useful results would have required a participant recruitment process substantially improved from MIP1.

One of the key insights this analysis provided was that the interactive question set was only being used by a small number of students compared to the tutorial questions and the Create tool. The first question received >110 unique visitors, but no question after the sixth received more than 30 unique visitors – students were willing to try the question set but few persisted far beyond the initial questions. There was a small group of users who made more extensive use of the question set, with 13 correctly answering 20 or more of the 25 questions. This figure could be an underestimate – users completing the questions from multiple browsers may be excluded – but not by a large degree, as 8 questions had fewer than 20 correct answers.

This information was useful in guiding development – it made it clear that further extensions to the question set would only be seen by a small number of users, whereas additions to the Create tool or improvements to the tutorials would reach more students.

3.2 Comparison to Existing Work

Another way of evaluating the success of the project is to compare the Workbench with other similar applications. As discussed in 1.2.1, JFLAP and JFLAP Web are the most similar in purpose to the FSM Workbench, so comparison will be limited to these applications.

3.2.1 JFLAP

This discussion is based primarily on the JFLAP 8.0 beta release, although JFLAP 7.0 was also examined during the research phase of this project.

The breadth of topics covered by JFLAP far exceeds the range covered by the Workbench, including pushdown automata, Turing machines, and context-free grammars.

This is not necessarily an advantage for Inf1:CL students however – it may simply provide opportunity for confusion.

This mirrors the author's main criticism of JFLAP – the application is incredibly rich in functionality but this complexity has made the system difficult for novice users to grasp, even if they are familiar with FSMs. JFLAP is presumably very powerful in the hands of an experienced user, but a student coming to JFLAP with little prior experience with FSMs may find the way it presents this breadth of features confusing. An illustrative example of the way JFLAP's design favours experienced users over novices is the way it handles transition creation. When a transition is created, it defaults to being an ε-transition. This may be a good default for a power user looking to quickly construct a machine, but it is likely to confuse a novice still learning the basic mechanics of deterministic machines.

JFLAP has support for different notation conventions – users can specify, among other options, how the empty string and the empty set are displayed. While this allows a student to adjust JFLAP's notation to match that used by Inf1:CL to an extent, it is not clear that the students who are most likely to be confused by notational differences will know to make use of this option. Even using this functionality, JFLAP does not support FSMs with multiple initial states as presented in Inf1:CL.

JFLAP has an on-demand layout system to rearrange machines, giving users the choice of 6 layout algorithms. While this sometimes produces useful output, it may encourage students to play with the layout of their machine rather than interacting meaningfully with the system. JFLAP does support repositioning of links through dragging, providing a solution to the issue with long links discussed in 4.1 in the absence of a real-time layout system.

JFLAP provides four ways of simulating the execution of a FSM. In the graphical modes, the current state(s) of the machine are highlighted but there is no transition highlighting or use of animation to make the process visually clearer.

The source code for JFLAP 8.0 has not been published yet, so the code for JFLAP 7.0 was examined in an attempt to assess how difficult it would be to extend or maintain. No obvious specific issues were identified, but only a small sample of the 473 Java source files were considered. Because of the size of JFLAP's codebase, any non-trivial modification to the system is likely to require a lot of work.

The FSM Workbench, with its far narrower scope, is designed around the needs of Inf1:CL students. Its interface is kept simple, favouring maximising learnability for novices (both to the system and to FSMs in general) over maximising the efficient operation of users experienced with the system. As the Workbench is tailored to the way that Inf1:CL is presented, students are not required to adjust settings to get the application to better match what they have seen in lectures.

The Workbench system for visualising the execution of a machine uses highlighting and animation to make the process visually clear, rather than relying on users having enough of an understanding of the topic to work out the details for themselves. Due to the permissive nature of the Workbench's licence, anyone teaching a course with different notation is free to host their own modified version of the application matching

that notation. Finally, the Workbench's real-time force-directed layout system provides simple, unobtrusive layout assistance, rather than relying on the user to choose an appropriate layout algorithm.

