



SAN JOSÉ STATE UNIVERSITY

CS252 Project Report Firefox Add-On for Scheme Language

Under the Guidance of
Dr. Thomas Austin

Saurabh Borwankar
San José State University
saurabhshridhar.borwankar@sjsu.edu

May 16, 2016

Contents

1	Introduction	3
2	Problem Statement	3
3	Scheme Programming Language	3
3.1	Syntax of Scheme	4
3.2	Examples	4
4	PEG.js	5
5	Firefox Addon	6
6	System Architecture	8
7	Result	9
7.1	Input Web Page	9
7.2	Output	9
7.2.1	Console Output	10
7.2.2	Web Page	10
8	Conclusion	11
9	Future work	11

List of Figures

1	<i>Parse Tree</i>	6
2	<i>Simple Add-On</i>	7
3	<i>System Architecture</i>	8
4	<i>Console Output</i>	10
5	<i>Modified Web Page</i>	10

Abstract

Scheme is a functional programming language. To use it with the web page it needs to get executed on the occurrence of script tag which contains Scheme code. By default JavaScript code gets executed by the SpiderMonkey engine and it is not designed to run the Scheme language script. This project illustrates the idea of how we can achieve this using JavaScript.

1 Introduction

All HTML pages are static. JavaScript makes the web pages more interactive and dynamic. JavaScript provides various functionalities in a web browser at the client side and thus reduces the workload of the server. JavaScript can be used for various applications like simple form validation, adding and creating visual effects. In the browser, JavaScript code gets executed by JavaScript engine. Each web browser has its own JS engine implementation. Google Chrome uses V8 engine whereas Microsoft's Internet Explorer uses Chakra. Mozilla Firefox has its own JS engine named SpiderMonkey. It provides support for JavaScript in Mozilla's Firefox web browser. It can be embed in other applications where host environment for JS is needed. SpiderMonkey is written C/C++ by Brendan Eich and it contains JIT interpreter and garbage collector.

To use any other language than the JavaScript in a web page, we have to convert that language in a way which JavaScript engine will understand or we have to implement evaluator engine similar to JavaScript engine for that particular language and then use that engine within a web browser for execution. This project focuses on an idea of transforming the source code for JavaScript engine. To do this, we need parser and interpreter. The parser will parse the source to build abstract syntax tree(AST) and then the interpreter will evaluate that AST. Also, we have to monitor web pages for detecting that language.

2 Problem Statement

To demonstrate the idea of using different language inside the web browser; this project is considering Scheme programming language and it will use Firefox addon to inspect web pages.

3 Scheme Programming Language

The scheme is a pure functional language which is derived from LISP programming language. It was primarily intended for research and teaching purpose. Few of the advantages of Scheme are listed below:

- Pure Functional Language
- Language of Parenthesis

- Functions are first class data types
- Compared to other programming languages, it has little syntax
- Easy to learn

3.1 Syntax of Scheme

Scheme program comprised of keywords, variables, structured form, constant data and white spaces. Keywords, variables and symbols are identified as “identifier”. An identifier can be made of letters, digits and some special characters[1]. Grammar for scheme program can be constructed as follows:

1. Programs: A program can be defined as sequence of definitions and expressions.

$$\begin{aligned} \langle program \rangle &\rightarrow \langle form \rangle^* \\ \langle form \rangle &\rightarrow \langle definition \rangle \mid \langle expression \rangle \end{aligned}$$

2. Definitions: Variable definition, begin construct and other definitions. Begin construct will have series of definitions.

$$\begin{aligned} \langle definition \rangle &\rightarrow \langle variable \ definition \rangle \\ &\mid \langle syntax \ definition \rangle \\ &\mid (begin \ \langle definition \rangle^*) \end{aligned}$$

$$\begin{aligned} \langle variable \ definition \rangle &\rightarrow (define \ \langle variable \rangle \ \langle expression \rangle) \\ \langle variable \rangle &\rightarrow \langle identifier \rangle \end{aligned}$$

3. Expressions: Expressions will have variables, constants, applications. Also it can be quote, lambda, if and !set expressions.

$$\begin{aligned} \langle expression \rangle &\rightarrow \begin{array}{l} \langle constant \rangle \\ \mid \langle variable \rangle \\ \mid (\lambda \ \langle formals \rangle \ \langle body \rangle) \\ \mid (if \ \langle expression \rangle \ \langle expression \rangle \ \langle expression \rangle) \\ \mid (if \ \langle expression \rangle \ \langle expression \rangle) \\ \mid (set! \ \langle variable \rangle \ \langle expression \rangle) \\ \mid \langle application \rangle \end{array} \end{aligned}$$

3.2 Examples

Examples of Scheme program:

1. Addition of two numbers

(+ 4 5) -> 9

2. Variable declaration

```
( define x 6 ) -> 0
```

This statement defines a variable "x" and binds it with value 6.

3. Variable value modification

```
( !set x 4 ) -> 0
```

This statement modifies the value of previously declared variable "x" and changes it to 4.

