Hale - Web Based Game to Teach Functional Programming

CS310 - Progress Report

Joe Moore, u1917702
Department of Computer Science
University of Warwick

WARWICK
THE UNIVERSITY OF WARWICK

Contents

1 Introduction

1.1 Motivation

With programming being such a fundamental skill in the field of Computer Science, there exist plenty of teaching tools to teach it. These tools range from beginner to advanced lessons and program. These tools are most commonly in the form of interactive websites that provide the ability for students to solve problems by typing code into one box and then seeing an output to its side. They almost always allow the student to see the correctness and less commonly but still regularly the efficacy of their solution. However there is a lack of these tools for functional programming, for example Codecademy does not currently offer any functional languages. In addition there is a lack of these tools that are engaging for young children. Most of them simply ask for the user to complete an arbitrary and often boring task, such as computing the n$^{\text{th}}$ number in a complicated mathematical sequence.

Functional programming should be taught to children from a younger age. It allows the explanation of an algorithm with fewer distracting elements. It excludes complicated syntax, order of evaluation and exceptions. However, there is a steep learning curve and Haskell alongside functional languages have a reputation in the academic community for being intellectual due to them relying heavily on abstract mathematical concepts. This is why there exists the need for a fun teaching tool aimed at making teaching dynamic and exciting for a simply functional programming language. The language will be a halfway house between Haskell and lambda calculus. Allowing syntax from either. The benefits of such are the simplicity of lambda calculus can be used to impart basic principles whilst the data structures, such as lists, and types alongside the way they are handled in Haskell are more beneficial. Functional programming has benefits and can often lead to better programming ability in the long run if learnt early [2].

1.2 Current Solutions

Codecademy is the most well known tool for teaching programming and has plenty of courses. These however are all for imperative languages. However, none are for functional programming languages. Scratch is a good example of an implementation of teaching through a game. The solutions they create can be visualized almost immediately in game form. They can take the solutions and algorithms they create and see how they directly affect the environment [3]. However Scratch makes use of a purely constructive approach. It gives way for experiments. This means that despite its strengths it has some gaping weaknesses for teaching new languages. The main weakness being that student learns through trial and error without guidance. Research shows this is more likely to lead to bad habits developing and also a hampered ability to grasp complex concepts [4].

There also exist other block based teaching methods that implement a more traditional teaching style, such as Reduct, a block based puzzle game to teach JavaScript. However studies showed this block approach (even when applied in a less creatively free setting) had serious negative side effects. The students knowledge transfer was very poor. And students struggled to write good JavaScript code once the blocks were removed [1].

1.3 Goals and Objectives

The general aim of this project is to create a tool to teach children and young adults (unfamiliar with functional programming) a pure form of functional programming. This should hopefully provide a foundation of understanding such that if they wanted to pick up a functional language, such as Haskell, as a later stage, they had the knowledge to do so, without feeling intimidated. The tool will use a game-based approach due its significant advantages particularly for younger users. Whereas learning new skills through games gathers growing enthusiasm among researchers and lecturers [5]. With child programmers who learnt the skill initially through games showing a stronger aptitude for the skill years later [6].

The problems should increase in difficulty to provide a sense of progression. For example the first problem will be to code the main characters move command. This is a simple incrementation problem, whereby the solution is to increment the character's x and y coordinates. These problems will give way to more advanced problems. It is also necessary that some of these later problems rely on initial problems knowledge, this will reinforce knowledge.

The tool will teach a custom functional programming language, Hale, I developed. It will be an extension on lambda calculus. To include types and primitives. This use of a version of lambda calculus will allow focus on teaching the key concepts of pure functional programming. Below is an example of a program written in Hale to compute the third power of a number:

$$\texttt{def}\ \ cube ::= \lambda x : \texttt{int.}\ \ x * x * x$$

The syntax will use the lambda to define the start of a function with parameters, followed by a colon type and then a full stop to signify start of the function. As seen below:

$$\texttt{def}\ \ \textit{[function name]} ::= \lambda\ \textit{[function parameters]} : \textit{[function type]}\ .\ \textit{[function body]}$$

2 Current Project State

2.1 Hale

So far plenty of work has gone into Hale and its interpreter.

Hale is based on Simply Typed Lambda Calculus. Therefore the AST tree for Hale can be based off of the syntax tree for lambda calculus. This will require some conversion to more reasonable syntax since the syntax of lambda calculus cannot be interpreted easily by an interpreter.

Lambda Calculus:	*Hale Syntax:*
$\dfrac{t_1 \to t_1'}{t_1\ t_2 \to t_1' t_2}$	t ::= x *variable*
	λx :T . t *abstraction*
	t t *application*

This syntax, implemented by Hale, is what is passed to the Python backend interpreter. this allows for easier conversion and interpretation in python. Naming context is used to map variable names bound by abstraction to their index.

$$\lambda x.\lambda y.x(yx) \to [y, x]$$

In this example the right hand side lambda calculus expression gives the following context, if a variable name was encountered outside of this context then it would be free and not bound to any of the abstractions.