3.2.2 JFLAP Web

JFLAP Web, unsurprisingly, has many similarities to the full version of JFLAP. It has advantages over JFLAP as a learning resource for Inf1:CL students that go beyond the advantages of being web-based.

Not having had the decades of work that have gone into JFLAP, JFLAP Web is far simpler. As a result, it does not suffer from the overwhelming array of functionality that makes JFLAP difficult for a novice user to use. Despite this, it still provides enough functionality to support Inf1:CL, including the provision of a procedure to convert machines to deterministic form.

JFLAP Web is not without issues however. In general, many of these issues can be characterised as lack of polish – bugs are frequently encountered, some operations require entering text into a default browser prompt without guidance on the expected format, and a naive approach to element placement is used which can lead to diagrams that are difficult to read.

JFLAP Web's visualisation of machine execution is easier to use than JFLAP's, and not just because only one way of performing the operation is offered. Some minor use is made of animation, with the active states fading out and in as input is consumed. There is no motion beyond this however, and no transitions are highlighted.

The source code for JFLAP Web is available as part of the OpenDSA repository [51], under the permissive MIT licence. A cursory inspection suggests that the code is of reasonable quality. The codebase is relatively small, which would make modifying or extending it significantly easier than modifying JFLAP.

Although JFLAP Web is part of the OpenDSA courseware project, it does not appear that any publicly available exercises or examples using the system have been produced.

From the perspective of an Inf1:CL student JFLAP Web is essentially comparable to the MIP1 version of the Workbench's Create tool, with the addition of a DFA conversion procedure. The system is adequate for creating and simulating finite state machines, but it lacks the graphical polish and novice-centric design of the Workbench.

3.2.3 Commonalities

Neither version of JFLAP has an integrated question set or other way of providing students with a more structured learning experience. Supplementary material exists for the Java version of JFLAP [6], [52], but this is entirely external to the application.

Integrating a question set into the Workbench not only puts it in easier reach of students, it allows for analytics data to be collected which can guide improvements to the

question set. Neither version of JFLAP has an analytics system, limiting the ability of their developers to adapt the systems to the needs of their users.

Finally, the Workbench has the advantage of being specifically designed for Inf1:CL. Both JFLAP applications offer some options to customise notation, but neither support the multiple initial states used in Inf1:CL. A customised solution, rather than a one-size-fits-all application with customisation left to the user has a clear advantage in terms of ease of use to a novice to the topic.

3.3 Project Objectives

The success of this project can also be judged by comparing what has been achieved against the goals described in 1.1

Efficacy A rigorous assessment of the impact of access to the Workbench on learning outcomes was not possible within the constraints of the project. However, less formal evidence was gathered to support continued use of the tool (see 3.1). The Workbench has received a positive response and the recorded usage data shows that students are choosing to make use of the system. This is compatible with the system being an effective learning tool, but not sufficient to conclude that.

Usability As discussed in 3.2, the usability of the Workbench compares favourably to other similar applications. In particular, the Workbench's focus on accommodating users new both to the topic and to the system makes it well suited to supporting an introductory course such as Inf1:CL. The design of the Workbench was shaped by repeated user testing in MIP1, which allowed a number of usability issues to be identified and addressed. MIP2 built on this, making use of tested designs and continuing to apply the lessons learned in MIP1. Additional evidence was gathered in MIP2 to ensure that the Workbench remained highly usable (see 3.1.2.1).

Compatibility Automated testing was used to check compatibility with Chrome and Firefox (see 2.6.3), this was supplemented by manual testing with other browsers. Various compatibility issues were identified and fixed, so that the Workbench works on the most popular modern browsers. The system's interface works with touch input, in addition to keyboard and mouse. Optimisation was done (see 2.4.1), ensuring that the Workbench works on less powerful devices.

