An Interactive Learning Platform for Compiler Data-Flow Analysis

Ayrton Massey

4th Year Project Report
Computer Science
School of Informatics
University of Edinburgh

2016

Abstract

Data-flow analysis is one of the cornerstones of modern compiler optimisation. A thorough understanding of the processes involved is essential to further exploration of the subject. A tool which allows exploration of data-flow analysis in an interactive environment would prove invaluable to students encountering the topic for the first time.

This report describes the design and implementation of an interactive system to simulate and visualise forms of data-flow analysis on simple assembly-like programs. The system is evaluated by in terms of user experience and the achievement of learning outcomes, through self-assessment and by examining usage data collected during the evaluation period.

The software was wildly successful in increasing the learning capacity of the evaluation subjects. The results collected present strong evidence of the need for more interactive learning tools, particularly in engineering subjects. Responses to the feedback survey were overwhelmingly positive, achievement of learning outcomes was well above average, users were kept engaged by the platform, and all but one participant stated that they would use the system alongside other methods of study.

Acknowledgements

First and foremost I would like to thank the project's supervisor, Hugh Leather, for his continued support throughout the project and for volunteering his time and effort during particularly unfortunate circumstances.

I am extremely grateful for the assistance provided by Drs. Alan Smaill and Iain Murray, and the UG4 project co-ordinator Prof. Don Sannella.

My sincere gratitude goes to Dr. Steve Gregory of the University of Bristol for distributing the application to his students, and to the participants of the evaluation without whom this project would not have been possible.

Finally, I would like to thank my family and friends for their support and encouragement throughout my academic career.

Table of Contents

1 Introduction 1

1.1 Data-Flow Analysis . . . 1

1.2 Motivations . . . 2

1.3 Objectives . . . 2

1.4 Summary of Contributions . . . 3

2 Background 5

2.1 Modern Compiler Design . . . 5

2.1.1 Control-flow Graph . . . 6

2.2 Data-Flow Analysis . . . 6

2.2.1 A Simple Example . . . 6

2.2.2 Properties of Data-Flows . . . 7

2.2.3 Lattices . . . 7

2.2.4 Bit-Vector Data-Flows . . . 8

2.3 Algorithms for Analysis . . . 9

2.3.1 Round-Robin Iterative Algorithm . . . 9

2.3.2 Worklist Algorithm . . . 10

2.3.3 Structural Algorithm . . . 10

2.4 Data-Flow Frameworks . . . 10

2.4.1 Algorithm for General Frameworks . . . 11

2.4.2 Conditions for Termination . . . 11

3 Related Work 13

3.1 Introduction . . . 13

3.2 Online Learning Platforms . . . 13

3.3 Simulators as a Teaching Aid . . . 15

3.4 Learning and Teaching Styles . . . 15

3.4.1 Ability to Identify Own Learning Style . . . 16

3.4.2 Criticism of Learning Modalities . . . 17

3.5 Summary . . . 17

4 Design 19

4.1 Introduction . . . 19

4.2 Design Constraints . . . 19

4.2.1 Technical Constraints . . . 20

4.2.2 Content Constraints . . . 20

4.3 System Architecture 21

5 Back-End Implementation 23

5.1 Overview 23
5.2 Value Sets 23
5.3 Data-Flow Frameworks 24
5.3.1 Included Data-Flows 25
5.4 Simulation Algorithm 26
5.5 Understanding Programs 27
5.5.1 Parsing Expression Grammar 27
5.5.2 Abstract Syntax Tree 28
5.5.3 Control-Flow Graph 29
5.5.4 Lattices 31
5.6 Summary 32

6 Front-End Implementation 33

6.1 Overview 33
6.2 View Model 33
6.3 Visual Components 34
6.3.1 Control-Flow Graphs 34
6.3.2 Hasse Diagrams 35
6.3.3 Table of Results 35
6.3.4 Code Display 36
6.4 Interactive Components 37
6.4.1 Simulator Controls 37
6.4.2 Questions 38
6.5 User Interface 39
6.5.1 Overview 39
6.5.2 Menu 39
6.5.3 Simulation Interface 40
6.5.4 Tutorial Interface 41
6.5.5 Testing Interface 43
6.6 Summary 45

7 Evaluation 47

7.1 Introduction 47
7.2 Method 47
7.2.1 Usage Data 49
7.2.2 Feedback Survey 50
7.3 Demographics 51
7.4 Critical Analysis 52
7.4.1 Time Spent 52
7.4.2 Lessons Completed 54
7.4.3 Achievement of Learning Outcomes 55
7.4.4 User Opinion 57
7.4.5 Written Feedback 59
7.5 Future Developments 60

7.6 Conclusion 62

Bibliography 63

Appendices 65

A Terminology 65

A.1 Glossary 65

A.2 Acronyms 67

B Types of Data-Flow Analysis 69

Chapter 1

Introduction

This chapter gives a short introduction to the topic of data-flow analysis, describes motivations and desired outcomes for the project and provides a brief summary of contributions.

1.1 Data-Flow Analysis

Data-flow analysis is a tool for analysing the flow of data through a program at various points in its execution. Analysis is performed over a control-flow graph, computing the properties of values flowing *in* and *out* of each node. Many forms of data-flow analysis exist to compute various properties, for example *liveness analysis* identifies variables which will be used in future instructions and *available expressions* identifies those expressions whose value has been previously computed at some point in the control-flow graph.

This analysis is used to inform optimisations which can be performed on a given program. Using the example of *liveness analysis*, the values computed can be used to optimise register allocation: a variable which is not live at a given point does not need to be allocated to a register, enabling more efficient use of available resources.

Data-flow analysis is not only useful in compiler optimisation. The information gathered can be used in other ways, such as identifying unsafe operations in PHP web applications[1] by monitoring sanitization[1] of variables which have been assigned to user input.

[1] To *sanitize* a user input is to remove any potentially dangerous elements from said input; for example, if a user input string is to be inserted into the HTML of a webpage it could be sanitized by replacing instances of < and > with `&lt;` and `&gt;`, respectively. This would prevent that input being misinterpreted as HTML and thus avoid malicious scripts contained within that input from being executed.

1.2 Motivations

This project was inspired by the project's supervisor, Hugh Leather. The original concept was an online tutor for data-flow analysis which would allow users to simulate an analysis on simple programs. The user could vary parameters, such as the data-flow in question or the order in which nodes are evaluated, and examine the resulting solution.

As lecturer of the Compiler Optimisations (COPT) course at the University of Edinburgh, Dr. Leather desired a system which could teach students the foundations of the course in a more interactive format than standard lectures. The system should be suitable for hosting on the course web page in order to provide access to all students.

My personal interest in this project stemmed from a desire to use my practical skills to increase my capacity for understanding theoretical content. As noted in our early discussions, many students find it difficult and time consuming to read and understand material from the course textbook. Presenting this information in such a way that it could be easily digested by even a novice to Computer Science proved an exciting challenge.

1.3 Objectives

The main aim of this project was to create an interactive system to teach students the basic principles of data-flow analysis in compilers.

