Programación Funcional

Instituto de Computación, Facultad de Ingeniería
Universidad de la República, Uruguay

Lazy evaluation

Modelo de computación

- En PF el modelo de computación es por reducción.
- La evaluación de una expresión se realiza mediante un proceso de reducción, en donde las definiciones de funciones actúan como reglas de cómputo.
- Vamos a escribir
  $e \longrightarrow e'$ para decir que e reduce en un paso a e', y
  $e \overset{*}{\longrightarrow} e'$ para decir que lo hace en varios pasos.
- En cada paso de la evaluación de una expresión se debe decidir que subexpresión reducir.
- Un **redex** (*reducible expression*) es una expresión formada por una función aplicada a argumentos, la cual está en condiciones de ser reducida.

Estrategias de evaluación

- Una expresión podría contener múltiples redexes. Surge entonces la pregunta de cuál redex reducir primero.
- Es en ese contexto que podemos plantearnos diferentes estrategias de evaluación.
- Hablaremos de dos estrategias que son clásicas.
- Se diferencian en el redex que seleccionan para reducir en cada paso.

Estrategias de evaluación

- Una expresión podría contener múltiples redexes. Surge entonces la pregunta de cuál redex reducir primero.
- Es en ese contexto que podemos plantearnos diferentes estrategias de evaluación.
- Hablaremos de dos estrategias que son clásicas.
- Se diferencian en el redex que seleccionan para reducir en cada paso.
- Para analizar estas dos estrategias consideremos la expresión:

  $$square\ (1 + 2)$$

  siendo

  $$square\ x = x * x$$

Estrategia leftmost innermost

En cada paso se reduce el redex más interno (no contiene otros redexes) y más a la izquierda.

$$
\begin{array}{ll}
& \mathit{square}\ \underline{(1+2)} \\
\longrightarrow & \quad \{\ \text{def. } +\ \} \\
& \underline{\mathit{square}\ 3} \\
\longrightarrow & \quad \{\ \text{def. } \mathit{square}\ \} \\
& \underline{3 * 3} \\
\longrightarrow & \quad \{\ \text{def. } *\ \} \\
& 9
\end{array}
$$

Estrategia leftmost innermost

En cada paso se reduce el redex más interno (no contiene otros redexes) y más a la izquierda.

$$
\begin{array}{ll}
& \mathit{square}\ \underline{(1+2)} \\
\longrightarrow & \quad \{\ \text{def. } +\ \} \\
& \underline{\mathit{square}\ 3} \\
\longrightarrow & \quad \{\ \text{def. } \mathit{square}\ \} \\
& \underline{3 * 3} \\
\longrightarrow & \quad \{\ \text{def. } *\ \} \\
& 9
\end{array}
$$

Corresponde a pasaje por valor (call-by-value).

Estrategia leftmost outermost

En cada paso se reduce el redex más externo (no contenido por otros redexes) y más a la izquierda.

$$
\begin{array}{ll}
& \underline{\mathit{square}\ (1+2)} \\
\longrightarrow & \quad \{\ \text{def. } \mathit{square}\ \} \\
& \underline{(1+2)} * (1+2) \\
\longrightarrow & \quad \{\ \text{def. } +\ \} \\
& 3 * \underline{(1+2)} \\
\longrightarrow & \quad \{\ \text{def. } *\ \} \\
& \underline{3 * 3} \\
\longrightarrow & \quad \{\ \text{def. } *\ \} \\
& 9
\end{array}
$$

Estrategia leftmost outermost

En cada paso se reduce el redex más externo (no contenido por otros redexes) y más a la izquierda.

$$
\begin{array}{ll}
& \underline{\mathit{square}\ (1+2)} \\
\longrightarrow & \quad \{\ \text{def. } \mathit{square}\ \} \\
& \underline{(1+2)} * (1+2) \\
\longrightarrow & \quad \{\ \text{def. } +\ \} \\
& 3 * \underline{(1+2)} \\
\longrightarrow & \quad \{\ \text{def. } *\ \} \\
& \underline{3*3} \\
\longrightarrow & \quad \{\ \text{def. } *\ \} \\
& 9
\end{array}
$$

Corresponde a pasaje por nombre (call-by-name).

Terminación

La evaluación de una expresión puede

- parar/terminar, retornando un valor, o
- seguir por siempre (no terminación).

Terminación

La evaluación de una expresión puede

- parar/terminar, retornando un valor, o
- seguir por siempre (no terminación).

Propiedad

Dos secuencias de reducción distintas de una expresión e que terminen lo van a hacer en el mismo valor v.

Terminación

La evaluación de una expresión puede

- parar/terminar, retornando un valor, o
- seguir por siempre (no terminación).

Propiedad

Dos secuencias de reducción distintas de una expresión e que terminen lo van a hacer en el mismo valor v.

Propiedad

Dada una expresión, si existe alguna secuencia de reducción que termina con cierto valor, entonces la estrategia call-by-name terminará produciendo ese resultado.

Terminación

La evaluación de una expresión puede

- parar/terminar, retornando un valor, o
- seguir por siempre (no terminación).

Propiedad

Dos secuencias de reducción distintas de una expresión e que terminen lo van a hacer en el mismo valor v.

Propiedad

Dada una expresión, si existe alguna secuencia de reducción que termina con cierto valor, entonces la estrategia call-by-name terminará produciendo ese resultado.

En cambio, call-by-value no garantiza llegar a ese resultado.