Accessibility The Workbench fails to meet the accessibility goals set out in the introduction. There are times when users are required to distinguish between several colours (e.g. in questions where the user must select which state a machine will move to), with nothing to mitigate the difficulties this may cause for colour-blind users. No testing has been performed using screen reading software, so problems are likely. Further

work is needed for the Workbench to work well with browser magnification settings – at present, only content around the editor viewport is scaled, with content within unaffected.

Maintainability A full evaluation of the maintainability of the Workbench would require an independent code audit, but this was not possible within the constraints of the project. An outside perspective is important, as the author of a program is likely to be a poor judge of how easily interpretable it is. With that said, steps such as the simplification and documentation of the deployment process (see 2.7) led to clear maintainability improvements. Further evidence for the maintainability of the Workbench is the success in improving the extensibility of the analytics system (see 2.5.1). While it is difficult to assess the maintainability of the system in absolute terms, there is no doubt that maintainability has been significantly improved from the MIP1 prototype.

Security The security status of the Workbench is largely unchanged from MIP1, with the Workbench being a low-priority target with a limited attack surface. Ceasing to store full user IP address (see 2.5.3) has further reduced the potential impact of a security breach. No further vulnerabilities in the Workbench have been identified since the XSS issue in MIP1.

Overall, the project has met most of its goals. Further work is needed to improve the accessibility of the Workbench, but this will be aided by the system's improved maintainability, which should facilitate the necessary changes.

Chapter 4

Further Work

There are many ways in which the Workbench could be extended or further refined.

4.1 The Long Link Problem

One problem with the way that the Workbench displays machines is that it provides no mechanism for drawing the kind of long, curved links that are often used when linking one side of a machine to the other (see Fig.8). The main difficulty in doing this is in minimising the increase in complexity such a system would add to the user interface. Further, as this tool is intended primarily for educational use, great care must be taken to ensure that users do not confuse graphical manipulations with meaningful changes in the system being displayed. For example, an obvious solution if the system were primarily a diagram editor for experienced users would be to add a second class of transition with different display properties. However, this could easily be misinterpreted by a user unfamiliar with FSMs as being a transition with some distinct meaning, undermining the educational goal of the system.

One possible solution would be to allow the user to bend transitions by clicking and dragging them (as is used in Evan Wallace's Finite State Machine Designer [11]). This could be adapted to work with the force layout system by making motion parallel to the transition adjust its target distance while making motion perpendicular to the transition adjust its curvature. This would give the user significantly more control over the appearance of their machines, but it would require more explanation and more of the user's attention. It may be possible to design a suitable affordance to help users to discover the functionality.

An alternative approach would be to try to integrate this behaviour into the force layout system more directly, so that the system is able to create output like this on its own. One system would be to allow the links between states to be permanently stretched or compressed, rather than modelling them as ideal springs. Links stretched beyond some threshold could then be made to bend to avoid overlapping by placing an invisible node at their centre that is repelled by states, making use of the same physics system that

