UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

Facultad de Ciencias

Lenguajes de Programación

MiniLisp

Proyecto 1

Presenta:

Lugo Díaz Ordaz Gretel Alexandra

Ramírez Juárez María Fernanda

Rojo Peña Manuel Ianluck

Profesor:

Manuel Soto Romero

Ayudantes:

Diego Méndez Medina

Erick Daniel Arroyo Martínez

Grupo: 7121, 2026-1

Fecha de entrega:

11 de octubre, 2025

Índice

1. Semántica Operacional **3**

1.1. Semántica Estructural (Paso pequeño) 3

1.1.1. Sistema de transición 3

1.1.2. Estados Finales en MiniLisp 4

1.1.3. Reglas de evaluación 6

1.2. Intérprete para MiniLisp 10

1.2.1. Función paso pequeño en MiniLisp 10

2. Resultados **11**

2.1. Menú interactivo 11

2.2. Funciones de prueba 11

2.2.1. Suma primeros n números naturales 11

2.2.2. Factorial 11

2.2.3. Fibonacci 11

2.2.4. Función `map` para listas 11

2.2.5. Función `filter` para listas 11

3. Conclusiones **13**

Bibliografía **14**

Capítulo 1

Semántica Operacional

Una vez hemos establecido nuestro sintaxis del lenguaje en núcleos, lo siguiente a realizar darle el significado a estos programas. Para ello recurrimos a la semántica operacional, un enfoque que describe la dinámica de ejecución de los programas mediante un conjunto de reglas de inferencia.

Desarrollar la introducción y explicación de la formalización de la semántica y semántica operacional.

La semántica operacional...

Continuar con la definicion de semantica operacional .

Existen dos estilos principales:
Semantica Natural conocido como paso grande (Big-step)

Definir brevemente lo que es paso pequeño y quien lo definio.

Semantica Estructural conocida como paso pequeño (Small-step)

Definir brevemente lo que es paso grande y quien lo definio.

Para nuestro lenguaje nos enfocaremos en este último.

1.1. Semántica Estructural (Paso pequeño)

Se le conoce también como semántica de paso pequeño o de transición describe paso a paso la ejecución mostrando los cómputos que genera en cada paso individualmente.

Aqui ya describimos toda la historia de paso pequeño, qué significa paso pequeño y por que debemos impleemntarlo

1.1.1. Sistema de transición

Dar la definicion y explicacion de un sistema de transision y como la aplicaremos a nustro proyecto. Tambien mencionamos y describimos los estados finales pero solo eso, no los definimos para el lenguaje

Explicar lo que es la cerradura

1.1.2. Estados Finales en MiniLisp

Definimos los estados finales para el lenguaje MiniLisp -0.2cm:

$$F = \{\, Num_V(n) \mid n \in \mathbb{Z} \,\} \cup \{\, Boolean_V(b) \mid b \in \{True, False\} \,\} \cup \{\, Pair_V(f,s) \mid f,s \in F \,\} \cup$$
$$\{\, Cons_V(h,t) \mid h,t \in C \,\} \cup \{\, Nil_V \mid \text{ es la lista vacía}\} \cup$$
$$\{\, Closure(f,\varepsilon) \mid f \text{ es una función y } \varepsilon \text{ es un ambiente léxico}\}$$

Intuitivamente implementamos los estados finales en nuestro lenguaje como un tipo de dato en Haskell:

```haskell
module ASV where

-- ASA Values
data ASV
  = VarV String
  | NumV Int
  | BoolV Bool
  | NiV
  | PairV ASV ASV
  | ConV ASV ASV
  | Closure String ASV [(String, ASV)]
  deriving (Show, Eq)
```

Código 1.1: Tipo de dato ASV, representan los estados finales

El tipo de dato `ASV` serán con el que modelaremos los estados finales del lenguaje. Sin embargo, en nuestra implementación, es necesario agregar las demás estructuras que trabajan con estos estados finales -como lo son los operadores aritméticos o las operaciones sobre pares o listas- ya que `ASV` sigue modelando una estructura de tipo árbol la cual sigue el mismo principio que las estructuras `ASA`:

Una expresión es `ASV` si y solo si sus hijos también son `ASV`.