This would take the form of a web application using visual components which could be combined in different ways, for example to present a series of tutorials on data-flow analysis or to provide a sandbox environment to explore. The content of the application would cover a range of topics from the basics of data-flow analysis to algorithms and frameworks for solving generic data-flow problems.

The application would be aimed at students of the COPT course and as such would be based on material from the course textbook *Engineering a Compiler, 2nd ed.*[2] by Keith D. Cooper and Linda Torczon. The application could then be extended to cover the topic in more depth using content from *Compilers: Principles, Techniques and Tools, 1st ed.*[3] by Alfred V. Aho, Ravi Sethi and Jeffery D. Ullman (commonly referred to as the *Dragon Book*).

Users would be able to interact with the system by providing simple assembly-like programs, altering parameters of the analysis and stepping through a simulation. Elements of the simulation such as the current state and the control-flow graph of the program would be visualised on-screen and update as the simulation progressed. Each of these elements would be linked visually to show how the concepts relate.

The application would be tested on real users. It would be evaluated in terms of user experience by analysing interactions with the system and conducting a user experience survey; achievement of learning outcomes would be assessed by examining responses to questions built into the software and self-assessment by the user.

1.4 Summary of Contributions

The final version of the software is an online learning platform featuring a sandbox simulation for data-flow analysis and a series of interactive tutorials which provide the background knowledge required in order to use it.

A summary of contributions is as follows:

- Simulation of pre-defined data-flows using generic framework models. (p. 24)
- Simulation of analysis on user-defined programs using the ILOC[2, appx. A] language from *Engineering a Compiler.* (p. 27)
- A parser for the ILOC language, with extended grammar rules. (p. 27)
- A built in code editor with the ability to share programs (p. 36)
- Simulation using the round-robin iterative algorithm allowing step-by-step, instant or automated playback. (p. 26)
- Visualisation of the following simulation elements:
  - Control-flow graph (p. 34)
  - Simulator state incl. currently evaluated node, framework etc.
  - Table of results displaying data flowing in / out of each node. (p. 35)
  - Hasse diagram of meet semi-lattice (p. 35)
- Tutorials covering basics of the topic with interactive elements (p. 41)
- Interactive test to assess achievement of learning outcomes (p. 43)
- An API to record user interaction events modelled on the Google Analytics event tracking system (p. 49).

Detailed explanations of the terminology mentioned above can be found in chapter 2. A quick summary of terms can be found in the glossary in appendix A.

$$\begin{aligned}
\mathit{nonconst} \wedge c &= \mathit{nonconst} && \text{for any constant } c \\
c \wedge d &= \mathit{nonconst} && \text{for any constants } c \neq d \\
c \wedge \mathit{undef} &= c && \text{for any constant } c \\
\mathit{nonconst} \wedge \mathit{undef} &= \mathit{nonconst} && \\
x \wedge x &= x && \text{for any value } x
\end{aligned}$$

Fig. 2.5: Equations describing the constant propagation meet function.

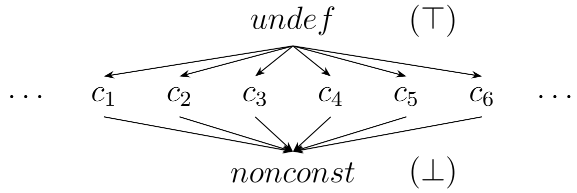

Fig. 2.6: A Hasse diagram for the meet function described in fig 2.5.

2.3 Algorithms for Analysis

There exist a number of algorithms for performing data-flow analysis. This section contains a brief overview of the methods proposed in the related reading[2] [3].

2.3.1 Round-Robin Iterative Algorithm

The simplest algorithm used to solve data-flow problems is the round-robin iterative method. We consider each node of the CFG in turn, calculating the In and Out sets using our data-flow equations. This is a fixed-point computation: we iterate until our value sets stop changing between iterations. Algorithm 2.1 shows the round-robin iterative method for solving reaching definitions.

Algorithm 2.1: Iterative Round-Robin Method for Reaching Definitions

```
for each (node $n$ in the CFG)
    In($n$) = $\emptyset$;

while (changes to any sets occur)
    for each (node $n$ in the CFG)
        In($n$)  = $\cup_{\text{predecessors } p \text{ of } n}$Out($p$);
        Out($n$) = DefGen($n$) $\cup$ (In($n$) $\setminus$ DefKill($n$));
```

2.3.2 Worklist Algorithm

The above solution holds value in its simplicity, but it is trivial to find a more efficient solution. A node's sets will only change if the Out sets of its predecessors change. Thus, we may use a worklist: starting with the initial node, we calculate the Out set for that node; if it changes we add the node's successors to the list. We continue this process until the worklist is empty. Algorithm 2.2 shows the worklist method for solving reaching definitions.

Algorithm 2.2: Worklist Method for Reaching Definitions

```
worklist = [ $n_0$ ];

while (worklist is not empty)
    n = pop(worklist);
    In($n$)  = $\cup_{\text{predecessors } p \text{ of } n}$Out($p$);
    Out($n$) = DefGen($n$) $\cup$ (In($n$) $\setminus$ DefKill($n$));
    if (Out($n$) has changed) append n to worklist;
```

2.3.3 Structural Algorithm

A third algorithm takes an entirely different approach. Instead of iterating over each node, we perform a number of simple transformations on the graph in order to reduce it to a single region. We then expand each node, calculating the In and Out sets of our reduced nodes as we expand them. A more detailed explanation of this concept may be found in the *Dragon Book*[3, p. 673].

2.4 Data-Flow Frameworks

It is possible to model data-flow problems using a generic framework. This allows us to use the same algorithm for multiple problems by specifying the following constraints[3, p. 680]:

- The *domain* of values on which to operate;
- The *direction* in which data flows;
- A set of *data-flow equations* including the *meet operator* $\wedge$ and the set of *transfer functions* F[2];
- The *boundary* value $v_{BOUNDARY}$ specifying the value at the entry or exit to the CFG; and
- The *initial value*, $\top$, at each point in the graph.

[2] The function corresponding to a particular node/block B is denoted F_B

2.4.1 Algorithm for General Frameworks

Algorithm 2.3, adapted from the one in the *Dragon Book*[3, p. 691], computes the value sets at each node using the elements of our general framework.

Algorithm 2.3: Data-Flow Analysis of General Frameworks

```
$\text{Meet}_{BOUNDARY} = \text{v}_{BOUNDARY}$;

for each (block $B$ in the CFG)
    $\text{Meet}_B = \top$;

while (changes to any Transfer occur)
    for each (block $B$ in the CFG)
        $\text{Meet}_B     = \wedge_{\text{priors } P \text{ of } B}$ $\text{Transfer}_P$;
        $\text{Transfer}_B = F_B(\text{In}_B)$;
```

Instead of referring to the value sets as In and Out as the *Dragon Book* does, we may call them Meet and Transfer. This allows us to generalise our algorithm to both forward and backward analyses; in the forward direction Meet is In whereas in the backward direction it is Out (and vice-versa for Transfer).

We first initialise the Transfer set using the boundary condition, then initialise each node's transfer set to our initial value $\top$.