2.2 Types and TypeChecking

Hale uses a function to check the type of a function before any evaluation takes place. In this way we would say Hale is *statically typed*. The check function is based off of the typing rule in lambda calculus.

$$\frac{\Gamma \vdash e_1 : (\tau_2 - \to \tau) \quad \Gamma \vdash e_2 : \tau_2}{\Gamma \vdash (e_1, e_2) : \tau}$$

This type checking is implemented by the following Python code:

```
checkType(self , e1 e2):
    if e1.type == e2:
        return f'({self.func} {self.arg})'
    else:
        message = u'Expected: {}, Found: {}'.format(e1, e2)
        super(ParserError, self).__init__(message)
        self.expected = expected
        self.found = found
```

2.3 Parsing

The code written by the user will be parsed into an AST through recursive decent parsing. The interpreter so far does successfully parse the majority of correct Hale functions. This however is still suffering from some bug issues which hope to be resolved in coming weeks.
No work has yet been started on the website or its user interface. It is the plan that such work will commence in a few weeks time.

2.4 Evaluation

The first key component in Hale evaluation is variable substitution. Take the expression,

$$(\lambda x : \texttt{int}. * x\ x)\ 2$$

this is obviously a function to square a number being applied to the value 2. However the function on the left only features the variable x, which clearly needs to be substituted for 2. However it was important to prevent any changes to meanings of functions.
In Hale the functions $\lambda x.(\lambda y.y\ x))$ and $\lambda a.(\lambda b.b\ a))$ are equivalent. Renaming x and y to a and b has no effect. From this one might assume any substitution is allowed. Any variable in any lambda term can be replaced with any other. To show this is not the case we must consider the expression:

$$(\lambda y\ .\ y\ x)$$

In this expression it is critical to observe x is *free*, or in other words not bound by a lambda abstraction. If you were to perform a substitution here to turn y to x the expression would become:

$$(\lambda x\ .\ x\ x)$$

The meaning of this is completely different since both x variables refer to the argument of the lambda abstraction. Therefore the implementation of a capture-avoiding substitution function was necessary in the interpreter:

```
CaptureAvoidingSub(x, y, function):
    if x == y:
        return function
    elif notElement(x, function.parameter) :
        return function
    elif notElement(function.parameter, y):
        function.body = CaptureAvoidingSub(c, y, function.body)
    else:
        for token in function.body:
            if token == x:
                token = y
        return function
```

2.5 Technologies and Frameworks

The frameworks and technologies used for the development thus far are:

- Ubuntu (for development)
- Python and Flask (for the back-end)
- Visual Studio Code (as a text editor)
- Git and GitHub (for version control)

Frameworks and technologies expected to be used for the continuation of the project are:

- REACT, HTML, CSS, JS (for the front-end)
- Google Chrome and Mozilla Firefox (to test the website)

3 Testing

3.1 Level difficulty

A selection of first year students will be used for some early testing of level difficulty. This will take place in January. If the progress that is expected has not been made by that time the tests will go ahead but using the interpreter and not the website. The level difficulty can still be determined even if the User interface is not up and running.

The testing will work by measuring how often students fail at each task. The solutions will also be stored and their efficacy analysed. This will allow a greater degree of precision since a more efficient solution shows a greater level of understanding. Whilst an inefficient one, despite being better than an ineffective one, shows gaps in knowledge and understanding. The primary tool to determine the efficacy is the average number of functions needed to solve a solution.

3.2 Pytest

Pytest is a testing framework which will be used to run pre-written tests and output the results. It will be how all the components of this project are tested. For example to ensure that the interpreter correctly performs reductions:

```
class Test(tdata.CheckConsistent):
def test(self, example, answer):
    new  = []
    state = parse.parse_expr(example)
    while state is not None:
        new.append(str(state))
        state = state.reduce_once({})
    assert new == answer
```

It is also possible to combine pyTest with random inputs. This is possible through validation the statistical behaviour of the function depending on the random distribution. Take the expected distribution of the functions results and compare the statistical metrics against success and fail criteria. For example are the mean, skew, etc. of the tested function matching their expected objective. This is achieved through a non frozen seed. It is important to note plenty of data needs to be collected for it to be statistically viable though.

4 Project Management

4.1 Methodologies

Overall the management of this project has thus far been successful. Progress has been made roughly in accordance. The back end is roughly working, and the interpreter is also close to completion.

The objective that the project should interpret code correctly and able to run correct code has been met. The analysis function objective to require more personalized feedback is to an extent met. This is where some changes have been made.

Instead of an algorithm to analyse code the parser and lexer will be further fleshed out. This is where specific code feedback will be assigned by returning type discrepancies and scope issues.

