

75.31 Teoría de Lenguaje

Adrián Mouly (92501)

Primer Cuatrimestre de 2012

1. Ejercicio N°1

Corregir



1.1. Enunciado

Considere el siguiente programa en OZ:

1	local Res in
2	local Pi in
3	thread
4	Pi = 3.14
5	Res = Pi+Y
6	end
7	thread
8	{Delay 10000}
9	Y = 6.86
10	{Browse Res}
11	end
12	end
13	end

Ejecutarlo en la máquina abstracta vista en clase. Responda y justifique las siguientes preguntas:

1. ¿La aparición del identificador de variable Pi en la 5ª línea está libre o ligada?.
2. ¿Cuando se comience a ejecutar la línea 5 del código, la variable a la que se refiere el identificador Res está ligada a un valor o su valor es indeterminado?.
3. ¿La aparición del identificador de variable Y en la 5ª línea está libre o ligada?.
4. Después de 10 segundos y luego que se haya ejecutado completamente el segundo hilo (entre las líneas 7 y 11) ¿Qué es lo que pasa?.

1.2. Resolución

Cada estado de la computación se representa como (ST, σ) , donde σ es el store de variables y ST es una pila de sentencias semánticas de la forma:

$[(\langle s \rangle_0, E_0), (\langle s \rangle_1, E_1), \dots]$, siendo $\langle s \rangle_i$ una sentencia y E_i el conjunto que mapea identificadores de esa sentencia con variables del store. La ejecución del programa anterior resulta entonces:

- a. $([(\text{local Res in } \dots \text{ end}, \{Y \rightarrow y\})], \{y\})$

El store está vacío y la única sentencia apilada es el programa completo, sin mapeos de variables.

- b. $([(\text{local Pi in } \dots \text{ end}, \{Y \rightarrow y, \text{Res} \rightarrow \text{res}\})], \{y, \text{res}\})$

En este paso se ligan las variables antes definidas y se procede al siguiente local.

- c. $(\{[(\text{thread } \dots \text{ end})], [(\text{thread Pi} = 3.14 \text{ Res} = \text{Pi} + Y \text{ end}), \{\text{Res} \rightarrow \text{res}, \text{Pi} \rightarrow \text{pi}\}]\}, \{Y \rightarrow y, \{y, \text{res}, \text{pi}\}\})$

Se crea un hilo a parte del principal, con el contenido del primer thread que fue declarado. En el thread principal continúa apilado el segundo thread declarado.

- d. $(\{[], [(\text{Pi} = 3.14 \text{ Res} = \text{Pi} + Y), \{\text{Res} \rightarrow \text{res}, \text{Pi} \rightarrow \text{pi}\}]\}, [(\text{thread } \{\text{Delay } 10000\} Y = 6.86 \{\text{Browse Res}\} \text{ end})], \{Y \rightarrow y, \{y, \text{res}, \text{pi} \rightarrow 3.14\}\})$

Se desapila el último thread, quedando conformados 3 threads, en el cual el principal se encuentra vacío.

- e. $(\{[(\text{Pi} = 3.14 \text{ Res} = \text{Pi} + Y), \{\text{Res} \rightarrow \text{res}, \text{Pi} \rightarrow \text{pi}\}]\}, [(\text{Delay } 10000) Y = 6.86 \{\text{Browse Res}\}]\}, \{Y \rightarrow y, \{y \rightarrow 6.86, \text{res}, \text{pi} \rightarrow 3.14\}\})$

El sistema mata el primer thread y se continúa con la interpretación del tercero, el cual liga el valor de "Y" luego de una espera de 10000 milisegundos.

- f. $(\{[(\text{Pi} = 3.14 \text{ Res} = 3.14 + 6.86), \{\text{Res} \rightarrow \text{res}, \text{Pi} \rightarrow \text{pi}\}]\}, [(\text{Delay } 10000) Y = 6.86 \{\text{Browse Res}\}]\}, \{Y \rightarrow y, \{y \rightarrow 6.86, \text{res} \rightarrow 10, \text{pi} \rightarrow 3.14\}\})$

El primer hilo estaba esperando el valor de Pi, el cual es finalmente asignado, y por ende prosigue su ejecución.

