Lenguaje PROC

Oscar Eduardo Galaviz Cuen

7 de Septiembre de 2022

Este lenguaje está basado en LET pero contempla la creación e invocación de procedimientos.

1 Sintaxis

Sintaxis concreta

$Expression ::= Number$
$Expression ::=$ **-**$(Expression, Expression)$
$Expression ::=$ **zero?**$(Expression)$
$Expression ::=$ **if** $Expression$ **then** $Expression$ **in** $Expression$
$Expression ::= Identifier$
$Expression ::=$ **let** $Identifier = Expression$ **in** $Expression$
$Expression ::=$ **proc** $(Identifier)$ $Expression$
$Expression ::= (Expression\ Expression)$

Sintaxis abstracta

```
(const-exp num)
(diff-exp exp1 exp2)
(zero?-exp exp1)
(if-exp exp1 exp2 exp3)
(var-exp var)
(let-exp var exp1 body)
(proc-exp var body)
(call-exp op-exp arg-exp)
```

2 Semántica

Los valores expresados $ExpVal$ (valores de expresiones) y los valores denotados $DenVal$ (valores asociados en entornos) son los mismos y corresponden a $Int + Bool + Proc$. Los procedimientos expval→num,expval→bool y expval→proc toman valores expresados y regresan los valores codificados en el

lenguaje de implementación.

Interpretación de expresiones

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

(value-of (var-exp var) env) = env(var)

(value-of (diff-exp exp1 exp2) env)
  = (num-val (- (expval->num (value-of exp1 env))
                (expval->num (value-of exp2 env))))

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

(value-of (if-exp exp1 exp2 exp3) env)
  = (if (expval->bool (value-of exp1 env))
        (value-of exp2 env)
        (value-of exp3 env))

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

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

(value-of (call-exp op-exp arg-exp) env)
  = (let ([proc (expval->proc (value-of op-exp env))]
          [arg (value-of arg-exp env)])
       (apply-procedure proc arg))

donde:
(apply-procedure (procedure var body env) val)
   =(value-of body [var = val]env)
```

Estrategias de implementación

Representación procedural:

```
(define (procedure var body env)
  (lambda (val)
     (value-of body [var = val]env)))
```

```
(define (apply-procedure proc1 val)
  (proc1 val))
```

Representación con estructuras de datos:

```
(define-datatype proc proc?
  (procedure
     (var identifier?)
     (body expression?)
     (saved-env environment?)))

(define (apply-procedure proc1 val)
  (cases proc proc1
     (procedure (var body env)
        (value-of body [var = val]env))))
```