Lenguajes de Programación 2023-1
Nota de clase 9: Maquinas Abstractas
Funcional (λ)

Javier Enríquez Mendoza

19 de noviembre de 2022

Las máquinas abstractas son un modelo teórico de computo compuesto de:

- Un conjunto de estados.
- Un conjunto de estados iniciales.
- Un conjunto de estados finales.
- Un conjunto de operaciones de la máquina que manipulan su estado.

El uso de máquinas abstractas en ciencias de la computación es de suma importancia, por ejemplo, para especificar la semántica operacional de un lenguaje de programación o en el estudio de la complejidad.Quizá el ejemplo mas conocido de una máquina abstracta sean las máquinas de Turing.

En este curso hemos estudiado las semántica operacional de los lenguajes mediante un sistema de transición, estos sistemas son también maquinas abstractas muy simples. Así que técnicamente ya hemos estudiado un ejemplo de máquina abstracta, el sistema de transición con el que se definió MinHs , que recibe el nombre de maquina $\mathcal{M}$.

Sin embargo está máquina funciona en un nivel de abstracción muy alto en donde no nos permite ver el proceso de evaluación de un programa sino únicamente el proceso de evaluación del estado actual del sistema.

El objetivo de esta nota es definir una máquina abstracta para la semántica operacional de MinHs , con un menor nivel de abstracción que nos permita ver la ejecución completa de los cómputos.

1 La Maquina $\mathcal{H}$

La primera máquina que vamos a estudiar es la máquina $\mathcal{H}$, esta máquina hace explicito el control de flujo del programa con una pila de control en donde se almacenan todos los cómputos pendientes en forma ce marcos. Primero vamos a formalizar estos conceptos para después definir la máquina $\mathcal{H}$.

1.1 Marcos y Pila de control

Definición 1.1 (Marcos)**.** Los marcos son esqueletos estructurales que registran los cómputos pendientes de una expresión. En nuestra definición, el símbolo $\square$ indica el lugar en donde se está llevando acabo la evaluación actual.

Operadores primitivos

$$\frac{}{\texttt{suma}(\square, e_2)\ \ \mathsf{marco}} \qquad \frac{}{\texttt{suma}(v_1, \square)\ \ \mathsf{marco}}$$

El resto de los operadores binarios son definidos análogamente.

Condicional

$$\frac{}{\texttt{if}(\square, e_2, e_3)\ \ \mathsf{marco}}$$

Aplicación de función

$$\frac{}{\texttt{app}(\square, e_2)\ \ \mathsf{marco}}$$

Observación. Los marcos están definidos en función de la semántica operacional. En particular no hay marcos para los valores del lenguaje (números, booleanos y funciones) esto se debe a que los marcos definen cómputos pendientes y dentro de un valor no hay ningún cómputo pendiente.

Definición 1.2 (Pila de control)**.** Una pila de control está formada a partir de marcos, y se define inductivamente como sigue:

$$\frac{}{\blacklozenge\ \ \mathsf{pila}}\ \mathsf{vacia} \qquad \frac{\mathfrak{m}\ \ \mathsf{marco} \qquad \mathcal{P}\ \ \mathsf{pila}}{\mathfrak{m}; \mathcal{P}\ \ \mathsf{pila}}\ \mathsf{top}$$

1.2 Estados

Definición 1.3 (Estados de la máquina $\mathcal{H}$)**.** Los estados están compuestos de una pila de control $\mathcal{P}$ y una expresión e cerrada y son de alguna de las siguientes formas:

- **Estados de evaluación**: Se evalúa e siendo $\mathcal{P}$ la pila de control y lo denotamos como $\mathcal{P} \succ e$
- **Estados de retorno**: Devuelve el valor v a la pila de control $\mathcal{P}$, que denotamos como $\mathcal{P} \prec v$

En donde se distinguen dos tipos de estados en particular:

- **Estados iniciales**: comienzan la evaluación con la pila vacía denotados como $\blacklozenge \succ e$.
- **Estados finales**: regresan un valor a la pila vacía y se denota $\blacklozenge \prec v$

1.3 Transiciones

La relación de transición modela la transición de la semántica operacional definida para MinHs .

Definición 1.4 (Transiciones para la máquina $\mathcal{H}$)**.** Las transiciones se definen por medio de la relación $\longrightarrow_{\mathcal{H}}$ y es de la forma:

$$\mathcal{P} \succ e \longrightarrow_{\mathcal{H}} \mathcal{P}' \succ e'$$

en donde los símbolos $\succ$ se pueden sustituir en ambos casos por $\prec$.
Las reglas para las expresiones de MinHs son:

Valores Los valores del lenguaje son números, booleanos y funciones y la evaluación de un valor simplemente lo regresa como resultado a la pila, pues un valor ya finalizo su proceso de evaluación.

$$\frac{}{\mathcal{P} \succ v \longrightarrow_{\mathcal{H}} \mathcal{P} \prec v}$$

Operaciones Veamos el ejemplo de la suma y el resto de las operaciones se definen análogamente. Para evaluar $\texttt{suma}(e_1, e_2)$ agregamos el marco $\texttt{suma}(\square, e_2)$ a la pila y evaluamos e_1.

