

Facultad de Ciencias, UNAM

Lenguajes de Programación

Tarea 3

Hernández Salinas Óscar
Rubí Rojas Tania Michelle

26 de octubre de 2020

1. Evalúa las siguientes expresiones usando a) `alcance estático` y b) `alcance dinámico`. Es necesario que muestres el ambiente de evaluación final en forma de pila y en forma de lista en cada caso.

a)

```
{with {a 2}
  {with {b 3}
    {with {foo {fun {x} {- {+ a b} x}}}}
    {with {a -2}
      {with {b -3}
        {with {foo {fun {x} {+ {- a b} x}}}}
        {foo -10}}}}}}}
```

SOLUCIÓN:

- Alcance estático

Como la expresión que tenemos que evaluar es `{foo 10}`, entonces

```
{foo -10} = {{fun {x} {+ {- a b} x}} -10}
           = {+ {- a b} -10}
           = {+ {- (-2) (-3)} -10}
           = {+ 1 -10}
           = -9
```

Así, la representación del ambiente final se vería como:

- En forma de lista

```
((foo {fun {x} {+ {- a b} x}}) (x -10) (b -3) (a -2) (foo {fun {x} {- {+ a b} x}})
  (b 3) (a 2))
```

- En forma de pila

...
foo {fun {x} {+ {- a b} x}}
x -10
b -3
a -2
foo {fun {x} {- {+ a b} x}}
b 3
a 2
...

■ Alcance dinámico

Como la expresión que tenemos que evaluar es {foo -10}, entonces

$$\begin{aligned}
 \{\text{foo } -10\} &= \{\{\text{fun } \{x\} \{+ \{- a b\} x\}\} -10\} \\
 &= \{+ \{- a b\} -10\} \\
 &= \{+ \{- (-2) (-3)\} -10\} \\
 &= \{+ 1 -10\} \\
 &= -9
 \end{aligned}$$

Así, la representación del ambiente final se vería como:

- En forma de lista

((x -10) (foo {fun {x} {+ {- a b} x}}) (b -3) (a -2) (foo {fun {x} {- {+ a b} x}}) (b 3) (a 2))

- En forma de pila

...
x -10
foo {fun {x} {+ {- a b} x}}
b -3
a -2
foo {fun {x} {- {+ a b} x}}
b 3
a 2
...

b) {with {foo {fun {x} {+ x {foo {- x 1}}}}}

 {foo 10}}

SOLUCIÓN:

- Alcance estático

Como la expresión que tenemos que evaluar es {foo 10}, entonces

```
{foo 10} = {{fun {x} {+ x {foo {- x 1}}}} 10}
          = {+ 10 {foo {- x 1}}}
          = error
```

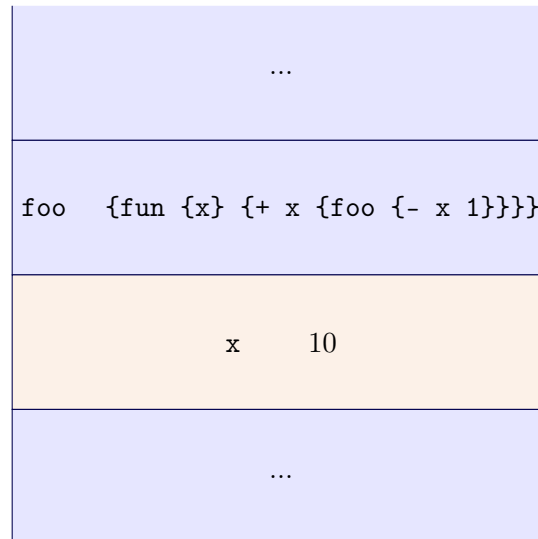
Por ser alcance estático, podemos encontrar el valor de x en la pila, pero no podemos volver a encontrar el valor de `foo` (ya que empezamos a buscar hacia abajo desde `foo`), así que tenemos un error de variable libre.