- g. $(\{[(\text{Delay } 10000) Y = 6.86 \{\text{Browse } 10\}]\}, \{Y \rightarrow y, \{y \rightarrow 6.86, \text{res} \rightarrow 10, \text{pi} \rightarrow 3.14\}\})$

Al terminar la ejecución del primer hilo y al estar todas sus variables ligadas, termina su ejecución y el sistema lo mata, pasando la ejecución principal al segundo hilo, donde se muestra por pantalla el resultado final.

- h. $(\{\phi\}, \{\phi\}, \{y \rightarrow 6.86, \text{res} \rightarrow 10, \text{pi} \rightarrow 3.14\})$

Al terminar de ejecutarse todo el segundo hilo, este queda vacío esperando que el sistema lo finalice.

1. El identificador de Pi, en la 5ta línea se encuentra ligado a una variable interna, y esta misma a su vez se encuentra ligada a un valor "3.14". ✓

2. Cuando se comience a ejecutar esta línea, la variable a la que se refiere el identificador "Res", tiene un valor indeterminado. ✓

3. El identificador de la variable "Y" se encuentra ligado, pero el valor de la variable a la que apunta es indeterminado. No 

4. En el segundo hilo, "Y" se liga con el valor "6.86" y la función "Browse" se detiene. Al mismo tiempo, en el otro hilo, se determina el valor de "Res", finalizando el mismo. En el segundo hilo, al estar determinado el valor de "Res", puede concluir su ejecución mostrando por pantalla el valor "10".

2. Ejercicio N°2

2.1. Enunciado

Dada la siguiente tabla con algunas de las características del lenguaje de programación Oz, completar la columna de características equivalentes en su lenguaje preferido.

No se olvide de completar en el título el nombre del lenguaje.

La característica equivalente a completar, debe tener una sintaxis definida en su lenguaje favorito y ejecutarse de forma equivalente a la característica en Oz.

S, S1 y S2 representan sentencias arbitrarias.


S1 S2 significa una secuencia de dos sentencias

V una expresión que se puede calcular (por ejemplo $A + 3$)

1. Describa cualquier inconveniente que encuentre al realizar el pasaje a su lenguaje favorito indicando claramente las diferencias semánticas.
2. Existe algún equivalente en su lenguaje favorito a la sentencia case de Oz, si existe descríbala y si no existe proponer como se podría simular en su lenguaje.

2.1. Resolución

Caractrística en Oz	Equivalente en Go
skip	{ }
S1 S2	S1; S2;
local X in S1 end	{ var x <type>; S1; }
X=V	var X <type>; var V <type> = X;
if X then S1 else S2 end	if X { S1 } else { S2 }
{X Y1 Y2 Y3}	X(Y1, Y2, Y3)

1. Uno de los inconvenientes encontrados al realizar el pasaje de lenguaje, es que la sentencia “skip” no presenta una traducción equivalente en el lenguaje elegido. 

Otra diferencia, es que la definición de variables en Oz se realiza para un determinado entorno al comienzo del bloque. En cambio en C/Go, esta definición puede ser realizada en cualquier momento dentro de un scope.

2. Actualmente existe un compilador para el lenguaje Go que permite utilizar Pattern Matching en el lenguaje, ampliando el compilador estándar de Go.

Al igual que Oz, en Go también se utiliza la sentencia “case”, pero al ser Go un lenguaje de tipado estático, se analiza no solo el valor de un patrón, sino también su tipo.

Un inconveniente surge a la hora de analizar patrones recursivamente, es decir, analizar un patrón dentro de otro patrón, ya que Go no soporta herencia de tipos, y sin esta característica sería imposible analizar una lista dentro de otra lista, por ejemplo. Para esto, se aplica el concepto de “trait”.

Mediante estas técnicas implementadas sobre el compilador estándar de Go, se da soporte al Pattern Matching en este lenguaje.

Para demostrar el funcionamiento de estos conceptos, utilizamos un ejemplo donde se aplican.

Código ejemplo Pattern-Matching:

```
1 package main
2
3 import "fmt"
4
5 /* Al definir un trait, va a ser madre de otros tipos. */
6 type Expr trait {}
7
8 /* Defino 3 subtipos de Expr. */
9 type Constante casestruct borrows Expr {
10     value int
11 }
12
13 type Variable casestruct borrows Expr {
14     name string
15 }
16
17 type Multiple casestruct borrows Expr {
18     left Expr
19     right Expr
20 }
21 /* Fin definición de subtipos. */
22
23 func main() {
24     /* Creo una entrada inicializa con las constantes 1 y 20. */
25     entrada := Multiple(Constante(1), Constante(20))
26     fmt.Printf("%s\n", entrada)
27
28     /* Trata de matchear con distintos casos. */
29     match entrada {
30         /* Caso en el cual coincide tipo y valor. */
31         case Multiple(Constante(10), Constante(20)):
32             fmt.Printf("Matchea tipo y valor (10,20)")
33
34         /* Caso donde se realiza la asignación
35            Constante(1) => Constante(x) -> x := 1
36            Constante(20) => y := Constante(20) */
37         case Multiple(Constante(x), y):
38             fmt.Printf("Machea x=%v, y=%v\n", x, y)
39     }
40 }
```

