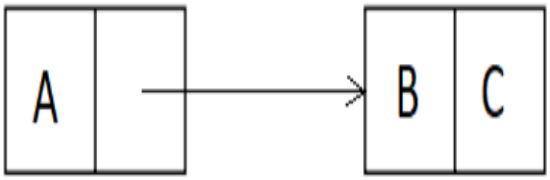
3.1

a)



b)

File Edit View Language Racket Insert Scripts Tabs Help

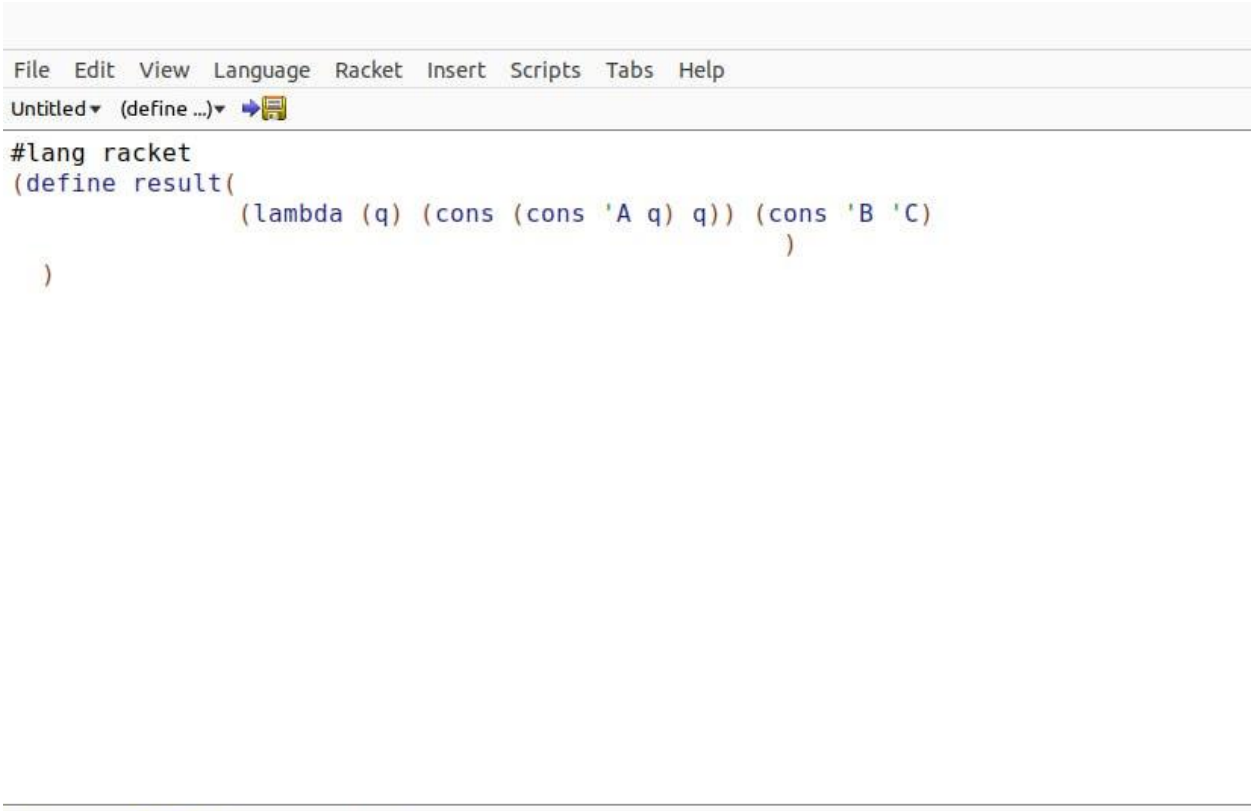
Untitled (define ...)

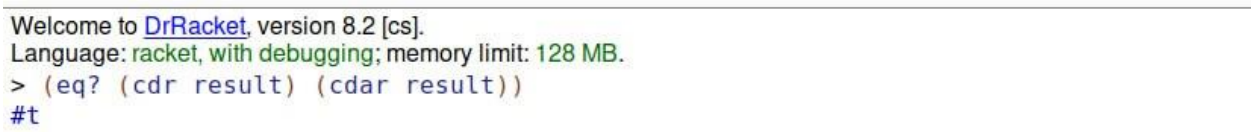
```
#lang racket
(define result
  (cons
 (cons 'A (cons 'B 'C))
 (cons 'B 'C)
 ))
```

