---

30. Prove that (length(reverse xs)) = (length xs)

base case that xs is nil:

```
(length (reverse '()))
        = { substitute actual parameter in definition of length}
(length
(if (null? '())
        '()
        (append (simple-reverse (cdr '())(list1 (car '()))))))
        = { null?- empty rule }
(if #t
        (length xs)
```

case that xs is not nil so xs = (cons z zs)

```
(length (reverse (cons z zs))
        = { substitute actual parameter in definition of length }
(length
(if (null? (cons z zs))
        (cons z zs)
        (append (simple-reverse (cdr (cons z zs)) (list1 (car (cons z zs))))))
        = { cdr-cons rule }
(length
(if (null? (cons z zs))
        (cons z zs)
        (append (simple-reverse zs) (list1 (car (cons z zs))))))
        = {car-cons rule}
(length
(if (null? (cons z zs))
        (cons z zs)
        (append (simple-reverse zs) (list1 z)))))
        = {null?-cons law}
(length
(if #f 0
        (cons z zs)
        (append (simple-reverse zs) (list1 z))))
        = {if #f law}
(length (append (simple-reverse zs) (list1 z)))
        = {length of append law (page 80 in the book)}
(+ (length (simple-reverse zs))(length (list1 z)))
        = { apply inductive hypothesis -length of simple reverse zs is length(zs)}
(+ (length zs)(length(list1 z)))
```

```
= { commutitive property of addition}
(+ (length(list1 z))(length zs))
= { length of append law (pg 80) }
(length (append (list1 z) zs))
= {substitute arg into definition of list1}
(length (append (cons z '()) zs))
= {substitute arg into definition of append}
(length
(if (null? (cons z '()))
zs
(cons (car (cons z '()))(append (cdr (cons z '()))ys))
= { null?-cons law}
(length
(if #f (cons z zs)
(cons (car (cons z '()))(append (cdr (cons z '()))ys))
= {if #f law}
(length (cons (car (cons z '()))(append (cdr (cons z '()))ys)))
= { cdr-cons law and car-cons law}
(length (cons '() (append '() ys))
= { append null law}
(length (cons '() ys))
= {null cons law}
(length ys)
```

a) Write a DefineGlobal rule to show the operational semantics of such a val

#37

$$\frac{\langle set(x, e), \rho\{x \mapsto l\}, \sigma\rangle \Downarrow \langle v, \sigma'\rangle}{\langle VAL(x, e), \rho, \sigma\rangle \rightarrow \langle \rho\{x \mapsto l\}, \sigma'\rangle}$$

(Notice that whether $x \in dom\ \rho$ or $x \notin dom\ \rho$ doesn't matter)

b) (define check_Val_semantics
(begin
(val x 2) ← if the DefineGlobal rule is applied, this will be defined and you can access that variable
(print x) [If it isn't, that will produce the error]
)

I'm trying to determine whether val has created a new variable

c) Compare and contrast the 2 ways of defining val.

I like the old method where a val binding of a name that is already bound is equivalent to set. Although it might be "easier" if val always creates a new binding, you could get some unexpected side effects when you are setting values when they don't actually exist. This would mean that when you try to access a variable it may or may not be there and it may or may not be what you are looking.

Ian Cross
#37