Lenguajes de Programación, 2017-1
Nota de clase 11: Máquinas Abstractas [*]

Favio E. Miranda Perea Lourdes del Carmen González Huesca
Facultad de Ciencias UNAM

27 de octubre de 2018

La técnica que hemos usado hasta ahora para especificar la semántica dinámica de un lenguaje se sirve de un sistema de transición muy sencillo y útil para propósitos teóricos, tales como probar la seguridad del lenguaje. Por otra parte se trata de una especificación a muy alto nivel poco recomendable para implementarse directamente. Una razón es que el uso de reglas de búsqueda (reglas con premisas) requiere atravesar y reconstruir una expresión para poder simplificar sólo una pequeña parte de élla. En una implementación real preferiríamos usar un mecanismo para registrar en que posición de la expresión nos encontramos para, después de evaluar la expresión actual, poder continuar el proceso de evaluación en esa misma posición. Este procedimiento puede implementarse mediante una *pila de control* que rastrea el contexto de una evaluación. Haciendo explícita la pila de control las reglas de transición no necesitan premisas –todas son instrucciones directas (axiomas). De esta manera se formaliza una idea informal que evita atravesar y reconstruir una expresión repetidamente.

En esta nota desarrollamos una máquina abstracta, llamada máquina $\mathcal{K}$, que utiliza una pila de control explícita. De esta manera nos acercamos más a una implementación real. Mientras describamos más detalladamente los mecanismos requeridos para una implementación nos acercamos más a una máquina real. En cada paso del camino iniciando con la semántica operacional estructural (es decir, la semántica dinámica que hemos usado hasta ahora) y continuando hasta una descripción a nivel ensamblador se utiliza un modelo particular de máquina abstracta o virtual.

Recientemente ha surgido un interés en usar máquinas virtuales para proporcionar una plataforma de cómputo independiente del hardware. La idea es definir una máquina abstracta a un nivel lo suficientemente bajo de manera que sea fácilmente implementable en las plataformas usuales de hardware y además sea compatible con lenguajes de alto nivel, en el sentido de que éstos puedan ser traducidos (compilados) fácilmente por dicha máquina, esto se conoce como código móvil. El ejemplo más prominente es la máquina virtual para Java (JVM). De esta manera se puede compilar código fuente en Java generando código de bytes, el cual puede transmitirse en redes, por ejemplo como un applet, y después interpretarse mediante la máquina virtual de Java. El uso de dicha máquina tiene dos aspectos de interés, el código en bytes es portable a cualquier arquitectura mediante un intérprete y el código recibido puede ser verificado fácilmente para detectar operaciones ilegales.

[*]Estas notas se basan en el libro de Harper y en material de Gerwin Klein y Frank Pfenning.

Es de primordial importancia que la máquina abstracta se defina de manera precisa, de otra forma no sería claro como implementarla o cómo usarla como compilador.

1. La Máquina $\mathcal{L}$

Una máquina abstracta consta esencialmente de dos componentes:

- Un conjunto de estados legales, algunos de ellos iniciales o finales.
- Un conjunto de instrucciones que alteran los estados de la máquina, de manera que sea posible simular dichas instrucciones en una máquina real en un número finito de pasos.

La semántica operacional estructural utilizada hasta ahora para definir las semánticas dinámicas de los lenguajes que hemos estudiado es una máquina abstracta en este sentido, llamada la máquina $\mathcal{L}$. Esta máquina se considera de muy alto nivel en dos sentidos:

- *Control*: el orden de ejecución se especifica mediante reglas complejas de búsqueda. Para manejar la búsqueda nos basamos en una pila implícita.
- *Datos*: el paso de parámetros se lleva a cabo mediante sustitución lo cual es complicado y genera código nuevo al vuelo.

Para manejar los datos de manera más simple existen otras alternativas como:

- El uso de una semántica de ambientes que combina reglas de inferencia con reglas de evaluación para un manejo más real del proceso de ligado de variables.
- El uso de semánticas de cerradura mediante las cuales se adjuntan contextos a las reglas de evaluación.

