Lenguaje LETREC

Este lenguaje está basado en PROC pero contempla la definición e invocación de procedimientos explícitamente recursivos.

Sintaxis

Sintaxis concreta		Sintaxis abstracta
Expression ::=	*Number*	(const-exp *num*)
Expression ::=	**-**(*Expression* , *Expression*)	(diff-exp *exp1* *exp2*)
Expression ::=	**zero?**(*Expression*)	(zero?-exp *exp1*)
Expression ::=	**if** *Expression* **then** *Expression* **else** *Expression*	(if-exp *exp1* *exp2* *exp3*)
Expression ::=	*Identifier*	(var-exp *var*)
Expression ::=	**let** *Identifier* = *Expression* **in** *Expression*	(let-exp *var* *exp1* *body*)
Expression ::=	**proc** (*Identifier*) *Expression*	(proc-exp *var* *body*)
Expression ::=	(*Expression* *Expression*)	(call-exp *op-exp* *arg-exp*)
Expression ::=	**letrec** *Identifier* (*Identifier*) = *Expression* **in** *Expression*	(letrec-exp *p-name* *b-var* *p-body* *letrec-body*)

Semántica

Interpretación de expresiones

```
(value-of (const-exp n) ρ) = (num-val n)

(value-of (var-exp var) ρ) = ρ(var)

(value-of (diff-exp exp1 exp2) ρ)
  = (num-val (- (expval→num (value-of exp1 ρ))
                (expval→num (value-of exp2 ρ))))

(value-of (zero?-exp exp1) ρ)
  = (let ([val1 (value-of exp1 ρ)])
      (bool-val (= 0 (expval→num val1))))

(value-of (if-exp exp1 exp2 exp3) ρ)
  = (if (expval→bool (value-of exp1 ρ))
        (value-of exp2 ρ)
        (value-of exp3 ρ))

(value-of (let-exp var exp1 body) ρ)
  = (let ([val1 (value-of exp1 ρ)])
      (value-of body [var = val1]ρ))

(value-of (proc-exp var body) ρ)
  = (proc-val (procedure var body ρ))

(value-of (call-exp op-exp arg-exp) ρ)
  = (let ([proc (expval→proc (value-of op-exp ρ))]
          [arg (value-of arg-exp ρ)])
      (apply-procedure proc arg))
donde:
  (apply-procedure (procedure var body ρ) val)
    =(value-of body [var = val]ρ)

(value-of (letrec-exp p-name b-var p-body letrec-body) ρ)
  = (value-of letrec-body
              [p-name = b-var ↦ p-body]ρ)
donde:
  Si ρ₁ = [p-name = b-var ↦ p-body]ρ, entonces
    (apply-env ρ₁ var) =
       (proc-val (procedure b-var p-body ρ₁))
  si var = p-name, y
    (apply-env ρ₁ var) = (apply-env ρ var)
  si var ≠ p-name.
```

Estrategias de implementación

Representación con estructuras de datos:

```
(define-datatype environment environment?
  (empty-env)
  (extend-env (var identifier?) (val expval?) (env environment?))
  (extend-env-rec (p-name identifier?) (b-var identifier?) (body expression?) (env environment?)))

(define (apply-env env search-var)
  (cases environment env
    (empty-env () ...señala un error...)
    (extend-env (saved-var saved-val saved-env)
      (if (eqv? saved-var search-var) saved-val
          (apply-env saved-env search-var)))
    (extend-env-rec (p-name b-var p-body env)
      (if (eqv? search-var p-name)
          (proc-val (procedure b-var p-body env))
          (apply saved-env search-var)))))
```