The next feature to consider is assignment (like \( x = E; \) in Java). In a assignment like \( x = x + 1; \), the variable \( x \) plays two different roles: one the right of the \( = \)-sign, we need its value (its R-value in C terminology). But on the left, we want the address in storage that is to be updated (L-value). We call the latter a reference (similar to a pointer in C).

The meaning of assignment will be given by updating references. These are defined as follows:

\[
\text{(define-datatype reference reference?}
  \text{ (a-ref}
    \text{ (position integer?)}
    \text{ (vec vector?))})
\]

A reference is given as a position in a vector (that is similar to an offset in a stack frame in compiler construction).

We need the following operations on references:

\[
\text{(define primitive-deref}
  \text{ (lambda (ref)}
    \text{ (cases reference ref}
      \text{ (a-ref (pos vec) (vector-ref vec pos))))})
\]

\[
\text{(define primitive-setref!}
  \text{ (lambda (ref val)}
    \text{ (cases reference ref}
      \text{ (a-ref (pos vec) (vector-set! vec pos val))))})
\]

\[
\text{(define deref}
  \text{ (lambda (ref)}
    \text{ (primitive-deref ref))}
\]

\[
\text{(define setref!}
  \text{ (lambda (ref val)}
    \text{ (primitive-setref! ref val))})
\]
The procedure `apply-env-ref` looks up a reference; `apply-env` also dereferences it.

```scheme
(define apply-env
  (lambda (env sym)
    (deref (apply-env-ref env sym))))

(define apply-env-ref
  (lambda (env sym)
    (cases environment env
      (empty-env-record ()
        (eopl:error 'apply-env-ref 'No binding for ~s' sym))
      (extended-env-record (syms vals env)
        (let ((pos (rib-find-position sym syms)))
          (if (number? pos)
            (a-ref pos vals)
            (apply-env-ref env sym)))))))
```

We add a new abstract syntax record `varassign-exp` with fields `id` and `rhs-exp`.

When we want to look up the (R-)value of a variable, we use `apply-env`. When we want to assign to a variable, we need the reference (its L-value), so we use `apply-env-ref`, and then call `setref!` to modify the contexts of the reference.

```scheme
(define eval-expression
  (lambda (exp env)
    (cases expression exp
      (var-exp (id) (apply-env env id))
      (varassign-exp (id rhs-exp)
        (begin
          (setref! (apply-env-ref env id)
            (eval-expression rhs-exp env))
          1))
    various cases as before))
```