Por lo que debemos hacer que nuestros Para diferenciar entre los verdaderos estados finales y las demás estructuras de tipo `ASV` será que nos referimos a los finales como ***valores canónicos***.[1] Lo que nos queda en la nueva definición del tipo de dato `ASV`:

```haskell
-- ASA Values
data ASV
  = VarV String
  | NumV Int
  | BoolV Bool
  | NiV
  | AddV ASV ASV
  | SubV ASV ASV
  | MulV ASV ASV
  | DiV ASV ASV
  | SqrtV ASV
```

[1] Más adelante enfatizaremos en como los diferenciamos entre Valores `ASV` y expresiones `ASV` en Haskell.

```
  | NotV ASV
  | EqualV ASV ASV
  | LessV ASV ASV
  | GreaterV ASV ASV
  | DiffV ASV ASV
  | LeqV ASV ASV
  | GeqV ASV ASV
  | PairV ASV ASV
  | FstV ASV
  | SndV ASV
  | IfV ASV ASV ASV
  | FunV String ASV
  | AppV ASV ASV
  | ConV ASV ASV
  | HeadV ASV
  | TailV ASV
  | Closure String ASV [(String, ASV)]
  deriving (Show, Eq)
```

Código 1.2: Tipo de dato ASV completo

Ya que hemos definido el tipo de dato `ASV` veamos como convertimos nuestras estructuras `AST` a estados finales `ASV`:

```
{-- Convertimos los AST a estados finales ASV --}
toFinalState :: AST -> ASV
toFinalState (VarC x) = VarV x
toFinalState (NumC n) = NumV n
toFinalState (BoolC b) = BoolV b
toFinalState (AddC i d) = AddV (toFinalState i) (toFinalState d)
toFinalState (SubC i d) = SubV (toFinalState i) (toFinalState d)
toFinalState (MulC i d) = MulV (toFinalState i) (toFinalState d)
toFinalState (DivC i d) = DiV (toFinalState i) (toFinalState d)
toFinalState (SqrtC n) = SqrtV (toFinalState n)
toFinalState (NotC x) = NotV (toFinalState x)
toFinalState (EqualC i d) = EqualV (toFinalState i) (toFinalState d)
toFinalState (LessC i d) = LessV (toFinalState i) (toFinalState d)
toFinalState (GreaterC i d) = GreaterV (toFinalState i) (toFinalState d)
toFinalState (DiffC i d) = DiffV (toFinalState i) (toFinalState d)
toFinalState (LeqC i d) = LeqV (toFinalState i) (toFinalState d)
toFinalState (GeqC i d) = GeqV (toFinalState i) (toFinalState d)
toFinalState (PairC f s) = PairV (toFinalState f) (toFinalState s)
toFinalState (FstC p) = FstV (toFinalState p)
toFinalState (SndC p) = SndV (toFinalState p)
toFinalState (ConS f s) = ConV (toFinalState f) (toFinalState s)
toFinalState (HeadC p) = HeadV (toFinalState p)
toFinalState (TailC p) = TailV (toFinalState p)
toFinalState (IfC c t e) = IfV (toFinalState c) (toFinalState t) (toFinalState e)
toFinalState (FunC p b) = FunV p (toFinalState b)
toFinalState (AppC f a) = AppV (toFinalState f) (toFinalState a)
toFinalState NiL = NiV
```

Código 1.3: Función `toFinalState` que transforma los núcleos `AST` a estados finales `ASV`

La función `toFinalState` transforma cada estructura del núcleo AST en su equivalente ASV. Aunque a primera vista parezca una simple correspondencia entre constructores, su propósito es fundamental: garantizar que todas las expresiones que se evalúan dentro del intérprete sean expresadas en términos de ASV, permitiendo así que la semántica del lenguaje opere únicamente sobre estructuras homogéneas y compatibles con los valores finales del lenguaje.

