

Development of robot in maze web application using Scheme Programming language

Yuh Guang law

Ylaw000@cittymail.cuny.edu

Fahad Uddin

Fuddin003@citymail.cuny.edu

Abstract

This paper represents the development process of a web application of robot in a maze problem using scheme programming language. Scheme is a functional programming language, also a descent of lisp programming language. Scheme has many things common with other commonly used programming language for web application development, like javascript, python etc. Scheme is dynamically typed, features closures and allow functions as first class citizen which are very important features for web application development. There are few tools to map javascript code to scheme. Even though a little optimization can be done by Bigloo and Jsigloo, but in this project we solely tried to solve maze problem and make an web application using scheme.

1. Introduction

Scheme is a functional programming language. One of the main dialect of Lisp. Scheme was created at MIT AI lab during 1970s. it was the first dialect of Lisp to have tail call optimization and lexical scope. It was also one of the first language to support first-class continuations.

Over the years there various releases of scheme implementations, such as Racket, Chez scheme, Ikarus. Racket formerly known as PLT scheme is a general purpose, multi paradigm programming language in the Lisp-Scheme family. Racket used in a variety of contexts such as scripting, general-purpose programming, Computer science education and research. The platform provides an implementation of the Racket language along with a development environment called DrRacket written in racket itself.

Racket was designed to be used as a general purpose language in production systems. The racket distribution features an extensive library that covers systems and network programming, web development, a uniform interface to the underlying operating system, a dynamic foreign function interface, several flavors of regular expressions, lexer/ parser generators, logic, programming and a complete GUI framework.

We will be using DrRacket to develop our web application.

Scheme is not easy to compile. Several of it's properties make it challenging to generate efficient code:

- Scheme is dynamically typed,
- Functions are first class citizens,

- Variables can be captured by functions,
- It contains eval function, which allows to compile and run code at run time.

Besides it's hard compilation there are some feature scheme have, which make it a suitable programming language for web application development. Contrary to Javascript much research has been spent in compiling scheme, and there exists several efficient scheme compilers now. In this paper we would try to light on some of the feature scheme has to make it a suitable programming language for web application development.

2. Usage of Scheme in world wide web

Web pages can be made directly as programs in a functional programming language instead through HTML or Other scripting language. Scheme is an attractive direct vehicle for authoring of internet material in the sense that the source of a web page becomes a scheme program. Abstraction from details in the underlying markup language constitute the main advantage in our approach. We find it useful to have the power of a high quality programming language available for automation of routine tasks during the authoring process.

We chose scheme because it's powerful. Scheme is best regarded as a mixed paradigm programming language, with strong support of classic functional programming and plain imperative programming. However, scheme lacks many of the hallmarks of contemporary functional programming languages, most notably lazy evaluation and pattern matching. Syntactically, scheme uses parenthesized prefix notation. It turns out that the angle bracket idea of HTML/SGML and the parenthesis idea of scheme are reasonably close to each other to

provide for easy translations between the two of them. Scheme has been used by others for internet programming purposes.

```
#lang web-server/insta
(define (start request)
  (response/xexpr
    `(html
      (head ( title "this is a title"))
      (body (h1"this is a heading"))))))
```

An example of a primary web page

The example shows a simple webpage. If we type this piece of code in Dracket and press run an web browser will pop up with the title “this is a title” and with a heading “this is a heading”.

“#lang web-server/insta” tag let dracket know that this program contains web page or web application. “(define (start request))” send a request through dracket that default web browser be start running. When a web browser visits our web page url, the browser construct a request structure and send it across the network to our application. We need a function, which we’ll call start, to consume such requests and produce responses to them. One basic kind of response is to show an HTML page; this is done by the function “response/xexpr”, which takes an X-expression representing the desired HTML. An X-expresion is define as

```
(define xexpr/c
  (flat-rec-contract
    Xexpr
    (or/c string?
      (cons/c symbol? (listof xexpr))
      (cons/c symbol?
        (con/c (listof (list/c symbol? String?))
          (listof xexpr))))))
```

And the following example illustrate how natural it is to use X-expressions to represent HTML.

