Your Paper

Kevin David Ruiz González, Daniel Ruben Ochoa Galván

27 de octubre de 2022

1. Exercise 4.8 [*]

Show exactly where in our implementation of the store these operations take linear time rather than constant time.
En new-ref cuando se usa lenght y append ocurre en un tiempo de ejecución lineal porque depende de la cantidad de elementos.
En der-ref, se usa list-ref por lo que tiene tiempo de ejecución lineal.
Y en setref!, setref-inner se ejecuta como un ciclo en el almacenamiento el cual toma tiempo lineal, por lo que setref! tiene tiempo de ejecución lineal

2. Exercise 4.9 [*]

Implement the store in constant time by representing it as a Scheme vector. What is lost by using this representation?

```
(define (empty-store)
  (vector))

(define the-store 'uninitialized)

(define (get-store)
  the-store)

(define (initialize-store!)
  (set! the-store (empty-store)))

(define (reference? v)
  (lambda (v)
       (integer? v)))

(define (extend-store store val)
  (let* ([store-size (vector-length store)]
         [new-store (make-vector (+ store-size 1))])
    (let loop ([i 0])
      (if (< i store-size)
          (let ([val (vector-ref store i)])
            (vector-set! new-store i val)
            (loop (+ i 1)))
          (vector-set! new-store i val)))
    (cons new-store store-size)))

(define (newref val)
  (let* ([new-store-info (extend-store the-store val)]
         [new-store (car new-store-info)]
         [new-ref (cdr new-store-info)])
    (set! the-store new-store)
    new-ref))

(define (deref ref)
  (vector-ref the-store ref))

(define (report-invalid-reference ref store)
  (eopl:error 'setref
              "illegal reference ~s in store ~s"
              ref
              store))

(define (setref! ref val)
  (if (and (reference? ref)
           (< ref (vector-length the-store)))
      (vector-set! the-store ref val)
      (report-invalid-reference ref the-store)))
```