Solutions

1. Evaluation [20 pts] (parts a–???)

For each of the following expressions determine what value would be printed by Dr. Scheme, or explain what kind of problem would occur.

(a) [5 pts]

```
(let ([greg (list "mini" "SUV" "elephant")])
  (car greg)) ; subtle joke?
```

Answer:

"mini"

(b) [5 pts]

```
(map car (list (list 1 2) (list (list "greg"))))
```

Answer:

(list 1 (list "greg"))

(c) [5 pts]

```
(let ([greg (lambda (x)
              (lambda (y) x))])
   ((greg 40) 2))
```

Answer:

40

(d) [5 pts]

```
(foldr - 0 (list 2 3 5))
```

Answer:

2 - (3 - (5 - 0)) = 2 - (3 - 5) = 2 - (-2) = 4

2. Semantics [20 pts] (parts a–???)

Some of the following expressions can be made to return the value 42 by first evaluating

`(define gregage EXP)`

for some expression `EXP`. For each of the following expressions, specify `EXP`, or state that the expression cannot be made to return 42 (no justification is needed).

(a) [5 pts] `(car gregage)`

Answer:

(list 42)

(b) [5 pts] `(gregage "car")`

Answer:

(lambda (x) 42)

(c) [5 pts] `((car gregage) (gregage "car"))`

Answer:

Can't be done–can't be a function and a list at the same time

(d) [5 pts]

```
(foldr
 (lambda (x r)
   (+ x
     (if (> r 0) (gregage 2) (gregage 3))))
 (gregage 4)
 (gregage 5))
```

Answer:

(lambda (x) (if (= x 5) (list 42) 0))

Answer:

(lambda (x) (if (= x 5) (list) 42))

3. Program correctness [25 pts] (parts a–???)

The procedure `copy` takes a list and returns a new list that prints the same way. Suppose we define a new version of it as follows:

```
(define (mycopy lst)
  (if (= (mylength lst) 0)
      empty
      (cons (first lst) (mycopy (rest lst)))))
```

Greg has done you a favor and written `mylength` as below:

```
(define (mylength lst)
  (if (empty? lst)
      0
      (+ 1 (mylength (rest lst)))))
```

(a) [18 pts] Prove by induction that the `mycopy` function is correct. You can assume that `mylength` is correct, much as you can assume that `cons` or `rest` is correct.

Answer:

Can be done either as structural induction on lst or as induction on the length of lst. We'll do structural induction here.

1) Doing structural induction on `lst`*, which is a list: either* `empty` *or* `(cons x r)`

2) P`[lst]` *is that* `(mycopy lst)` *returns a copy of the list that is identical to* `lst`*.*

3) Base case: Need to show that `(mycopy empty)` *works correctly. Given that* `mylength` *works correctly, and the evaluation rules for* `if`*,* `(mycopy empty)` *evaluates to* `empty`*, which is a correct copy of the empty list.*

4) Inductive case: we assume that P`[k]` *holds, and show that P*`[(cons x k)]` *holds for any* `x`*. Because* `mylength` *is correct,* `(mylength (cons x k))` *will not be 0, so the value of* `(mycopy (cons x k))` *will be the value of*

`(cons (first (cons x k)) (mycopy (rest (cons x k))))`*.*

By the evaluation rules for the list functions, this evaluates to `(cons x (mycopy k))`*.*

By the induction hypothesis P`[k]`*, we have that* `(mycopy k)` *is a correct copy of k, so* `(cons x (mycopy k))` *is a correct copy of* `(cons x k)`*.*

(b) [5 pts] Write the recurrence relation for $T(n)$, the running time of `mycopy`, in terms of the length n of the input list. Don't forget the base case, $T(0)$. You don't need to justify your answer.

Answer:

$$T(0) = c$$

$$T(n) = O(n) + c * T(n-1)$$

Note: the O(n) is because mylength takes time linear in the length of the list. This is what leads to the quadratic running time below.

(c) [2 pts] What is the running time of `mycopy`, expressed in big-O notation? Again, you need not justify your answer.

Answer:

$$O(n^2)$$

4. Lists, higher-order procedures [20 pts] (parts a–???)

Suppose we have the following definition: `(define-struct pair (x y))`

(a) [5 pts] Write a function `zip` which when given two lists of numbers, returns a list of pairs of numbers. For example:

```
(zip (list 1 2 3) (list 4 5 6)) =
   (list (make-pair 1 4) (make-pair 2 5) (make-pair 3 6))
```

Your function should fail when given lists of unequal length using the expression `(error 'zip "bad arguments")`.

Answer:

```
(define (zip x y)
  (cond [(and (empty? x) (empty? y)) empty]
        [(or (empty? x) (empty? y))
          (error 'zip "bad arguments")]
        [else (cons (make-pair (car x) (car y))
                    (zip (cdr x) (cdr y)))]))
```

Note: if you used a check of the length of x and y instead of using empty? in the base case, your function suddenly has quadratic runtime. We didn't take off points this time, but it's something to watch out for.

(b) [5 pts] Now, given two lists of numbers, we might wish to compute their dot-product. For instance:

```
(dot-prod (list 1 2 3 5) (list 4 5 6 2)) =
     1*4 + 2*5 + 3*6 + 5*2 = 4 + 10 + 18 + 10 = 42
```

Write the function `dot-prod`. Your code **must** use `zip` and `foldr`.

Answer:

```
(define (dot-prod v1 v2)
  (foldr (lambda (x r)
           (+ (* (pair-x x) (pair-y x)) r))
         0
         (zip v1 v2)))
```

(c) [10 pts] Recall that `reverse` takes a list and reverses it, so for example `(reverse (list 1 2 3))` evaluates to `(list 3 2 1)`. It is possible to write `reverse` using a single call to `foldr`, as shown below. Write the code that should be in EXP, **using only lambda and cons**.

```
(define (reverse lst)
  ((foldr
    EXP
    (lambda (l) l)
    lst)
   empty))
```

Answer:

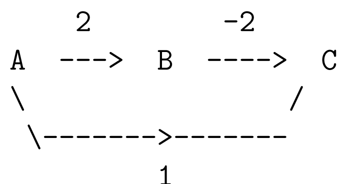
```
(lambda (x f) (lambda (L) (f (cons x L))))
```

5. Graphs [15 pts] (parts a–???)

Recall that Dijkstra's shortest path algorithm requires that the input graph has non-negative edge weights.

(a) [8 pts] Draw a directed graph, in which only one edge has a negative weight, where Dijkstra's algorithm gives the wrong answer. Specify (a) which source and destination nodes it gives the wrong answer for; (b) the length of the shortest path; and (c) the length of the path computed by Dijkstra's algorithm.

Answer:



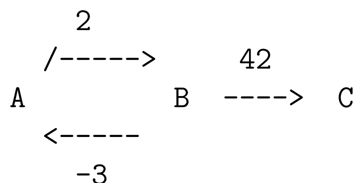
a) Wrong answer from A to C

b) Length of the shortest path is 0

c) Length computed by Dijkstra's is 1

(b) [7 pts] Jim Scheme has come up with a shortest path algorithm to handle the case of one negative edge weight. His idea is to add a large positive constant to the weight of every edge, thereby making all the weights positive, and then run Dijkstra's algorithm to find the shortest path in the new graph. So for example, if the negative edge weight is -42, he could add 50 to all edges. If this works, explain why. If it does not work, draw a graph where it fails.

Answer:



There is no shortest path from A to C in this graph, but when constant weights are added, there will be one. There are also other examples where shortest paths exist in both the original and modified graphs, but are different (in order to be correct, the modified algorithm has to find the correct path for the original graph).