Con respecto al control, la máquina $\mathcal{L}$ proporciona una implementación engañosa al estar basada en una pila de almacenamiento implícita (en nuestro caso la pila de memoria de Haskell). Dos defectos importantes son:

- Las reglas de evaluación, que juegan el papel de reglas de búsqueda, tienen una o más premisas que deben ser verificadas recursivamente.
- El intérprete implementado así no es recursivo de cola lo cual es inaceptable.

En conclusión una implementación de verdad no puede basarse en el manejo implícito del almacenamiento que hemos usado hasta ahora. Es mandatorio buscar semánticas dinámicas que se asemejen más a una implementación real.
A continuación presentamos una máquina abstracta, llamada la máquina $\mathcal{K}$, cuya característica principal es el uso explícito de una pila de control cercana a la usada por una implementación real.

2. La máquina $\mathcal{K}$

Para implementar una regla de evaluación de nuestra semántica dinámica actual, por ejemplo

$$\frac{e_1 \to e_1'}{\texttt{app}(e_1, e_2) \to \texttt{app}(e_1', e_2)}$$

debemos hacer lo siguiente:

- Guardar el estado actual del proceso de evaluación $\texttt{app}(-, e_2)$
- Ejecutar un paso de evaluación $e_1 \to e_1'$
- Restaurar el estado a $\texttt{app}(e_1', e_2)$.

La pila de memoria de la implementación se encarga de guardar el estado de evaluación $\texttt{app}(-, e_2)$ siendo el caracter "$-$" el indicador de la posición de evaluación.

Las implementaciones reales no pueden basarse en un manejo implícito de la memoria de manera que a continuación definimos una máquina abstracta que utiliza una pila explícita. Esta máquina, llamada máquina $\mathcal{K}$, representa el flujo de control de la evaluación de manera explícita en cada estado mediante una pila de control incorporada a él, la cual permite que las reglas de evaluación no tengan premisas, generando así una implementación recursiva de cola.

2.1. Marcos y pilas de control

Una pila de control se forma mediante marcos. La pila vacía se denota con $\Box$.

$$\frac{}{\Box \text{ pila}} \qquad \frac{m \text{ marco} \quad \mathcal{P} \text{ pila}}{m \,;\, \mathcal{P} \text{ pila}}$$

donde $m; \mathcal{P}$ denota a la pila con el marco m en el tope.

Los *marcos* son esqueletos estructurales que registran los cómputos pendientes. En la siguiente definición el espacio marcado con guión indica la posición correspondiente al valor devuelto en la evaluación actual, el cual corresponde al lugar donde se está llevando a cabo la evaluación y por lo tanto implementa la búsqueda de reglas en la semántica operacional.

- Operadores primitivos:

$$\frac{}{\texttt{suma}(-, e_2) \text{ marco}} \qquad \frac{}{\texttt{suma}(v_1, -) \text{ marco}}$$

  analogamente para cualquier otro operador como producto, sucesor, etc.

- Condicional booleano

$$\frac{}{\texttt{if}(-, e_1, e_2) \text{ marco}}$$

- Aplicaciones de función:

$$\frac{}{\texttt{app}(-, e_2) \text{ marco}} \qquad \frac{}{\texttt{app}(v_1, -) \text{ marco}}$$

Obsérvese que los marcos usan la estrategia de evaluación de la semántica operacional. En particular no hay marcos para abstracciones lambda pues éstas son valores. Como ya se dijo un marco representa la estructura de un cómputo pendiente por lo que ningún valor necesita un marco, dado que en ese caso ya no hay cómputos pendientes. Por la misma razón no hay marcos que involucren las ramas de un `if`.

2.2. Estados

Los estados se forman mediante una pila de control $\mathcal{P}$ y una expresión cerrada e y son de alguna de las siguientes dos formas:

- Evalua e siendo $\mathcal{P}$ la pila de control actual, denotado $\mathcal{P} \succ e$
- Devuelve el valor v a la pila de control $\mathcal{P}$, denotado $\mathcal{P} \prec v$
- Los estados iniciales son los de la forma $\Box \succ e$, indicando que se empezará a evaluar la expresión e con la pila de control vacía.
- Los estados finales son los de la forma $\Box \prec v$ indicando que se devolverá el valor calculado v a la pila vacía.

