Lisp -> Ruby

**PROJECT BY: RUCHIR PATEL. F002BZ1**

Submission date June 4, 2016

**Task:** Convert a "defun lisp" into "def ruby".

**Description:** Run a ruby.rb file, input a valid "defun lisp" function, (lexically binded), and file will output the corresponding ruby function which takes the same amount of parameters and does the same thing as lisp function.

**Language:** ruby

**Partners:** none. Would prefer to work alone on this project.

**Code:** <https://github.com/ruchir594/ruby2lisp>

Please ignore the files with “\_wc.rb”. They have not been updated to support latest functionalities.

**Test Cases:** <https://github.com/ruchir594/ruby2lisp/blob/master/SupportingDoc.txt>

**Concept:** The idea is to write a language translator which will take as input, a valid LISP Defun, and output a Ruby Def Function which will have the same functionalities. Consider following for example. If you write a defun to find the maximum of a list,

(defun maxall (x)

(if (null (cdr x)) (car x)

(if (> (car x) (maxall (cdr x))) (car x) (maxall (cdr x)))))

maxall

(maxall '(1 2 3 4 5 2 3))

5

and input the function *“(defun maxall (x)(if (null (cdr x)) (car x) (if (> (car x) (maxall (cdr x))) (car x) (maxall (cdr x)))))”* to the program run by file “**rb2l.rb**”, it will output the following Ruby code.

def maxall(x )

if [] == x[1..-1]

return x[0]

else

if x[0] > maxall(x[1..-1])

return x[0]

else

maxall(x[1..-1])

end

end

end

Which is a valid Ruby function and performs as expected.

ruchirs-air:~ atulpatel65$ irb

2.1.2 :001 > require './Desktop/BigGreen/S16/CS59/ruby2lisp/ruby\_is.rb'

 => true

2.1.2 :002 > maxall([1,2,1,4,2])

 => 4

2.1.2 :003 > maxall([1,2,1,2])

 => 2

2.1.2 :004 > maxall([1,2,1,2,5])

 => 5

2.1.2 :005 > maxall([1,2,8,2,5])

 => 8

So I wrote a ruby program to achieve this. This program has four ruby files

1. rb2l.rb
2. rb2l\_header.rb
3. simhead.rb
4. condhand.rb

**rb2l.rb** is the only file user will interact with. All you have to do is run the file. The program will ask you to input a Valid Lisp Defun. And it will parse it to output the valid ruby function in **ruby\_is.rb** which can be directly run.

**Approach:**

**Step 1:** The place of the name of the function and the list of all parameters is known. It follows right after “defun” word. So we got that statically.

**Step 2:** Once we have this, we can move towards the body of the function. Now, clearly, lisp is very well structured in the sense that every thing is in these parenthesis ‘(‘ and ‘)’  
  
I extract these string between parenthesis. I also extract all the words in the string.

For example,

a string between parenthesis may be “(maxall (cdr x))” <- Parameter\_2

The corresponding words will be [“maxall”, “cdr”, “x”] <- Parameter\_1

They are printed on the terminal under “parameter\_1” and “parameter\_2” tag.

Let us call “parameter\_1” Build Blocks and;

Let us call “parameter\_2” All Blocks for the rest of this document.

**Step 3:** Recursive parsing of these Build Blocks and All Blocks

My code reads the Build Block to understand if there is an “if” statement or a switch-case like structure using “cond” in lisp.

If there is an “if” structure, it needs following things.

1. What type of comparison <condition statement> it is?   
     
   if (null (cdr x)) is type of compression where there are only two blocks. Is (cdr x) null?  
     
   if (> (car x) (maxall (cdr x))) is a type where there are three blocks.   
   the > sign  
   (car x)  
   recursive call (maxall (cdr x)).   
     
   It does identify these different type using the build blocks.
2. Is there an Else statement  
     
   This is done by measuring the length of <condition statement> and <if true>. If the length of sum of these two +1 (for “if” keyword in defun) is less than the entire length of build block, then there is <else true> statement which needs to be parsed.
3. Is there an else-if ladder  
     
   Step 2 is done recursively. And hence this functionality is added.

If there is “cond” structure, it does following things.

1. Get all the switch cases
2. Get all the corresponding action of those cases.  
     
   For example,   
   (cond ((null x) 0);

((atom x) 1);  
)

There are cases of (null x) and (atom x). Corresponding action is to return 0 and 1 respectively.

**Step 4:** Language Translation

Once I know what are those Build Blocks and All Blocks and where they fit using Step 3, I have to translate them into corresponding ruby code.

Here are a few quick and easy examples.

|  |  |
| --- | --- |
| LISP | Ruby |
| (car x) | x[0] |
| (cdr x) | x[1..-1] |
| null | nil |
| (atom x) | x.length == 1 |

Please look at the “**beautify**” function in **condhead.rb** for more information.

**Step 3 and Step 4 work simultaneously to produce the final result visible at “ruby\_is.rb” in the same directory.**