The first alternative in xexpr/c is string?. For example, the HTML hello is represented as “hello”. To guarantee valid HTML, strings are automatically escaped when output. Thus, the X-expression “Unfinished tag” is rendered as the HTML < b> unfinished tag, and not as unfinished tag. Similarly, “<i>finished\ntag</i>” is rendered as < i> finished tag</i>, and not as <i>finished tag</i>.

The second alternative in xexpr/c is the recursive contract (cons/c symbol? (listof xexpr)). For example, the HTML <p>This is an example</p> is represented by the X-expression ‘(p “this is an example”)

And finally, the third alternative in xexpr/c allows for parameters in HTML tags. As examples:

Past is represented by

‘(a ((href “link.html”)) “past”)

And <p> this <div class=”emph”> “another”> example.”)

It has been demonstrated that an webpage can be authored as a scheme program. The scheme program makes up the high level document source, which by means of program execution is translated to HTML. The availability of high quality abstraction mechanisms is the main asset of scheme approach. The main drawback is the splitting of a document into many relatively small strings which are passed as parameters to scheme functions. Using scheme we are approximating the potential of the forthcoming xml technology, but on a much simpler basis which can be used today. In addition, the possibility of automating routine work via integrated, programmed solutions turns out to be very useful.

3. Scheme features for web application development

Although scheme is not that much popular among web developers, but it has many features common with popular web languages. In this section we will be discussing those features.

3.1 Binding of variables

Procedure and variable bindings are the fundamental building blocks of scheme programming. Create procedures and manipulate variable can begin with the two most fundamental building blocks of scheme programs, variable references and lambda expressions and continues with descriptions of the variable binding and assignment forms such as define, let and set.

The lambda syntactic form is used to create procedures. Any operation that creates a procedure or establishes local variable bindings is ultimately defined in terms of lambda.

The variables in formals are the formal parameters of the procedure, and the sequence of expressions $\text{exp}_1 \text{exp}_2 \dots$ is its body.

Let establishes local variable bindings. Each variable var is bound to the value of the corresponding expression val . The body of the let, in which the variables are bound, is the sequence of expressions $\text{exp}_1 \text{exp}_2 \dots$.

The body of a `let` expression may begin with a sequence of definitions, which establish bindings local to the body of the `let`.

`letrec` is similar to `let` and `let*`, except that all of the expressions `val....` are within the scope of all of the variables `var ...`. `letrec` allows the definition of mutually recursive procedures.

`Set!` Assigns a new value to an existing variable. The value of the variable `var` is changed to the value of `exp`. Any subsequent reference to `var` evaluates to the new value.

3.2 Global variables

Simply spoken, the global object represents the scope holding all global variables. It is hence possible to modify global variables through an object.

Scheme has both local and global variables. In Scheme, a variable is a name which is bound to some data object. There are no type declarations for variables. The rule for evaluating symbols: a symbol evaluates to the value of the variable it names. We can bind variables using the special form `define`.

`(define symbol expression)`

Using `define` binds `symbol` to the result of evaluating `expression`. `define` is special form because the first parameter, `symbol`, is not evaluated.

3.3 Eval Function

Scheme has eval function, which allows to compile and execute code at runtime. Scheme gives the developer the choice between the null- environment, scheme-report-environment or the interaction-environment. The Null-environment and scheme-report-environment are completely independent of the running program and an expression evaluated in them will always yield the same result. The optional interaction-environment however allows to interact with the running program. The visibility of this environment is usually restricted to the top-level of the running program, and it is certainly independent from the location where eval is executed.