$$\frac{}{\mathcal{P} \succ \texttt{suma}(e_1, e_2) \longrightarrow_{\mathcal{H}} \texttt{suma}(\square, e_2); \mathcal{P} \succ e_1}$$

Si tenemos en el tope de la pila el marco $\texttt{suma}(\square, e_2)$ y se regresa como resultado un valor v, entonces, evaluamos e_2 y sustituimos el tope de la pila por el marco $\texttt{suma}(v_1, \square)$.

$$\frac{}{\texttt{suma}(\square, e_2); \mathcal{P} \prec v_1 \longrightarrow_{\mathcal{H}} \texttt{suma}(v_1, \square); \mathcal{P} \succ e_2}$$

Si se devuelve un valor a la pila que tiene como tope el marco $\texttt{suma}(\texttt{num}[n], \square)$ entonces podemos devolver al resto de la pila el resultado de la suma de ambos valores.

$$\frac{}{\texttt{suma}(\texttt{num}[n], \square); \mathcal{P} \prec \texttt{num}[m] \longrightarrow_{\mathcal{H}} \mathcal{P} \prec \texttt{num}[n+m]}$$

Condicional Para evaluar la expresión $\texttt{if}(e_1, e_2, e_3)$ agregamos el marco $\texttt{if}(\square, e_2, e_3)$ al tope de la pila y evaluamos e_1.

$$\frac{}{\mathcal{P} \succ \texttt{if}(e_1, e_2, e_3) \longrightarrow_{\mathcal{H}} \texttt{if}(\square, e_2, e_3); \mathcal{P} \succ e_1}$$

Si se regresa `true` a la pila con el marco $\texttt{if}(\square, e_2, e_3)$ en el tope, entonces evaluamos e_2 con el resto de la pila.

$$\frac{}{\texttt{if}(\square, e_2, e_3); \mathcal{P} \prec \texttt{true} \longrightarrow_{\mathcal{H}} \mathcal{P} \succ e_2}$$

Si se regresa `false` a la pila con el marco $\texttt{if}(\square, e_2, e_3)$ en el tope, entonces evaluamos e_3 con el resto de la pila.

$$\frac{}{\texttt{if}(\square, e_2, e_3); \mathcal{P} \prec \texttt{false} \longrightarrow_{\mathcal{H}} \mathcal{P} \succ e_3}$$

Asignaciones locales Si se quiere evaluar la expresión $\texttt{let}(e_1, x.e_2)$ con la pila $\mathcal{P}$ entonces se evalúa e_2 en donde se sustituyen las apariciones de x por e_1 con la misma pila.

$$\frac{}{\mathcal{P} \succ \texttt{let}(e_1, x.e_2) \longrightarrow_{\mathcal{H}} \mathcal{P} \succ e_2[x := e_1]}$$

Aplicación de función Para evaluar una aplicación $\texttt{app}(e_1, e_2)$ en una pila $\mathcal{P}$ se agrega el marco $\texttt{app}(\square, e_2)$ como tope y se evalúa e_1.

$$\frac{}{\mathcal{P} \succ \texttt{app}(e_1, e_2) \longrightarrow_{\mathcal{H}} \texttt{app}(\square, e_2); \mathcal{P} \succ e_1}$$

Si se regresa un valor $\texttt{lam}(x.e_1)$ a la pila con tope $\texttt{app}(\square, e_2)$ entonces se quita el tope y se evalúa e_1 sustituyendo x por e_2.

$$\frac{}{\texttt{app}(\square, e_2); \mathcal{P} \prec \texttt{lam}(x.e_1) \longrightarrow_{\mathcal{H}} \mathcal{P} \succ e_1[x := e_2]}$$

Si se regresa un valor $\texttt{recfun}(f.x.e_1)$ a la pila con tope $\texttt{app}(\square, e_2)$ entonces se quita el tope y se evalúa e_1 sustituyendo f por su punto fijo y x por e_2.

$$\frac{}{\texttt{app}(\square, e_2); \mathcal{P} \prec \texttt{recfun}(f.x.e_1) \longrightarrow_{\mathcal{H}} \mathcal{P} \succ e_1[f := \texttt{fix}(f.x.e_1), x := e_2]}$$

El operador de punto fijo Para evaluar la expresión $\texttt{fix}(f.e)$ en la pila $\mathcal{P}$ se evalúa e sustituyendo f por $\texttt{fix}(f.e)$.

$$\frac{}{\mathcal{P} \succ \texttt{fix}(f.e) \longrightarrow_{\mathcal{H}} \mathcal{P} \succ e[f := \texttt{fix}(f.e)]}$$

Observación. Es importante notar que todas las reglas de transición de la máquina $\mathcal{H}$ son axiomas, lo que facilita su implementación.

