CSU290 Lecture Notes Lecture 6 22 Sept 2008

Peter Dillinger

Announcements

* First homework due 23rd (Tuesday)

* First exam on 24th (Wednesday)

Review for exam

Assume we have a function DIVISIBLE-BY like in the homework. Write PRIMEP, a recognizer for prime numbers.

In class I wrote this helper function:

```
; DIVISOR<=P: nat nat -> boolean
; (Old helper for PRIMEP)
; Whether there is a divisor for first argument >= 2 and <= second argument
(defun divisor<=p (x i)
  (if (or (zp i)
          (= i 1))
    nil
    (if (divisible-by x i)
      t
      (divisor<=p x (- i 1)))))

(check= (divisor<=p 6 3) t)
(check= (divisor<=p 6 5) t)
(check= (divisor<=p 5 4) nil)
(check= (divisor<=p 12 1) nil)
(check= (divisor<=p 12 0) nil)
(check= (divisor<=p 12 5) t)
(check= (divisor<=p 97 96) nil)
(check= (divisor<=p nil nil) nil)
(check= (divisor<=p 42 t) nil)
```

But this is better suited for constructing PRIMEP:

```
; COUNT-DIVISORS<=: nat nat -> nat
; (Helper for PRIMEP)
; Counts the number of positive integer divisors of the first argument
; that are less than or equal to the second argument.
(defun count-divisors (x i)
  (if (zp i)
    0
    (if (divisible-by x i)
      (+ 1 (count-divisors x (- i 1)))
      (count-divisors x (- i 1)))))

(check= (count-divisors 6 3) 3)
(check= (count-divisors 5 5) 2)
(check= (count-divisors 12 1) 1)
(check= (count-divisors 12 0) 0)
(check= (count-divisors 97 97) 2)
(check= (count-divisors nil nil) 0)
(check= (cound-divisors 42 t) 0)
```

```
; PRIMEP: nat -> boolean
; Recognizer for prime numbers, which must have exactly two distinct
; divisors (1 and itself).
(defun primep (x)
  (and (natp x)
       (= (count-divisors x x) 2)))

(check= (primep 0) nil)
(check= (primep 1) nil)
(check= (primep 2) t)
(check= (primep 3) t)
(check= (primep 4) nil)
(check= (primep 5) t)
(check= (primep 6) nil)
(check= (primep 97) t)
(check= (primep 98) nil)
(check= (primep 1/2) nil)
(check= (primep -4) nil)
(check= (primep t) nil)
```

==========

Write SQUAREP, a recognizer for perfect squares--numbers that are the square of a natural number. (Square means multiplied by itself.)

```
; SQRT<=P: nat nat -> boolean
; Whether the first argument has a natural number square root <= second
; argument
(defun sqrt<=p (x i)
  (if (zp i)
    (= x 0)
    (if (= x (* i i))
      t
      (sqrt<=p x (- i 1)))))

(check= (sqrt<=p 4 2) t)
(check= (sqrt<=p 4 1) nil)
(check= (sqrt<=p 5 5) nil)
(check= (sqrt<=p 25 10) t)
(check= (sqrt<=p 25 4) nil)
(check= (sqrt<=p 4 0) nil)
(check= (sqrt<=p 0 0) t)
(check= (sqrt<=p t t) nil)
(check= (sqrt<=p 0 t) t)
(check= (sqrt<=p "hi" 0) nil)

; SQUAREP: nat -> boolean
; Recognizer for perfect squares--numbers that are the square of a natural
; number.
(defun squarep (i)
  (sqrt<=p i i))

(check= (squarep 25) t)   ; 5 * 5
(check= (squarep 24) nil)
(check= (squarep 6)  nil)
(check= (squarep 4)  t)   ; 2 * 2
(check= (squarep 3)  nil)
(check= (squarep 1)  t)   ; 1 * 1
(check= (squarep 0)  t)   ; 0 * 0
(check= (squarep 1/2) nil)
(check= (squarep -4) nil)
(check= (squarep t) nil)
```