The eval function takes a representation of an expression or definition and evaluates it:

```
>(eval '( + 1 2))
3
```

The power of eval is that an expression can be constructed dynamically:

```
>(define (eval-formula formula)
  (eval (let ((x 2) (y 3))
    formula)))
>(eval-formula '(+ x y))
5
>(eval-formula '(+ (* x y) y))
9
```

Also, eval is often used directly or indirectly on whole modules. For example, a program might load a module on demand using dynamic-require, which is essentially a wrapper around eval to dynamically load the module code.

3.4 Object System

There are two kind of object models, the closed model or the open model. The closed model comes from statically type systems like C++, java. This model is characterized by the following, classes are encapsulation units, methods must be known at class-definition time, there's only one receiver for a message. This model has good compilation properties since most types are known at compile time.

The open model is more dynamic. It acknowledges the fact that all behaviors cannot be expressed at class definition time, that is should be possible to add new behaviors to existing instances, that is desirable and useful for serialization, debug and all sorts of walkers, that some way to customize or extend the implementation should also exist as well.

A define-class form creates a new class. The first parameter is the name of the class, the second is the name of the super-class, specifications of fields appear as third parameter, additional parameters may appear after. Except initializes that are evaluated, no parameters or expressions within parameters are evaluated.

A class is made of fields which may be regular or indexed. A field may be mutable or not (mutable by default). A default initialize may be specified (a think for a regular field, a unary function for an indexed field). The form that define these initializes are evaluated at class-definition time. The prototype class option forbids instantiation of the class.

When a class is defined, some functions are created: a predicate and some read/write assessors.

The define-generic form defines generic functions. When a generic function is defined, it does not possess any method. However it may have a default body which is invoked when no specific method is found. The generic function must specify the discriminating variables. Syntactically a discriminating variable appears within parentheses. If a class is associated to a discriminating variable, it restricts the set of methods that may be adjoined to this generic function. A generic function may have more than one discriminating variable.

The define-method form adjoins a method to a generic functions. Methods may be adjoined to any generic functions provided that the method has a compatible signature is the same number of discriminating variables at the same position associated to classes that are subclasses of the classes mentioned in the generic function definition. If the generic function has a final dotted variable, then methods should also have such a final dotted variable.

4. Maze problem

Before solving any problem, the main requirement is to understand that problem. In this section we are going to explain a robot in a maze problem.

A robot is asked to navigate a maze. It is placed at a certain position in the maze which can be named the starting position and asked to try to reach another position which we can call the goal position. Positions in the maze can be opened or blocked by an obstacle. Positions are identified by (x, y) coordinates.

At any given moment, the robot can only move 1 step to any one of the four directions. Valid moves are:

Go North (x, y) -> (x, y-1)

Go East: (x, y) -> (x+1, y)

Go South: (x, y) -> (x, +1)

Go Wes: (x, y) -> (x-1, y)

Positions are specified in zero based coordinates.

The robot can only move to positions without obstacles and must stay within the maze. The robot should search for a path from the starting positions to the goal position until it finds one or until it finds one.

O make this problem more solid, we can consider a maze represented by a matrix of characters. An example 6*6 input maze is

```
S#####
.....#
#.####
#.####
...#.G
##...#
```

‘.’ where he robot can move

‘#’ obstacles

‘S’ start position

‘G’ Goal

5. Applying Scheme to Maze on the Web

In our program, thanks to the web application feature in DrRacket, we were able to combine the power of both html and Scheme. To implement the features of web server in DrRacket, we will use `#lang racket`, instead of `#lang web-server/insta`. But the choices not too important. With:

```
#lang racket

(require web-server/servlet
         web-server/servlet-env
         web-server/formlets)
```

to precede the rest of the code, we allow features of a basic web-server such as form parsing and more importantly, the ability to serve our webpage to the public over the internet, instead of merely hosting it on a local computer, unless privacy is a main concern. Our program ended with:

```
(serve/servlet start
  #:listen-ip #f
  #:servlet-path "/standalone.rkt"
  #:extra-files-paths (list (build-path (current-directory))))
```

Where `#:listen-ip #f` opens up the ability to host our server online. While the other two lines made serving image files possible.