Ejemplo 1.1 (Ejecución de la máquina $\mathcal{H}$)**.** Veamos el comportamiento de la máquina $\mathcal{H}$ evaluando la siguiente expresión:

```
let x = 5 in if x <= 3 then x + 2 else x - 2 end
```

Que en sintaxis abstracta se representa con el árbol:

$$\texttt{let}(5, x.\texttt{if}(\texttt{leq}(x, 3), \texttt{suma}(x, 2), \texttt{sub}(x, 2)))$$

La ejecución en la máquina $\mathcal{H}$ se ve de la siguiente forma:

$$\begin{aligned}
\blacklozenge &\succ \texttt{let}(5, x.\texttt{if}(\texttt{leq}(x, 3), \texttt{suma}(x, 2), \texttt{sub}(x, 2))) \\
\blacklozenge &\succ \texttt{if}(\texttt{leq}(x, 3), \texttt{suma}(x, 2), \texttt{sub}(x, 2))[x := 5] \\
\blacklozenge &\succ \texttt{if}(\texttt{leq}(5, 3), \texttt{suma}(5, 2), \texttt{sub}(5, 2)) \\
\texttt{if}(\square, \texttt{suma}(5, 2), \texttt{sub}(5, 2)); \blacklozenge &\succ \texttt{leq}(5, 3) \\
\texttt{leq}(\square, 3); \texttt{if}(\square, \texttt{suma}(5, 2), \texttt{sub}(5, 2)); \blacklozenge &\succ 5 \\
\texttt{leq}(\square, 3); \texttt{if}(\square, \texttt{suma}(5, 2), \texttt{sub}(5, 2)); \blacklozenge &\prec 5 \\
\texttt{leq}(5, \square); \texttt{if}(\square, \texttt{suma}(5, 2), \texttt{sub}(5, 2)); \blacklozenge &\succ 3 \\
\texttt{leq}(5, \square); \texttt{if}(\square, \texttt{suma}(5, 2), \texttt{sub}(5, 2)); \blacklozenge &\prec 3 \\
\texttt{if}(\square, \texttt{suma}(5, 2), \texttt{sub}(5, 2)); \blacklozenge &\succ 5 \leq 3 \\
\texttt{if}(\square, \texttt{suma}(5, 2), \texttt{sub}(5, 2)); \blacklozenge &\prec \texttt{false} \\
\blacklozenge &\succ \texttt{sub}(5, 2) \\
\texttt{sub}(\square, 2); \blacklozenge &\succ 5 \\
\texttt{sub}(\square, 2); \blacklozenge &\prec 5 \\
\texttt{sub}(5, \square); \blacklozenge &\succ 2 \\
\texttt{sub}(5, \square); \blacklozenge &\prec 2 \\
\blacklozenge &\succ 5 - 2 \\
\blacklozenge &\prec 3
\end{aligned}$$

Ejemplo 1.2 (Ejecución máquina $\mathcal{H}$)**.** Para ver el funcionamiento de la máquina $\mathcal{H}$ vamos a evaluar la siguiente expresión:

```
let x = 3 in
    let f = fun y => x + y in
        let x = 5 in
            f 4
        end
    end end
```

En sintaxis abstracta:

$$\texttt{let}(3, x.\texttt{let}(\texttt{lam}(y.\texttt{suma}(x, y)), f.\texttt{let}(5, x.\texttt{app}(f, 4))))$$

La evaluamos en la máquina $\mathcal{H}$.

$$\begin{array}{rcl}
\blacklozenge & \succ & \texttt{let}(3, x.\texttt{let}(\texttt{lam}(y.\texttt{suma}(x, y)), f.\texttt{let}(5, x.\texttt{app}(f, 4)))) \\
\blacklozenge & \succ & \texttt{let}(\texttt{lam}(y.\texttt{suma}(x, y)), f.\texttt{let}(5, x.\texttt{app}(f, 4)))[x := 3] \\
\blacklozenge & \succ & \texttt{let}(\texttt{lam}(y.\texttt{suma}(3, y)), f.\texttt{let}(5, x.\texttt{app}(f, 4))) \\
\blacklozenge & \succ & \texttt{let}(5, x.\texttt{app}(f, 4))[f := \texttt{lam}(y.\texttt{suma}(3, y))] \\
\blacklozenge & \succ & \texttt{let}(5, x.\texttt{app}(\texttt{lam}(y.\texttt{suma}(3, y)), 4)) \\
\blacklozenge & \succ & \texttt{app}(\texttt{lam}(y.\texttt{suma}(3, y)), 4)[x := 5] \\
\blacklozenge & \succ & \texttt{app}(\texttt{lam}(y.\texttt{suma}(3, y)), 4) \\
\texttt{app}(\square, 4); \blacklozenge & \succ & \texttt{lam}(y.\texttt{suma}(3, y)) \\
\texttt{app}(\square, 4); \blacklozenge & \prec & \texttt{lam}(y.\texttt{suma}(3, y)) \\
\blacklozenge & \succ & \texttt{suma}(3, y)[y := 4] \\
\blacklozenge & \succ & \texttt{suma}(3, 4) \\
\texttt{suma}(\square, 4); \blacklozenge & \succ & 3 \\
\texttt{suma}(\square, 4); \blacklozenge & \prec & 3 \\
\texttt{suma}(3, \square); \blacklozenge & \succ & 4 \\
\texttt{suma}(3, \square); \blacklozenge & \prec & 4 \\
\blacklozenge & \succ & 3 + 4 \\
\blacklozenge & \prec & 7
\end{array}$$

2 La Maquina $\mathcal{J}$

Con la máquina $\mathcal{H}$ es evidente el flujo de ejecución de un programa lo que nos facilita implementar un lenguaje de forma eficiente en una computadora. Las operaciones nativas necesarias para el funcionamiento de la máquina abstracta son: aritméticas, lógicas, de comparación y la operación de sustitución. La mayoría de éstas son operaciones nativas en una computadora, por lo que las podemos implementar de forma eficiente.

