Semántica en Paso Pequeño (Small Step):

Para nuestro lenguaje, emplearemos una máquinas abstracta CE compuesta por dos componentes principales [15]:

- Control: La expresión que se evalúa.
- Ambiente: Asociaciones entre variables y sus valores.

El uso de esta máquina permite tener una complejidad lineal en la búsqueda de los valores de variables (si se implementa en una pila o una lista, como es nuestro caso) , pues tiene un ambiente que guarda los valores de dichas variables y permite resolverlos dinámicamente [15].

Ahora, definimos el sistema de transición de paso pequeño con el uso de la máquina CE de la siguiente forma: La tupla (C, →, I, F), donde:

- C = { (e, ε) | e ∈ ASA, ε un ambiente léxico } … configuraciones
- → ⊆ C × C … la relación de reducción (entre configuraciones ⟨e, ε⟩).En small step reducimos configuraciones.
- I = ASA … estados iniciales, pues en paso pequeño todas las expresiones pueden ser estados iniciales.
- F = { Num(n) | n ∈ Z } ∪ { Bool(b) | b ∈ {True, False} } ∪ { Pair(v1, v2) | v1, v2 ∈ F } ∪ { Closure(p, c, ε0) | p : String, c ∈ ASA, ε0 es un ambiente léxico } … valores canónicos.

Expresiones del ASAV:

Dado que en la implementación del lenguaje se debe poder diferenciar entre los estados que son finales de los que no lo son, se construye el conjunto ASAV (Árbol de Sintaxis Abstracta Value) para dicho fin. Estas expresiones (valores canónicos), son las que se regresan en cada ejecución de un programa en el lenguaje.

ASAV de <Int>:

n ∈ Z

—-----------------

Num(n) ASAV

ASAV de <Bool>:

b ∈ { True, False }

—-----------------------

Bool(b)   ASAV

ASAV de los pares:

l ASAV r ASAV

—----------------------

Pair(l,r) ASAV

ASAV de las cerraduras de función:

p : String c   ASAV ε un ambiente léxico

—---

Closure(p,c,ε) ASAV

Ambientes (ε) y búsqueda de variables (Lookup):

Usamos un ambiente léxico ε que asocia identificadores a **valores canónicos** (ASAV). El ambiente se modela como una lista (o pila) de enlaces con sombreados por derecha:

- ε := [] | ε[x → v]
- Función de búsqueda (toma el **enlace más reciente**): (x, v) ∈ ε si ε = ε'[x → v] o (x, v) ∈ ε'

Reglas de Lookup:

—--------------------------
⟨Id(x), ε⟩ → ⟨v, ε⟩

si (x, v) ∈ ε (Lookup-Hit)

—---------------------

⟨Id(x), ε⟩ → error

si x ∉ dom(ε) (Lookup-Miss)

Régimen de evaluación que usaremos (para todo el small-step):

- **Estrategia: Call-by-Value (CBV)** con orden izquierda→derecha.

Primero se reduce el operando/guardia del lado izquierdo, luego el derecho, y solo cuando ambos son valores se aplica la regla base del constructor (suma, comparación, aplicación, etc.).

- **Alcance: estático (léxico).**

Las funciones se evalúan como cerraduras Closure(p, c, $ε_0$) que capturan el ambiente del punto de definición. La regla de βv sustituirá enlazando p ↦ v en $ε_0$, no en el ambiente de llamada.

- **Determinismo (esbozo):**

Con CBV y el orden L→R, para toda configuración no final ⟨e, ε⟩ existe a lo más un contexto activo y, por tanto, un siguiente paso ⟨e', ε'⟩. Esto asegura que la relación → sea determinista a nivel de un paso.

Esquemas contextuales (patrones de evaluación):

Bajo **CBV con orden izquierda→derecha**, toda construcción con subexpresiones se evalúa primero a la **izquierda** y después a la **derecha**. Para evitar repetir reglas, usamos **familias** que luego se instancian por operador.

Binarios (familia general):

$$\frac{\langle e_1, \varepsilon\rangle \to \langle e_1', \varepsilon\rangle}{\langle Bin(e_1, e_2), \varepsilon\rangle \to \langle Bin(e_1', e_2), \varepsilon\rangle}\ (\text{Bin-Left})$$

$$\frac{\langle e_2, \varepsilon\rangle \to \langle e_2', \varepsilon\rangle, \qquad v \in F}{\langle Bin(v, e_2), \varepsilon\rangle \to \langle Bin(v, e_2'), \varepsilon\rangle}\ (\text{Bin-Right})$$

Instanciación: si Bin = Add obtienes Add-Left y Add-Right, si Bin = Lt, obtienes Lt-Left/Lt-Right, etc.