The specifications of our maze can be seen briefly as a table of m times n cells which make up the maze puzzle. Each cell is filled with an image, leaving no spacing with other images from adjacent cells. The point is, we move/replace these images in the game, according to the user's choice, which is done by the submit buttons in the (form) html tag, implemented by formlets in Racket.

The image files define the walls of the maze. There are basically 4 sets of these image files needed in this program. One set with just the walls, another with the robot icon in each of the previous set. Like the ones with the robot icon in them, the other two sets have either the target icon or the target-found (robottarget prefix) icons instead of the robot icon. The image files are named in a special naming format for convenient manipulation with strings when we replace one image file from one set with another image file from a different set.

The perimeter wall of the maze looks thinner than the interior walls. The reason for that is to make the coding easier. The interior walls' thickness is due to the adjacent cells requiring the same wall requirement on the side where these cells touch. This is done so in order for the robot to detect walls locally, instead of having to write extended and more complex coding to detect walls around it from adjacent cells, which is not the goal of this project.

In the program, we defined ORIGINAL-MAZE as the default maze map, which the user can reset to at anytime. It is possible, but we do not implement a way to generate new maze map here as that is beyond the scope of our purpose in this project. Some of the powerful features of Scheme we have applied in this program is the (map) function which generates the rows and columns of the table in html most elegantly. The recursive methods of manipulating the table

cells to make the game so interactive graphically with the help of html are another caviars Scheme has to offer in this.

This simple illustration of the Scheme/functional programming language shows us just how powerful Scheme can be in various areas such as the web and games.

6. Conclusion

What we presented in this paper is small solution of much bigger problem. This paper presents the process to make an web application of robot in maze problem using scheme programming language, where the robot explore and exploit the maze. Different examples have been plotted to provide in to behavior of the algorithm for different scenarios. Also this examples have been useful in providing important information about the optimal number of the agents in a group required for solving the maze problem. The scalability of the algorithm has been shown for different directions in the maze.

In this project we have learned about how to solve a robot in a maze problem. We also learned about usefulness of scheme programming language in web application development.

7. Future work

This project is not done here. There are plenty of scope to work on that problem. Using scheme programming to develop an web application is just the beginning work.

While standard maze solving algorithms by a robot involve a graph traversal approach, the performance of such algorithms by a robot was found to be lacking in comparison to a learning algorithm by a robot. It was observed that improving a particular skill has direct impact on the ability to perform other tasks that make use of the same skill set by a robot .

There is still a lot more work to be done in this field of work and within this problem in general. One area would be to benchmark other tasks that make use of learned skill sets by a robot.

A robot to navigate a maze is an important subject for autonomous robot and route optimization. An example application may be finding the optimized route in an urban area during a disaster. Teaching the robot to navigate through an unknown environment and find the optimal route is a very difficult task. A simplified version of this problem can be simulated by using a random 2d synthetic maze.

For more, a comprehensive analysis of performance will be performed for the number of clusters relative to the size of the maze. Also, using clustering in combination with additional external inputs may improve results further. Detailed analysis of the effects of different parameters may also offer a better understanding of what information will help to improve the robot's performance.

8. Acknowledgement

Developers of this project and authors of this paper Yuh Guang law and Fahad Uddin

Would like to acknowledge Professor Douglas Troeger for his guidance and oversight in this work.

9. References

- [1] Kurt Nørmark. "Programming World Wide Web pages in scheme". Aalborg University. December 12 1999
- [2] Chris Pearce, Yu Bai, Pat Langley and Charlotte Worsfold. "Problem Solving through Successive Decomposition". Department of Computer Science, University of Auckland. January 1, 2015
- [3] Florian Loitsch. "Javascript to Scheme Compilation". Inria Sophia Antipolis.
- [4] <https://en.wikipedia.org/wiki/Racket>
- [5] <https://en.wikipedia.org/wiki/Scheme>
- [6] <https://www.scheme.com/>
- [7] <http://docs.racket-lang.org/continue/>