Sin embargo la operación de sustitución no se puede implementar eficientemente. Esta operación tiene una complejidad de $O(n)$ sobre el tamaño de la expresión, lo cuál no está ni cerca de la complejidad de las instrucciones de la computadora. Para que nuestro interprete de MinHs sea eficiente, es necesario deshacernos de la operación de sustitución y para eso agregaremos un ambiente a los estados de nuestra máquina que almacenen el valor de las variables. Un ambiente va a mapear una

variable con el valor que le fue asignado.

Llamaremos a esta nueva máquina $\mathcal{J}$, y se define como una extensión de la máquina $\mathcal{H}$ evitando el uso de sustitución en la evaluación de expresiones. En su lugar se contará con un ambiente en donde se guardan los valores de las variables.

Observación. Para simplificar la definición de esta máquina se cambiará la estrategia de evaluación sobre el lenguaje, es decir, optaremos por una evaluación ansiosa. El motivo es simplificar el contenido de los ambientes, sin embargo, la persona que lee debe convencerse de que una implementación de evaluación perezosa no solo es posible sino que solo implica modificaciones menores sobre la definición que se presentará.

Definición 2.1 (Marcos)**.** En la máquina $\mathcal{J}$ se usa el mismo conjunto de marcos que los presentados para la máquina $\mathcal{H}$, y para el uso de evaluación ansiosa como estrategia de evaluación en la máquina $\mathcal{J}$ se agregan los siguientes marcos:

Asignaciones locales

$$\frac{}{\texttt{let}(\square, x.e_2)\ \ \mathsf{marco}}$$

Aplicación de función

$$\frac{}{\mathsf{app}(f, \square)\ \ \mathsf{marco}}$$

Con los que se modela la evaluación ansiosa del lenguaje al evaluar los valores de las asignaciones locales así como el argumento de la función en la aplicación.

Los ambientes de variables se definen como una colección de identificadores con el valor que les corresponde.

Definición 2.2 (Ambientes)**.** Un ambiente es una estructura que almacena asignaciones de variables con su valor y la definimos inductivamente como:

$$\frac{}{\bullet\ \ \mathsf{env}}\ \mathsf{vacio} \qquad \frac{x\ \ \mathsf{var} \qquad v\ \ \mathsf{valor} \qquad \mathcal{E}\ \ \mathsf{env}}{x \leftarrow v; \mathcal{E}\ \ \mathsf{env}}\ \mathsf{asig}$$

La forma de acceder a los elementos del ambiente es mediante el nombre de la variable, entonces si el ambiente $\mathcal{E}$ tiene la asignación $x \leftarrow v$. Podemos acceder al valor de x como $\mathcal{E}[x]$ y el resultado es v.
En caso de tener mas de una asignación sobre el mismo nombre de variable $\mathcal{E}[x]$ nos regresa la primera aparición de x en el ambiente. Por ejemplo en el ambiente $\mathcal{E} =_{def} x \leftarrow v_1; x \leftarrow v_2; \bullet$ la operación $\mathcal{E}[x]$ nos arroja como resultado v_1.

Los estados de la máquina entonces ahora tendrán un ambiente.

Definición 2.3 (Estados de la máquina $\mathcal{J}$)**.** Los estados ahora son una relación ternaria de una pila de control $\mathcal{P}$ un ambiente $\mathcal{E}$ y una expresión e, denotados como:

- **Estados de evaluación**: $\mathcal{P} \mid \mathcal{E} \succ e$
- **Estados de retorno**: $\mathcal{P} \mid \mathcal{E} \prec e$

Un estado inicial es de la forma:

$$\blacklozenge \mid \bullet \succ e$$

Mientras que los estados finales son de la forma:

$$\blacklozenge \mid \mathcal{E} \prec v$$

Notemos que en los estados finales no importa que está guardado en el ambiente solo importa que la pila de control esté vacía.

2.1 Transiciones

Ahora es necesario definir una reglas de transición para la máquina $\mathcal{J}$.

Definición 2.4 (Transiciones de variables en la máquina $\mathcal{J}$)**.** Se definen las transiciones de la máquina sin utilizar la operación de sustitución.