2.3. Transiciones

La relación de transición se denotará con $\longrightarrow_{\mathcal{K}}$, en caso de no haber ambigüedad se suprimirá el índice $\mathcal{K}$. Una transición toma entonces la forma

$$\mathcal{P} \succ e \longrightarrow_{\mathcal{K}} \mathcal{P}' \prec e'$$

donde los símbolos $\succ, \prec$ pueden intercambiarse en cualquiera de los dos lados.

Veamos cada regla para el lenguaje PCF con cierto detalle:

- Valores (números, booleanos, funciones). Para evaluar un valor con una pila dada basta retornarlo a dicha pila.

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

- Operaciones primitivas, como ejemplo damos la suma:
  - Para evaluar $\texttt{suma}(e_1, e_2)$ con la pila $\mathcal{P}$, basta agregar el marco $\texttt{suma}(-, e_2)$ a $\mathcal{P}$ y con esta nueva pila evaluar e_1

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

  - Si estando el marco $\texttt{suma}(-, e_2)$ en el tope de la pila se devuelve el valor v entonces estando $\texttt{suma}(v, -)$ en el tope de la pila se evalua e_2. Obsérvese que *devolver* un valor a la pila significa ponerlo en la posición vacía denotada por $-$.

$$\frac{}{\texttt{suma}(-, e_2); \mathcal{P} \prec v \longrightarrow_{\mathcal{K}} \texttt{suma}(v, -); \mathcal{P} \succ e_2}$$

- Si se desea devolver el valor $\texttt{num}[n_2]$ a la pila $\texttt{suma}(\texttt{num}[n_1], -); \mathcal{P}$ entonces basta con devolver el valor $\texttt{num}[n_1 + n_2]$ a la pila $\mathcal{P}$. Esto índica que el marco de suma desaparecio, lo cual es claro pues en tal momento la suma ya fue completamente evaluada.

$$\overline{\texttt{suma}(\texttt{num}[n_1], -); \mathcal{P} \prec \texttt{num}[n_2] \longrightarrow_{\mathcal{K}} \mathcal{P} \prec \texttt{num}[n_1 + n_2]}$$

- Condicional booleano:

  - Para evaluar una expresión $\texttt{if}(e, e_1, e_2)$ con una pila $\mathcal{P}$ basta agregar el marco $\texttt{if}(-, e_1, e_2)$ a la pila y evaluar la guardia e.

$$\overline{\mathcal{P} \succ \texttt{if}(e, e_1, e_2) \longrightarrow_{\mathcal{K}} \texttt{if}(-, e_1, e_2); \mathcal{P} \succ e}$$

  - Si se desea devolver el valor true a la pila con tope $\texttt{if}(-, e_1, e_2)$ esto indica que la guardia del `if` se evaluo a true por lo que se elimina el marco del `if` y se procede a evaluar e_1 con $\mathcal{P}$.

$$\overline{\texttt{if}(-, e_1, e_2); \mathcal{P} \prec \mathsf{true} \longrightarrow_{\mathcal{K}} \mathcal{P} \succ e_1}$$

  - Si se desea devolver el valor false a la pila con tope $\texttt{if}(-, e_1, e_2)$ esto indica que la guardia del `if` se evaluo a false por lo que se elimina el marco del `if` y se procede a evaluar e_2 con $\mathcal{P}$.

$$\overline{\texttt{if}(-, e_1, e_2); \mathcal{P} \prec \mathsf{false} \longrightarrow_{\mathcal{K}} \mathcal{P} \succ e_2}$$

- Aplicación de funciones

  - Para evaluar una aplicación $\texttt{app}(e_1, e_2)$ con la pila $\mathcal{P}$ se agrega el marco $\texttt{app}(-, e_2)$ a la pila y se procede a evaluar el primer argumento e_1.

$$\overline{\mathcal{P} \succ \texttt{app}(e_1, e_2) \longrightarrow_{\mathcal{K}} \texttt{app}(-, e_2); \mathcal{P} \succ e_1}$$