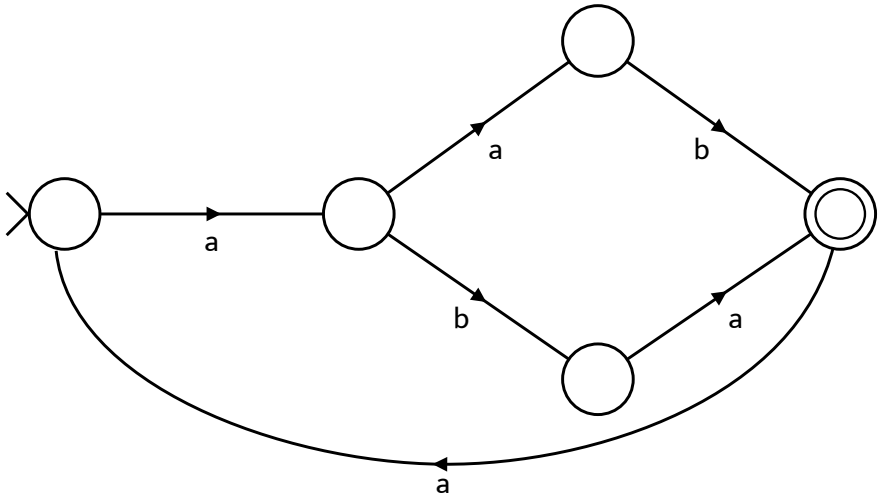

Figure 8: Long links as shown here would be a useful addition to the system

causes states to repel each other. While it is less clear that this method would lead to the desired behaviour, a successful implementation would reduce the amount of work that users need to do to lay out their machines. Increasing the burden on the physics system could also have an adverse impact on performance, necessitating further optimisation work.

Any attempted solution to this problem would need to be validated with user testing to ensure that it does not confuse users.

4.2 Improving Learnability

It is important that new users are able to quickly learn how to use the system. Ideally, the system's interface should be sufficiently simple and sufficiently intuitive to be self-explanatory. This is not always achievable however, and is made more difficult if the user's experience with FSMs is too limited to guide their understanding of the system. There are some additions that could be made to the system to help new users familiarise themselves with it.

Firstly, the existing help page could be expanded significantly. At present, it consists only of a very brief summary of the function of each editing tool. It was not considered worthwhile to expand it further as very few page views were recorded to it during MIP1. However, for reasons which are not clear, many more users visited the page in this phase of the project (~50 by January 2017), so the demand for user-facing documentation may be greater than previously thought. At a minimum, the help page should be extended to cover the full functionality of the system, making use of diagrams and animations to show how to interact with the system. It may be beneficial to expand the scope of the help documentation even further to cover finite state machines themselves, rather than just the mechanics of the Workbench. This could be done in the style of an interactive textbook, with the existing system used to provide example machines for students to interact with.

A short introduction to the editing tools, shown the first time a user accesses them, may also help users to familiarise themselves with the Workbench. This could be as simple as displaying the brief summary of the editing tools' functionality that is used on the current help page. It could also be more involved, perhaps showing the user each tool being used using a short animation. In designing this introduction, it would be important not to bore users and not to intimidate or confuse them by presenting too much information at once.

4.3 Making Best Use of Screen Space

While the core editor interface of the Workbench is implemented using SVG, large parts of its functionality rely on a HTML interface separated from the editor viewport. Developing with this mixture of HTML and SVG elements worked well within the constraints of the project, with the HTML content being easy to create and alter quickly. However, this approach does not necessarily make the best use of the available screen space. A more sophisticated approach may be to expand the editor viewport to fill the browser window, and then draw the rest of the interface on top as SVG elements. The user could then be given control of hiding these elements, so that they could have the full screen to work with. This could also make the tool more useful for giving demonstrations in lectures.

4.4 Touch-Centric Interface

While the Workbench is compatible with touch-based devices, it does not make full use of their potential. A system allowing users to directly draw FSMs and have them recognised by the Workbench could provide a more natural and fluid way of interacting with the system. To avoid frustrating users, the system would need to recognise drawings with a high degree of accuracy. This may be more easily achievable for FSMs than for other domains, as only a small number of elements need to be recognised.

4.5 Other Domains

Much of the work done on this project is not specific to finite state machines. Many aspects of the Workbench's interface, both in design and implementation, could be applied directly to other domains involving specialised graph editing. Some areas that could be suitable and which are used in the School of Informatics are Markov models, Petri nets, UML activity diagrams, and queuing networks.

For instance, editing Petri nets (used to model distributed or parallelised systems) is in many ways a similar task to editing a finite state machine – they are simply another kind of directed graph. As with FSMs, the task can become tedious and time-consuming if the user is required to arrange each element by hand. And as with finite state machines,

the existing software (such as PIPE - the Platform Independent Petri net Editor [53], [54] or ORIS [55], [56]) consists of heavyweight Java applications which are more focused on providing powerful analysis functionality than on supporting novices. Given this, the FSM Workbench could be a useful starting point for developing a web-based Petri net editor, designed for use in an education setting, and using a version of the Workbench's force-directed layout to ease the drudgery of creating nets to let students focus on the concepts rather than the diagrams.