Variables En la máquina $\mathcal{H}$ una variable representaba un estado bloqueado, pues la única forma de llegar a ella era que se tratara de una variable libre. En la máquina $\mathcal{J}$ como no aplicamos sustitución tenemos que evaluar las variables buscándolas en el ambiente. Lo que definimos con la siguiente regla:

$$\frac{}{\mathcal{P} \mid \mathcal{E} \succ x \longrightarrow_{\mathcal{J}} \mathcal{P} \mid \mathcal{E} \prec \mathcal{E}[x]}$$

La idea es utilizar el ambiente de las variables para buscar su valor. Sin embargo todas las expresiones que requieren de sustitución para su evaluación definen un alcance para sus variables. Por ejemplo en la evaluación de la expresión

$$\mathtt{app}(\mathtt{lam}(x.e_1), e_2)$$

se debe agregar al ambiente la asignación $x \leftarrow e_2$ y evaluar e_1 para obtener el resultado. Pero una vez que termine la evaluación de la aplicación de función esta asignación debe quitarse del ambiente pues ya terminó su alcance. Para lograrlo es necesario extender la definición de la pila de ejecución para que guarde no sólo marcos sino también ambientes.

Definición 2.5 (Pila de control para $\mathcal{J}$)**.**

$$\frac{}{\blacklozenge \ \mathsf{pila}}\ \mathsf{vacia} \qquad \frac{\mathfrak{m}\ \ \mathsf{marco} \qquad \mathcal{P}\ \ \mathsf{pila}}{\mathfrak{m};\mathcal{P}\ \ \mathsf{pila}}\ \mathsf{top} \qquad \frac{\mathcal{E}\ \ \mathsf{env} \qquad \mathcal{P}\ \ \mathsf{pila}}{\mathcal{E};\mathcal{P}\ \ \mathsf{pila}}\ \mathsf{top}$$

De esta forma cuando evaluemos una expresión que genera un alcance distinto se agrega el ambiente anterior a la pila de control y se definen las nuevas asignaciones dentro del ambiente actual. Una vez que termine la ejecución y se regrese un valor a la pila con un ambiente en el tope, continuamos con la ejecución del programa tomando ese ambiente como actual.

Definición 2.6 (Transiciones con alcance en la máquina $\mathcal{J}$)**.** Ahora reescribimos los casos que involucran usar sustitución para su evaluación.

Asignaciones locales

$$\frac{}{\mathcal{P} \mid \mathcal{E} \succ \texttt{let}(e_1, x.e_2) \longrightarrow_{\mathcal{J}} \texttt{let}(\square, x.e_2); \mathcal{P} \mid \mathcal{E} \succ e_1}$$

$$\frac{}{\texttt{let}(\square, x.e_2); \mathcal{P} \mid \mathcal{E} \prec v \longrightarrow_{\mathcal{J}} \mathcal{E}; \mathcal{P} \mid x \leftarrow v; \mathcal{E} \succ e_2}$$

Aplicación de función

$$\frac{}{\texttt{app}(\square, e_2); \mathcal{P} \mid \mathcal{E} \prec v_1 \longrightarrow_{\mathcal{J}} \texttt{app}(v_1, \square); \mathcal{P} \mid \mathcal{E} \succ e_2}$$

$$\frac{}{\texttt{app}(\texttt{lam}(x.e_1), \square); \mathcal{P} \mid \mathcal{E} \prec v_2 \longrightarrow_{\mathcal{J}} \mathcal{E}; \mathcal{P} \mid x \leftarrow v_2; \mathcal{E} \succ e_1}$$

$$\frac{}{\texttt{app}(\texttt{recfun}(f.x.e_1), \square); \mathcal{P} \mid \mathcal{E} \prec v_2 \longrightarrow_{\mathcal{J}} \mathcal{E}; \mathcal{P} \mid f \leftarrow \texttt{fix}(f.x.e_1); x \leftarrow v_2; \mathcal{E} \succ e_1}$$

Liberación del ambiente Esta regla se agrega para liberar el ambiente de la pila de control.

$$\frac{}{\mathcal{E}; \mathcal{P} \mid \mathcal{E}_1 \prec v \longrightarrow_{\mathcal{J}} \mathcal{P} \mid \mathcal{E} \prec v}$$

Observemos como la evaluación del valor a almacenar en las asignaciones locales se volvió un punto estricto pues los ambientes solo almacenan valores.

Para ejemplificar el funcionamiento de la máquina $\mathcal{J}$ veamos un ejemplo de evaluación de una expresión de MinHs en ésta.

Ejemplo 2.1 (Ejecución máquina $\mathcal{J}$)**.** Para ver el funcionamiento de la máquina $\mathcal{J}$ vamos a evaluar la siguiente expresión:

```
let x = 3 in
    let f = fun y => x + y in
        let x = 5 in
            f 4
        end
    end
end
```

En sintaxis abstracta:

$$\texttt{let}(3, x.\texttt{let}(\texttt{lam}(y.\texttt{suma}(x, y)), f.\texttt{let}(5, x.\texttt{app}(f, 4))))$$

La evaluamos en la máquina $\mathcal{J}$.