4. Function Definition

```
(define square (lambda (x) (* x x)))
```

This statement defines a simple function to calculate square of a number.

5. Sequential Evaluation

```
(begin (define square (lambda (x) (* x x))) (square 4) ( + 4 5 ) )
```

"begin" construct is used to sequentially evaluate all the statements between the parenthesis. It will first define a function square, then it will calculate square of 4 and then it will perform the addition.

4 PEG.js

The first step in executing any code is to break that code into pieces of tokens and analyse them. Parser comes to help in this process. The parser will take input a string and analyse it to build abstract syntax tree. Abstract syntax tree will represent the structure of source code. PEG.js is a simple parser generator for JavaScript [2]. It is used to build a fast and simple parser. PEG.js combines both syntactical and lexical analysis.

PEG.js will generate a parser based on a provided grammar. Also, we can specify actions to take if any rule matches. To build a parser, buildParser method is used. It takes an input your grammar and returns parser object. After building a parser is done, we can use parse method from PEG.js to build an AST. This method will take input a string and it returns an AST for that string.

Consider the following example:

```
Multiply : left:Num "*" right:Num { return (left*right) }
```

```
Num: n:[0-9]+ { return Number(n.join('')) }
```

```
Parser.parse(2*3) -> 6
```

In the above example, we are defining multiplication rule. This will accept a number followed by multiplication symbol (*) followed by another number. We are declaring a action for this rule in {}. If the match is found, parser will return the product of two numbers. In the second line, we are defining rule to recognize number. By default, parser will return single digit. So to avoid this, we need to form a whole number. This is done by using join method which will return combined digits. Last line represents the use of parser.

Following figure shows the parsed output of the simple scheme code. Parser is defined to parse simple string which represents a multiplication of two

```
(define square (lambda (x) (* x x)))
```

Input parsed successfully.

Output

```
[
  "define",
  "square",
  [
    "lambda",
    [
      "x"
    ],
    [
      "*",
      "x",
      "x"
    ]
  ]
]
```

Figure 1: *Parse Tree*

number.

This project takes an advantage of this library to build the parser for scheme language. It will a source code of Scheme language and based on the code; it will return an AST. Grammar is built using the rules mentioned in the above section.

5 Firefox Addon

Firefox provides a software development kit(SDK) to create a Firefox Add-Ons. These add-ons can be implemented with the use of various standard web technologies. We can use JavaScript, HTML and CSS for developing an

add-on. SDK provided by Firefox contains different APIs which help developers to create, test and run their add-on.

jpm tool is used for testing, running and packaging purpose. jpm can be installed using node package manager.

1. Install jpm tool
2. Intialize the default add-on template
3. Modify an index.js
4. Run and test it.

This code snippet[3] shows the implementation of a simple add-on. On click, this add-on will open a molliza.org in the tab. It defines a button and gives attributes to it like label,id and icon image. It also defines the onClick action which will call handleClick function. handleClick function will do the task of opening a provided url in the tab

```
var buttons = require('sdk/ui/button/action');
var tabs = require('sdk/tabs');

var button = buttons.ActionButton({
  id: "mozilla-link",
  label: "Visit Mozilla",
  icon: {
    "16": "./icon-16.png",
    "32": "./icon-32.png",
    "64": "./icon-64.png"
  },
  onClick: handleClick
});

function handleClick(state) {
  tabs.open("http://www.mozilla.org/");
}
```

Figure 2: *Simple Add-On*

This project implements a firefox add-on which will listen for every page load. To do this, we need to write content scripts which will manipulate the web contents. These scripts are injected into web pages using page-mod and panel APIs of the SDK. Content script file will contain parser and evaluate function which will execute scheme code.

6 System Architecture

Here add-on continuously monitors the every web page which gets load into Firefox tab. After successfully page load is done, add-on injects content script file to the web page. This script does following functions:

1. Get the source code of the page
2. Scan it for the script tag which will contain scheme code
3. For every tag, do:
 - Parse the code present between `<script>` and `</script>` tag.
 - Evaluate the generated AST
 - Writes the result back to web page