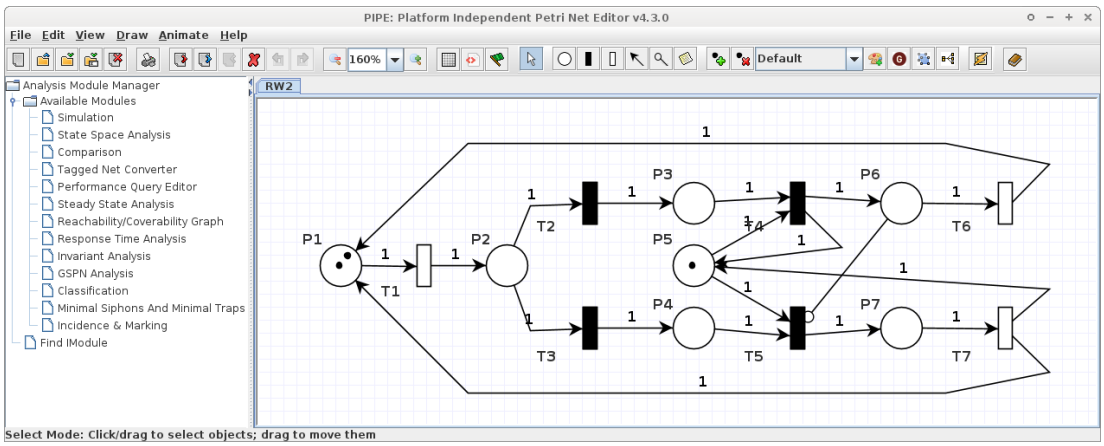

Figure 9: A generalised stochastic Petri net (GSPN) in PIPE 4.3

Chapter 5

Conclusion

5.1 Work Done

Extensive re-engineering work was performed in MIP2 to build a reliable, maintainable application from the prototype developed in MIP1. This resulted in a codebase of high enough quality to support future extensions and maintenance. Improving the codebase also made the Workbench more reliable, leading to a better user experience.

These re-engineered foundations supported further improvements to the system in MIP2. A particular focus was placed on increasing the visual clarity of the tool, to ensure that the Workbench is able to convey key concepts clearly to its users. The system's use of animation was revised, removing extraneous distractions and adding more animation at key points. The way in which graphical elements are placed was improved, with a real-time system to reduce overlapping introduced.

Significant new functionality was added – an interactive system guiding students through the procedure for converting FSMs to deterministic form, with an iterative process used to improve the design.

A number of smaller features were also added, designed to improve the experience of using the tool – undo/redo functionality, local save/load functionality, and the ability to export SVG images.

Two sets of tutorial questions were co-written and added to the Workbench. This drove significant usage of the system and provided a way to integrate the Workbench even more directly into Inf1:CL.

5.2 Insight Gained

Software Engineering

As discussed in 2.2, the work required to create a reliable, high quality application from a prototype is easy to underestimate. That was certainly the case in this project. With the MIP1 prototype already providing all of the Workbench's core functionality, it seemed a simple task to go back and improve the quality of the implementation. However, this proved to be a significantly more substantial undertaking.

Despite the task being far larger than anticipated, completing it was still the best way to meet the goals of this project. It would have been possible to work around the deficiencies of the MIP1 codebase – doing so is likely to have left time to add additional features. For the Workbench to see continued use however, it is more important that the application is reliable and has a high quality, extensible codebase.

Software Design

Beyond the software engineering issues, the most important consideration of this project, spanning MIP1 and MIP2, has been the problem of designing software best able to meet the needs of Inf1:CL students. An iterative design approach, guided by user testing, user feedback, and analytics data was essential in solving this problem. As well as helping to refine individual design elements, this process also identified broader insights into the problem.