Unarios (Familia general)

$$\frac{\langle e, \varepsilon\rangle \to \langle e', \varepsilon\rangle}{\langle Un(e), \varepsilon\rangle \to \langle Un(e'), \varepsilon\rangle}\ (\text{Un-Arg})$$

Instanciación: Un = Not, Un = Fst, Un = Snd, etc. (Las reglas base de cada uno van en su sección.)

Condicional (patrón del guardia):

$$\frac{\langle g, \varepsilon\rangle \to \langle g', \varepsilon\rangle}{\langle If(g, t, f), \varepsilon\rangle \to \langle If(g', t, f), \varepsilon\rangle}\ (\text{If-Guard})$$

Las bases (If-True/If-False) irán en el bloque de condicional.

Aplicación (solo el patrón; las bases van después):

$$\frac{\langle e_1, \varepsilon\rangle \to \langle e_1', \varepsilon\rangle}{\langle App(e_1, e_2), \varepsilon\rangle \to \langle App(e_1', e_2), \varepsilon\rangle}\ (\text{App-Fun})$$

$\langle e_2, \varepsilon \rangle \rightarrow \langle e_2', \varepsilon \rangle \quad v \in F$

—---(App-Arg)

$\langle App(v, e_2), \varepsilon \rangle \rightarrow \langle App(v, e_2'), \varepsilon \rangle$

La βv (aplicación por valor sobre closures) aparecerá en su bloque respectivo.

Catálogo de side-conditions:

Para mantener la semántica clara y determinista, fijamos estas condiciones (se referenciarán en las reglas base de cada operador):

- **Tipos esperados**
  - Aritmética Add, Sub, Mult, Div, Expt: ambos argumentos deben ser **Num**
  - Comparadores Eq, Lt, Le, Gt, Ge: argumentos **Num**
  - Not: argumento **Bool**
  - Fst/Snd: argumento **Pair(v_1,v_2)**
  - If: guardia **Bool(True/False)**
  - App: la **fun** debe ser **Closure(p,c,ε_0)** antes de aplicar βv

Reglas que comparten las operaciones aritméticas:

Ar = { Add, Sub, Mult, Div, Expt }

$<i, \varepsilon> \rightarrow <i', \varepsilon>$

—---------------------------------- (`Add-Left/Right`, `Sub-Left/Right`, `Mult-Left/Right`, `Div-Left/Right`, `Expt-Left/Right`)

$<Ar(i, d), \varepsilon> \rightarrow <Ar(i', d), \varepsilon>$

<d, ε> → <d’, ε>

—--(`Add-Left/Right`, `Sub-Left/Right`, `Mult-Left/Right`, `Div-Left/Right`, `Expt-Left/Right`)

<Ar(Num(n), d), ε> → <Ar(Num(n), d’), ε>

Reglas únicas de Add:

—-- (Add-Num)

<Add(Num(n), Num(m)), ε> → <Num(n +Z m), ε>

Reglas únicas de Sub:

—-- (Sub-Num)

<Sub(Num(n), Num(m)), ε> → <Num(n -Z m), ε>

.**Reglas únicas de Mult:**

—-- (Mult-Num)

<Mult(Num(n), Num(m)), ε> → <Num(n xZ m), ε>

Reglas únicas de Div:

—--- (Div-Cero)

<Div(Num(n), Num(0)), ε> → error

m ≠ 0

—---(Div-Num)

<Div(Num(n), Num(m)), ε> → <Num(n /Z m), ε>

Reglas únicas de Expt:

—--- (Expt-Cero)

<Expt(Num(n), Num(0)), ε> → <Num(1), ε>

—--- (Expt-Num)

<Expt(Num(n), Num(m)), ε> → <Num(n^m), ε>

Condicional (If):

Primero se evalúa la **guardia** (patrón ya definido como *If-Guard*). Las reglas base aplican sólo cuando la guardia es **boolean**:

—-------------------------------------(If-True)

⟨If(Bool(True), t, f), ε⟩ → ⟨t, ε⟩

—-------------------------------------(If-False)

⟨If(Bool(False), t, f), ε⟩ → ⟨f, ε⟩

Si la guardia no es **boolean**, el condicional es inválido:

—--------------------------(If-Err)

⟨If(v, t, f), ε⟩ → error

si v ∉ {Bool(True), Bool(False)}

Side-condition usada: la guardia debe ser Bool(b). Esto respeta CBV y evita ambigüedad.