Next, we perform a fixed-point computation on the CFG, evaluating each node's Meet and Transfer sets using our data-flow equations until the sets stop changing.

The meet is taken over a node's *priors*: in the forward direction, the node's predecessors; in the backward direction, the node's successors.

This algorithm can be applied to any framework. In fact, all of the data-flow problems in appendix B may be solved using this process.

2.4.2 Conditions for Termination

We must be careful when constructing our general frameworks. If our value sets continuously change we may never reach a fixed-point and thus our computation will never halt.

To avoid this, our frameworks must satisfy the following conditions[3, p. 684]:

- The set of transfer functions, F, contains the identity function[3];
- F is closed under composition: that is, for any two functions f and g, $f(g(x))$ is also in F;
- F is monotone; and
- The domain and the meet operator, $\wedge$, must form a meet semi-lattice.

[3] The identity function maps its input to its output, i.e. $F(x) = x$

These conditions ensure that during every iteration of the algorithm the values at each point will either become *smaller* (with respect to the partial ordering) or stay the same. Since F is monotone, all of the sets must eventually stop changing **or** reach the bottom element of the lattice, $\perp$, at which point they cannot change any further.

3.3 Simulators as a Teaching Aid

In his article *Evaluating A System Simulator For Computer Architecture Teaching And Learning Support*[7], Mustafa discusses the design & implementation of a system which integrates the simulation of a compiler, CPU and operating system to aid in the teaching of undergraduate computer architecture and operating systems modules.

Although brief, the article provides valuable insight into the design and evaluation of such teaching software. The primary source of feedback came from an opinion survey using a 5-point Likert scale for quantitative analysis and open-ended questions for qualitative feedback. In addition, students were administered a test to assess their knowledge pre- and post- use of the system.

The evaluation results highlight an important point to consider when designing teaching software; over 20% of respondents indicated that they spent more time learning how to use the software than they did completing the given exercises. An equal number of students reported that the simulator left them more confused than before. In addition, 7.1% of respondents said that the simulator was too complicated to use effectively in their tutorials.

However, the converse of these results gives a positive outlook for this project: 72.4% of respondents disagreed with the above statements and 79.3% believed that the system improved their understanding of the topics covered in lectures. Over 95% of respondents agreed that the simulator was more useful than reading textbooks or searching the internet in helping them understand the material.

3.4 Learning and Teaching Styles

A 1988 report by Richard M. Felder and Linda K. Silverman, entitled *Learning and Teaching Styles in Engineering Education*[8], categorises students' learning methods and describes ways in which professors may target specific categories in their teaching strategy.

Felder refers to the learning modalities (or VAK) model proposed by Walter Burke Barbe, which defines three modalities:

- **visual** – those who learn best by viewing images and diagrams;
- **auditory** – those who learn best by listening or speaking aloud; and
- **kinaesthetic** – those who learn best by actively experiencing things, learning by doing.

The report claims that most people of college age and above, the intended audience of this project, identify as *visual* learners – those who benefit from charts, diagrams and such. In contrast, the presentation of content in university courses is primarily *auditory* (via lectures) or a visual representation of auditory information (i.e. words or mathematical formulae).

The survey results from Mustafa[7, p. 103] support this observation: although only a small sample (N=54) responded, 31% of respondents identified as visual learners, while a mere 6% identified as auditory learners. The majority of students (52%) identified as kinaesthetic learners, indicating that taking a hands-on approach is a valuable tool for learning.

It should be noted that Felder discounts kinaesthetic learning from his report as he considers the *learning by doing* to be a separate category, perceiving the remaining attributes of kinaesthetic learning to have little value in engineering. A later observation by Felder agrees with Mustafa, stating that engineers are "more likely to be active than reflective learners" [8, p. 678].

3.4.1 Ability to Identify Own Learning Style

In her PhD dissertation *"Individual Differences in Learning: Predicting One's More Effective Learning Modality"*[9], Beatrice J. Farr claims that students are able to accurately predict the learning style in which they perform best:

> "An experiment with 72 college students confirmed that individuals could accurately predict the modality in which they could demonstrate superior learning performance. The data also revealed that it is advantageous to learn and be tested in the same modality and that such an advantage is reduced when learning and testing are both conducted in an individual's preferred modality." [10, p. 242]

Coffield[11, p. 120] disagrees with this, based on an observation by Merrill[12] that "most students are unaware of their learning styles and so, if they are left to their own devices, they are most unlikely to start learning in new ways". This would indicate that basing teaching methods on a student's preferences is damaging to their education. This is, however, a misinterpretation; the actual text of Merrill reads:

> "... a student must engage in those activities ... that are required for them to acquire a particular kind of knowledge or skill ... Most students are unaware of these fundamental instructional (learning) strategies and hence left to their own are unlikely to engage in learning activities most appropriate for acquiring a particular kind of knowledge or skill." [12, p. 4]

Merrill later argues that the optimal strategy for teaching is decided first and foremost by the content being taught, then fine-tuned to the learner's preferred style[12, p. 4]. By not knowing the most effective strategy for learning the content they are studying, a student limits his or her potential. However, the appropriate strategy *should* in fact be tailored to the student's preferred learning style to obtain the best results.

3.4.2 Criticism of Learning Modalities

Some of the criticism levied at the concept of learning styles is that the idea of *matching* – exclusively teaching a student based on his or her preferred learning style – is harmful to a student's education. Although Coffield's[11, p. 120] reasoning is flawed, the conclusion drawn by him and Merrill[12, p. 4] is sound: by allowing students to exclusively use their preferred way of learning, students are at risk of missing out on more effective methods of study.

Felder's report mentions a study carried out by the Socony-Vacuum Oil Company, which concludes that:

> "...students retain 10 percent of what they read, 26 percent of what they hear, 30 percent of what they see, 50 percent of what they see and hear, 70 percent of what they say, and 90 percent of what they say as they do something." [8, p. 677]

This indicates that by relying solely upon auditory methods, professors can only hope to convey as little as 26% of the desired material to their students. Felder advocates using a mixture of teaching methods in order to appeal to all students' preferred learning styles.

3.5 Summary

While there may be some disagreement over the validity of learning styles or modalities, even its critics seem to agree that there is value in varying the methods of teaching in use. At present, the only available resources for learning data-flow analysis are lecture slides, videos and textbooks. These are mostly auditory and sometimes visual methods of teaching, with little kinaesthetic learning involved. There is a clear need for more active study; although 52% of students identified as learning best through kinaesthetic learning[7, p. 103], there is almost no support for this method of study.

Given the relative success of the examples discussed in this chapter and the apparent lack of any such resources for data-flow analysis, there is a strong precedent for this project and as such a system will be developed to fulfil this role.

However, it is important to note the mistakes made by the examples discussed here. The simulator designed by Mustafa was deemed too complex[7, p. 103] and a detriment to their learning by over 20% of respondents. This project will seek to ensure that that content is clear and concise on order to produce an effective learning resource.

