Gael Balderrama Dominguez
Tarea-07
Noviembre 2023

1.-Extiende el lenguaje agregando un nuevo operador *minu* que toma como argumento n y regresa $-n$.Por ejemplo, el valor de minus(-(minus(5),9)) debe ser 14
Respuesta:

Especificacion Lexica :

Minus = Minus

Especificacion Sintactica :

Concreta :Expression ⇒ minus (Expression)

Abstracta : (minus-exp exp1)

Especificacion Semantica :

(value-of (minus-exp $exp1)\rho$) = (-(num-val n))

$$\frac{\mathcal{E}(-(0, exp1), \rho) = x}{\mathcal{E}(minus(exp1), \rho) = x}$$

2.-Extiende el lenguaje agregando operadores para la suma, multiplicacion y conciente de enteros.
Respuesta:

Especificacion Lexica :

Suma = +

Multiplicacion = *

Division = /

Especificacion Sintactica :

Concreta :Expression ⇒ +(Expression,Expression)

Concreta :Expression ⇒ *(Expression,Expression)

Concreta :Expression ⇒ /(Expression,Expression)

Abstracta : (Sum-exp exp1 exp2)

Abstracta : (Mult-expexp1 exp2)

Abstracta : (Div-exp exp1 exp2)

Especificacion Semantica :

Regla:

(value-of (Sum-exp exp1 exp2) ρ) =
— (num-val (+ (expval→num (value-of exp1 ρ))
— (expval→num (value-of exp2 ρ))))

(value-of (Mult-exp exp1 exp2) ρ) =
— (num-val (* (expval→num (value-of exp1 ρ))
— (expval→num (value-of exp2 ρ))))

(value-of (Div-exp exp1 exp2) ρ) =
— (num-val (quotient (expval→num (value-of exp1 ρ))
— (expval→num (value-of exp2 ρ))))

Regla Breve:

$$\mathcal{E}(*(exp1, exp2), \rho) = [[\mathcal{E}(exp1, \rho)] * [\mathcal{E}(exp2, \rho)]]$$

$$\mathcal{E}(+(exp1, exp2), \rho) = [[\mathcal{E}(exp1, \rho)] + [\mathcal{E}(exp2, \rho)]]$$

$$\frac{\mathcal{E}(exp2, \rho) \neq 0}{\mathcal{E}((/(exp1, exp2), \rho)) = [[\mathcal{E}(xep1, \rho)]/[\mathcal{E}(exp2, \rho)]]}$$

3.- Agrega un predicado de igualdad numerica *equal*? y predicados de orden *greater*? y *less*? al conjunto de operaciones de lenguajes LET.
Respuesta:

Especificacion Lexica :

Equal? = Equal?

Greater? = >

less? = <

Especificacion Sintactica :

Concreta:Expression ⇒ equal?(Expression,Expression)

Concreta:Expression ⇒<(Expression,Expression)

Concreta:Expression ⇒>(Expression,Expression)

Abstracta:(Equal?-exp exp1 exp2)

Abstracta:(Greater?-exp exp1 exp2)

Abstracta:(less?-exp exp1 exp2)

Especificacion Semantica :

Regla:

[(value-of(equal?-exp exp1 exp2)
— (bool-val (equal? (expval→num (value-of exp1 env))
— (expval→num (value-of exp2 env)))))]

[(value-of(greater?-exp exp1 exp2)
— (bool-val (> (expval→num (value-of exp1 env))
— (expval→num (value-of exp2 env)))))]

[(value-of(less?-exp exp1 exp2)
— (bool-val (< (expval→num (value-of exp1 env))
— (expval→num (value-of exp2 env)))))]

Regla Breve:

$$\mathcal{E}(Equal?(exp1, exp2), \rho) \begin{cases} \top si\ \mathcal{E}(-(exp1, \rho)(exp2, \rho)) = 0 \\ \bot si\ \mathcal{E}(-(exp1, \rho)(exp2, \rho)) \neq 0 \end{cases}$$

$$\mathcal{E}(> (exp1, exp2), \rho) \begin{cases} \top si\ \mathcal{E}(-(exp1, \rho)(exp2, \rho)) > 0 \\ \bot si\ \mathcal{E}(-(exp1, \rho)(exp2, \rho)) \ngtr 0 \end{cases}$$

$$\mathcal{E}(< (exp1, exp2), \rho) \begin{cases} \top si\ \mathcal{E}(-(exp1, \rho)(exp2, \rho)) < 0 \\ \bot si\ \mathcal{E}(-(exp1, \rho)(exp2, \rho)) \nless 0 \end{cases}$$