Negación (Not):

Primero se reduce su argumento (patrón Un-Arg). Bases:

—---(Not-Base)

$\langle \mathrm{Not(Bool}(b)), \varepsilon \rangle \to \langle \mathrm{Bool}(\neg b), \varepsilon \rangle$

Argumento inválido:

—-------------------------(Not-Err)

$\langle \mathrm{Not}(v), \varepsilon \rangle \to \mathrm{error}$

si $v \notin \{ \mathrm{Bool(True)}, \mathrm{Bool(False)} \}$

Comparadores (Eq, Lt, Le, Gt, Ge)

Patrón contextual (hereda Bin-Left / Bin-Right):

Para cada comparador binario $\mathrm{Cmp} \in \{\mathrm{Eq, Lt, Le, Gt, Ge}\}$ usamos las familias genéricas:

$\langle e_1, \varepsilon\rangle \to \langle e_1', \varepsilon\rangle$
—---(Cmp-Left)
$\langle \mathrm{Cmp}(e_1, e_2), \varepsilon\rangle \to \langle \mathrm{Cmp}(e_1', e_2), \varepsilon\rangle$

$\langle e_2, \varepsilon\rangle \to \langle e_2', \varepsilon\rangle \; v \in F$
—---(Cmp-Right)
$\langle \mathrm{Cmp}(v, e_2), \varepsilon\rangle \to \langle \mathrm{Cmp}(v, e_2'), \varepsilon\rangle$

Reglas base (tipadas a enteros):

(Usan la side-condition global: los comparadores trabajan sobre Num, Num. Si no se cumple, es error.)

—--(Eq-Num)

⟨Eq(Num(n), Num(m)), ε⟩ → ⟨Bool(n = m), ε⟩

—--(Lt-Num)

⟨Lt(Num(n), Num(m)), ε⟩ → ⟨Bool(n < m), ε⟩

—---(Le-Num)

⟨Le(Num(n), Num(m)), ε⟩ → ⟨Bool(n ≤ m), ε⟩

—--(Gt-Num)

⟨Gt(Num(n), Num(m)), ε⟩ → ⟨Bool(n > m), ε⟩

—---(Ge-Num)

⟨Ge(Num(n), Num(m)), ε⟩ → ⟨Bool(n ≥ m), ε⟩

Tipos inválidos (mismo criterio para todos los comparadores):

—-------------------------⟨Cmp-Err),

⟨Cmp(v_1, v_2), ε⟩ →Error

Erorerror si $(v_1, v_2) \notin$ Num × Num

Esto mantiene la semántica determinista y coherente con el catálogo: sólo Num, Num son válidos para Cmp, el resultado siempre es Bool(b).

Pares y proyecciones (Pair / Fst / Snd)

Valores con pares

Extendemos F con pares anidados:

$F \ni Pair(v_1, v_2)$ si $v_1 \in F \wedge v_2 \in F$

Construcción de pares

Patrón contextual:

$$\frac{\langle e_1, \varepsilon\rangle \to \langle e_1', \varepsilon\rangle}{\langle Pair(e_1, e_2), \varepsilon\rangle \to \langle Pair(e_1', e_2), \varepsilon\rangle}\ (\text{Pair-Left})$$

$$\frac{\langle e_2, \varepsilon\rangle \to \langle e_2', \varepsilon\rangle \quad v \in F}{\langle Pair(v, e_2), \varepsilon\rangle \to \langle Pair(v, e_2'), \varepsilon\rangle}\ (\text{Pair-Right})$$

Regla base (creación de valor par):

$$\frac{}{\langle Pair(v_1, v_2), \varepsilon\rangle \to \langle Pair(v_1, v_2), \varepsilon\rangle}\ (\text{Pair-Val})$$

Observación. Si `v1,v2 ∈ F` entonces `Pair(v1,v2) ∈ F` (no hay paso).