  - Si se desea devolver el valor v a la pila con tope $\texttt{app}(-, e_2)$ entonces basta sustituir el tope de la pila con el marco $\texttt{app}(v, -)$ y continuar con la evaluación de e_2.

$$\overline{\texttt{app}(-, e_2); \mathcal{P} \prec v \longrightarrow_{\mathcal{K}} \texttt{app}(v, -); \mathcal{P} \succ e_2}$$

  - Si se desea devolver un valor v a la pila con tope $\texttt{app}(\texttt{lam}(\mathsf{T}, x.e), -)$ esto indica que los argumentos de la aplicación terminaron de evaluarse, siendo el primero una abstracción lambda, por lo que se elimina el marco de aplicación que está en el tope de la pila y se continua evaluando $e[x := v]$ con la pila $\mathcal{P}$.

$$\overline{\texttt{app}(\texttt{lam}(\mathsf{T}, x.e), -); \mathcal{P} \prec v \longrightarrow_{\mathcal{K}} \mathcal{P} \succ e[x := v]}$$

- Recursión general: si se desea evaluar $\texttt{fix}(\mathsf{T}, x.e)$ con pila $\mathcal{P}$ basta evaluar $e[x := \texttt{fix}(\mathsf{T}, x.e)]$ con la misma pila $\mathcal{P}$.

$$\overline{\mathcal{P} \succ \texttt{fix}(\mathsf{T}, x.e) \longrightarrow_{\mathcal{K}} \mathcal{P} \succ e[x := \texttt{fix}(\mathsf{T}, x.e)]}$$

  Obsérvese que la evaluación de un operador `fix` no requiere espacio en la pila.

A continuación mostramos con detalle un ejemplo de evaluación con la máquina $\mathcal{K}$:

$$
\begin{array}{ll}
 & \Box \succ \mathtt{app}\Big(\mathtt{if}\big((\lambda x : \mathsf{Bool}.\mathsf{not}\, x)\,\mathsf{false}, \lambda y : \mathsf{Nat}\,.y + 3, \lambda z : \mathsf{Nat}\,.z * 2\big), 0\Big) \\
\longrightarrow_{\mathcal{K}} & \mathtt{app}(-, 0); \Box \succ \mathtt{if}\big((\lambda x : \mathsf{Bool}.\mathsf{not}\, x)\,\mathsf{false}, \lambda y : \mathsf{Nat}\,.y + 3, \lambda z : \mathsf{Nat}\,.z * 2\big) \\
\longrightarrow_{\mathcal{K}} & \mathtt{if}\big(-, \lambda y : \mathsf{Nat}\,.y + 3, \lambda z : \mathsf{Nat}\,.z * 2\big); \mathtt{app}(-, 0); \Box \succ (\lambda x : \mathsf{Bool}.\mathsf{not}\, x)\,\mathsf{false} \\
\longrightarrow_{\mathcal{K}} & \mathtt{app}(-, \mathsf{false}); \mathtt{if}\big(-, \lambda y : \mathsf{Nat}\,.y + 3, \lambda z : \mathsf{Nat}\,.z * 2\big); \mathtt{app}(-, 0); \Box \succ \lambda x : \mathsf{Bool}.\mathsf{not}\, x \\
\longrightarrow_{\mathcal{K}} & \mathtt{app}(-, \mathsf{false}); \mathtt{if}\big(-, \lambda y : \mathsf{Nat}\,.y + 3, \lambda z : \mathsf{Nat}\,.z * 2\big); \mathtt{app}(-, 0); \Box \prec \lambda x : \mathsf{Bool}.\mathsf{not}\, x \\
\longrightarrow_{\mathcal{K}} & \mathtt{app}(\lambda x : \mathsf{Bool}.\mathsf{not}\, x, -); \mathtt{if}\big(-, \lambda y : \mathsf{Nat}\,.y + 3, \lambda z : \mathsf{Nat}\,.z * 2\big); \mathtt{app}(-, 0); \Box \succ \mathsf{false} \\
\longrightarrow_{\mathcal{K}} & \mathtt{app}(\lambda x : \mathsf{Bool}.\mathsf{not}\, x, -); \mathtt{if}\big(-, \lambda y : \mathsf{Nat}\,.y + 3, \lambda z : \mathsf{Nat}\,.z * 2\big); \mathtt{app}(-, 0); \Box \prec \mathsf{false} \\
\longrightarrow_{\mathcal{K}} & \mathtt{if}\big(-, \lambda y : \mathsf{Nat}\,.y + 3, \lambda z : \mathsf{Nat}\,.z * 2\big); \mathtt{app}(-, 0); \Box \succ (\mathsf{not}\, x)[x := \mathsf{false}] \\
\equiv & \mathtt{if}\big(-, \lambda y : \mathsf{Nat}\,.y + 3, \lambda z : \mathsf{Nat}\,.z * 2\big); \mathtt{app}(-, 0); \Box \succ \mathsf{not}\ \mathsf{false} \\
\longrightarrow_{\mathcal{K}} & \mathsf{not}(-); \mathtt{if}\big(-, \lambda y : \mathsf{Nat}\,.y + 3, \lambda z : \mathsf{Nat}\,.z * 2\big); \mathtt{app}(-, 0); \Box \succ \mathsf{false} \\
\longrightarrow_{\mathcal{K}} & \mathsf{not}(-); \mathtt{if}\big(-, \lambda y : \mathsf{Nat}\,.y + 3, \lambda z : \mathsf{Nat}\,.z * 2\big); \mathtt{app}(-, 0); \Box \prec \mathsf{false} \\
\longrightarrow_{\mathcal{K}} & \mathtt{if}\big(-, \lambda y : \mathsf{Nat}\,.y + 3, \lambda z : \mathsf{Nat}\,.z * 2\big); \mathtt{app}(-, 0); \Box \prec \mathsf{true} \\
\longrightarrow_{\mathcal{K}} & \mathtt{app}(-, 0); \Box \succ \lambda y : \mathsf{Nat}\,.y + 3 \\
\longrightarrow_{\mathcal{K}} & \mathtt{app}(-, 0); \Box \prec \lambda y : \mathsf{Nat}\,.y + 3 \\
\longrightarrow_{\mathcal{K}} & \mathtt{app}(\lambda y : \mathsf{Nat}\,.y + 3, -); \Box \succ 0 \\
\longrightarrow_{\mathcal{K}} & \mathtt{app}(\lambda y : \mathsf{Nat}\,.y + 3, -); \Box \prec 0 \\
\longrightarrow_{\mathcal{K}} & \Box \succ (y + 3)[y := 0] \\
\equiv & \Box \succ \mathtt{suma}(0, 3) \\
\longrightarrow_{\mathcal{K}} & \mathtt{suma}(-, 3); \Box \succ 0 \\
\longrightarrow_{\mathcal{K}} & \mathtt{suma}(-, 3); \Box \prec 0 \\
\longrightarrow_{\mathcal{K}} & \mathtt{suma}(0, -); \Box \succ 3 \\
\longrightarrow_{\mathcal{K}} & \mathtt{suma}(0, -); \Box \prec 3 \\
\longrightarrow_{\mathcal{K}} & \Box \prec 3
\end{array}
$$

3. Completud y Correctud de la máquina $\mathcal{K}$

Con la introducción de la máquina $\mathcal{K}$ hemos definido una nueva semántica dinámica para el lenguaje PCF. Es natural preguntarse si ambas semánticas serán equivalentes, es decir, si el proceso de evaluación mediante la máquina $\mathcal{L}$ arrojará el mismo resultado que el de la máquina $\mathcal{K}$, formalmente

Proposición 1 (Correctud y completud de la máquina $\mathcal{K}$) *Sean e una expresión del lenguaje* PCF *y v un valor. Entonces:*

$$e \to^\star v \;\; \textit{si y sólo si} \;\; \Box \succ e \longrightarrow_{\mathcal{K}} \Box \prec v$$

Se omiten los intrincados detalles de la demostración.

4. Seguridad de la máquina $\mathcal{K}$

Hasta ahora no hemos relacionado la nueva máquina $\mathcal{K}$ con la semántica estática de PCF . Para definir y probar la seguridad del lenguaje debemos darle sentido a los teoremas de preservación y progreso. Esta situación requiere nuevos juicios que describimos ahora.