En este ejemplo, se puede observar que fueron creados 4 tipos, de los cuales el primero sería el padre de los otros 3 subtipos. El tipo “Multiple”, puede almacenar dos valores, los cuales pueden ser del tipo padre “Expr” (Expresión).

En la función “main” se inicializa una estructura del tipo “Multiple” con dos valores constantes. Luego, con la sentencia “match” analiza los distintos casos, e imprime aquél caso que matchea.

El primer “case” lo que hace es crear un nuevo “Multiple” con dos valores

constantes (10 y 20), y el lenguaje se encarga de comparar el tipo (en este caso tipo "Constante", el cual matchea) y luego el valor con el cual fue creado (en este caso 10 y 20, lo cual no matchea con la estructura almacenada en "entrada"), dando por resultado, en este caso, "false", ya que no coincide completamente con la estructura "entrada".

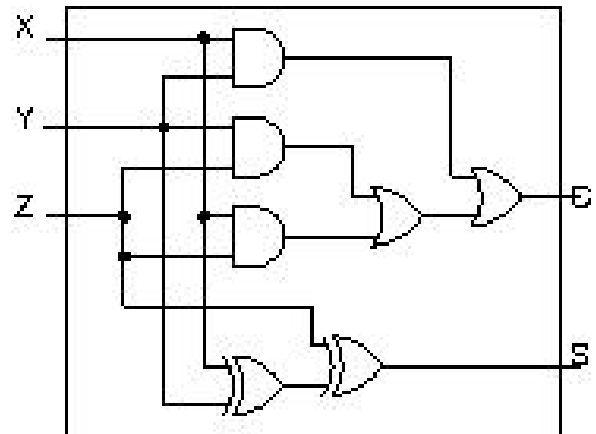
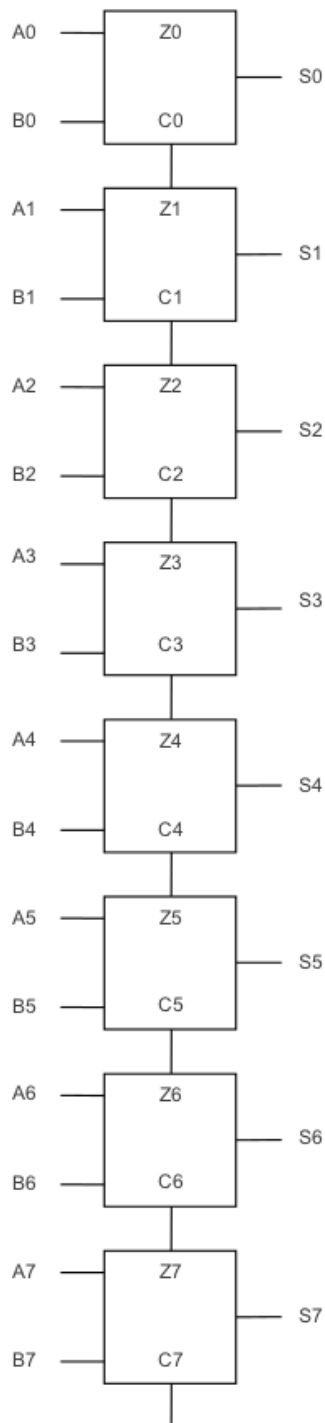
El segundo "case" crea un nuevo "Multiple", pero al ser inicializado con dos variables (x, y) las cuales no fueron declaradas, el lenguaje se encarga de asignarlas como alias de los valores de inicialización de "entradas". Esto produce que el matcheo sea positivo, ya que "x" es una "Constante" y "y" solo se trata de una variable, tomando el valor inicializado.