No terminación

Consideremos la siguiente definición:

$$inf :: Int$$
$$inf = 1 + inf$$

La evaluación de *inf* no termina independiente de la estrategia de evaluación.

$$\begin{aligned} inf &\longrightarrow 1 + inf \\ &\longrightarrow 1 + 1 + inf \\ &\longrightarrow 1 + 1 + 1 + inf \\ &\longrightarrow \ldots \end{aligned}$$

Decimos que su valor es indefinido.

Terminación: call-by-value

$$
\begin{aligned}
& \mathit{fst}\ (\mathit{length}\ [\,], \mathit{inf}\,) \\
\longrightarrow\ & \\
& \mathit{fst}\ (0, \mathit{inf}\,) \\
\longrightarrow\ & \\
& \mathit{fst}\ (0, 1 + \mathit{inf}\,) \\
\longrightarrow\ & \\
& \mathit{fst}\ (0, 1 + 1 + \mathit{inf}\,) \\
\longrightarrow\ & \\
& \mathit{fst}\ (0, 1 + 1 + 1 + \mathit{inf}\,) \\
\longrightarrow\ & \\
& \ldots
\end{aligned}
$$

La evaluación no termina.

Terminación: call-by-name

$$
\begin{array}{l}
\quad \mathit{fst}\ (\mathit{length}\ [\,], \mathit{inf}\,) \\
\longrightarrow \\
\quad \mathit{length}\ [\,] \\
\longrightarrow \\
\quad 0
\end{array}
$$

El redex inicial más externo es la propia expresión (*fst* aplicado al par).

Esto muestra que en call-by-name los argumentos son evaluados a demanda.

Terminación: call-by-name

$$three :: Int \to Int$$
$$three\ x = 3$$

$$three\ inf \longrightarrow 3$$

Terminación: call-by-name

$$
\begin{aligned}
& \mathit{length}\ [\mathit{inf}, 2] \\
\longrightarrow\ & \\
& 1 + \mathit{length}\ [2] \\
\longrightarrow\ & \\
& 1 + (1 + \mathit{length}\ [\,]) \\
\longrightarrow\ & \\
& 1 + (1 + 0) \\
\longrightarrow\ & \\
& 2
\end{aligned}
$$

Valor indefinido

- Al indefinido se lo suele denotar con el símbolo especial $\bot$, que se pronuncia "bottom".
- Denota tanto la no terminación como la detención anormal de un programa.
- Semánticamente, podemos pensar que existe un valor $\bot$ en todos los tipos.

  Por ejemplo,
  - el valor de *inf* es el valor $\bot$ del tipo *Int*.
  - el valor de $1/0$ es el valor $\bot$ del tipo *Float*.

Funciones estrictas

- Se dice que una función es **estricta** si cumple que

  $$f \perp = \perp$$

  Esto es, el resultado de la función es indefinido cuando la aplicamos a un argumento cuyo valor es indefinido.

- Caso contrario se dice que la función es **no estricta**.

- Con call-by-value todas las funciones son estrictas.

- Los operadores aritméticos $+$, *, -, $/$, *div*, *mod* son todas funciones estrictas incluso con call-by-name.

- La función *three* es no estricta con call-by-name.

Problema de call-by-name

Puede haber subexpresiones que se evaluan múltiples veces.
En nuestro ejemplo $1 + 2$.

$$
\begin{aligned}
&\textit{square}\ (1+2) \\
&\quad \longrightarrow\ (1+2) * (1+2) \\
&\quad \longrightarrow\ 3 * (1+2) \\
&\quad \longrightarrow\ 3 * 3 \\
&\quad \longrightarrow\ 9
\end{aligned}
$$

Propiedad

Con call-by-value los argumentos son evaluados una sola vez, en cambio con call-by-name pueden llegar a ser evaluados muchas veces.

Lazy evaluation

- Para evitar la múltiple evaluación de subexpresiones se utiliza una suerte de punteros que apuntan a aquellas subexpresiones que ocurren en varios lugares de una expresión.

- Las expresiones se representan entonces como grafos en lugar de como árboles.

- Supongamos que tenemos definida una función $f\ x = b$.

  Al hacer una llamada $(f\ e)$, en lugar de copiar físicamente el argumento e en todos los lugares de b donde ocurre el parámetro x, se va a mantener una única copia de e y se va a apuntar a dicha expresión desde las posiciones que estaba x.

$$f\ e \longrightarrow \textbf{let}\ p = e\ \textbf{in}\ b\ [x := p]$$

  donde hemos simulado el puntero mediante una variable local.

Lazy evaluation (2)

- De esta manera, cualquier reducción que se haga sobre e va a ser compartida por todas las posiciones en que ocurre p.
- Por lo tanto, la evaluación de un argumento que sería copiado en múltiples posiciones por call-by-name ahora se hace a lo más una vez.
- El uso de call-by-name junto con esta forma de sharing de subexpresiones se conoce como **lazy evaluation** o **call-by-need**.
- Notar que este es un mecanismo de evaluación que posee la máquina abstracta que reduce las expresiones y no algo en lo que interviene el programador.

Ejemplo de lazy evaluation

$$square\ x = x * x$$

$$\begin{aligned}
&square\ (1+2) \\
&\longrightarrow \textbf{let } p = 1+2 \\
&\qquad \textbf{in } p * p \\
&\longrightarrow \textbf{let } p = 3 \\
&\qquad \textbf{in } p * p \\
&\longrightarrow 9
\end{aligned}$$