1.1.3. Reglas de evaluación

Dar una introduccion de que son las reglas de evaluacion y por que son necesarias, ademas completar con la explicacion de las reglas de evaluacion. Hablar de los ambientes

- Expresiones atómicas:
  - VarV(i):

$$\frac{lookup\ i\ \varepsilon = v}{\langle VarV(i), \varepsilon\rangle \to \langle v, \varepsilon\rangle}$$

$$\frac{lookup\ i\ \varepsilon \text{ no está definido}}{\langle Varv(i), \varepsilon\rangle \to \text{Error en la ejecución de la evaluación}}$$

Aqui se puede poner la explicacion breve de lo que hace lookup

  - NumV(n):

$$\frac{}{\langle NumV(n), \varepsilon\rangle \to \langle NumV(n), \varepsilon\rangle}$$

  - BoolV(b):

$$\frac{}{\langle BoolV(b), \varepsilon\rangle \to \langle BoolV(b), \varepsilon\rangle}$$

  - NiV:

$$\frac{}{\langle NivV, \varepsilon\rangle \to \langle NiV, \varepsilon\rangle}$$

- AddV(i,d):

$$\frac{\langle i, \varepsilon\rangle \to \langle i', \varepsilon\rangle}{\langle AddV(i,d), \varepsilon\rangle \to \langle AddV(i',d), \varepsilon\rangle}$$

$$\frac{\langle d, \varepsilon\rangle \to \langle d', \varepsilon\rangle}{\langle AddV(NumV(n),d), \varepsilon\rangle \to \langle Add(NumV(n),d'), \varepsilon\rangle}$$

$$\frac{}{\langle AddV(NumV(n),NumV(m)), \varepsilon\rangle \to \langle NumV(n +_{\mathbb{Z}} m), \varepsilon\rangle}$$

- SubV(i,d):

$$\frac{\langle i, \varepsilon\rangle \to \langle i', \varepsilon\rangle}{\langle SubV(i,d), \varepsilon\rangle \to \langle SubV(i',d), \varepsilon\rangle}$$

$$\frac{\langle d, \varepsilon\rangle \to \langle d', \varepsilon\rangle}{\langle SubV(NumV(n),d), \varepsilon\rangle \to \langle SubV(NumV(n),d'), \varepsilon\rangle}$$

$$\frac{}{\langle SubV(NumV(n),NumV(m)), \varepsilon\rangle \to \langle NumV(n -_{\mathbb{Z}} m), \varepsilon\rangle}$$

- MulV(i,d):

$$\frac{\langle i, \varepsilon \rangle \to \langle i', \varepsilon \rangle}{\langle MulV(i, d), \varepsilon \rangle \to \langle MulV(i', d), \varepsilon \rangle}$$

$$\frac{\langle d, \varepsilon \rangle \to \langle d', \varepsilon \rangle}{\langle MulV(NumV(n), d), \varepsilon \rangle \to \langle MulV(NumV(n), d'), \varepsilon \rangle}$$

$$\frac{}{\langle MulV(NumV(n), NumV(m)), \varepsilon \rangle \to \langle NumV(n *_{\mathbb{Z}} m), \varepsilon \rangle}$$

- DiV(i,d):

$$\frac{\langle i, \varepsilon \rangle \to \langle i', \varepsilon \rangle}{\langle DiV(i, d), \varepsilon \rangle \to \langle DiV(i', d), \varepsilon \rangle}$$

$$\frac{\langle d, \varepsilon \rangle \to \langle d', \varepsilon \rangle}{\langle DiV(NumV(n), d), \varepsilon \rangle \to \langle DiV(NumV(n), d'), \varepsilon \rangle}$$

$$\frac{}{\langle DiV(NumV(n), NumV(0)), \varepsilon \to \text{Error en la ejecución de la evaluación}}$$

$$\frac{}{\langle DiV(NumV(n), NumV(m)), \varepsilon \rangle \to \langle NumV(n /_{\mathbb{Z}} m), \varepsilon \rangle} \quad (m \neq 0)$$

- SqrtV(n):

$$\frac{\langle n, \varepsilon \rangle \to \langle n', \varepsilon \rangle}{\langle SqrtV(n), \varepsilon \rangle \to \langle SqrtV(n'), \varepsilon \rangle}$$

$$\frac{n < 0}{\langle SqrtV(NumV(n)), \varepsilon \to \text{Error en la ejecución de la evaluación}}$$

$$\frac{}{\langle SqrtV(NumV(n)), \varepsilon \rangle \to \langle NumV(\sqrt{n}_{\mathbb{N}}), \varepsilon \rangle} \quad (n \geq 0)$$

