2nd Interpreter of the Term

(Due to Friedman and Felleisen, the authors of TLS)

An L-expression is an S-expression which is either:

(AND l1 l2), or (OR l1 l2), or

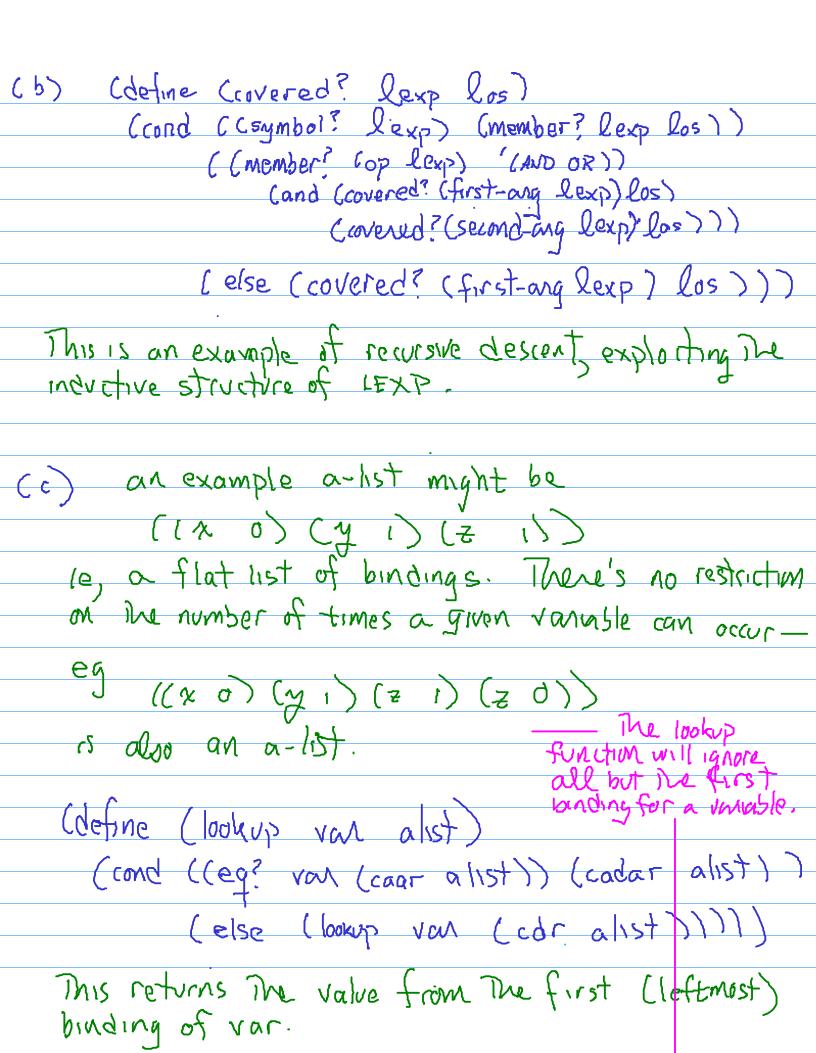
(NOT I1), or

an arbitrary symbol, which we call a variable

Here I1 and I2 are arbitrary L-expressions, so this is an inductive definition once we add 'and nothing else is an L-expression'

- (a) Write and certify a function lexp? which checks whether an S-expression is an L-expression.
- (b) Write and certify a function covered? of an L-expression lexp and a list of symbols los that tests whether all the variables in lexp are in los.
- (c) For the evaluation of L-expressions we need association lists, or alists. An alist for L-expressions is a list of (variable, value) pairs. The variable component is always a symbol, and the value component is either the number 0 (for false) or 1 (for true). Write and certify a function lookup of the symbol var and the association list al, so that (lookup var al) returns the value of the first pair in al whose car is eq? to var.
- (d) If the list of symbols in an alist for L-expressions contains all the variables of an L-expression lexp, then lexp is called _closed_ with respect to this alist. A closed L-expression can be evaluated essentially by substituting the values of the variables given in the alist for the variable occurrences in the L-expression. You are asked to write and certify the function value of an L-expression lexp and an alist al, which, after verifying that lexp is closed with respect to al, determines whether lexp means true or false. If lexp is not closed wrt al, then (value lexp al) should return the symbol not-covered.