- El estado s de la máquina $\mathcal{K}$ es correcto, denotado s ok.
- La pila $\mathcal{P}$ espera un valor de tipo T y devuelve una respuesta final de tipo S, denotado

$$\mathcal{P} : \mathsf{T} \Rightarrow \mathsf{S}$$

- El marco m espera un valor de tipo T y calcula un valor de tipo S, denotado

$$m : \mathsf{T} \Rightarrow \mathsf{S}$$

No debe surgir ambigüedad con los juicios de tipado $f : \mathsf{T} \to \mathsf{S}$ pues ahora la flecha es gorda y el objeto de la izquierda es una pila o un marco y no una expresión.

Estos juicios se definen como sigue:

- Si la pila $\mathcal{P}$ espera un valor de tipo T y responde con un valor de tipo S y e es una expresión cerrada de tipo T entonces el estado $\mathcal{P} \succ e$ es correcto.

$$\frac{\mathcal{P} : \mathsf{T} \Rightarrow \mathsf{S} \qquad \vdash e : \mathsf{T}}{(\mathcal{P} \succ e)\ \mathsf{ok}}$$

- Si la pila $\mathcal{P}$ espera un valor de tipo T y responde con un valor de tipo S y v es un valor de tipo T entonces el estado $\mathcal{P} \prec v$ es correcto.

$$\frac{\mathcal{P} : \mathsf{T} \Rightarrow \mathsf{S} \qquad \vdash v : \mathsf{T} \quad v \text{ valor}}{(\mathcal{P} \prec v)\ \mathsf{ok}}$$

- La pila vacía espera un valor de tipo T y devuelve una respuesta final del mismo tipo T.

$$\frac{}{\Box : \mathsf{T} \Rightarrow \mathsf{T}}$$

- Si la pila $\mathcal{P}$ espera un valor de tipo R y devuelve una respuesta de tipo S y el marco m espera un valor de tipo T y calcula un valor de tipo R entonces la pila $m;\mathcal{P}$ espera un valor de tipo T y devuelve una respuesta de tipo S.

$$\frac{\mathcal{P} : \mathsf{R} \Rightarrow \mathsf{S} \quad m : \mathsf{T} \Rightarrow \mathsf{R}}{m;\mathcal{P} : \mathsf{T} \Rightarrow \mathsf{S}}$$

- Se necesitan además las reglas para todos los marcos particulares, por ejemplo para la aplicación de funciones tenemos:

  - Si e_2 es una expresión cerrada de tipo S entonces el marco $\mathtt{app}(-, e_2)$ espera un valor de tipo $\mathsf{S} \to \mathsf{T}$ y calcula un valor de tipo T.

$$\frac{\vdash e_2 : \mathsf{S}}{\mathtt{app}(-, e_2) : (\mathsf{S} \to \mathsf{T}) \Rightarrow \mathsf{T}}$$

  - Si v es un valor de tipo $\mathsf{T} \to \mathsf{S}$ entonces el marco $\mathtt{app}(v, -)$ espera una expresión de tipo T y calcula un valor de tipo S.

$$\frac{\vdash v : \mathsf{T} \to \mathsf{S} \quad v \text{ valor}}{\mathtt{app}(v, -) : \mathsf{T} \Rightarrow \mathsf{S}}$$

Proposición 2 (Preservación de tipos para la máquina $\mathcal{K}$) *Si s ok y $s \longrightarrow_{\mathcal{K}} s'$ entonces s' ok.*

Proposición 3 (Progreso de la relación $\longrightarrow_{\mathcal{K}}$) *Si s ok entonces o bien s es final, es decir existe un valor v tal que $s = \Box \prec v$ o existe un s' tal que $s \longrightarrow_{\mathcal{K}} s'$.*

En conclusión la máquina $\mathcal{K}$ produce los mismos resultados que la evaluación mediante la semántica operacional usual (máquina $\mathcal{L}$) pero con una pila de control explícita que describe de manera más adecuada una implementación real.

Nuestros siguientes temas dentro del paradigma imperativo son las excepciones y las continuaciones, y se servirán de extensiones adecuadas a la máquina $\mathcal{K}$.