- NotV(b):

$$\frac{\langle b, \varepsilon \rangle \to \langle n', \varepsilon \rangle}{\langle NotV(b), \varepsilon \rangle \to \langle NotV(b'), \varepsilon \rangle}$$

$$\frac{}{\langle NotV(BoolV(b)), \varepsilon \rangle \to \langle BoolV(\neg_{\mathbb{P}} b) \varepsilon \rangle}$$

- EqualV(i,d):

$$\frac{\langle i, \varepsilon \rangle \to \langle i', \varepsilon \rangle}{\langle EqualV(i, d), \varepsilon \rangle \to \langle EqualV(i', d), \varepsilon \rangle}$$

$$\frac{\langle d, \varepsilon \rangle \to \langle d', \varepsilon \rangle}{\langle EqualV(NumV(n), d), \varepsilon \rangle \to \langle EqualV(NumV(n), d'), \varepsilon \rangle}$$

$$\frac{}{\langle EqualV(NumV(n), NumV(m)), \varepsilon \rangle \to \langle BoolV(n =_{\mathbb{Z}} m), \varepsilon \rangle}$$

- LessV(i,d)

$$\frac{\langle i, \varepsilon \rangle \to \langle i', \varepsilon \rangle}{\langle LessV(i, d), \varepsilon \rangle \to \langle LessV(i', d), \varepsilon \rangle}$$

$$\frac{\langle d, \varepsilon \rangle \to \langle d', \varepsilon \rangle}{\langle LessV(NumV(n), d), \varepsilon \rangle \to \langle LessV(NumV(n), d'), \varepsilon \rangle}$$

$$\frac{}{\langle LessV(NumV(n), NumV(m)), \varepsilon \rangle \to \langle BoolV(n <_{\mathbb{Z}} m), \varepsilon \rangle}$$

- GreaterV(i,d)

$$\frac{\langle i, \varepsilon\rangle \to \langle i', \varepsilon\rangle}{\langle GreaterV(i,d), \varepsilon\rangle \to \langle GreaterV(i',d), \varepsilon\rangle}$$

$$\frac{\langle d, \varepsilon\rangle \to \langle d', \varepsilon\rangle}{\langle GreaterV(NumV(n),d), \varepsilon\rangle \to \langle GreaterV(NumV(n),d'), \varepsilon\rangle}$$

$$\frac{}{\langle GreaterV(NumV(n),NumV(m)), \varepsilon\rangle \to \langle BoolV(n >_{\mathbb{Z}} m), \varepsilon\rangle}$$

- DiffV(i,d)

$$\frac{\langle i, \varepsilon\rangle \to \langle i', \varepsilon\rangle}{\langle DiffV(i,d), \varepsilon\rangle \to \langle DiffV(i',d), \varepsilon\rangle}$$

$$\frac{\langle d, \varepsilon\rangle \to \langle d', \varepsilon\rangle}{\langle DiffV(NumV(n),d), \varepsilon\rangle \to \langle DiffV(NumV(n),d'), \varepsilon\rangle}$$

$$\frac{}{\langle DiffV(NumV(n),NumV(m)), \varepsilon\rangle \to \langle BoolV(n \neq_{\mathbb{Z}} m), \varepsilon\rangle}$$

- LeqV(i,d)

$$\frac{\langle i, \varepsilon\rangle \to \langle i', \varepsilon\rangle}{\langle LeqV(i,d), \varepsilon\rangle \to \langle LeqV(i',d), \varepsilon\rangle}$$

$$\frac{\langle d, \varepsilon\rangle \to \langle d', \varepsilon\rangle}{\langle LeqV(NumV(n),d), \varepsilon\rangle \to \langle LeqV(NumV(n),d'), \varepsilon\rangle}$$

$$\frac{}{\langle LeqV(NumV(n),NumV(m)), \varepsilon\rangle \to \langle BoolV(n \leq_{\mathbb{Z}} m), \varepsilon\rangle}$$

- GeqV(i,d)

$$\frac{\langle i, \varepsilon\rangle \to \langle i', \varepsilon\rangle}{\langle GeqV(i,d), \varepsilon\rangle \to \langle GeqV(i',d), \varepsilon\rangle}$$

$$\frac{\langle d, \varepsilon\rangle \to \langle d', \varepsilon\rangle}{\langle GeqV(NumV(n),d), \varepsilon\rangle \to \langle GeqV(NumV(n),d'), \varepsilon\rangle}$$

$$\frac{}{\langle GeqV(NumV(n),NumV(m)), \varepsilon\rangle \to \langle BoolV(n \geq_{\mathbb{Z}} m), \varepsilon\rangle}$$