To evaluate the success of the system this project will build upon the methods presented by Mustafa, aggregating opinion using a Likert scale and examining this data to judge overall satisfaction with the software and identify specific areas for improvement.

Chapter 4

Design

This chapter discusses the high-level design and architecture of the system, including motivations for design choices and solutions to conceptual problems.

4.1 Introduction

The original goal of this project was to produce an online simulation of data-flow analysis to show students how data-flow works. As discovered in the related reading, however, a large portion of users of a similar system found it to be too complex and claimed to have spent more time studying the software itself than gaining useful knowledge. The literature also raised another key point; a wide variety of learning techniques are necessary to gain a true understanding of a topic.

For these reasons the proposal has been extended to create a comprehensive learning platform. In addition to the simulator the software will provide supporting lectures to explain the basic concepts and gradually introduce each element of the simulation. These lectures will include visual and interactive elements to engage the user through a range of learning styles. It is hoped that the system will prove a valuable tool for learning alongside existing resources such as textbooks and lectures.

4.2 Design Constraints

In this section, design constraints are identified and potential solutions are suggested.

4.2.1 Technical Constraints

As the intention is to make such a system available to COPT students via the University, the following technical considerations must be made:

- The system must be distributed to all students in some format;
- This format must be functional on and compatible with a wide range of devices owned by said students;
- The system must be secure and require little maintenance; and
- The system must be hosted on some platform available to the University.

In order to meet these criteria the system must rely on as little technology as possible. The easier the platform is to host and distribute, the more likely it is to be made available to students – a system which uses new technology and requires its own dedicated hosting would be more difficult to set up and maintain than one which can be deployed to existing hardware. The application could be developed using purely client-side technologies such as JavaScript, HTML and CSS. These static files may be distributed over HTTP using any standard web server such as the one already used to host the course webpage.

Efforts must be made to keep the performance of the system consistent across a range of web browsers and devices; although all modern web browsers are capable of interpreting these types of content there are subtle differences which may break functionality on one platform but not others.

It is also necessary to ensure that the system is secure to avoid damaging University or student property. Provided that user input is sanitized and scripts are only included from trusted sources, the sandboxed environment inherent to modern web browsers should meet the remaining security requirements.

Use of popular libraries and frameworks such as jQuery and Bootstrap will be encouraged as this provides a number of benefits. The user will experience reduced load times since they have likely already downloaded the required files, and such libraries will provide security and robustness due to their wide use and active development. Likewise, the more popular a library is the more documentation and resources will be available to aid in developing the best possible system.

The specific technologies used in each are detailed in the implementation (chapters 5 & 6).

4.2.2 Content Constraints

To be viable as a learning platform, the system must:

- Appeal to a range of learning styles;
- Provide comprehensive coverage of the topic at hand;

- Ensure the content included is correct, clear and concise; and
- Maintain a shallow learning curve, gradually introducing students to each topic or element of the simulation.

Extending the proposed system into a comprehensive learning platform will provide a range of ways in which to study data-flow analysis. Students will have the choice to use the areas of the software which most appeal to them and content will be presented using a mixture of interactive, visual and textual formats.

As the aim is to assist students of the COPT course, coverage of topics will be prioritised based on their inclusion in the course syllabus and whether they are required to understand the simulation:

- Basic principles of data-flow analysis;
- A few simple data-flows, including:
    - Reaching Definitions
    - Liveness Analysis
    - Available Expressions
- Round-robin fixed-point algorithm;
- Effect of orderings on analysis efficiency;
- Generic frameworks for data-flow analysis;
- Hasse diagrams and lattice representation of values; and
- Conditions for analysis termination.

If the content delivered by the platform is incomplete, incorrect or too difficult to understand, the system will not present a viable alternative method of study. When evaluating the system it will be important to assess the clarity and presentation of the material, and the ease of use of the platform as a whole.

4.3 System Architecture

The system will be divided into two main areas of development. The back-end of the system controls the logic and state of a data-flow analysis simulation. The front-end components will hook into and expose elements of the back-end, visualising the state of the simulation and allowing the user to interact with it. See fig. 4.1 for a visual representation.

Object-oriented design will be paramount, allowing functionality to be extended and shared between similar elements. In addition to speeding up development of the system by reducing the need to duplicate code, this is generally good design practice. Components will be written as re-usable modules which can be combined in different ways, for example to present the user with a series of lectures or an interactive simulation. See §6.5 for more details on the user interface implementation.

Each component needs to be be self contained; it should be possible to instantiate more than one of each type of component at once, including the simulator itself. This would allow, say, displaying the annotated control-flow graphs of two simulations side-by-side. The simulation will also be independent of the user interface so that it may be adapted for re-use in other projects.