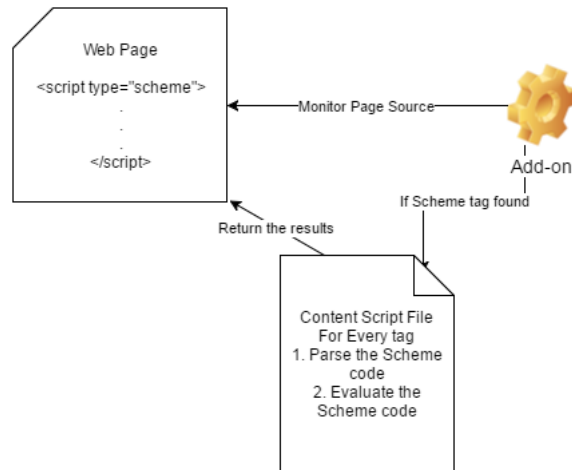


Figure 3: *System Architecture*

7 Result

7.1 Input Web Page

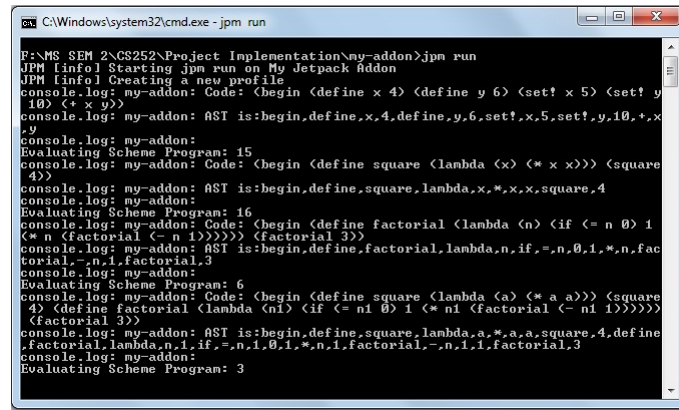
Following snippet is the sample input web page. It contains 4 scheme tags which will get executed.

```
<!DOCTYPE html>
<!--
To change this license header, choose License Headers in Project Properties.
To change this template file, choose Tools | Templates
and open the template in the editor.
-->
<html>
  <head>
    <title>Scheme Script Testing</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <div>Evaluate Scheme Script embedded in script tag</div>
    <script type="text/scheme" > (begin (define x 4) (define y 6)
      (set! x 5) (set! y 10) (+ x y)) </script>
    <script type="text/scheme" > (begin (define square (lambda (x) (* x x)))
      (square 4)) </script>
    <script type="text/scheme" > (begin (define factorial (lambda (n) (if (= n 0) 1
      (* n (factorial (- n 1)))))) (factorial 3)) </script>
    <script type="text/scheme" > (begin (define square (lambda (a) (* a a)))
      (square 4)
      (define factorial (lambda (n1) (if (= n1 0) 1 (* n1 (factorial (- n1 1))))))
      (factorial 3)) </script>
  </body>
</html>
```

7.2 Output

Console outputs shows the parsed tree of a scheme code and result after evaluation of that tree.

7.2.1 Console Output



```
C:\Windows\system32\cmd.exe - jpm run
F:\MS SEM 2\CS252\Project Implementation\my-addon>jpm run
JPM [info] Starting jpm run on My Jetpack Addon
JPM [info] Creating a new profile
console.log: my-addon: Code: <begin <define x 4> <define y 6> <set! x 5> <set! y 10> <+ x y>
console.log: my-addon: AST is:begin,define,x,4,define,y,6,set!,x,5,set!,y,10,+,x,y
console.log: my-addon:
Evaluating Scheme Program: 15
console.log: my-addon: Code: <begin <define square <lambda <x> <+ x x>>> <square 4>
console.log: my-addon: AST is:begin,define,square,lambda,x,+,x,x,square,4
console.log: my-addon:
Evaluating Scheme Program: 16
console.log: my-addon: Code: <begin <define factorial <lambda <n> <if <= n 0> 1 <+ n <factorial <- n 1>>>>> <factorial 3>
console.log: my-addon: AST is:begin,define,lambda,n,if,=,n,0,1,+,n,factorial,-,n,1,factorial,3
console.log: my-addon:
Evaluating Scheme Program: 6
console.log: my-addon: Code: <begin <define square <lambda <a> <+ a a>>> <square 4> <define factorial <lambda <n!> <if <= n! 0> 1 <+ n! <factorial <- n! 1>>>>> <factorial 3>
console.log: my-addon: AST is:begin,define,square,lambda,a,+,a,a,square,4,define,factorial,lambda,n!,if,=,n!,0,1,+,n!,factorial,-,n!,1,factorial,3
console.log: my-addon:
Evaluating Scheme Program: 3
```

Figure 4: Console Output

7.2.2 Web Page

After executing all the scripts, Add-On is writing results back to web page.



Figure 5: Modified Web Page

8 Conclusion

This add-on successfully provides the support for scheme in web pages. It extracts, transforms and evaluates the scheme code. It considers basic arithmetic operations, conditional structure, list operations, functions and lambda expression for evaluation. With this concept and add-on, we can provide support for user-defined languages.

9 Future work

Currently, this add-on provides support for scheme language with limited functionalities but we can extend interpreter implemented in this add-on to provide support for all the functionalities of scheme. Also, we can implement parser and interpreter to provide support for other languages also.

References

- [1] R Kent Dybvig. *The Scheme Programming Language Fourth Edition*. The MIT Press, 2009.
- [2] David Majda. Peg.js documentation. <http://pegjs.org/documentation>. Last Accessed: 16th May 2016.
- [3] wbamberg Erik Vold kosmodrey aviav. Firefox add on documentation and tutorials. <https://developer.mozilla.org/en-US/Add-ons/SDK/Tutorials>. Last Accessed: 14th May 2016, Last updated by: wbamberg, Jan 15, 2016.