$$
\begin{aligned}
\blacklozenge \mid \bullet &\succ \texttt{let}(3, x. \dots) \\
\texttt{let}(\square, x. \dots); \blacklozenge \mid \bullet &\succ 3 \\
\texttt{let}(\square, x. \dots); \blacklozenge \mid \bullet &\prec 3 \\
\bullet; \blacklozenge \mid x \leftarrow 3; \bullet &\succ \texttt{let}(\texttt{lam}(y.\texttt{suma}(x, y)), \dots) \\
\texttt{let}(\square, \dots); \bullet; \blacklozenge \mid \underbrace{x \leftarrow 3; \bullet}_{\mathcal{E}_1} &\succ \texttt{lam}(y.\texttt{suma}(x, y)) \\
\texttt{let}(\square, \dots); \bullet; \blacklozenge \mid x \leftarrow 3; \bullet &\prec \texttt{lam}(y.\texttt{suma}(x, y)) \\
\mathcal{E}_1; \bullet; \blacklozenge \mid f \leftarrow \texttt{lam}(y.\texttt{suma}(x, y)); x \leftarrow 3; \bullet &\succ \texttt{let}(5, x.\texttt{app}(f, 4)) \\
\texttt{let}(\square, x.\texttt{app}(f, 4)); \mathcal{E}_1; \bullet; \blacklozenge \mid f \leftarrow \texttt{lam}(y.\texttt{suma}(x, y)); x \leftarrow 3; \bullet &\succ 5 \\
\texttt{let}(\square, x.\texttt{app}(f, 4)); \mathcal{E}_1; \bullet; \blacklozenge \mid \underbrace{f \leftarrow \texttt{lam}(y.\texttt{suma}(x, y)); x \leftarrow 3; \bullet}_{\mathcal{E}_2} &\prec 5 \\
\mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid x \leftarrow 5; f \leftarrow \texttt{lam}(y.\texttt{suma}(x, y)); x \leftarrow 3; \bullet &\succ \texttt{app}(f, 4) \\
\texttt{app}(\square, 4); \mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid x \leftarrow 5; f \leftarrow \texttt{lam}(y.\texttt{suma}(x, y)); x \leftarrow 3; \bullet &\succ f \\
\texttt{app}(\square, 4); \mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid \underbrace{x \leftarrow 5; f \leftarrow \texttt{lam}(y.\texttt{suma}(x, y)); x \leftarrow 3; \bullet}_{\mathcal{E}_3} &\prec \texttt{lam}(y.\texttt{suma}(x, y)) \\
\mathcal{E}_3; \mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid y \leftarrow 4; x \leftarrow 5; f \leftarrow \texttt{lam}(y.\texttt{suma}(x, y)); x \leftarrow 3; \bullet &\succ \texttt{suma}(x, y) \\
\texttt{suma}(\square, y); \mathcal{E}_3; \mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid y \leftarrow 4; x \leftarrow 5; f \leftarrow \texttt{lam}(y.\texttt{suma}(x, y)); x \leftarrow 3; \bullet &\succ x \\
\texttt{suma}(\square, y); \mathcal{E}_3; \mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid y \leftarrow 4; x \leftarrow 5; f \leftarrow \texttt{lam}(y.\texttt{suma}(x, y)); x \leftarrow 3; \bullet &\prec 5 \\
\texttt{suma}(5, \square); \mathcal{E}_3; \mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid y \leftarrow 4; x \leftarrow 5; f \leftarrow \texttt{lam}(y.\texttt{suma}(x, y)); x \leftarrow 3; \bullet &\succ y \\
\texttt{suma}(5, \square); \mathcal{E}_3; \mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid y \leftarrow 4; x \leftarrow 5; f \leftarrow \texttt{lam}(y.\texttt{suma}(x, y)); x \leftarrow 3; \bullet &\prec 4 \\
\mathcal{E}_3; \mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid y \leftarrow 4; x \leftarrow 5; f \leftarrow \texttt{lam}(y.\texttt{suma}(x, y)); x \leftarrow 3; \bullet &\prec 5 + 4 \\
\mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid \mathcal{E}_3 &\prec 9 \\
\mathcal{E}_1; \bullet; \blacklozenge \mid \mathcal{E}_2 &\prec 9 \\
\bullet; \blacklozenge \mid \mathcal{E}_1 &\prec 9 \\
\blacklozenge \mid \bullet &\prec 9
\end{aligned}
$$

3 Alcance

Consideremos la siguiente expresión

```
let x = 3 in
    let f = fun y => x + y in
        let x = 5 in
            f 4
        end
    end
end
```

Si evaluamos esta expresión en la máquina $\mathcal{H}$ el resultado es 7 mientras que si la evaluamos en la máquina $\mathcal{J}$ el resultado será 9. Esto se debe al alcance utilizado en cada una de las máquinas. En general se tienen dos tipos de alcance: estático y dinámico.

Definición 3.1 (Alcance estático)**.** En un lenguaje con alcance estático, el alcance de una variable es la región en la cual se encuentra definida.

Por ejemplo la máquina $\mathcal{H}$ implementa alcance estático y esto se debe al uso de sustitución para encontrar el valor de sus variables. Y de esta forma en el ejemplo anterior la expresión

```
fun y =>x + y
```

toma el valor de `x` de la región que delimita a esa función, es decir, 3.

