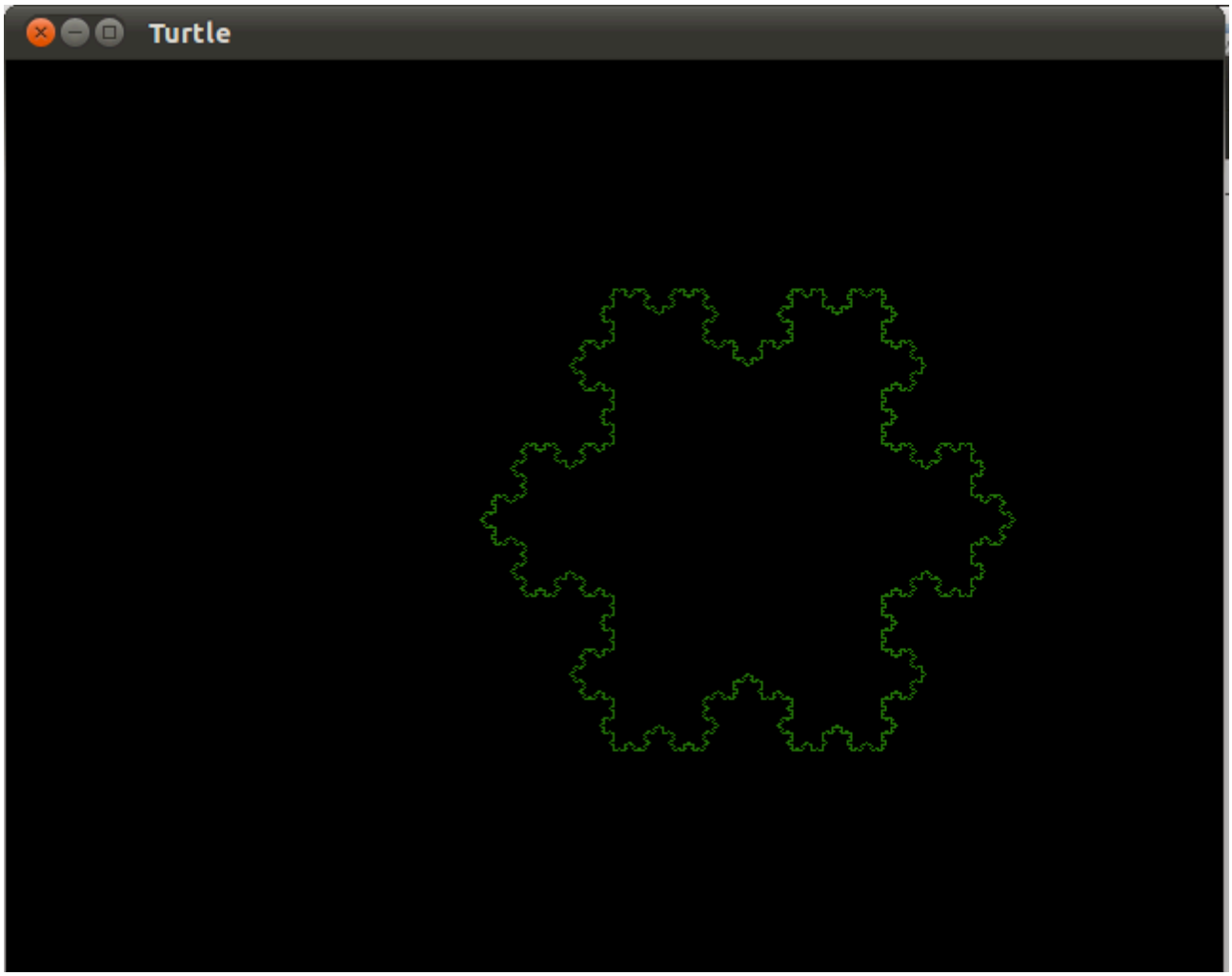
Haskell Project 2: Turtle Graphics

Table of Contents

Haskell Project 2: Turtle Graphics

In this project a simple turtle graphics language will be designed, implemented and tested. A most famous example of turtle graphics is that in Logo, a programming language which was created at MIT in 1967[i].

