Problem 1 - Scheme programming

Write the following functions on lists in Scheme. The semantics of the functions is described through examples.

1. (define remove-all
   (lambda (l)
     ...))
   ... (remove-all '(a((b)(c d)(((e f)))))) --> '(((())()())))

2. (define reverse-list
   (lambda (l)
     ...))
   ... (reverse-list '(a((b)(c d)(((e f g)))))) --> '((((g f e))(d c)(b))a)

3. (define make-single
   (lambda (l)
     ...))
   ... (make-single '(a((b)(c c)(((e e e)))))) --> (a((b)(c)(((e)))))

Problem 2 - let and let* in TINY

\[
\begin{align*}
x & \in \text{Variables} \\
n & \in \text{Integers} \\
c & ::= n \mid \#t \mid \#f \mid + \mid - \mid \ast \mid / \\
v & ::= c \mid (\lambda (x \ldots) e) \\
e & ::= v \mid x \mid (e_1 \ldots e_k) \mid (\text{if } e_1 \ e_2 \ e_3) \mid \\
    & \quad \quad \quad \left(\text{let } (x_1 \ e_1) \ldots (x_n \ e_n) \ e\right) \mid \\
    & \quad \quad \quad \left(\text{let* } (x_1 \ e_1) \ldots (x_n \ e_n) \ e\right) \\
p & ::= e \\
\end{align*}
\]
How can the let and let* constructs be expressed in the TINY language? In other words, can you rewrite a let and let* with TINY language constructs? If so, specify how to do this.

Note: In the second project, you asked to implement an interpreter extension for let and let*. This is different from what you are asked here.

* Problem 3 – Closures

```
(define test
  (lambda()
    (let* ((a 10)
           (b 11)
           (c 12)
           (foo (lambda(y)
                   (let* ((a 2)
                          (b (+ a y))
                          (f (lambda(x) (+ (* a x) b))))
                        f)))
         ((foo 3) a))) ;; (*1*)

(define run (test)) ;; (*2*)
```

To specify a closure, please use the following notation:

\[
<< \text{lambda abstraction , environment} >>
\]

where environment is a finite mapping from variables to values.

1. Give the closure value for foo on line (*1*)
2. Give the closure value for (foo 3) on line (*1*)
3. Give the closure value of test on line (*2*)
4. What is the value of (test) on line (*2*)?