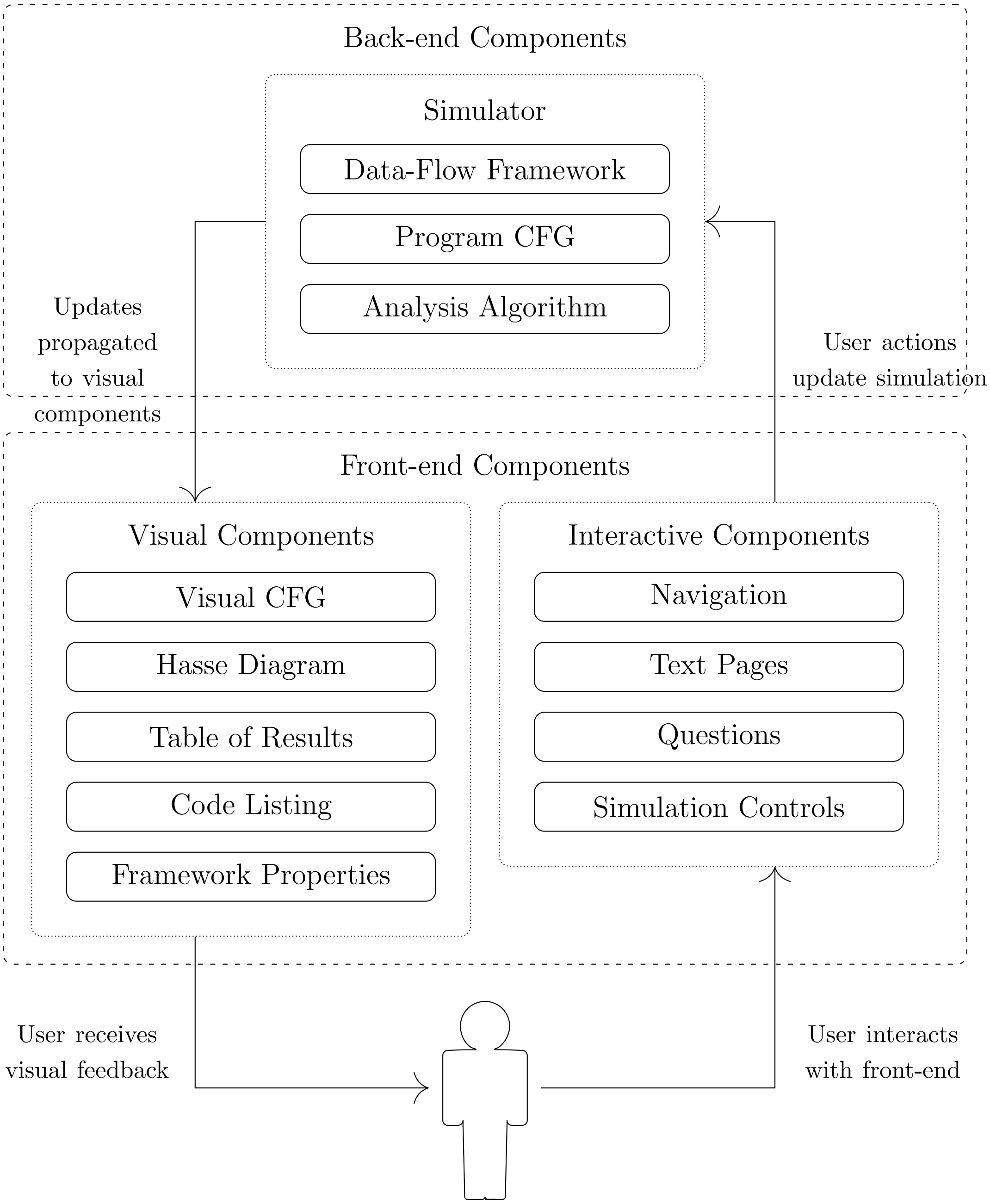

Fig. 4.1: An overview of the proposed system architecture.

Chapter 5

Back-End Implementation

This chapter discusses the implementation of the data-flow analysis simulation – the back-end elements which control the logic and state of the analysis, which is then visualised by the user interface components described in chapter 6.

5.1 Overview

The implementation of the simulator is based upon the concept of a general framework as described in §2.4 of this report and §10.11 of the *Dragon Book*[3]. The simulator takes as input a program written in the ILOC (see §5.5) language, a data-flow framework and an order in which to evaluate nodes. It parses the program and produces an AST, then uses this to build a CFG. The data-flow may then be simulated on said graph using the round-robin fixed-point algorithm for general frameworks (algorithm 2.3).

Playback of the simulation can be controlled through the simulator's API. Function calls which update the state of the simulation trigger an event handling system to update the visual components. However, the simulation is entirely independent of the rest of the system; re-using it in another project is as simple as copying the directory containing the simulator code and including the required libraries.

The simulator is implemented entirely in JavaScript. Snippets of code taken from external sources are explicitly commented as such in the source code and will be mentioned here.

5.2 Value Sets

JavaScript includes a native `Set` data structure for representing collections of unique objects. Unfortunately, two objects are only seen as equal if they refer to the exact same instance. For the purposes of this simulation it would be useful to compare objects which share some attributes but refer to different instances; for example, when operating on sets of operands it would be useful to consider two operands of different instructions as the same if they refer to the same variable.

For this reason the simulator uses a new `ValueSet` data structure, backed by a native JavaScript `array`. The `ValueSet` stores objects which inherit from a `ValueMixin`[1] and must define a `compare` function. Objects are checked for equality using this function so that two objects may be considered equal based on selected attributes.

`ValueSet`s support the following operations, listed here along with their estimated worst-case time complexities:

`s.size()`	`s.add(v)`	`s.delete(v)`	`s.has(v)`
$O(1)$	$O(\|s\|)$	$O(\|s\|)$	$O(\|s\|)$

`s.union(t)`	`s.intersect(t)`	`s.difference(t)`
$O(\|s\| \cdot \|t\|)$	$O(\|s\| \cdot \|t\|)$	$O(\|s\| \cdot \|t\|)$

Using an alternate backing structure such as a hash table would provide constant-time implementations of `has`, `add` and `delete`, and thus also improve the runtime of operations which make use of these functions (`union`, `intersect` and `difference`). Given that the expected user input will only operate on small `ValueSet`s this is not a major concern, but a potential improvement that should be noted for future development.

5.3 Data-Flow Frameworks

Each data-flow framework defines the elements outlined in §2.4, namely:

- The *domain* of values on which to operate;
- The *direction* in which data flows;
- A set of *data-flow equations* including the *meet operator* $\wedge$ and the set of *transfer functions* F.
- The *boundary* value specifying the initial value at the entry or exit to the CFG; and
- The *initial value*, $\top$, at each point in the graph.

The meet and transfer functions operate on `ValueSets`. Frameworks must also specify the following additional information:

- A name and identifier;
- LaTeX representations of the meet and transfer functions; and

[1] A mixin class contains methods which may be inherited by another class, but is not necessarily the parent of that class.

5.5 Understanding Programs

The simulation operates on programs written in ILOC, as described in *Engineering a Compiler*[2, appx. A]. ILOC is an assembly-like language designed for toy compilers, in that it is both simple to parse and human-readable. This makes it the perfect input language for the learning platform, with the added benefit that students will be able to step through examples from the textbook using the simulator.

The simulator supports the following ILOC features:

- The entire set of opcodes listed in *Engineering a Compiler*;
- Three types of literal: registers, labels, and integers;
- Simple programs with a single operation per instruction; and
- Distinction between three types of memory: registers, main memory, and the comparison register.

There is also limited support for multi-operation instructions, which may be used to emulate data-flow analysis on basic blocks. This functionality remains untested as priority was assigned to simulation on nodes as single instructions.

In order to understand ILOC programs, they must be to converted from text into a CFG. The first stage in this process is parsing the text. The PEG.js[13] was used to generate a parser (written in JavaScript) from an input grammar. This parser produces an abstract syntax tree (AST), which is then transformed into a CFG representing the program. The following sections describe this process in more detail.

5.5.1 Parsing Expression Grammar

A Parsing Expression Grammar (PEG) is very similar to a context-free grammar in that it describes a formal language using a set of rules for recognising strings in that language. The description of ILOC in *Engineering a Compiler* describes a simple PEG for the language, which has been adapted for use in this system.

Some rules in the grammar have been extended as shown in fig. 5.2. The reason for these changes is that ILOC allows the user to write to three types of storage: registers, main memory and comparison flags. Changing the grammar allows disambiguation in the abstract syntax tree between operations which write to or read from different types of storage. This is an important distinction to make because the built-in data-flows only consider register accesses when calculating value sets at each point (see §5.3.1 for an explanation).

$$
\begin{array}{rcl}
Operation & \rightarrow & NormalOp \\
 & | & ControlFlowOp \\
 & | & MemoryLoadOp \\
 & | & MemoryStoreOp
\end{array}
$$

$$
\begin{array}{rcllcl}
MemoryLoadOp & \rightarrow & LoadOpcode & OperandList & \Rightarrow & OperandList \\
MemoryStoreOp & \rightarrow & StoreOpcode & OperandList & \Rightarrow & OperandList \\
NormalOp & \rightarrow & NormalOpcode & OperandList & \Rightarrow & OperandList \\
ControlFlowOp & \rightarrow & ControlFlowOpcode & OperandList & \Rightarrow & OperandList
\end{array}
$$

$$
\begin{array}{rcl}
LoadOpcode & \rightarrow & \texttt{loadI} \\
 & | & \texttt{loadAO} \\
 & | & \ldots \\
 \\
StoreOpcode & \rightarrow & \texttt{storeI} \\
 & | & \texttt{storeAO} \\
 & | & \ldots \\
 \\
NormalOpcode & \rightarrow & \texttt{addI} \\
 & | & \texttt{rshift} \\
 & | & \ldots \\
 \\
ControlFlowOpcode & \rightarrow & \texttt{jumpI} \\
 & | & \texttt{cbr\_GE} \\
 & | & \ldots \\
 \\
Operand & \rightarrow & \textit{register} \\
 & | & \textit{num} \\
 & | & \textit{label} \\
 & | & \textit{cc} \\
 \\
\textit{cc} & \rightarrow & \texttt{cc}
\end{array}
$$