Proyecciones: Fst/Snd

Patrón contextual (instancia de Un-Arg):

$$\frac{\langle e, \varepsilon\rangle \to \langle e', \varepsilon\rangle}{\langle Fst(e), \varepsilon\rangle \to \langle Fst(e'), \varepsilon\rangle}\ (\text{Fst-Arg})$$

$$\frac{\langle e, \varepsilon\rangle \to \langle e', \varepsilon\rangle}{\langle Snd(e), \varepsilon\rangle \to \langle Snd(e'), \varepsilon\rangle}\ (\text{Snd-Arg})$$

Reglas base (sobre valores par):

$$\frac{}{\langle Fst(Pair(v_1, v_2)), \varepsilon\rangle \to \langle v_1, \varepsilon\rangle}\ (\text{Fst-Pair})$$

$$\frac{}{\langle Snd(Pair(v_1, v_2)), \varepsilon\rangle \to \langle v_2, \varepsilon\rangle}\ (\text{Snd-Pair})$$

Errores por tipo inválido:

—----------------------(Fst-Err)

$\langle Fst(v), \varepsilon\rangle \to error$

si $v \neq Pair(v_1, v_2)$

—-----------------(Snd-Err)

$\langle Snd(v), \varepsilon\rangle \to error$

si $v \neq Pair(v_1, v_2)$

Side-condition aplicada: los argumentos de Fst/Snd deben ser **valores par**. Esto mantiene determinismo y unifica el tratamiento de errores.

Listas

Patrón, instancia de Un-Arg:

$\langle e, \varepsilon\rangle \to \langle e', \varepsilon\rangle$
—------------------------------------(Head-Arg)
$\langle Head(e), \varepsilon\rangle \to \langle Head(e'), \varepsilon\rangle$

$\langle e, \varepsilon\rangle \to \langle e', \varepsilon\rangle$
—-------------------------------(Tail-Arg)
$\langle Tail(e), \varepsilon\rangle \to \langle Tail(e'), \varepsilon\rangle$

Bases:
—--------------------------------------(Head-Cons)
$\langle Head(Cons(v, xs)), \varepsilon\rangle \to \langle v, \varepsilon\rangle$

—--------------------------------------(Tail-Cons)
$\langle Tail(Cons(v, xs)), \varepsilon\rangle \to \langle xs, \varepsilon\rangle$

Errores:

—---(Head-Err)
$\langle Head(v), \varepsilon\rangle \to error$ si $v \neq Cons(v1, xs)$

—---(Tail-Err)
$\langle Tail(v), \varepsilon\rangle \to error$ si $v \neq Cons(v1, xs)$

Funciones y Aplicación (Lambda / App / βv con

closures)

Valores de función y cerraduras

En nuestro conjunto de valores F las funciones son **cerraduras**:

F $\ni$ Closure(p, c, ε_0) donde p $\in$ Id, c $\in$ ASA, ε_0 es el ambiente léxico capturado

Lambdas como cerraduras (su creación): cuando una lambda aparece como expresión, captura el ambiente actual:

—--(Lam-Clos)

$\langle$Lambda(p, c), ε$\rangle$ → $\langle$Closure(p, c, ε), ε$\rangle$

Con esto, el **valor** de una función **siempre** es closure(…), lo cual refleja el **alcance estático** (léxico): se captura ε del punto de **definición**. Al definir una lambda se crea una **cerradura** que captura el ambiente **ε** vigente (alcance estático).

Aplicación (patrones contextuales; hereda App-Fun / App-Arg)

Recordatorio de patrones (ya fijados en el Paso 2):

$\langle e_1, \varepsilon\rangle \to \langle e_1', \varepsilon\rangle$
—---(App-Fun)
$\langle App(e_1, e_2), \varepsilon\rangle \to \langle App(e_1', e_2), \varepsilon\rangle$

$\langle e_2, \varepsilon\rangle \to \langle e_2', \varepsilon\rangle$, v $\in$ F
—---(App-Arg)
$\langle App(v, e_2), \varepsilon\rangle \to \langle App(v, e_2'), \varepsilon\rangle$

Regla base de aplicación (βv por ambiente)

Cuando el operador es una **cerradura** y el **operando** ya es un **valor**, hacemos el paso β-por-valor extendiendo el ambiente capturado:

—--(App-βv)

$\langle App(Closure(p, c, \varepsilon_0), v), \varepsilon\rangle \to \langle c, \varepsilon_0[p \mapsto v]\rangle$

El paso βv sustituye en el ambiente capturado ε_0: el nuevo ambiente es $\varepsilon_0[p \mapsto v]$.

La estrategia CBV L→R: primero función, luego argumento; solo se dispara **βv** cuando la función ya es **Closure** y el argumento un **valor**

Aplicación inválida (no-función):

—-------------------------------(App-Err)

$\langle App(v, w), \varepsilon\rangle \rightarrow error$

si $v \neq Closure(p, c, \varepsilon_0)$

Side-conditions: por CBV, **siempre** evaluamos (1) la fun hasta valor, (2) el arg hasta valor, y **solo entonces intentamos** (App-Bv). Si la "fun" no es closure, caemos en **App-Err**.