(a) comprises the pre-evaluation sanity check
(b) and (c) comprise the environment subsystem

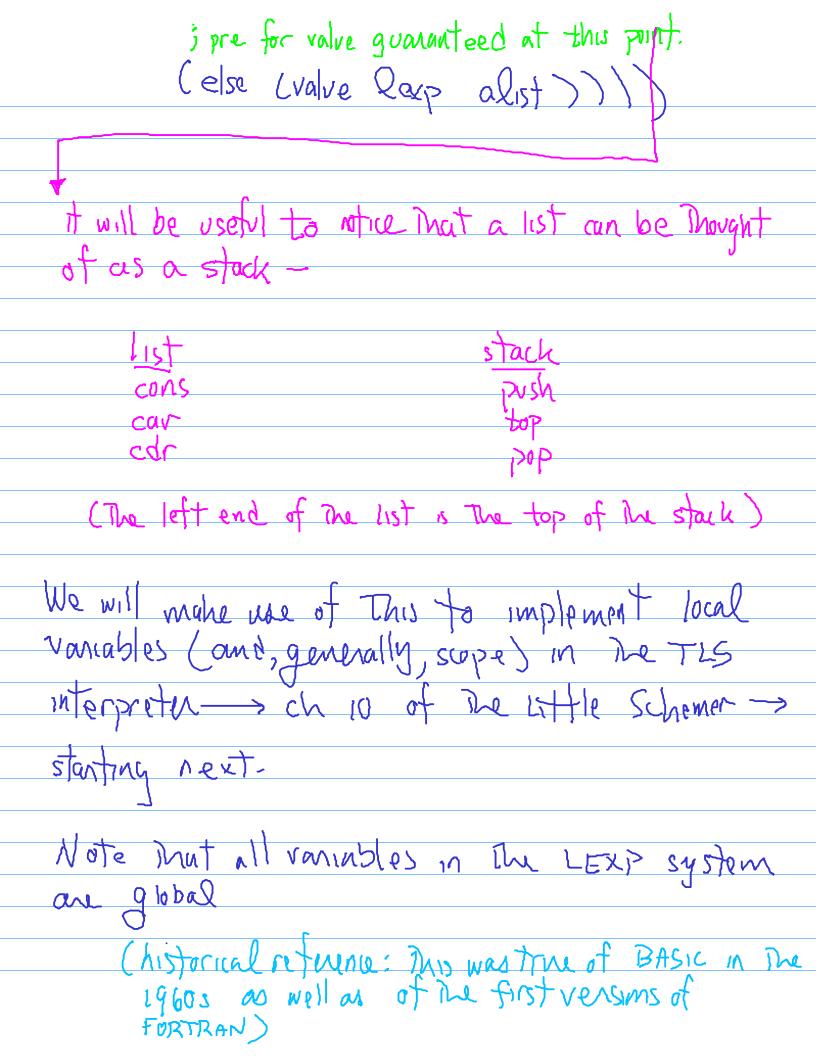


Note That (AND XX) means one thing it alist is ((x o) (y 1)), and means something else if alists
((x 1) (y 1)) explains the need for the alist assuming pre for value is satisfied— ie lexp really is in LEXP, and also muit (covered? lexp (map can alist)) is true. (define (value lexp alist) representation of (cond ((Symbol? (exp) (lookup Lexp alist))

((and? lexp)

(and (value (first-ang lexp) alist)

(value (sound-ang lexp) alist) (101) Jap (mt? lexp) (define (check-and-evaluate lexp alist) (cond ((not (lexp? lexp)) syntax-error) (C'not (rovered? lexp (map can alist))) not-covered



; For scheme, we need somewhat more complex environment structures.
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
; to see this, let's consider some lambda forms and their associated environments
; example 1
((lambda (a b) environment by adding the cib
2 3)
; the expression (+ a b) is evaluated in this environment
; (a b) (2 3)
; essentially the same flat structure used for L-exps, perhaps represented as a pair of lists the ; first is a list of names, i.e., (a b), and the second is the list of corresponding values, (2 3)
The lookup procedure here would be to smuttaneously cdr
down The vous and values justs
But this simple one-rib environment is not
always adequate

Consider;

