Lista zagadnień nr 9 – Zadania domowe

Zadanie 16.

Leniwe listy to struktura danych, która jest podobna do list, ale oblicza swój ogon leniwie. Można to uzyskać poprzez zamknięcie ogona pod lambdą. Ciało lambdy obliczamy, gdy potrzebujemy znać ogon listy, czyli robiąc `cdr`-a:

```
(define (lcons x f) (cons x f))

(define lnull null)

(define lnull? null?)

(define (lcar xs) (car xs))

(define (lcdr xs) ((cdr xs)))
```

Należy też uczulić użytkownika, żeby jako drugi argument `lcons`-a przekazywał bezargumentową procedurę.

Najbardziej efektownym zastosowaniem leniwych list jest tworzenie list nieskończonych. Np. listę wszystkich liczb naturalnych możemy zdefiniować tak:

```
(define (from n)
  (lcons n (lambda () (from (+ n 1)))))

(define nats
  (from 0))
```

Kolejny przykład to obliczanie n-tej liczby pierwszej przez stworzenie nieskończonej listy liczb pierwszych i wzięcie jej n-tego elementu:

```
(define (lnth n xs)
  (cond [(= n 0) (lcar xs)]
        [else (lnth (- n 1) (lcdr xs))]))

(define (lfilter p xs)
  (cond [(lnull? xs) lnull]
        [(p (lcar xs))
         (lcons (lcar xs) (lambda () (lfilter p (lcdr xs))))]
        [else (lfilter p (lcdr xs))]))
```

```
(define (prime? n) ; definicja umyslnie malo wydajna
  (define (factors i)
    (cond [(>= i n) (list n)]
          [(= (modulo n i) 0) (cons i (factors (+ i 1)))]
          [else (factors (+ i 1))]))
  (= (length (factors 1)) 2))

; lista wszystkich liczb pierwszych
(define primes (lfilter prime? (from 2)))
```

Teraz można poznać n-tą liczbę pierwszą używając `(lnth n primes)`.

Leniwe listy mają jedną wadę: za każdym razem, gdy prosimy o ogon listy, jego wartość jest obliczana od początku. Widać to, gdy prosimy o kilka większych liczb pierwszych:

```
> (time (lnth 1000 primes))
cpu time: 1161 real time: 1166 gc time: 12
7927
> (time (lnth 1001 primes))
cpu time: 1142 real time: 1146 gc time: 11
7933
> (time (lnth 1002 primes))
cpu time: 1128 real time: 1130 gc time: 0
7937
```

Zadanie: Dodaj do leniwych list *spamiętywanie*. Czyli zmodyfikuj reprezentację leniwych list tak, by używała ona mutowalnych struktur danych. W momencie obliczania ogona listy, niech `lcdr` nie tylko zwraca wartość, ale także modyfikuje odpowiednio struktury w pamięci, by kolejne wywołanie procedury `lcdr` na tej samej liście nie obliczało ogona, tylko zwracało spamiętaną wartość.

Przykładowo, autorowi zadania udało się osiągnąć w ten sposób następujące rezultaty:

```
> (time (lnth 1000 primes))
cpu time: 1149 real time: 1154 gc time: 0
7927
> (time (lnth 1001 primes))
cpu time: 2 real time: 3 gc time: 0
7933
> (time (lnth 1002 primes))
cpu time: 2 real time: 2 gc time: 0
7937
```

– licząc tysiąc pierwszą liczbę pierwszą, nie musimy najpierw obliczać tysiąca pierwszych liczb pierwszych, bo są one już spamiętane. Stąd tak szybkie obliczenie wyników dla argumentów `1001` i `1002`.

Zadanie 17.

Programy w języku WHILE składają się z instrukcji, a częścią składową niektórych instrukcji są wyrażenia. Definiując język WHILE użyliśmy tych samych wyrażeń, których używaliśmy w definicji języka funkcyjnego. Konsekwencją tego jest to, że wartościami WHILE-owych zmiennych mogą być dowolne wartości, także domknięcia. Oczywiście, używamy środowiska (pamięci) z momentu definicji, więc w programie

```
{(x := 5)
 (f := (lambda (y) (+ x y)))
 (x := 10)
 (z := (f 0))}
```

końcową wartością zmiennej z jest `5`. Zmodyfikuj interpreter WHILE-a tak, by zmienne WHILE-owe (ale nie lokalne w wyrażeniach) były przekazywane przez *referencję*, czyli podczas wywoływania procedury jej ciało brało pod uwagę wartości zmiennych WHILE-owych z momentu wywołania. Np. wynikiem programu powyżej powinno być `10`.

Inne przykłady:

```
{(x := 5)
 (f := (let [x 50] (lambda (y) (+ x y))))
 (x := 10)
 (z := (f 0))}
```

Końcowa wartość zmiennej z: `50`

```
{(x := 5)
 (f := (let [x 50]
        (let [foo (lambda (y) (+ x y))]
        (let [x 100]
        foo))))
 (x := 10)
 (z := (f 0))}
```

Końcowa wartość zmiennej z: `50`

```
{(x := 5)
 (f := (let [x x] (lambda (y) (+ x y))))
 (x := 10)
 (z := (f 0))}
```

Końcowa wartość zmiennej z: `5`