Fig. 5.2: Extensions to ILOC Parsing Expression Grammar

5.5.2 Abstract Syntax Tree

A parse tree represents the exact structure of parsed data. In contrast, an AST represents the structure of data independent of its original representation; this may involve adding or removing nodes or re-structuring the tree entirely.

To construct an AST one would traditionally obtain a parse tree and then transform it into the desired structure. In contrast, parsers generated by the PEG.js library are capable of constructing the AST as they parse the input program. The developer may supply JavaScript code to be executed when a rule matches and this code is used to create an abstract representation of the data matched by that rule. This feature can also be used to annotate the tree as it is built.

Listing 5.3 shows a snippet of the PEG.js grammar for ILOC. Groups of symbols may be assigned an identifier and referred to in the JavaScript code; on line 9

Chapter 6

Front-End Implementation

This chapter discusses the implementation of the front-end elements which make up the user interface, including the visualisation and interactive teaching components.

6.1 Overview

The implementation of the user interface components uses a modular design. Each element is a `View`, a component which renders content to a HTML element referred to as that component's canvas. An inheritance model is used so that functionality is shared between `Views`, for example, each of the visual components inherits from `SimulatorView`, which handles registering callback functions with a simulator's event handler.

The front-end is implemented using JavaScript, HTML and CSS. A number of libraries provide additional functionality, including:

- d3.js and the extension dagre-d3 to draw directed graphs;
- jQuery and extensions such as tipsy for navigation and visual elements;
- MathJax for rendering of $\LaTeX$ math formulae;
- Handlebars.js for HTML templating; and
- Bootstrap v4 alpha for page layout and theming.

6.2 View Model

Each user interface component is implemented as its own class, or `View`. A `View` component renders HTML content inside another HTML element referred to as its canvas. Multiple `Views` may exist at one time, and a `View` may contain other views.

Beside from the individual view classes, there are multiple types of `View`; for example, all of the visual components inherit from the `SimulatorView` class. Each `SimulatorView` is required to implement callback functions for the `update` and `reset` events. These functions are called by the simulator's event handler when an event is triggered, which may cause the component to be re-rendered or the information contained within to be modified.

In addition to specifying the interface for various classes, the `View` model allows functionality to be shared between components. The `TutorialView` class implements navigation through and display an interactive tutorial, so that each one need only implement functions to control the content shown at each step.

This model is a very simple implementation of what is known as a single page app: a web application in which the page is only loaded once, after which JavaScript is used to update the page content and asynchronous calls request more data from the server. Some existing libraries are based around this concept; for example AngularJS[15] provides a similar framework for creating single page apps using re-usable components.

6.3 Visual Components

This section describes the implementation of the components which visualise the simulation, including the choice of any libraries and frameworks and the justification of design decisions.

6.3.1 Control-Flow Graphs

Control-flow graphs are rendered using the dagre-d3 extension for the graph visualisation library d3.js. An SVG component is created and the nodes and edges added to the dagre-d3 graph. The layout of the graph is handled entirely by the graphing library.

The dagre-d3 library was chosen due to the strength of the demonstrations on the project's website[16]. The available examples showed graphs very similar to the desired result, with only a small set of API calls required to draw a simple control-flow graph. Given more time, perhaps using a more fully-featured graphing library such as Cytoscape.js[17] or vis.js[18] would improve the aesthetics and performance of the component. However, dagre-d3 is more than adequate for a proof of concept.

To reduce clutter in the graph, the user has three options for displaying nodes: *current*, in which only the points relating to the current step are shown; *all*, in which every point is displayed; or *none*, in which only the nodes of the CFG are shown. Fig. 6.1 shows the CFG in the *current* setting.

6.6 Summary

In summary, the user interface is fairly well-designed but lacks polish. Some small features such as progress indicators were overlooked during development; the loss of these elements slightly reduces the usability of the interface.

However, the design has many positive aspects. The interface was optimised for display on smaller screens to make the application accessible to all students, and the added code sharing functionality should help students share examples with their classmates or professors in order to develop their learning through discussion.

The simulation interface visualises the content clearly and is certainly the highlight of the application's front-end. The choice of popular external libraries allows the system to be easily modified and extended by most JavaScript or web developers, in addition to increasing performance and security as stated in 4.2.2. The system's modular design allows its components to be re-used across the application, reducing the need to duplicate code and providing consistency throughout.

Even without the improvements mentioned throughout this chapter, the platform is fully-functional and should prove itself a valuable tool for learning. I hope that the techniques I have employed in developing a truly interactive and visual learning experience have set a good example for similar projects in future.

Chapter 7

Evaluation

This chapter describes the evaluation methodology for the project, followed by a critical analysis of the design and implementation of the system and suggestions for future development. The chapter concludes with an assessment of the success or failure of the project as a whole.

7.1 Introduction

The goal of this project was to create an interactive system to teach students the basic principles of data-flow analysis in compilers. The final system took the form the form of an online learning platform, containing a data-flow analysis simulator and a series of interactive tutorials.

To determine whether the final product meets the original aim of the project, there are three main points against which it should be evaluated:

- Interactions with the system, to analyse the users' engagement with the platform;
- Achievement of learning outcomes by the participants, in order to assess its validity as a learning platform; and
- Feedback from the participants, to determine whether it is a useful tool for the intended audience.

The following sections present a thorough evaluation of the system against these three objectives.

7.2 Method

The intended audience of the project is primarily students of the COPT course. However, the number of students available is limited; the course has around 30 registered students and anecdotal evidence shows that around 10 of those regularly attend class. In order to collect a suitable amount of data to perform an evaluation the net must be widened to include participants from outside the course, perhaps including those with prior experience in STEM subjects or students in general. The aim was to obtain around 30 responses, as this is generally seen as a reasonable sample size in statistical analysis.

Evaluating the system requires the collection of data and supporting evidence from which conclusions may be drawn. There are a number of ways in which this data could be collected; for example, in-person evaluations may be performed and the sessions recorded. These recordings would then be analysed to obtain feedback. However, this method presents some issues: firstly, the participant may feel pressured to give a positive response in order to please the person conducting the evaluation. Second, whilst observing the participant would provide valuable insight into how participants use the application, it is very time consuming and difficult to find willing participants given the required time investment. To obtain the desired 30 responses, a more efficient method of data collection was required.

