COMP-105, μScheme Assignment

Benjamin Tanen

October 6, 2015

Ramsey, p.181 #31

Problem: Prove that (length (reverse xs)) = (length xs)

For this proof, we will be using `simple-reverse` in the place of the generic `replace` function. The first step is the base case in which `xs` is empty (xs = *nil*).

(length (simple-reverse '()))
={substitute actual parameters in definition of length function}
(if (null? (simple-reverse '()))
    0
    (+ 1 (length (cdr '()))))
={substitute actual parameters in definition of simple-reverse function}
(if (null?
    (if (null? '())
        '()
        (append(simple-reverse cdr '())(list1 (car '())))))
    0
    (+ 1 (length (cdr (simple-reverse '())))))
={null? - empty list law}
(if (null?
    (if #t '() (append (simple-reverse cdr '()) list1 (car '()))))
    0
    (+ 1 (length (cdr (simple-reverse '())))))
={if - #t law}
(if (null? '())
    0
    (+ 1 (length (cdr (simple-reverse '())))))
={null? - empty list law}
(if #t)
    0
    (+ 1 (length (cdr (simple-reverse '())))))

={if - #t law}

0

={since, by definition, the length of the empty list is 0, we can say 0 = (length '())}

(length '())

Thus we have shown that the base case (`xs` = *nil*) is true.

The next step is to show the inductive step, where we assume `xs` $\neq$ *nil*. We can think of this list as being made up of two parts, the front y and back ys. Since we have shown the base case already, we will next try to show, considering xs = (cons y ys), the inductive hypothesis of:

(length (reverse xs)) = (length xs)

(length (simple-reverse xs))

={under assumption xs $\neq$ *nil*, xs = (cons y ys)}

(length (simple-reverse (cons y ys)))

={substitute actual parameters into definition of simple-reverse function}

```
(length
    (if (null? (cons y ys))
        (cons y ys)
        (append
            (simple-reverse (cdr (cons y ys)))
            (list1 (car (cons y ys)))))
```

={null?-cons law}

```
(length
    (if #f
        (cons y ys)
        (append
            (simple-reverse (cdr (cons y ys)))
            (list1 (car (cons y ys))))))
```

={if #f law}

```
(length
    (append
        (simple-reverse (cdr (cons y ys)))
        (list1 (car (cons y ys)))))
```

={car-cons law}

```
(length
    (append
        (simple-reverse (cdr (cons y ys)))
        (list1 (y))))
```

={cdr-cons law}

(length (append (simple-reverse ys) (list1 (y))))

={substitute in definition of list1 function}

```
(length (append (simple-reverse ys) (cons y '())))
={using (length (append xs ys)) = (+ (length xs) (length ys)) law, from Ramsey p.94}
(+ (length (simple-reverse ys)) (length (cons y '())))
={substitute parameters into definition of length}
(+
    (length (simple-reverse ys))
    (if (null? (cons y '()))
        0
        (+ 1 (length (cdr (cons y '()))))))
={cdr-cons law}
(+
    (length (simple-reverse ys))
    (if (null? (cons y '()))
        0
        (+ 1 (length '()))))
={null-cons law}
(+
    (length (simple-reverse ys))
    (if #f
        0
        (+ 1 (length '()))))
={if #f law}
(+
    (length (simple-reverse ys))
    (+ 1 (length '()))))
={induction hypothesis, (length (reverse ys)) = (length ys)}
(+
    (length ys)
    (+ 1 (length '()))))
={substitute parameters into definition of length}
(+
    (length ys)
    (+ 1
        (if (null? '())
            0
            (+ 1 (length (cdr '()))))))
={null? - '() law}
(+
    (length ys)
    (+ 1
        (if #t
            0
```

(+ 1 (length (cdr '())))))))

={if-true law}

```
(+
    (length ys)
    (+ 1 0))
```

={addition}

(+ (length ys) 1)

={addition is commutative}

(+ 1 (length ys))

={using (length (cons x xs)) = (+ 1 (length xs)) law, from Ramsey p.93}

(length (cons y ys))

={using previous assumption xs = (cons *y* *ys*)}

(length xs)

Thus was have shown (length (reverse xs)) = (length xs) is **true**.