Posibles errores:

Convención: error es un estado terminal (no es un valor). Toda regla que lo produzca termina la ejecución (no hay pasos desde error).

1. **Variable no ligada** (Lookup-Miss)

—--------------------

$\langle Id(x), \varepsilon\rangle \rightarrow error$

si $x \notin dom(\varepsilon)$

2. **División entre cero**

—--

$\langle Div(Num(n), Num(0)), \varepsilon\rangle \rightarrow error$

3. **Tipos inválidos** (operadores aritméticos)

—----------------------------

$\langle Ar(v_1, v_2), \varepsilon\rangle \rightarrow error$

si $(v_1, v_2) \notin Num \times Num$

con Ar $\in$ {Add, Sub, Mult, Div, Expt}

4. **Exponentes negativos** (si aritmética entera)

—---

$\langle Expt(Num(n), Num(m)), \varepsilon\rangle \to error$

si $m < 0$

5. **Comparadores fuera de tipo**

—---------------------------

$\langle Cmp(v_1, v_2), \varepsilon\rangle \to error$

si $(v_1, v_2) \notin Num \times Num$

con $Cmp \in \{Eq, Lt, Le, Gt, Ge\}$

6. **If con guardia no booleana**

—------------------------

$\langle If(v, t, f), \varepsilon\rangle \to error$

Si $v \notin \{Bool(True), Bool(False)\}$

7. **Not con argumento no booleano**

—-------------------

$\langle Not(v), \varepsilon\rangle \to error$

Si $v \notin \{Bool(True), Bool(False)\}$

8. **Proyecciones sobre no-par**

—------------------------

$\langle Fst(v), \varepsilon\rangle \to error$

—----------------------------

$\langle Snd(v), \varepsilon\rangle \to$ error si $v \notin Pair(v_1, v_2)$

9. **Aplicación a no-función**

—-------------------------

$\langle App(v, w), \varepsilon\rangle \to$ error

si $v \notin Closure(p, c, \varepsilon_0)$

Determinismo CBV y orden L→R

En nuestro núcleo evaluamos **por valor** y **de izquierda→derecha**. En cada constructor hay **un solo contexto activo** (primero el operando/guardia/argumento izquierdo; hasta que sea valor pasamos al derecho). Las **reglas base no se solapan** gracias a sus side-conditions, por eso desde cualquier configuración no final ⟨e,ε⟩ **existe a lo más un** siguiente paso ⟨e′,ε′⟩: la relación → es **determinista**.

En otras palabras con CBV y orden L→R hay un único contexto activo y las bases están separadas por side-conditions; por tanto, desde ⟨e,ε⟩ no final hay **a lo más un** paso ⟨e′,ε′⟩: → es determinista.

Derivaciones completas:

(A) β-por-valor con cierre + aritmética

Expresión: App(Lambda(x, Add(x, Num(1))), Num(3))

⟨App(Lambda(x, Add(x, Num(1))), Num(3)), ε⟩
→ ⟨App(Closure(x, Add(x, Num(1)), ε), Num(3)), ε⟩ (Lam-Clos)
→ ⟨Add(x, Num(1)), ε[x → Num(3)]⟩ (App-βv)
→ ⟨Add(Num(3), Num(1)), ε[x → Num(3)]⟩ (Lookup-Hit + Bin-Left)
→ ⟨Num(4), ε[x → Num(3)]⟩ (Add-Num)

(B) fst correcto y error por tipo

1. **Correcto:**

⟨Fst(Pair(Num(1), Num(2))), ε⟩
→ ⟨Num(1), ε⟩ (Fst-Pair)

2. **Error por tipo:**

⟨Fst(Num(7)), ε⟩
→ error (Fst-Err)

(C) cond desazucarado a if +

corto-circuito Expresión concreta:

(cond [Eq(Num(2), Num(2)) => Num(10)]

[Else => Num(20)])

Desazúcar al núcleo :
If(Eq(Num(2), Num(2)), Num(10), Num(20))

⟨If(Eq(Num(2), Num(2)), Num(10), Num(20)), ε⟩
→ ⟨If(Bool(True), Num(10), Num(20)), ε⟩ (Eq-Num)
→ ⟨Num(10), ε⟩ (If-True)