Sample Output



Features

The language which was designed includes the following features:

- Turtle commands to move forward/backwards, turn left/right, pen up/down and change pen color (to either a pre-defined color or a RGB-triple).
- Scripts contain one of more functions which can take arguments and contain multiple statements.
- Main function can take arguments from the command line.
- Statements can be turtle commands, function calls, repeat statements, if, or if-else.
- Because of this the language has conditionals, programmers can use recursion to simulate loops. Thus anything which can be done with loops can be done in the language.

Personal Participation

My personal participation to the project was:

- implementing, testing and commenting LangParser.hs, Lang.hs, Main.hs and Turtle.hs
  - including eval, interp, parsers to convert Strings to executable structures, adding additional methods to Turtle
  - implemented interp as an Either String Command (rather than a maybe)
- wrote Test.sc, Nsides.sc, etc.

Design

When designing a programming language there are a number of factors which need to be taken into account, including features such as conditionals, functions, how different types are stored in memory and the syntax of the language.

Language Syntax

The syntax of a programming language is the set of rules which define the combinations of symbols which are considered valid in that language. There are several important aspects which should be considered when designing a programming language. These include readability, write-ability, verifiability, translatability, and lack of ambiguity[ii]. An important aspect of both readability and write-ability is whether the language accept, and/or requires, white-space. Ideally a language is so readable that the semantics of algorithms are apparent by inspection, but support for comments can help increase readability by providing text which is ignored by the language's interpreter.

A program in the language consists of a series of function definitions which can take a number of arguments, and contains a sequence of commands.

Programs will be executed by calling their “main”-method, and thus all programs must have such a method.

```
blueCircle(x, n){

   Color (0, n, n)

   Fd x

   Rt 1

   if (n < 360) { blueCircle(x,n+1) }

}

main(){

   PenUp

   MoveTo -100 0

   PenDown
```

```
    blueCircle(2,1)

}
```

This simple script displays how several turtle-commands work, as well as functions and conditionals. The use of curly braces is similar to that of Java, function calls are done in the same way as in many scripting and programming languages and arguments can only be floats. Possible additions to this could be to be able to pass colors, booleans and functions to other functions. The language allows a sequence of statements on the same line without any delimiting characters (other than white space, which can loop a bit strange at times (see scripts/Koch1Line.sc). This would be a possible improvement.

Operators

The language supports a number of operators. All of these require two operands of the Expr type.

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
<	Less than
<=	Less than or equals
>	Greater than
>=	Greater than or equals
==	Equals

The first four operators result in Expr which can be evaluated to Floats while the last five result in Boolean values which can be used in conditional-statements.

Modular organisation

The implementation will consist of five modules:

Module	**Purpose**
Main	Contains the main method. Reads arguments from the command line, reads a script from the filename given as the first argument and executes the script with the remaining arguments as parameters passed into the script's main method.
Lang	Defines the different types of constructs of the language and how they are executed and evaluated. Includes mathematical and boolean expressions, turtle commands and function-definitions. Converts programs to commands which are executed and displayed in Turtle.
LangParser	Parser which interprets a script file and converts it a format which can then be evaluated and interpreted.
Turtle	Deals with the graphical operations and display.

Parsing	Functional parsing library[iii].

Implementation

Main

The main method gets the command line arguments using getArgs, uses the first one as the filename of the script to be parsed and parses the remaining arguments to expressions using Main.parseArgs before passing them onto the main method of the parsed script, when is then executed and displayed. Using a case-of-statement the result of interp is extracted in order to either display some graphics or report an error. The use of the Either monad is discussed in more detail in the next section.

Lang

The project specification suggests making interp return a Maybe Command, but a better way of doing it is to use a monadic instance for Either, which allows Either a Command or a String error message to be returned. It is implemented in the following way:

```
instance Monad (Either String) where

   return v = Right v

   fail s   = Left s

   x >>= f = case x of

      Left s  -> Left s

      Right v -> f v
```

This essentially means that a function is called with an “Either String a”-type as its argument it will return either the function executed on the Right value if it is of that type, and the original Left value otherwise.