Así, la representación del ambiente final se vería como:

- En forma de lista

```
((foo {fun {x} {+ x {foo {- x 1}}})) (x 10))
```

- En forma de pila



- Alcance dinámico

Como la expresión que tenemos que evaluar es {foo 10}, entonces

```
{foo 10} = {{fun {x} {+ x {foo {- x 1}}}} 10}
          = {+ 10 {foo {- x 1}}}
          = {+ 10 {{fun {x} {+ x {foo {- x 1}}}} {- x 1}}}
          = {+ 10 {{fun {x} {+ x {foo {- x 1}}}} {- 10 1}}}
          = {+ 10 {{fun {x} {+ x {foo {- x 1}}}} 9}}
          = {+ 10 {+ 9 {foo {- x 1}}}}
          = ...
```

Notemos que, usando alcance dinámico, esta expresión se cicla. Puesto que buscamos desde el tope de la pila, vamos a ir encontrando los valores de x y nunca terminamos de llamar a la función `foo` (ésta siempre se llama a sí misma). Por lo tanto, esta expresión nunca termina.

Así, la representación del ambiente final se vería como:

- En forma de lista

```
(... (x 9) (x 10) (foo {fun {x} {+ x {foo {- x 1}}}}))
```

- En forma de pila

...
⋮
x 9
x 10
foo {fun {x} {+ x {foo {- x 1}}}}
...

c)

```

{with {x 2}
  {with {foo {fun {a} {+ x 2}}}}
  {with {y 3}
    {with {foo {fun {b} {- y b}}}
      {with {x 4}
        {with {goo {fun {b} {+ {foo x} {foo y}}}}
          {goo 3}}}}}}}
```

SOLUCIÓN:

- Alcance estático

Como la expresión que tenemos que evaluar es {goo 3}, entonces

```

{goo 3} = {{fun {b} {+ {foo x} {foo y}}} 3}
        = {+ {foo x} {foo y}}
        = {+ {foo x} {foo y}}
        = {+ {{fun {b} {- y b}} x} {{fun {b} {- y b}} y}}
        = {+ {{fun {b} {- y b}} 4} {{fun {b} {- y b}} 3}}
        = {+ {- y b} {- y b}}
        = {+ {- 3 4} {- 3 3}}
        = {+ (-1) 0}
        = -1
```

Así, la representación del ambiente final se vería como:

- En forma de lista

```

((goo {fun {b} {+ {foo x} {foo y}}}) (b 3) (x 4) (foo {fun {b} {- y b}}) (b 3) (b
4) (y 3) (foo {fun {a} {+ x 2}}) (x 2))
```

- En forma de pila

...
goo {fun {b} {+ {foo x} {foo y}}}
b 3
x 4
foo {fun {b} {- y b}}
b 3
b 4
y 3
foo {fun {a} {+ x 2}}
x 2
...

- Alcance dinámico

Como la expresión que tenemos que evaluar es {goo 3}, entonces

$$\begin{aligned}
 \{\text{goo } 3\} &= \{\{\text{fun } \{b\} \{+ \{\text{foo } x\} \{\text{foo } y\}\}\} 3\} \\
 &= \{+ \{\text{foo } x\} \{\text{foo } y\}\} \\
 &= \{+ \{\{\text{fun } \{b\} \{- y b\}\} x\} \{\{\text{fun } \{b\} \{- y b\}\} y\}\} \\
 &= \{+ \{\{\text{fun } \{b\} \{- y b\}\} 4\} \{\{\text{fun } \{b\} \{- y b\}\} 3\}\} \\
 &= \{+ \{- y b\} \{- y b\}\} \\
 &= \{+ \{- 3 4\} \{- 3 3\}\} \\
 &= \{+ (-1) 0\} = -1
 \end{aligned}$$

Así, la representación del ambiente final se vería como:

- En forma de lista

```
((b 3) (b 4) (b 3) (goo {fun {b {+ {foo x} {foo y}}}}) (x 4) (foo {fun {b} {- y
b}}) (y 3) (foo {fun {a} {+ x 2}}) (x 2))
```

- En forma de pila

...
b 3
b 4
b 3
goo {fun {b} {+ {foo x} {foo y}}}
x 4
foo {fun {b} {- y b}}
y 3
foo {fun {a} {+ x 2}}
x 2
...

2. Las primeras versiones del lenguaje LISP hacían uso de alcance dinámico y sus diseñadores se negaban a cambiarlo debido a una gran ventaja que traía consigo el uso de este tipo de alcance. Con base en los resultados obtenidos en el inciso (b) del ejercicio anterior, menciona esta ventaja.