Definición 3.2 (Alcance dinámico)**.** En un lenguaje con alcance dinámico, el alcance de un identificador es todo el programa, es decir, se toma la última asignación hecha al mismo.

La máquina $\mathcal{J}$ implementa este tipo de alcance. En el ejemplo anterior la expresión

```
fun y =>x + y
```

toma el valor de `x` de la última asignación hecha sobre esa variable, es decir, 5.

Si bien el usar alcance dinámico no es un error, en nuestro caso representa un problema pues se busca que ambas máquinas evalúen al mismo valor las expresiones de MinHs . Para corregir esto, es necesario que las funciones usen el ambiente en el cual fueron creadas.

3.1 Closures

Debemos revisar nuestra definición de función para garantizar que use el ambiente adecuado en su ejecución. Para lograr evaluar las expresiones en el ambiente adecuado se usa el concepto de *Closure* o cerradura, que se define como sigue.

Definición 3.3 (Closure)**.** Un *Closure* es una pareja de una expresión de función de MinHs y un ambiente que se denota como:

$$\langle\langle \mathcal{E}, f \rangle\rangle$$

y se interpreta como, que el ambiente adecuado para evaluar la función f es $\mathcal{E}$, de esta forma se respeta el ambiente en el que se define una función para usar el mismo en su ejecución y de esta forma definir una evaluación con alcance estático.

Ahora se modifican las transiciones definidas en la sección anterior para que usen *closures* y modelen una evaluación con alcance estático, en lugar de la evaluación con alcance dinámico presentada anteriormente.

Definición 3.4 (Transición para funciones)**.** En lugar de regresar las funciones como una expresión, se guardará como una pareja de la expresión y el ambiente en el que fue definido.

$$\frac{}{\mathcal{P} \mid \mathcal{E} \succ \texttt{lam}(x.e) \longrightarrow_{\mathcal{J}} \mathcal{P} \mid \mathcal{E} \prec \langle\!\langle \mathcal{E}, x.e \rangle\!\rangle}$$

$$\frac{}{\texttt{app}(\square, e_2); \mathcal{P} \mid \mathcal{E} \prec \langle\!\langle \mathcal{E}_f, x.e_1 \rangle\!\rangle \longrightarrow_{\mathcal{J}} \texttt{app}(\langle\!\langle \mathcal{E}_f, x.e_1 \rangle\!\rangle, \square); \mathcal{P} \mid \mathcal{E} \succ e_2}$$

$$\frac{}{\texttt{app}(\langle\!\langle \mathcal{E}_f, x.e_1 \rangle\!\rangle, \square); \mathcal{P} \mid \mathcal{E} \prec v \longrightarrow_{\mathcal{J}} \mathcal{E}; \mathcal{P} \mid x \leftarrow v; \mathcal{E}_f \succ e_1}$$

$$\frac{}{\mathcal{P} \mid \mathcal{E} \succ \texttt{recfun}(f.x.e) \longrightarrow_{\mathcal{J}} \mathcal{P} \mid \mathcal{E} \prec \texttt{fix}(f.\langle\!\langle \mathcal{E}, x.e \rangle\!\rangle)}$$

$$\frac{}{\texttt{app}(\square, e_2); \mathcal{P} \mid \mathcal{E} \prec \texttt{fix}(f.\langle\!\langle \mathcal{E}_f, x.e_1 \rangle\!\rangle) \longrightarrow_{\mathcal{J}} \texttt{app}(\texttt{fix}(f.\langle\!\langle \mathcal{E}_f, x.e_1 \rangle\!\rangle), \square); \mathcal{P} \mid \mathcal{E} \succ e_2}$$

$$\frac{}{\texttt{app}(\texttt{fix}(f.\langle\!\langle \mathcal{E}_f, x.e_1 \rangle\!\rangle), \square); \mathcal{P} \mid \mathcal{E} \prec v \longrightarrow_{\mathcal{J}} \mathcal{E}; \mathcal{P} \mid x \leftarrow v; f \leftarrow \texttt{fix}(f.\langle\!\langle \mathcal{E}_f, x.e_1 \rangle\!\rangle); \mathcal{E}_f \succ e_1}$$

Con estas reglas se general los *closures* en la evaluación de una función para que de esta forma el ambiente con el que se ejecutan en la aplicación sea el mismo ambiente en el que se definió la función y las variables tomen el valor esperado según el alcance estático.

Ahora veamos un ejemplo de la evaluación de una expresión usando *closures* para diferenciar respecto al ejemplo anterior en el que se usa un alcance dinámico.

