L14. Laziness 2

유형	강의
복습 여부	✓
Status	Done

Redundant Evaluation

```
{{fun {x} {+ {+ x x} {+ x x}}}
   {- {+ 4 5} {+ 8 9}}}
```

How many times is {+ 8 9} evaluated?

Since the result is always the same, we'd like to evaluate {- {+ 4 5} {+ 8 9}} at most once.

save the result of {- {+ 4 5} {+ 8 9}} somewhere else. Then use that saved value when we need {- {+ 4 5} {+ 8 9}}

Where should we save the evaluated value ?

Box

> A box is like a single-element vector, normally used as minimal mutable storage.

a simple data structure that can contain any type of one value

```
; box
(define mybox (box 0))
(unbox mybox)
(set-box! mybox 17)
(unbox mybox)
(set-box! mybox #f)
(unbox mybox)
(set-box! mybox 'b)
(unbox mybox)
```

```
0
17
#f
'b
```

Why do we use box instead of cache?

Identifiers and its corresponding values should be saved into a sort of linked lists so that we can find the values of the identifier that we called.

However, we are now implementing laziness concept.

In the below example, the value of {- {+ 4 5} {+ 8 9}} is needed only for the given function parameter ‘x. Argument expression {- {+ 4 5} {+ 8 9}} is always bound with the function parameter ‘x. Using cache would be too costly because we only need to save one value.

{{fun {x} {+ {+ x x} {+ x x}}} {- {+ 4 5} {+ 8 9}}}

수정사항

```
; LFAE-Value
(define-type LFAE-Value
  [numV (n number?)]
  [closureV (param symbol?) (body LFAE?) (ds DefrdSub?)]
  [exprV (expr LFAE?) (ds DefrdSub?)
         (value (box/c (or/c false LFAE-Value?)))]) ; store t
; box contains either two types of values : false or LFAE-Val
; strict
(define (strict v)
  (type-case LFAE-Value v
    [exprV (expr ds v-box)
           (if (not (unbox v-box)) ; if box contains #f, eval
               (local
                 [(define v (strict (interp expr ds)))] ; v :
                 (begin (set-box! v-box v)
                        v)) ; return v after evaluating it
               (unbox v-box))] ; if box contains a value, jus
    [else v] ; for numV or closureV
    )
  )
```

```
; lookup
(define (lookup name ds)
  (type-case DefrdSub ds
    [mtSub () (error 'lookup "free identifier")]
    [aSub (i v saved) (if (symbol=? i name)
                          (strict v)
                          (lookup name saved))]
    )
  )
```

```
(define (interp lfae ds)
  (type-case LFAE lfae
    [num (n) (numV n)]
    [add (l r) (num+ (interp l ds) (interp r ds))]
    [sub (l r) (num- (interp l ds) (interp r ds))]
    [id (s) (lookup s ds)]
    [fun (p b) (closureV p b ds)]
    [app (f a) (local
                 [(define ftn-v (strict (interp f ds)))
                  (define arg-v (exprV a ds (box #f)))]
                 (interp (closureV-body ftn-v)
                         (aSub (closureV-param ftn-v)
                               arg-v
                               (closureV-ds ftn-v))
                         )
                 )
         ]
    )
  )
```

```
(run '{{fun {x} 0} {+ 1 {fun {y} 2}}} (mtSub)) ; produces (nu
(run '{{fun {x} {+ x x}} {+ 1 {{fun {y} 2} 1}}} (mtSub)) ; pr
;(run '{{fun {x} {+ x x}} {+ 1 {fun {y} 2}}} (mtSub)) ; produ
;(run '{{fun {x} x} {+ 1 {fun {y} 2}}} (mtSub)) ; produces er
```

```
(numV 0)
(numV 6)
```