One such insight was the importance of maximising the learnability of the system, sometimes even at the expense of adding features. Often when trying to make learnable software it is possible for the designer to leverage the user's existing domain knowledge to shape the user's expectation of how the software will behave. There were limited opportunities to do this when designing the Workbench however, as many users will be learning about finite state machines for the first time. Optimising for learnability can come at the expense of the experience of more advanced users, but in the context of the Workbench that is worthwhile – few Inf1:CL students are likely to spend enough time with the system to become expert users. To achieve this maximisation, the impact on learnability of every potential new feature had to be considered – with features not added if a way of limiting this impact was not found.

A related insight was that the desire to demonstrate originality was sometimes at odds with the goal of delivering high quality, usable software. Adding new, novel features does not necessarily lead to an improved user experience. JFLAP is a prime example of this problem (see 3.2), where an over-abundance of features makes the application difficult to use. The other side of this issue is that features which do have the potential to improve the user experience can be neglected if they are commonly seen elsewhere. In this project, features such as undo/redo and save/load were almost overlooked due to a desire to focus on more novel work. The omission of more standard functionality

is appropriate at the prototype stage, as less user testing is likely to be required, but not when moving beyond prototypes to produce a real application.

5.3 Project Outcome

In summary, the MIP2 phase of this project succeeded in producing a high quality application from the MIP1 prototype. This application was expanded with new functionality, with emphasis placed on ensuring that a high level of usability was maintained. The application was successfully integrated into the delivery of Inf1:CL, seeing widespread usage.

Taken as a whole, this project has identified a need for an application, shaped the design of the application through iteration guided by user testing, refined the implementation of the application to ensure it is of high quality, and successfully deployed the application to its target users.

References

[1] Matthew Hepburn. Tools for Learning: Computation and Logic (Part 1). MIP1 Project, University of Edinburgh, 2016. https://project-archive.inf.ed.ac.uk/ug4/20160738/ug4_proj.pdf.

[2] Matt Chapman. The Finite State Machine Explorer. http://www.mathcs.bethel.edu/~gossett/DiscreteMathWithProof/fsme/fsme.html (accessed 2017/04/04).

[3] Kyril Faenov and Stephan Merz. State Machine Simulator. http://www.pst.ifi.lmu.de/personen/merz/software/fsm/Simulator.html (accessed 2016/03/06).

[4] Susan H. Rodger. JFLAP. http://www.jflap.org/ (accessed 2017/01/10).

[5] JFLAP 7.0 tutorial. http://www.jflap.org/tutorial (accessed 2017/01/10).

[6] Susan H Rodger and Thomas W Finley. *JFLAP: an interactive formal languages and automata package*. Jones & Bartlett Learning, 2006.

[7] Bill Yu. My projects. http://billyu.me/index.php/2016/09/09/my-projects/, 2016.

[8] Bill Yu. Finite Automaton Editor. http://billyu.me/OpenDSA/AV/FLA/billyu/ui/FAEditor.html (accessed 2017/03/29).

[9] Ivan Zuzak and Vedrana Jankovic. FSM Simulator. http://ivanzuzak.info/noam/webapps/fsm_simulator/ (accessed 2017/04/04).

[10] E4Coder. SMCube. http://www.e4coder.com/content/smcube-state-machine-editor (accessed 2017/04/04).

[11] Evan Wallace. Finite State Machine Designer. http://madebyevan.com/fsm/ (accessed 2017/03/30).

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

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

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

[19] 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.

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

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

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

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

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

[25] 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.

[26] Apple Inc. iOS human interface guidelines. `https://developer.apple.com/ios/human-interface-guidelines`.

[27] Gerry Gaffney. Why consistency is critical. `https://www.sitepoint.com/why-consistency-is-critical/`, 2005.

[28] Dorothy Shamonsky. The most important reason for consistency in the design of a UX. http://www.ics.com/blog/most-important-reason-consistency-design-ux, 2013.

[29] Jonathan Grudin. The case against user interface consistency. *Communications of the ACM*, 32(10):1164–1173, 1989.