4.- Agrega opreaciones de procesamiento de listas al lenguaje, incluyendo *cons*,*car*,*cdr*,*null*? y *emptylist*. Una lista debe poder contener cualquier valor expresado, in-

cluyendo otra lista Respuesta:

Especificacion Lexica :

cons=cons

car=car

cdr=cdr

null?=null?

emptylist=emptylist

Especificacion Sintactica :

Concreta: Expression ⇒ cons(Expression,Expression)

Concreta: Expression ⇒ car(Expression)

Concreta: Expression ⇒ cdr(Expression)

Concreta: Expression ⇒ null?(Expression)

Concreta: Expression ⇒ emptylist()

Abstracta: (cons-exp exp1 exp2)

Abstracta: (car-exp exp1)

Abstracta: (cdr-exp exp1)

Abstracta: (null?-exp exp1)

Abstracta: (emptylist-exp())

Especificacion Semantica :

Regla:

(Value of (cons-exp exp1 exp2) =
(pair-val (value-of exp1)
(value-of exp2)))

(Value of (car-exp (cons-exp exp1 exp2)) =
(value-of exp1))

(Value of (cdr-exp (cons-exp exp1 exp2)) =
(value-of exp2))

(Value of (null?-exp exp1) =
(bool-val (if (= exp1 emptylist) $\top$ $\bot$)))

(Value of (emptylist-exp ()) =
(emptylist))

Regla Breve:

$$\frac{\mathcal{E}(\ exp1,\ \rho) = val1\ \ \mathcal{E}(exp2, \rho) = val2}{\mathcal{E}(cons(exp1, exp2), \rho) = pair(val1, val2)}$$

$$\frac{\mathcal{E}(\ exp1,\ \rho) = pair(exp1, exp2)}{\mathcal{E}(car(exp1), \rho) = exp1}$$

$$\frac{\mathcal{E}\ (exp1,\ \rho) = pair(exp1, exp2)}{\mathcal{E}(cdr(exp1), \rho) = exp2}$$

$$\mathcal{E}(null?(exp1), \rho) \begin{cases} \top si\ \mathcal{E}(exp1, \rho) = emptylist \\ \bot si\ \mathcal{E}(exp1, \rho) \neq emptylist \end{cases}$$

$$\mathcal{E}(emptylist, \rho) = emptylist$$

5.- Agrega una operacion *list* al lenguaje. Esta operacion debe tomar cualquier cantidad de argumentos y regresar un valor expresado de la lista de sus valores.
Respuesta:

Especificacion Lexica :

list=list

Especificacion Sintactica :

Concreta:

Expression = List(Expressions)

Expressions = $\mathcal{E}$ | Exp1

Exp1 = Expression | Expression,Exp1

Abstracta: (list-exp exps)

Especificacion Semantica :

Regla:

(Value-of (list-exp exps) =
(if (null? (value-of exps))
(emptylist)
(pair-val (value-of (car-exp exps)) (value-of (cdr-exp exps)))))

Regla breve:

$$\frac{\mathcal{E}\ (null?(exps), \rho) = \top}{\mathcal{E}(list(exps)\rho) = emptylist}$$

$$\frac{\mathcal{E}\ (exp1, \rho) = val1,\ \mathcal{E}(list(exp2, ...)\rho) = val2}{\mathcal{E}(list(exp1, exp2, ..., expn), \rho) = pair(val1, val2)}$$

7.-Incorpora al lenguaje expresiones *cond*. Usa la gramatica

$$Expression \Rightarrow condExpression \Rightarrow Expression * end$$

En esta expresión, las expresiones de los lados izquierdos de los $\Rightarrow$ son evaluadas en orden hasta que una de ellas regresa un valor verdadero. Entonces el valor de toda la expresión es el valor de la expresión correspondiente al lado derecho de esa $\Rightarrow$. Si ninguno de los lados izquierdos es verdadero, la expresión debe reportar un error.
Respuesta:

Especificacion Lexica :

Cond=Cond

End=End

$\Rightarrow = \Rightarrow$

Especificacion Sintactica :

Concreta:

$$Expression \Rightarrow condExpression \Rightarrow Expression * end$$

Abstracta: (cond-exp exps)
Especificacion Semantica :

Regla:

(Value-of (cond-exp exps) =
(if (value-of exp1) (value-of exp2)
(if (value-of exp3) (value-of exp4)
...
(if (value-of expn-1) (value-of expn) error))))

Regla breve :

$$\frac{\mathcal{E}\ (exp1, \rho) = \top,\ \mathcal{E}(exp2, \rho) = x}{\mathcal{E}(cond(exp1, exp2, ..., expn), \rho) = x}$$

$$\frac{\mathcal{E}\ (exp1, \rho) = \bot,\ \mathcal{E}(cond(exp2, ...), \rho) = x}{\mathcal{E}(cond(exp1, exp2, ..., expn), \rho) = x}$$

$$\frac{\mathcal{E}(cond(exp2, ...), \rho) = error}{\mathcal{E}(cond(exp1, exp2, ..., expn), \rho) = error}$$