Two possible methods which would enable remote participation involve collecting usage information or distributing an online survey to gather feedback. These methods are useful but can be unreliable; when collecting usage information if the user has JavaScript disabled or their internet connection drops then loss of data could occur. It is also difficult to determine whether respondents to an online survey have actually used the software without witnessing them do so.

The most effective solution would therefore involve a mixture of all three methods: performing a few evaluation sessions in person would ensure that the methodology was sound and that the automated data collection functioned as intended, whilst enabling participants to complete the evaluation in their own time and removing the need for an assessor to be present. This also alleviates any pressure the participant may feel to provide a positive response.

This was the method chosen to evaluate the project: participants were given a link to the application and usage information was collected with their consent. Upon accessing the application for the first time, users were asked two questions in order to categorize them (see §7.3). A link in the application gave users the option to participate in a feedback survey. Three in-person evaluation sessions were held: two one-to-one sessions in order to validate the evaluation method itself, and one in-class session with the COPT students.

These sessions went well and thus an invitation was extended to the following sources:

- The remaining COPT students who did not attend the in-class session;
- Friends and family through my personal Facebook feed;
- Members of the "Informatics – Class of 2016 – University of Edinburgh" and "Informatics – University of Edinburgh" Facebook pages;
- Students of the Compilers courses at the Universities of Bristol, Birming-

ham, Durham, Oxford, Manchester, Warwick, Imperial College London and UCL, whose organisers' e-mail addresses were publicly available; and

- Members of the /r/compilers, /r/computerscience and /r/webdev communities on Reddit [20].

This should encourage participants from a wide range of backgrounds, including those with prior experience with or an interest in studying data-flow analysis, those with knowledge of similar material such as Computer Science and STEM students, developers of web applications who should be familiar with user experience (UX) and UI design, and those completely unrelated to any technical fields.

All participants were presented with the same version of the software and were encouraged to be truthful and honest with their feedback.

7.2.1 Usage Data

To collect usage information, the use of some tracking system was required. A variety of web platforms exist for this purpose including Google Analytics and Piwik. However, these platforms are intended for large-scale data collection and as such are not suited for the fine-grained analysis necessary in the evaluation of this project. For example, it is not possible to link tracked events to specific users or view a complete list of triggered events using Google Analytics. Furthermore, extended use of these platforms often comes at significant financial cost. It is also important to consider the ethical issues that may arise from sharing usage data with companies like Google; using such a platform would require a much deeper discussion around user privacy and use of the data for financial gain.

Developing a custom event-tracking API would allow full control of the data collected both in terms of accessing the raw data and maintaining user privacy. The data model used in this API must be able to store the following on a per-user basis:

- Page views;
- Button clicks and interactions with components;
- Page load times; and
- Question attempts and overall scores.

This data can be collected using an API with a single endpoint[1], based on the Google Analytics event model. Each `event` record has the following fields:

- A `tracking token` used to group events by user;
- A `type` such as page view or button click;

[1] An endpoint is a URL at which a particular API service may be accessed, e.g. `api.com/users/` may provide access to user data

- The `datetime` at which the event occurred; and
- Four levels of categorization: `category`, `action`, `label`, and `value`. Each field can be used to group similar events and store values such as question scores or which element was clicked. The actual content of these fields does not matter as long as the grouping is consistent.

The tracking token is generated when the user first visits the page and persists across visits to the site. Storing this token enables analysis of how a user's actions affect their achievement of learning outcomes, such as whether completing a given tutorial or using a simulator component increases question scores in the related topic. It would be possible to determine which elements of the software users find difficult to understand based on their interactions with each component.

This design was implemented in Python using Django REST Framework (DRF) [21], due to the ease with which one may develop a simple web API. This application was hosted on an Apache server using the `mod_wsgi` module. The API receives events as JSON data and stores them in a PostgreSQL database hosted on the same server. A small JavaScript library was written to generate identification tokens and send user events to the API.

All of the data sent to the API is completely anonymous; the tracking token is randomly generated and exists only to tie records to a given session. In addition to the events outlined above, the application sends the user's categorization answers to the API to provide context to the data collected.

7.2.2 Feedback Survey

After using the software, participants had the option to fill out a short evaluation survey. This survey was produced using Google Forms [22] because of its flexibility and low risk regarding loss of data.

The first page of the form, which requests the same categorization information as the application, is automatically filled if the user follows the link to the survey from the application's main menu. This reduces the amount of time users must spend filling out the survey and increases consistency between the data collected via the API and the survey.

Following the example set by Mustafa [7] (see §3.3), the survey consists of a number of Likert scale questions ranging from "Strongly Agree" to "Strongly Disagree". Participants were required to select a response to each Likert item, but were given a "No Response" option instead of the usual "Neutral" response which allowed them to explicitly indicate that they did not wish to respond or that they had no strong feelings one way or another. This provided context as to why participants did not respond to a particular item, ensuring that they did not accidentally miss a question.

More information about the content of the survey can be found throughout the remainder of this chapter.

Bibliography

[1] A. Rimsa, M. d'Amorim, and F. Pereira. Tainted Flow Analysis on e-SSA-form Programs. *International Conference on Compiler Construction*, 2011.

[2] K. Cooper and L. Torczon. *Engineering a Compiler*. Morgan Kaufmann Publishers, 2nd edition, 2011.

[3] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques and Tools*. Addison-Wesley Publishing Company, 1st edition, 1986.

[4] Coursera. About Coursera. `https://www.coursera.org/about/`, 2015. Accessed: 2015-09-20.

[5] HackerRank. Reverse a linked list: Challenges. `https://www.hackerrank.com/challenges/reverse-a-linked-list/`, 2015. Accessed: 2015-09-21.

[6] Khan Academy. Implementing binary search of an array. `https://www.khanacademy.org/computing/computer-science/algorithms/binary-search/a/implementing-binary-search-of-an-array`, 2015. Accessed: 2015-09-21.

[7] B. Mustafa. Evaluating A System Simulator For Computer Architecture Teaching And Learning Support. *Innovation in Teaching and Learning in Information and Computer Sciences*, 9(1):100–104, 2010.

[8] R. M. Felder and L. K. Silverman. Learning and Teaching Styles In Engineering Education. *Engineering Education*, 78(7):674–681, 1988.

[9] B.J. Farr. *Individual Differences in Learning: Predicting One's More Effective Learning Modality*. PhD thesis, Catholic University of America, 1970.

[10] R. S. Dunn and K. J. Dunn. Learning Styles / Teaching Styles: Should They... Can They... Be Matched? *Educational Leadership*, 36:238 – 244, 1979.

[11] F. Coffield, D. Moseley, E. Hall, and K. Ecclestone. *Learning styles and pedagogy in post-16 learning*. The Learning and Skills Research Centre, 2004.

[12] M. D. Merrill. Instructional Strategies and Learning Styles: Which takes Precedence? *Trends and Issues in Instructional Technology*, 2000.

[13] David Majda. PEG.js – Parser Generator for JavaScript. `http://pegjs.org/`, 2016. Accessed: 2016-03-25.