Ejemplo 3.1 (Ejecución máquina $\mathcal{J}$ con *closures*)**.** Para ver el funcionamiento de la máquina $\mathcal{J}$ vamos a evaluar la siguiente expresión:

```
let x = 3 in
    let f = fun y => x + y in
        let x = 5 in
            f 4
        end
    end
end
```

En sintaxis abstracta:

$$\texttt{let}(3, x.\texttt{let}(\texttt{lam}(y.\texttt{suma}(x, y)), f.\texttt{let}(5, x.\texttt{app}(f, 4))))$$

La evaluamos en la máquina $\mathcal{J}$.

$$
\begin{array}{rcl}
\blacklozenge \mid \bullet & \succ & \texttt{let}(3, x. \dots) \\
\texttt{let}(\square, x. \dots); \blacklozenge \mid \bullet & \succ & 3 \\
\texttt{let}(\square, x. \dots); \blacklozenge \mid \bullet & \prec & 3 \\
\bullet; \blacklozenge \mid x \leftarrow 3; \bullet & \succ & \texttt{let}(\texttt{lam}(y.\texttt{suma}(x, y)), \dots) \\
\texttt{let}(\square, \dots); \bullet; \blacklozenge \mid \underbrace{x \leftarrow 3; \bullet}_{\mathcal{E}_1} & \succ & \texttt{lam}(y.\texttt{suma}(x, y)) \\
\texttt{let}(\square, \dots); \bullet; \blacklozenge \mid x \leftarrow 3; \bullet & \prec & \langle\!\langle \mathcal{E}_1, y.\texttt{suma}(x, y) \rangle\!\rangle \\
\mathcal{E}_1; \bullet; \blacklozenge \mid \underbrace{f \leftarrow \langle\!\langle \mathcal{E}_1, y.\texttt{suma}(x, y) \rangle\!\rangle; x \leftarrow 3; \bullet}_{\mathcal{E}_2} & \succ & \texttt{let}(5, x \dots) \\
\texttt{let}(\square, \dots); \mathcal{E}_1; \bullet; \blacklozenge \mid f \leftarrow \langle\!\langle \mathcal{E}_1, y.\texttt{suma}(x, y) \rangle\!\rangle; x \leftarrow 3; \bullet & \succ & 5 \\
\texttt{let}(\square, \dots); \mathcal{E}_1; \bullet; \blacklozenge \mid f \leftarrow \langle\!\langle \mathcal{E}_1, y.\texttt{suma}(x, y) \rangle\!\rangle; x \leftarrow 3; \bullet & \prec & 5 \\
\mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid \underbrace{x \leftarrow 5; f \leftarrow \langle\!\langle \mathcal{E}_1, y.\texttt{suma}(x, y) \rangle\!\rangle; x \leftarrow 3; \bullet}_{\mathcal{E}_3} & \succ & \texttt{app}(f, 4) \\
\texttt{app}(\square, 4); \mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid \mathcal{E}_3 & \succ & f \\
\texttt{app}(\square, 4); \mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid \mathcal{E}_3 & \prec & \langle\!\langle \mathcal{E}_1, y.\texttt{suma}(x, y) \rangle\!\rangle \\
\texttt{app}(\langle\!\langle \mathcal{E}_1, y.\texttt{suma}(x, y) \rangle\!\rangle, \square); \mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid \mathcal{E}_3 & \succ & 4 \\
\texttt{app}(\langle\!\langle \mathcal{E}_1, y.\texttt{suma}(x, y) \rangle\!\rangle, \square); \mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid \mathcal{E}_3 & \prec & 4 \\
\mathcal{E}_3; \mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid y \leftarrow 4; \mathcal{E}_1 & \succ & \texttt{suma}(x, y) \\
\texttt{suma}(\square, y); \mathcal{E}_3; \mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid y \leftarrow 4; \mathcal{E}_1 & \succ & x \\
\texttt{suma}(\square, y); \mathcal{E}_3; \mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid y \leftarrow 4; \mathcal{E}_1 & \prec & 3 \\
\texttt{suma}(3, \square); \mathcal{E}_3; \mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid y \leftarrow 4; \mathcal{E}_1 & \succ & y \\
\texttt{suma}(3, \square); \mathcal{E}_3; \mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid y \leftarrow 4; \mathcal{E}_1 & \prec & 4 \\
\mathcal{E}_3; \mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid \mathcal{E}_1 & \prec & 3 + 4 \\
\mathcal{E}_2; \mathcal{E}_1; \bullet; \blacklozenge \mid \mathcal{E}_3 & \prec & 7 \\
\mathcal{E}_1; \bullet; \blacklozenge \mid \mathcal{E}_2 & \prec & 7 \\
\bullet; \blacklozenge \mid \mathcal{E}_1 & \prec & 7 \\
\blacklozenge \mid \bullet & \prec & 7
\end{array}
$$

Por lo que el resultado de la evaluación del programa es el valor 7 como se esperaba.

Referencias

[1] Keller G., O'Connor-Davis L., Class Notes from the course Concepts of programming language design, Department of Information and Computing Sciences, Utrecht University, The Netherlands, Fall 2020.

[2] Miranda Perea F., González Huesca L., Nota de Clase del curso de Lenguajes de Programación, Facultad de Ciencias UNAM, Semestre 2021-1.

[3] Ramírez Pulido K., Soto Romero M., Nota de Clase del curso de Lenguajes de Programación, Facultad de Ciencias UNAM, Semestre 2021-2

[4] Krishnamurthi S., Programming Languages Application and Interpretation; Version 26.04.2007.

[5] Harper R., Practical Foundations for Programming Languages. Working draft, 2010.

[6] Mitchell J., Foundations for Programming Languages. MIT Press 1996.