Como se puede ver en las reglas para los comparadores, la regla final cuando ambas expresiones del comparador son el resultado deriva en

- PairV(f,s):

$$\frac{\langle f, \varepsilon\rangle \to \langle f', \varepsilon\rangle}{\langle PairV(f,s), \varepsilon\rangle \to \langle PairV(f',s), \varepsilon\rangle}$$

$$\frac{\langle s, \varepsilon\rangle \to \langle s', \varepsilon\rangle}{\langle PairV(v_f,s), \varepsilon\rangle \to \langle PairV(v_f,s'), \varepsilon\rangle}$$

$$\frac{v_f, v_s \text{ son valores canónicos}}{\langle PairV(v_f,v_s), \varepsilon\rangle \to \langle PairV(v_f,v_s), \varepsilon\rangle}$$

- FstV(p):

$$\frac{\langle p, \varepsilon\rangle \to \langle p', \varepsilon\rangle}{\langle FstV(p), \varepsilon\rangle \to \langle FstV(p'), \varepsilon\rangle}$$

$$\frac{v_1, v_2 \text{ son valores canónicos}}{\langle FstV(PairV(v_1, v_2)), \varepsilon\rangle \to \langle v_1, \varepsilon\rangle}$$

- SndV(p):

$$\frac{\langle p, \varepsilon\rangle \to \langle p', \varepsilon\rangle}{\langle SndV(p), \varepsilon\rangle \to \langle SndV(p'), \varepsilon\rangle}$$

$$\frac{v_1, v_2 \text{ son valores canónicos}}{\langle SndV(PairV(v_1, v_2)), \varepsilon\rangle \to \langle v_2, \varepsilon\rangle}$$

- IfV(c,t,e), IfV solo evalúa la condicional hasta llegar a un `BoolV` mas no evalúa el **then** o **else** dependiendo del resultado de la condición, solo se encarga de retornar alguno de los dos dependiendo del caso:

$$\frac{\langle c, \varepsilon\rangle \to \langle c', \varepsilon\rangle}{\langle IfV(c, t, e), \varepsilon\rangle \to \langle IfV(c', t, e), \varepsilon\rangle}$$

$$\frac{}{\langle IfV(BoolV(\#t), t, e), \varepsilon\rangle \to \langle t, \varepsilon\rangle}$$

$$\frac{}{\langle IfV(BoolV(\#f), t, e), \varepsilon\rangle \to \langle e, \varepsilon\rangle}$$

- ConV(i,d):

$$\frac{\langle i, \varepsilon\rangle \to \langle i', \varepsilon\rangle}{\langle ConV(i, d), \varepsilon\rangle \to \langle ConV(i', d), \varepsilon\rangle}$$

$$\frac{v_i \text{ es valor canónico } \langle d, \varepsilon\rangle \to \langle d', \varepsilon\rangle}{\langle ConV(v_i, d), \varepsilon\rangle \to \langle ConV(v_i, d'), \varepsilon\rangle}$$

$$\frac{v_i, v_d \text{ son valores canónicos}}{\langle ConV(v_i, v_d), \varepsilon\rangle \to \langle ConV(v_i, v_d), \varepsilon\rangle}$$

- HeadV(l):

$$\frac{\langle p, \varepsilon\rangle \to \langle p', \varepsilon\rangle}{\langle HeadV(l), \varepsilon\rangle \to \langle HeadV(l'), \varepsilon\rangle}$$

$$\frac{}{\langle HeadV(ConV(v_i, v_d)), \varepsilon\rangle \to \langle v_i, \varepsilon\rangle}$$