[14] Troels Knak-Nielsen. Calculate All Combinations (permutations). `https://dzone.com/articles/calculate-all-combinations`, 2016. Accessed: 2016-03-25.

[15] Google Inc. AngularJS – Superheroic JavaScript MVW Framework. `https://angularjs.org/`, 2016. Accessed: 2016-03-26.

[16] Chris Pettitt. cpettitt/dagre-d3 Wiki. `https://github.com/cpettitt/dagre-d3/wiki#demos`, 2016. Accessed: 2016-03-26.

[17] Cytoscape Consortium. Cytoscape.js. `http://js.cytoscape.org/`, 2015. Accessed: 2016-03-26.

[18] Almende B.V. vis.js. `http://visjs.org/`, 2016. Accessed: 2016-03-26.

[19] Alan Williamson. jQuery Lined TextArea. `http://alan.blog-city.com/jquerylinedtextarea.htm`, 2016. Accessed: 2016-03-26.

[20] Reddit, Inc. reddit: the front page of the internet. `https://www.reddit.com/`, 2016. Accessed: 2016-03-30.

[21] Christie, T. Django REST Framework. `http://www.django-rest-framework.org/`, 2016. Accessed: 2016-03-30.

[22] Google, Inc. Google Forms. `https://forms.google.com/`, 2016. Accessed: 2016-03-30.

[23] Wikipedia. Context-free grammar. `https://en.wikipedia.org/wiki/Context-free_grammar`, 2016. Accessed: 2016-03-25.

Appendix A

Terminology

A.1 Glossary

available expression An expression is *available* if it has been computed along all paths leading to the current node. 1, 25, 69

bit-vector data-flow Data-flows in which values come as single items, and are either included or excluded in each set. Such data-flows may be represented by a vector of bits, each bit representing a value and a 0 or 1 indicating the presence or absence of that value in the set.. 8, 25, 31, 61

boundary The initial value at the starting point of an analysis, i.e. In($ENTRY$) or Out($EXIT$). 10, 24

context-free grammar A context-free grammar describes a formal language using rules of the form $A \to \alpha$, in which A is a single non-terminal symbol and α is a series of terminal or non-terminal symbols. The grammar is "context-free" in that its rules may be applied regardless of the context of the non-terminal A[23]. 27

control-flow graph A graph representing the possible execution paths in a program. Nodes represent instructions, edges represent possible jumps between said instructions. 1–3, 6, 7, 34, 35, 61, 65

data-flow A system of equations and conditions which constitute a data-flow problem, that is, a problem which may be solved through data-flow analysis. 2, 3, 7–11, 19, 23, 25, 29, 31, 32, 40, 61, 62, 65–67

data-flow analysis A technique for gathering information at various points in a control-flow graph. i, 1–3, 6, 9, 13, 17, 19, 21, 23, 27, 39, 47, 49, 51

data-flow equations A system of equations which determine how data flows through a CFG. 7, 9–11, 24

direction The direction in which data flows in a data-flow problem. Either forward (from the entry point of the CFG to the exit point) or backward (the opposite). 7, 10, 24

domain The domain of values considered in a data-flow problem, e.g. definitions or expressions. 10, 11, 24

dominate In a CFG a node n_i is said to dominate a node n_j if every path from n_0 (the entry node) to n_j must go through n_i[2, p. 478]. 66, 69

fixed-point A computation in which the required process is repeated until the state stops changing. 9, 11, 23

Hasse diagram A diagram used to represent partially ordered sets. Nodes represent elements of the sets, edges represent an ordering between a pair of elements. 3, 8, 9, 31, 32, 35

Likert scale A method of gauging opinion on a topic, consisting of a collection of Likert items. Each Likert item contains a statement followed by an odd number of responses, containing an equal number of positive and negative responses balanced such that the difference between responses is uniform. An example Likert item is the oft-used Strongly Agree / Strongly Disagree scale, which may be weighted from from 1-5. The Likert scale is the sum of weights of responses to Likert items. 15, 17, 50, 57

liveness analysis A variable is *live* if its current value will be used later in the program's execution. 1, 25, 69

local information Information specific to a given node in the CFG, e.g. in reaching definitions DefGen is the set of definitions which a given node generates.. 7

meet An equation (or set of equations) which determines how data flows *between* nodes in a CFG. 7, 11, 24

meet operator An operator which defines how the meet of two sets is obtained, such as $\cup$, $\cap$ or another operator entirely. 10, 11, 24

meet semi-lattice A partially ordered set in which there exists a greatest lower bound (or meet) for any non-empty, finite subset. 3, 7, 8, 11, 31, 32

reaching definition A definition of a variable *reaches* a block if there exists at least one path from its definition to the block along which it is not overwritten. 6–10, 25, 29, 66, 69

region A set of nodes in a graph which includes a *header* node which dominates all other nodes in the region[3, p. 669]. 10

single page app a web application in which the page is only loaded once, after which JavaScript is used to update the content and asynchronous calls are used to request more data.. 34

transfer An equation (or set of equations) which determines how data flows *through* a node in a CFG. 7, 10, 11, 24, 25

tuple-valued data-flow Data-flows in which the values come in tuples; for example, in constant propagation each variable is paired with a value indicating whether it holds a constant.. 8, 25, 61

A.2 Acronyms

API application programming interface. 3, 23, 34, 35, 37, 41, 49, 50

AST abstract syntax tree. 5, 23, 27–31

CDN content delivery network. 39

CFG control-flow graph. 6–11, 23–25, 27, 29, 31, 32, 34, 35, 40, 41, 43, 61, 65–67

COPT Compiler Optimisations. 2, 20, 21, 47, 48, 55

DRF Django REST Framework. 50

DRY Don't Repeat Yourself. 41

ILOC Intermediate Language for Optimising Compilers. 3, 23, 25, 27–31

IR intermediate representation. 5, 6

JSON JavaScript Object Notation. 50

PEG Parsing Expression Grammar. 27

UI user interface. 39, 49

UX user experience. 49

Appendix B

Types of Data-Flow Analysis

Data-Flow	**Purpose**	**Applications**
Dominators	Computes the set of nodes which dominate the current node.	Computing SSA form.
Reaching definitions	Computes the set of variable definitions which are available at points in the CFG.	Generating def-use chains for other analyses.
Liveness analysis	Computes the set of variables whose current value will be used at a later point in the control flow graph.	Register allocation. Identifying useless store operations. Identifying uninitialised variables.
Available expressions	Identifies expressions which been computed at a previous point in the CFG.	Code motion.
Anticipable Expressions	Computes expressions which will be computed along all paths leading from the current point.	Code motion.
Constant Propagation	Computes the set of variables which have a constant value based on previous assignments.	Constant propagation. Dead code elimination.
Copy Propagation	Computes the set of variables whose values have been copied from another variable.	Dead code elimination. Code motion.
Tainted Flow Analysis[1]	Identifies unsafe operations which have been passed unsanitized *(tainted)* input as a parameter.	Preventing security vulnerabilities such as SQL injection and XSS attacks.

Table B.1: Types of data-flow analysis.