Welcome to DrRacket, version 8.2 [cs].
Language: racket, with debugging; memory limit: 128 MB.

```
> (eq? (cdr result) (cdar result))
#f
```

c)





3.2

No, we cannot, because the halting problem is non-deterministic. Therefore, we cannot present an algorithm to determine whether 'p' for a given condition is defined or not.

b)

We can concurrently evaluate 'p' for conditions and 'e' as well. The first 'p' that evaluates to true will be selected, and the corresponding 'e' will be the output.

c)

(defun odd (x)
(cond ((eq x 0) nil)
((eq x 1) t)
((> x 0) (odd (- x 2)))
((< x 0) (odd (+ x 2)))))

d)

(a) is suitable for Scor, and (b) is appropriate for Por. The difficulty in using (a) for Por and (b) for Scor arises from their concurrent nature.

3.4

a)

```
1  #lang racket
14
15 (define maplist
16   (lambda (f xs)
17     (cond
18       ((eq? xs null) null)
19       (#t (cons (f xs) (maplist f(cdr xs))))
20       )
21     )
22   )
23
24 (define inc (lambda (x) (+ x 1)))
25
26 (define compose2
27   (lambda (m c)
28     (lambda (f xs)
29       (m (lambda (x) (f (c x))) xs)
30       )
31     )
32   )
33
```

```
Welcome to DrRacket, version 8.11.1 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (equal?
   ((compose2 maplist car) inc '(1 2 3 4)) (mapcar inc '(1 2 3 4)))
#t
>
```

b)

g -> maplist h-> car

c)

(compose f h)

3.5

a)

No. If a memory location is garbage according to our definition (inaccessible from the current point of execution), it may or may not align with McCarthy's definition. This discrepancy arises because our definition deals with all memory locations, while McCarthy's definition specifically concerns registers reachable through car and cdr operations from base registers.

b)

If a location is considered garbage according to McCarthy's definition, it will also be garbage according to our definition, but the opposite may not always be true.
McCarthy's definition identifies garbage as registers that cannot be reached via a chain of specific operations (car and cdr operations) from any base register. In this case, if a register is unreachable through these operations, it essentially becomes inaccessible according to the program's flow.

c)

No. McCarthy’s definition of garbage is actually a subset of the definition given by the book, but
because in Lisp there is only one interaction model between program and memory (the program’s access to
memory is limited to the chains of car and cdr), the definition given by McCarthy It covers the entire state

space in such a way. Another difference between McCarthy's method and the definition of the book is that
after the execution of a part of the program that causes the loss of communication between the program
and a part of the memory, it performs the garbage collection operation. In general, it is not practical to
implement the garbage collection method by the book's definition, because it is not possible to determine
a non-obvious feature about the programs (at any point of execution).

3.6

a)

Initially, two cells, `(cons a b)` and `(cons c d)`, are created with each having a single reference. Then, a new cell `((a . b) . (c . d))` is formed, also with one reference. When the `car` operation is applied to this cell, the pointer to it is lost, rendering it garbage. Consequently, the reference count for `(a . b)` drops to zero due to the lost pointer. Eventually, when the `car` operation is executed on `(c . d)`, the pointer to this cell is also destroyed. As a result, all three cons cells become unreferenced and can be considered garbage, eligible for release from memory.

b)

When using `rplaca` and `rplacd` in Lisp to create a circular structure like:

```
 (setq x (cons 'a 'b))
(setq y (cons 'c 'd))

(rplacd x x)
(rplacd y x)
```

The resulting circular reference between `x` and `y` forms a structure that the reference-counting garbage collection algorithm cannot handle. The circular referencing prevents any cell's reference count within the cycle from reaching zero, causing the algorithm to fail in identifying this circular structure as garbage. Consequently, even if pointers are reset, the cells won't be freed as they're still part of the cycle referenced by other cells, leading to memory leaks.

Exercise:

```
1  #lang racket
2
3  (define Z_combinator (lambda (f)
4    ((lambda (x) (f (lambda (v) ((x x) v)))) (lambda (x) (f (lambda (v) ((x x) v)))) )
5                         )
6    )
7
8
9  (define fact (lambda (f)
10   (lambda (n)
11     (if (equal? n 0) 1 (* n (f (- n 1))))
12     )
13                 )
14   )
15
```

```
Welcome to DrRacket, version 8.11.1 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> (Z_combinator fact)
#<procedure>
> ((Z_combinator fact) 5)
120
```