Manual testing has taken place throughout development so far in order to influence development of new unit tests. Once the parser and interpreter is finished the next stage is to start work on the website. Despite the original timetable there is confidence that this can be completed before week 1 term 2. Thus a more aggressive timetable has been developed.

13 December	**Christmas Break**	Finish interpreter and start front-end work
10 January	**Weeks 1-2**	Connect frontend to backend, bug test and ensure the website is fully working.
24 January	**Weeks 3-4**	Start writing the initial report. This will focus on the introduction and the details of development stages.
7 February	**Weeks 5-6**	inish writing the development sections of the report and start on the conclusion.
21 February	**Weeks 7-8**	Finish the report and analyse the success of the project from the tests and interviews.
7 March	**Weeks 9-10**	Finish and submit the final report.

It is not necessary for this to be met, since it is more ambitious. However attempting to stick to it leaves wiggle room in case of unforeseen situations down the line.

4.2 Risk Assessment

Summary	Risk Level	Mitigation	Residual Level
A third-party cloud service being used becomes unavailable	Low	Offline backups regularly made	Low
Underestimation of task complexity and scope or debugging ease	High	Internal deadlines earlier than need be to allow for leeway	Medium
Personal illness or an unforeseen personal event encountered	Low	New timetable is more ambitious but also therefore features more leeway	Low
Outage of cloud based server of choice	Medium	Website can be simulated locally and development still continue	Low
Lack of data due to small number of test subjects. Therefore inability to perform statistical analysis on data	High	Ask if message can be put out to functional programming labs	Medium

4.3 Requirements

The requirements have been greatly advanced. The progress made on this project has made it clear that more specificity in the requirements is necessary. As such here is a newly devised set of requirements.

4.3.1 Functional Requirements

1. The backend of the website should successfully interpret and run all of the code written by the user. The scope of this is defined as the program should be able to interpret:
    (a) Lambda expressions
    (b) Types:
        i. Integers

        ii. Bools
        iii. Lists
    (c) Function application
    (d) List operations
    (e) Higher Order functions
    (f) Recursion

2. Errors should be properly handled and feedback to the user. No code should be able to break the website.
3. The game should work if the user inputs correct code and the use of arrow keys etc should execute the code they haven written in order to play the game.
4. The type-level system designed will use an Embedded Domain-Specific Language in order to make it accessible. It should use the type-level model to type-check inputted code and provide feedback specifically about type issues.
5. Feedback should be given whenever:
    (a) A program runs
    (b) A program fails to run
    (c) A program runs but does not give an expected result
6. Success of a challenge should factor in:
    (a) Length of code
    (b) Program efficacy
    (c) Program result

4.3.2 Non-functional Requirements

1. The teaching tool should allow students to fulfil the *MUSIC* model in their learning.
    (a) Present the user with problems with multiple solutions so as to provide them with significant choices and place them in control of their learning.
    (b) The content should increase in difficulty with later tasks being possible but fairly tricky. This should provide a challenging learning environment and such improve knowledge whilst allowing the user to be certain they have built up the skills in previous tasks to attempt the current one.

   (c) Explanations as to the usefulness and applications of different aspects of functional programming should be explained with each task.

2. The total learning project should result in the creation of an overarching project, so as to facilitate *project based learning*.
3. Students should feel that their understanding of functional programming has improved by engaging with the teaching tool.
4. The code feedback should be helpful and understandable to students who are unfamiliar with functional programming.

5 Legal and Ethical Issues

5.1 Software Licencing

All libraries and software used thus far are free to use for development, without attribution.

5.2 Ethics

An application will need to be submitted if feedback is to be collected from students enrolled on CS141 Functional Programming. If this is successful then we will proceed. If not I will ask individuals I know to participate and get them to sign a contract indicating their consent to have their data monitored.

References

[1] I. Arawjo, C.-Y. Wang, A. C. Myers, E. Andersen, and F. Guimbretière. Teaching programming with gamified semantics. pages 4911–4923, 2017.

[2] M. M. CHAKRAVARTY and G. KELLER. The risks and benefits of teaching purely functional programming in first year. *Journal of Functional Programming*, 14:113–123, 01 2004.

[3] M. S. Gunbatar and H. Karalar. Gender differences in middle school studentsâ attitudes and self-efficacy perceptions towards mblock programming. *European Journal of Educational Research*, 7:925–933, 10 2018.

[4] F. Kalelioglu and Y. Gulbahar. The effects of teaching programming via scratch on problem solving skills: A discussion from learners' perspective. *Informatics in Education*, 13:33–50, 04 2014.

[5] M.-C. Li and C.-C. Tsai. Game-based learning in science education: A review of relevant research. *Journal of Science Education and Technology*, 22(6):877–898, 2013.

[6] M. Pivec. Editorial: Play and learn: potentials of game-based learning. *British Journal of Educational Technology*, 38(3):387–393, 2007.