COMP301 PROJECT 3

Ahmet Uyar
ID: 72847

Part A)

(1) Create an initial environment that contains 3 different variables (x, y, and z)

It is defined in **environments.rkt** and i, v, x are initialized as 4, 2, 3. We must add y and z since the question wants x, y, and z.

```
(define init-env
  (lambda ()
    (extend-env
     'i (num-val 1)
     (extend-env
      'v (num-val 5)
      (extend-env
       'x (num-val 10)
       (extend-env
        'y (num-val 15) ; added for part A
        (extend-env
         'z (num-val 20) ; added for part A
        (empty-env)))))))))
```

(init-env) = [i=1, v=5, x=10, y=15, z=20, empty-env]

(2) Using the environment abbreviation shown in the lectures, write how the environment changes at each variable addition.

[] p0 : after empty-env
[z=20]p1: after (extend-env ’z (num-val 3) (empty-env))))))
[y=15, z=20]p2: after (extend-env ’y (num-val 2) (extend-env ’z (num-val 3) (empty-env))))))
[x=10, y=15, z=20]p3: after (extend-env ’x (num-val 4) (extend-env ’y (num-val 2) (extend-env ’z (num-val 3) (empty-env))))))
[v=5, x=10, y=15, z=20]p4: after adding v
[i=1, v=5, x=10, y=15, z=20]p5: after adding i

Part B) Specify expressed and denoted values for MyProc language.

Expressed and denoted values are the same in the MyProc language.

ExpVal = Int + Bool + Proc + List<Int>
DenVal = ExpVal

Part C)

I have added all the components needed and all tests pass.

Part D) Discussion

To implement stacks natively in MyProc without relying on delegating operations to Scheme, we would need to focus on enhancing the language's capabilities to manage memory and manipulate data structures directly. This involves introducing features such as dynamic memory allocation, mutable data structures, and primitive operations for stack manipulation. By enabling MyProc with dynamic memory allocation, it can allocate memory as needed to store stack elements dynamically. Mutable data structures, like arrays or linked lists, are essential for representing and modifying the stack's contents efficiently within the language itself. Primitive operations such as push, pop, peek, and merging would directly manipulate these data structures without relying on external languages or low-level constructs. Additionally, incorporating resource management mechanisms within MyProc ensures efficient use of memory and prevents resource leaks. By enhancing MyProc's capabilities in these aspects, it becomes self-sufficient in implementing stacks, providing a more native and integrated approach to stack manipulation within the language.

Furthermore, To implement stacks natively in MyProc while allowing low-level memory access, we can introduce pointers as a fundamental feature. With pointers, MyProc gains the ability to directly manipulate memory addresses, enabling more efficient implementation of data structures like stacks. By incorporating pointers into MyProc, we can manage memory dynamically, allocate and deallocate memory blocks as needed, and implement stack operations directly at the memory level. This approach allows for fine-grained control over memory usage and enables MyProc to efficiently manage stack data without relying on external languages or constructs. Additionally, with pointers, MyProc can implement stack operations such as push, pop, peek, and merging more directly and efficiently, optimizing performance and resource utilization.

So we need:

- Memory Management

- Pointer Operations
- Mutable Data Structures
- Primitive Operations
- Resource Management