8.- Cambia los valores del lenguaje para que los enteros sean los unicos valores expresados. Modifica if para que le valor de 0 sea tratado como falso y todos los otros sean tratados como verdaders. Modifica los predicados de manera consistente.
Respuesta:

Especificacion Lexica :

Queda igual

Especificacion Sintactica :

Concreta: Queda igual

Abstracta: Queda igual

Especificacion Semantica :

Regla:

(value-of(if-exp exp1 exp2 exp3)
(if (not (= (value-of exp1 env) 0))
(value-of exp2 env)
(value-of exp3 env)))

Regla breve:

$$\mathcal{E}(zero?\ (exp1)\ ,\rho) = \begin{cases} \top\ si\ \mathcal{E}(exp1,\rho) \neq 0 \\ \bot\ si\ \mathcal{E}(exp1,\rho) = 0 \end{cases}$$

$$\mathcal{E}(if\ exp1\ then\ exp2\ else\ exp3,\rho) = \begin{cases} \mathcal{E}(exp2,\rho) si\ \mathcal{E}(exp1,\rho) = 0 \\ \mathcal{E}(exp3,\rho) si\ \mathcal{E}(exp1,\rho) \neq 0 \end{cases}$$

9.-Como una alternativa al ejercicio anterior, agrega una nueva categoria sintáctica $Bool - exp$ de expresiones booleanas al lenguaje. Cambia la producción para expresiones condicionales para que sea

$$Expression \Rightarrow if\ Bool - exp\ then\ Expression\ else\ Expression$$

Escribe Producciones apropiadas para Bool-exp y especifica su semantica con Value-of-bool-exp (Puedes abreviarlo como B). ¿En dónde terminan estando los predicados del ejercico 3 con este cambio?

Respuesta:

Especificacion Lexica :

Queda igual

Especificacion Sintactica :

Concreta: Expression ⇒ if bool-exp then Expression else Expression

Concreta: Bool-exp ⇒ $\top$

Concreta: Bool-exp ⇒ $\bot$

Concreta: Bool-exp ⇒ zero?(Expression)

Concreta: Bool-exp ⇒ equal?(Expression,Expression)

Concreta: Bool-exp ⇒< (Expression,Expression)

Concreta: Bool-exp ⇒> (Expression,Expression)

Abstracta: (if-exp bool-exp exp2 exp3)

Abstracta: (bool-exp bool)

Abstracta: True-exp

Abstracta: False-exp

Abstracta: (Zero?-boolexp exp1)

Abstracta: (equal?-boolexp exp1, exp2)

Abstracta: (Greater?-boolexp exp1, exp2)

Abstracta: (less?-boolexp exp1, exp2)

Especificacion Semantica :

(value-of (bool-exp B) = (bool-val B))

$$\mathcal{E}(if\, boolexp\, then\, exp2\, else\, exp3, \rho) = \begin{cases} \mathcal{E}(exp2, \rho) si\ \mathcal{E}(boolexp, \rho) = True exp \\ \mathcal{E}(exp3, \rho) si\ \mathcal{E}(boolexp, \rho) = False exp \end{cases}$$

10.-Modifica la implementacion del intérprete agragando una nueva operacion *print* que tome un argumento, lo imprime y regresa el entero 1. ¿Por qué esta operacion no es expresable en nuestro metodo de especificacion formal?
Respuesta:

La operacion *print* no puede ser expresado de manera sencilla en los metodos utilizados de especificacion formal debido a el efecto que tiene la operacion *print* dentro del interprete, teniendo un efecto que no se puede defiinir de manera sencilla dentro de la especificaciones, siendo esta la de imprimir en la consola, la especificacion formal se centra en la relacion logica de entradas y salidas de las operaciones, el imprimir no se puede modelar facilmente en terminos de entradas y salidas predecibles.

El imprimir agrega un componente no determinista a la especificacion formal, ya que depende directamente del entorno externo (la consola), alejandose de las propiedades matematicas y logicas de las funciones.