[30] Michael Zuschlag. Achieving and balancing consistency in user interface design. http://www.uxmatters.com/mt/archives/2010/07/achieving-and-balancing-consistency-in-user-interface-design.php, 2010.

[31] Jakob Nielsen. Enhancing the explanatory power of usability heuristics. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 152–158. ACM, 1994.

[32] Jakob Nielsen. 10 usability heuristics for user interface design. https://www.nngroup.com/articles/ten-usability-heuristics/, 2005.

[33] Mattias Petter Johansson. Always rewrite EVERYTHING! https://youtube.com/watch?v=XcUUY8ziTuk, Febuary 2016.

[34] Curtis Cooley. Hazards and safeguards for software rewrites. https://industriallogic.com/blog/rewrites-hazardous/, April 2014.

[35] Dan Tao. The prototype is your friend (if you care about perf). https://coderwall.com/p/jj6fwa/the-prototype-is-your-friend-if-you-care-about-perf.

[36] Mozilla Developer Network. Classes. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes.

[37] Ian Ward. JSON Lines. http://jsonlines.org/ (accessed 2017/04/05).

[38] Data Protection Act 1998.

[39] 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 2017/03/26).

[40] Information Commissioner's Office. Guidance on the rules on use of cookies and similar technologies. https://ico.org.uk/media/for-organisations/documents/1545/cookies_guidance.pdf (accessed 2017/03/27), May 2012.

[41] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). http://eur-lex.europa.eu/eli/reg/2016/679/oj.

[42] Gabe Maldoff. Top 10 operational impacts of the GDPR: Part 3 – consent. https://iapp.org/news/a/top-10-operational-impacts-of-the-gdpr-part-3-consent/, January 2016.

[43] Information Commissioner's Office. Consultation: GDPR consent guidance. https://ico.org.uk/media/about-the-ico/consultations/2013551/draft-gdpr-consent-guidance-for-consultation-201703.pdf (accessed 2017/03/27), March 2017.

[44] Software Freedom Conservancy. Selenium documentation – waits. https://seleniumhq.github.io/docs/wd.html#waits.

[45] Bug 309612 - SVG marker "not found" when style file used (new builds only, after 2005072109). https://bugzilla.mozilla.org/show_bug.cgi?id=309612.

[46] Len Bass, Ingo Weber, and Liming Zhu. *DevOps: A Software Architect's Perspective*. Addison-Wesley Professional, 2015.

[47] Chris Williams. How one developer just broke Node, Babel and thousands of projects in 11 lines of JavaScript. https://www.theregister.co.uk/2016/03/23/npm_left_pad_chaos/, March 2016.

[48] Ludy T Benjamin. A history of teaching machines. *American Psychologist*, 43(9):703, 1988.

[49] S.L. Pressey. Machine for intelligence tests, March 4 1930. US Patent 1,749,226.

[50] Perdita Stevens. Software design and modelling. http://www.inf.ed.ac.uk/teaching/courses/sdm/, 2016.

[51] OpenDSA/fa – GitHub. https://github.com/OpenDSA/OpenDSA/tree/master/AV/FLA/billyu/fa (accessed 2017/03/29).

[52] JFLAP modules and exercises. http://www.jflap.org/modules/ (accessed 2017/04/02).

[53] Nicholas J Dingle, William J Knottenbelt, and Tamas Suto. Pipe2: a tool for the performance evaluation of generalised stochastic petri nets. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):34–39, 2009.

[54] PIPE – Platform Independent Petri Net Editor. https://github.com/sarahtattersall/PIPE(accessed 2017/03/22).

[55] Giacomo Bucci, Laura Carnevali, Lorenzo Ridi, and Enrico Vicario. Oris: a tool for modeling, verification and evaluation of real-time systems. *International Journal on Software Tools for Technology Transfer (STTT)*, 12(5):391–403, 2010.

[56] Oris Tool – analysis of timed and stochastic petri nets.
`http://www.oris-tool.org/` (accessed 2017/03/22).