- TailV(l);

$$\frac{\langle l, \varepsilon\rangle \to \langle l', \varepsilon\rangle}{\langle TailV(l), \varepsilon\rangle \to \langle TailV(l'), \varepsilon\rangle}$$

$$\frac{v_d \text{ es valor pero } v_d = ConV(i, d) \;\; v_d \to v_d'}{\langle TailV(ConV(v_i, v_d)), \varepsilon\rangle \to \langle TailV(v_i, v_d'), \varepsilon\rangle}$$

$$\frac{v_d \text{ es valor pero} v_d \neq ConV(i, d)}{\langle TailV(ConV(v_i, v_d)), \varepsilon\rangle \to \langle v_d, \varepsilon\rangle}$$

- FunV(p,c):

$$\frac{}{\langle FunV(p,c),\varepsilon\rangle \to \langle Closure(FunV(p,c),\varepsilon),\varepsilon\rangle}$$

- AppV(f,a):

$$\frac{\langle f,\varepsilon\rangle \to \langle f',\varepsilon\rangle}{\langle AppV(f,a),\varepsilon\rangle \to \langle AppV(f',a),\varepsilon\rangle}$$

$$\frac{\langle a,\varepsilon\rangle \to \langle a',\varepsilon\rangle}{\langle AppV(Closure(FunV(p,c),\varepsilon'),a),\varepsilon\rangle \to \langle AppV(Closure(FunV(p,c),\varepsilon'),a'),\varepsilon\rangle}$$

$$\frac{\text{es un valor canónico}}{\langle AppV(Closure(FunV(p,c),\varepsilon'),),\varepsilon\rangle \to \langle c,\varepsilon'[p \leftarrow]\rangle}$$

Puede que sea buena idea dar un cierre a esto antes de entrar en Haskell

1.2. Intérprete para MiniLisp

Una vez definimos formalmente lo que será la ***Semántica Operacional Estrcutural*** para nuestro lenguaje podemos programar el interprete del mismo que será el encargado de aplica la evaluación al programa del usuario, más precisamente a las expreciones `ASV` que ya han depurado el programa original.

creo podemos extendernos mas aqui

1.2.1. Función paso pequeño en MiniLisp

Para nuestro lenguaje definimos la función de evaluación como sigue:

Sin embargo, esta implementación no nos sirve para la aplicación de funciones derivada de `LetRec`, pues veamos el resultado de evaluar el `LetRec`:

```
[MiniLisp]> (letrec (sum (lambda (n) (if0 n 0 (+ n (sum (- n 1)))))) (sum 3))
[Error]: Var 'Z' no definida
CallStack (from HasCallStack):
  error, called at ./Interprete.hs:154:18 in main:Interprete
[MiniLisp]>
```

Capítulo 2

Resultados

Bien, una vez hemos visto toda la teoría de nuestro MiniLisp y además de que hemos mostrado el código que lo implementa en Haskell, veamos como funciona:

2.1. Menú interactivo

2.2. Funciones de prueba

2.2.1. Suma primeros n números naturales

2.2.2. Factorial

2.2.3. Fibonacci

2.2.4. Función `map` para listas

2.2.5. Función `filter` para listas

Capítulo 3

Conclusiones

Bibliografía

[1] https://weblibrary.mila.edu.my/upload/ebook/engineering/2017_Book_FoundationsOfProgr

[2] Aho, A. V., Lam, S. M., Sethi, R., & Ullman, J. D. Compilers: Principles, Techniques, and Tools. [Second Edition]. 2007.

[3] Disponible en: https://lambdasspace.github.io/LDP/notas/ldp_n04.pdf

[4] Disponible en: https://lambdasspace.github.io/LDP/notas/ldp_n05.pdf

[5] Documentación Haskell. Disponible en: https://www.haskell.org

[6] Documentación Alex(Haskell) The Alex Lexer Generator for Haskell Programming in Haskell (Graham Hutton, 2nd Edition). Sección sobre parsers y lexers. Disponible en: https://www.haskell.org/alex/

[7] Marlow, S., Gill, A. (2009). Happy. Disponible en: https://www.haskell.org/happy/

[8] Disponible en: https://lambdasspace.github.io/LDP/notas/ldp_n08.pdf

[9] Disponible en: https://lambdasspace.github.io/LDP/notas/ldp_n09.pdf

[10] Disponible en: https://lambdasspace.github.io/LDP/notas/ldp_n10.pdf

[11] Disponible en: https://docs.racket-lang.org/reference/let.html

[12] Disponible en: https://www.lispworks.com/documentation/HyperSpec/Body/s_let_l.htm