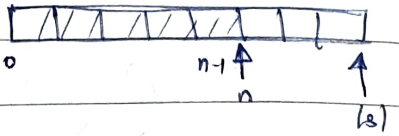
INTERPRETER FOR EXPLICIT REF

(define store-extend

;;; a store is a pair
(s, n)



s: vector

n: size of store $0 \le n \le |s|$

(define (newref store v)
(store-extend store v)
(match-let ([(list s n) store])
(if (= n |s|)
(error 'store-extend "out of memory!")
(cases storable v
[loc (m) (if (>
(begin
(vector-set! s n v)
(list s (add1 n)))...)
;;;

(define (store-get store m)
(match-let ([(list s n)]))
(if (>= m n)
(error 'store-get "ref. out of bounds!" m)
(else
(vector-ref s m)))))

(define *global-store (make-vector 0 50) 0)
(cases ast of e.
[num (n) n]
[bool (b) b]

[getref (r) (store-get *global-store* r)]
[getref (e)
(let ([r (eval-ast e env)])
(store-get *global-store* r)]
(seq (as)

STORE PASSING

(define (eval-ast a env store)
((cases ast a
"
(if (test then else)
(match-let ([(list val newsto)
(eval-ast test env sto)])

IMPLICIT REFERENCES

(2)

expval = num + bool + proc

storeval = expval.

store = L → storeval.

env = id → denval.

denval = L

(1)

expval = num + bool + proc

env = (id → expval)

e1

(3)

Den.Val = L + proc.