The Either Monad is currently only used in interp and LangParser.parseScript, but should also be used for eval and evalCondition, in order to make them return Either String Float and Either String Bool, respectively. This would mean that error reporting would be done more elegantly.

LangParser

Parsing is done using the given parsing library. The ||| operator allows strings to be attempted parsed using several different methods. For example in parseTurtleCmd:

```
parseTurtleCmd :: Parser TurtleCmd

parseTurtleCmd = parseFd

            ||| parseBk

            ||| parseRt

            ||| parseLt
```

```
    ||| parsePenUp

    ||| parsePenDown

    ||| parseChgColor

    ||| parseMoveTo

    ||| parseJumpTo
```

It first tries to use parseFd and if that fails, it goes onto the next one, and so on. This makes parsing expressions relatively simple.

In order for the language to allow white space the parser needs to be able to tokenise scripts. This is done using the symbol-function in Parsing.hs which allows strings to be parsed, ignoring white space on either side.

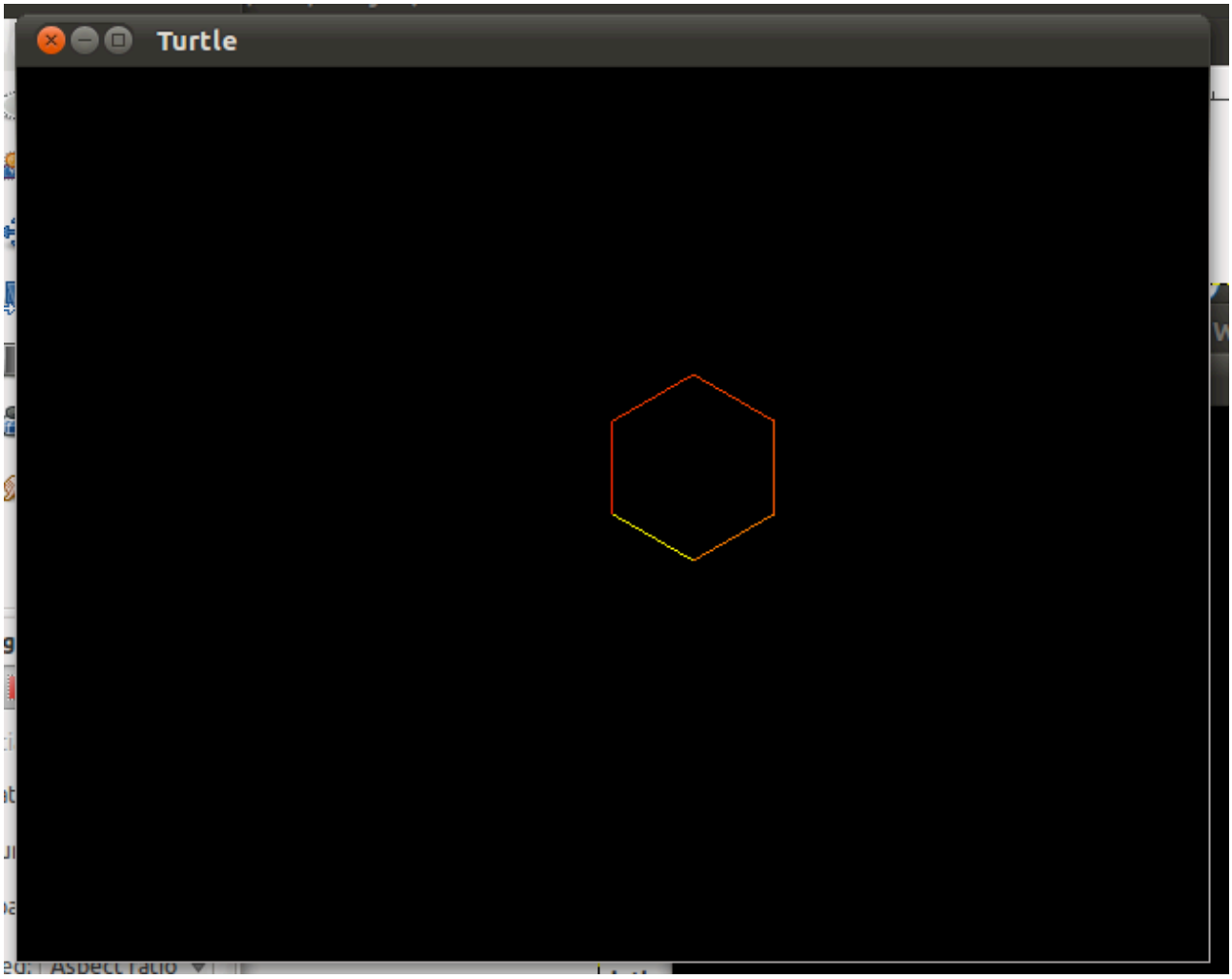
Turtle

The +> operator allows results of Command-type-methods to be combined. This is used in Lang to combine the results of a sequence of programs and to interpret repeat-statements.

Testing

A number of test scripts have been attached in the scripts-folder.

Here is the output from “NSides.sc 50 6” as an argument:



This shows that the main method read arguments from the command line, that the Color command works with self-defined colors, that methods can take arguments, and that if-statements work. Additionally it shows that division and subtraction works.

Parsing: LangParser

Test output of parsing methods can be found in TestParser.txt in the appendix-folder.

Uses

The language can be used in the real world to learn about basic programming and possibly to produce modern art.

Limitations

The language has several limitations, including the following:

- Hard to debug scripts because if a single statements in a function is invalid the parser will return an error message with the the remainder of the script, including the whole function.
- No error message is specified if too many arguments are passed into a function.
- The language is ambiguous in the way Color is parsed: “Color(r,g,b)” could be either a method or a command to change the pen color. However, the way the parser is implemented it will be interpreted as a pen color change.

- Mathematical expressions cannot contain more than one operator or brackets.
- There are no boolean operators, like not, and, or.

Potential improvements and additions

- Define eval and evalCondition using Either String in order to provide better error handling.