11.-Extiende el lenguaje para que las expresiones *let* puedan vincular una cantidad arbitraria de variables, usando la produccion,

$$Expression \Rightarrow let\, \{identifier = Expression*\}\, in\, Expression$$

Respuesta:

Especificacion Lexica :

Queda igual

Especificacion Sintactica :

Concreta: Expression ⇒ let { identifier = Expression}* in Expression

Concreta: iden-Exps ⇒ $\mathcal{E}$ | iden-Exp

Concreta: iden-Exp ⇒ identifier Expression | identifier Expression iden-Exp

Abstracta: (let-exp iden-Exps rest)

Especificacion Semantica :

Reglas:

(value-of (let-exps iden-Exps body) ρ) =
(value-of body (extend-env ρ identifier1 (value-of Expression1 ρ))
(extend-env ρ identifier2 (value-of Expression2 ρ))
...)

Reglas breves:

$$\frac{\mathcal{E}\ (exp1, \rho) = val1\ , ...,\ \mathcal{E}(expN, \rho) = valN}{\mathcal{E}(let\{id = exp1, ..., idN = expN\}\ in\ body, \rho) = \mathcal{E}(body, [id1 : val1, ..., idN : valN]\rho)}$$

12.- Extiende el lenguaje con unan expression Let* que funciona como en racket.
Respuesta:

Especificacion Lexica :

Let* = Let*

Especificacion Sintactica :

Concreta: Expression ⇒ let { identifier = Expression}* in Expression

Concreta: iden-Exps ⇒ $\mathcal{E}$ | iden-Exp

Concreta: iden-Exp ⇒ identifier Expression | identifier Expression iden-Exp

Abstracta: (let-exp* iden-Exps rest)

Especificacion Semantica :

Reglas:

(value-of (let*-exps iden-Exps body) ρ) =
(value-of body (extend-env ρ identifier1 (value-of Expression1 ρ))
(extend-env ρ identifier2 (value-of Expression2
(extend-env ρ identifier1 (value-of Expression1 ρ))))
...
(extend-env ρ identifierN (value-of ExpressionN
(extend-env ρ identifierN-1 (value-of ExpressionN-1 ...
(extend-env ρ identifier1 (value-of Expression1 ρ)))))))

Reglas breves:

$$\frac{\mathcal{E}(exp_1, \rho) = val_1,\ \ldots,\ \mathcal{E}(exp_N, [id_1 : val_1,\ \ldots,\ id_{N-1} : val_{N-1}]\rho) = val_N}{\mathcal{E}(let*\ \{id_1 = exp_1,\ \ldots,\ id_N = exp_N\}\ in\ body, \rho) = \mathcal{E}(body, id_1 : val_1,\ \ldots,\ id_N : val_N)\rho}$$

13.- Agrega una expresion al lenguaje de acuerdo a la siguiente regla

$$Expression \Rightarrow unpack\ \ \{Identifier\}* = Expression\ in\ Expression$$

tal que unpack x y z = lst in ... vincula x , y y z a los elementos de lst si lst es una lista con exactamente tres elementos, reportando un error en otro caso.
Respuesta:

Especificacion Lexica :

unpack = unpack

Especificacion Sintactica :

Concreta: Expression ⇒ unpack {Identifier}* = Expression in Expression

Concreta: iden-list ⇒ $\mathcal{E}$ | iden-list iden-Exp

Concreta: iden-Exp ⇒ Identifier | Identifier, iden-Exp

Abstracta: (Unpack-exp Exp1 exp body)

Especificacion Semantica :

Reglas:

(value-of (unpack-exps iden-Exps exp body) ρ) =
(match (value-of exp ρ)
((pair val1 rest) (value-of body
(extend-env ρ id1 val1) (extend-env ρ id2 rest) ... (extend-env ρ idN rest)))
(emptylist (value-of body
(extend-env ρ id1 emptylist) (extend-env ρ id2 emptylist) ... (extend-env ρ idN emptylist)))
(else error))

Reglas breves:

$$\frac{\mathcal{E}(Exp1, \rho) = pair(val1, rest)}{\mathcal{E}(unpack((id...idN)exp1body)) = \mathcal{L}(list(id, ..., idN), pair(val1, rest), body, \rho)}$$

$$\frac{\mathcal{E}(Exp, \rho) = val\ \mathcal{E}(body, \rho) = Val}{\mathcal{E}(unpack(()\ exp\ body)\rho) = val}$$

$$\frac{\mathcal{E}(body, \rho) = Val}{\mathcal{E}(unpack((\ )\ (\ )\ body)\ rho) = val}$$

$$\frac{\mathcal{E}Exp1, \rho = emptylist}{\mathcal{E}(\text{Exp1}, \rho) = [\text{Exp1} : \text{emptylist}]\rho}$$

$$\frac{\mathcal{E}(Exp1, \rho) \neq \text{pair}(val1, \text{rest}) \quad \mathcal{E}(Exp1, \rho) \neq \text{list}(...)}{\mathcal{E}(\text{unpack}((id...idN)\ Exp1\ body)) = \text{error}}$$