SOLUCIÓN: En el ejercicio 1.b podemos notar que una ventaja del alcance dinámico es que nos permite hacer o *simular* ciclos (**while**, **for**). En este caso, no tenemos un caso base para poder evitar que se cicle, pero LISP sí tiene la posibilidad de poner condicionales (y por ende, casos base); así que esa ventaja está bonita porque en lenguajes donde no hay ciclos y sólo hay recursión, el alcance dinámico puede hacer más fácil el uso o *simulación* de los ciclos (ya que a veces pasar de usar un montón de ciclos a sólo tener recursión es un poco confuso y difícil).

Pero, también podemos mencionar un par de ventajas más sobre el alcance dinámico:

- Es más sencillo implementar un lenguaje de programación con este alcance, que con alcance estático.
 - En algunos casos, los parámetros que se pasan de un subprograma a otro son variables que se definen en el *llamador*. No es necesario pasar ninguno de ellos en un lenguaje de alcance dinámico, porque son implícitamente visibles en el subprograma *llamado*.
3. En clase se revisó que la función de *sustitución* es ineficiente, ya que en el peor caso es de orden cuadrático en relación al tamaño del programa (considerando el tamaño del programa como el número de nodos en el árbol de sintaxis abstracta). Por otro lado, se expuso una alternativa a este algoritmo de sustitución usando ambientes. Sin embargo, el implementar un ambiente usando una pila no parece ser mucho más eficiente.

- a) Da un programa que ilustre la no linealidad de la implementación basada en pilas y explica brevemente por qué su ejecución en tiempo no es lineal con respecto al tamaño de la entrada.

SOLUCIÓN: Supongamos que tenemos el siguiente programa, el cual tiene n variables de-ligado.

```
{with {x 1}
  {with {a_1 x}
    {with {a_2 x}
      ...
        {with {a_{n-1} x}}
          {+ x {+ a_1 {+ a_2 + {+ ... {+ a_{n-2} a_{n-1}} ... }}}}}}}
```

Notemos que esta expresión tiene n variables **de-ligado**. Usando un ambiente, por cada **with** guardamos el valor de x y a_i con $i \in \{1, 2, \dots, n-1\}$ en la pila; pero el valor de a_i es justamente el valor que tiene x (y éste ya fue agregado a la pila del ambiente), por lo que la complejidad será de $O(n^2)$ (sobre el número de variables en el ambiente) ya que se estará revisando cada uno de los nodos del árbol de sintaxis abstracta n veces.

- b) Describe una estructura de datos para un ambiente que un intérprete de FWAE pueda usar para mejorar su complejidad. Muestra cómo el intérprete usaría esta nueva estructura de datos. Indica además, cuál es la nueva complejidad del intérprete (análisis del peor caso).

SOLUCIÓN: Los diccionarios (o *hash tables*) son estructuras de datos que nos ayudan a asociar llaves con valores. La operación principal que soporta de manera eficiente es la búsqueda, ya que permite el acceso a los elementos almacenados a partir de una llave generada. Funciona transformando la llave con una *función hash* en un hash, un número que identifica la posición donde la tabla hash localiza el valor deseado.

Los diccionarios se suelen implementar sobre arreglos, por lo que proveen tiempo constante de búsqueda, sin importar el número de elementos en la tabla. Sin embargo, en casos particularmente malos, el tiempo de búsqueda puede llegar a ser lineal.

Así pues, nuestra propuesta es utilizar un diccionario como estructura de datos, ya que el código hash de cada una de las variables es su propia llave y su valor, lo que nos permitirá poder buscar en el ambiente el valor de las variables. Por lo que, si la primer variable ligada que nos encontramos es x (con un valor de 1), entonces podemos agregar esta información al diccionario (nuestro ambiente) y aplicar esto a cada una de las variables que vayamos encontrando.

Así, cada vez que mandemos a llamar una función, nos tomará tiempo constante (salvo casos particularmente malos) tomar el valor del último ambiente y ponerlo en nuestra función. En el peor caso, la función que mandamos a llamar tiene todas las variables que tenemos hasta el momento, lo que hará que nos tome tiempo lineal (con respecto al número de elementos en la *tabla hash*).

Referencias

- [1] Hash tables
https://en.wikipedia.org/wiki/Hash_table
- [2] What are the advantages of dynamic scoping
<https://stackoverflow.com/questions/321000/what-are-the-advantages-of-dynamic-scoping>
- [3] What are advantages and disadvantages of dynamic scoping
<http://ngocchan-nguyen.blogspot.com/2011/03/what-are-advantages-and-disadvantages.html>
- [4] What are the advantages of dynamic scoping
<https://www.quora.com/What-are-the-advantages-of-dynamic-scoping>