Problem Set 6
COMP301 Fall 2019
14.11.2019 17:30 - 18:45

Read me first! Please download the *Codes* file. In the scheme codes, you will see some hints regarding where to modify. You will use DrRacket. We have also edited the `tests.rkt` for each of them so that if you solve the problem, running `tests.rkt` should have no errors. You can also write your own program in `let`, `letrec` or `proc` by writing your code in a string and run it from the console of the respective `tests.rkt` like this:

```
(display (run ''your code here''))
```

Make sure you run the file itself to update the definitions before running your code from console. Regarding problem 3, just write the answer to a text file and save itr. As for your submission, you will submit your modified code and the text file for problem 3 in a zip folder.

Problem 1[1]: Extend the `let` language so that it can use `cond` expression. Use the grammar below:

$$\mathit{Expression} ::= \texttt{cond}\ \{\mathit{Expression} \texttt{==>}\ \mathit{Expression}\}^*\ \texttt{end}$$

In this expression, tests, which are the expressions on the left-hand sides of the ==>'s, are evaluated in order until one of them returns a true value. When one of the tests returns true, the value of the entire expression is the value of the corresponding right-hand expression. If none of the tests succeeds, the expression should report an error.

Problem 2[2]: Extend the `proc` language so that it includes procedures with multiple arguments and calls with multiple operands, as suggested by the grammar below:

$$\mathit{Expression} ::= \texttt{proc}\ (\{\mathit{Identifier}\}^{*(,)})\ \mathit{Expression}$$

$$\mathit{Expression} ::= (\mathit{Expression}\ \{\mathit{Expression}\}^*)$$

Continue to the next page for 2 more questions.

[1] EOPL p.74 Exercise 3.12

[2] EOPL p.80-81 Exercise 3.21

Problem 3[3]: The tricks of the previous exercises in the book can be generalized to show that we can define any recursive procedure in `proc` language, though it is a bit trickier than it would have been in `letrec`. Consider the following bit of code:

```
let makerec = proc (f)
   let d = proc (x)
      proc (z) ((f (x x)) z)
   in proc (n) ((f (d d)) n)
in let maketimes4 = proc (f) proc (x)
                    if zero?(x) then 0 else -((f -(x,1)), -4)
   in let times4 = (makerec maketimes4)
      in (times4 3)
```

Show that this returns 12. Write your answer in `problem3.txt`.

Problem 4[4]: Implement `cond` from Problem 1 for the lexical addressing language (the language in the folder `problem4-lexaddr-lang`). You will almost do the same modifications you did for Problem 1, but for this one you will also have to modify the code in `translator.rkt`.

[3]EOPL p.81 Exercise 3.25

[4]EOPL p.101 Exercise 3.38