- Add support for multiple mathematical operators per expressions.
- Making the delimiter between statements semicolon or newline, so that programmers are forced to make the program more readable.
- Variable assignment.
- For and while loops.
- Make the Color command unambiguous by changing its syntax.
- The parser currently does not report what is wrong with
- Create a data type for all values, such as floats, booleans, colors, and possibly strings.
- Statements such as turtle-commands do not require spaces in between them, it would be preferable if, like in most programming languages, for example Python, statements have to be separated by semicolons or new-line-symbols.

Conclusion

In this project a programming language for turtle graphics was designed, implemented and tested. Features include are functions with arguments, simple math expressions, repeat-statements, and conditionals. Despite being very simple, the language is capable doing everything a language with more sophisticated features, such as assignment, loops and more complicated mathematical operators, could. However, programming is more tedious and may be less readable than it otherwise could have been with these features.

The project has given me significant insight into monads and the ease of parsing using Haskell. It has also furthered my general computer knowledge by the use of svn and other Unix commands.

Once again, working as a group proved difficult. We assigned tasks to all members, I finished the basic code for the main method (reading a script from argument) and tried to help the person who did not do anything for the previous project with the code for interp and eval, but he seemed not to have any understanding of the language. I checked up on the other group member who was meant to write the parser many times to see how he was doing and he always told me he was “working on it”, but on Thursday he asked me to write it.

Glossary

Script – A file a main function and zero or more other functions.

Program – Sometimes refers to a sequence of functions.

Function – Can take a number of arguments and contains a sequence of commands.

Command – A TurtleCommand, function call, repeat- or conditional-statement.

TurtleCommand – Can be either move forward or backwards, turn left or right, pen-up or -down, change pen color, move or jump to a specific location.

Function call – Invokes a function given zero or more arguments.

---

i http://el.media.mit.edu/logo-foundation/logo/index.html, Accessed: 12/12/10
ii http://faculty.simpson.edu/lydia.sinapova/www/cmsc315/LN315_Pratt/PPT/L03_Syntax.ppt, Accessed: 12/12/10
iii https://studres.cs.st-andrews.ac.uk/CS2006-APP/Practicals/p4/code/Parsing.hs, Accessed: 12/12/10