Este sistema permite realizar cosas más complejas, por ejemplo "Constante(Var(Expr(Constante(1))))" y de esta forma realizar un Pattern-Matching recursivo sobre cualquier tipo. ✓

3. Ejercicio N°3

3.1. Enunciado

Leer el capítulo 4 del libro y simular un sumador de 8 bits con compuertas lógicas, según el diagrama esquemático presentado a continuación. Cada compuerta debe ejecutarse en su propio hilo.



3.2. Resolución

```
1  declare R1 R2 R3 R4 R5
2
3  fun {ConstruirCompuerta F}
4      fun {$ Xs Ys}
5          fun {CicloCompuerta Xs Ys}
6              case Xs#Ys of (X|Xr)#(Y|Yr) then
7                  {F X Y}||{CicloCompuerta Xr Yr}
8              end
9          end
10     in
11         thread {CicloCompuerta Xs Ys} end
12     end
13 end
14
15 proc {SumadorCompleto X Y Z ?C ?S}
16     K L M CAnd COr CNand CNor CXor
17 in
18     CAnd = {ConstruirCompuerta fun {$ X Y} X*Y end}
19     COr = {ConstruirCompuerta fun {$ X Y} X+Y-X*Y end}
20     CNand= {ConstruirCompuerta fun {$ X Y} 1-X*Y end}
21     CNor = {ConstruirCompuerta fun {$ X Y} 1-X-Y+X*Y end}
22     CXor = {ConstruirCompuerta fun {$ X Y} X+Y-2*X*Y end}
23
24     K={CAnd X Y}
25     L={CAnd Y Z}
26     M={CAnd X Z}
27     C={COr K {COr L M}}
28     S={CXor Z {CXor X Y}}
29 end
30
31
32 fun {SumadorOchoBit X1 X2 X3 X4 X5 X6 X7 X8 Y1 Y2 Y3 Y4 Y5 Y6 Y7
33     Y8}
34     A0=X8|_ B0=Y8|_ C0 Z0=0|_ S0
35     A1=X7|_ B1=Y7|_ C1 Z1      S1
36     A2=X6|_ B2=Y6|_ C2 Z2      S2
37     A3=X5|_ B3=Y5|_ C3 Z3      S3
38     A4=X4|_ B4=Y4|_ C4 Z4      S4
39     A5=X3|_ B5=Y3|_ C5 Z5      S5
40     A6=X2|_ B6=Y2|_ C6 Z6      S6
41     A7=X1|_ B7=Y1|_ C7 Z7      S7
42 in
43     {SumadorCompleto A0 B0 Z0 C0 S0}
44     {SumadorCompleto A1 B1 C0 C1 S1}
45     {SumadorCompleto A2 B2 C1 C2 S2}
46     {SumadorCompleto A3 B3 C2 C3 S3}
47     {SumadorCompleto A4 B4 C3 C4 S4}
48     {SumadorCompleto A5 B5 C4 C5 S5}
```

49	{SumadorCompleto A6 B6 C5 C6 S6}
50	{SumadorCompleto A7 B7 C6 C7 S7}
51	
52	C7 S7 S6 S5 S4 S3 S2 S1 S0
53	end
54	
55	R1={SumadorOchoBit 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1}
56	R2={SumadorOchoBit 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1}
57	R3={SumadorOchoBit 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 1}
58	R4={SumadorOchoBit 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1}
59	R5={SumadorOchoBit 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1}
69	
61	{Browse R1}
62	{Browse R2}
63	{Browse R3}
64	{Browse R4}
65	{Browse R5}

En este código se implementa un SumadorOchoBit, el cual recibe dos números de 8 bits y devolviendo la suma de ellos, más un valor de acarreo para su mejor verificación.

Como funciones auxiliares se definieron las distintas compuertas lógicas enunciadas en el gráfico, siendo estas parte del SumadorCompleto. Este se encarga de abstraer los distintos casos enunciados en el gráfico. Las compuertas son creadas en base a la función ConstruirCompuerta, la cual recibe una condición de compuerta y la ejecuta recursivamente.

Finalmente, la función SumadorOchoBit se encarga de controlar el stream de datos que se quiere sumar, y enviando estos valores al SumadorCompleto. Una vez realizada la suma, se devuelve una lista con el resultado final y un valor de acarreo.

En el programa principal se realiza una prueba del contador, sumando varios pares de números por separado, y luego mostrando los resultados por pantalla.

4. Ejercicio N°4

4.1. Enunciado

Implementar en Oz, un procedimiento que reciba una lista con letras y muestre por pantalla la frecuencia de aparición de cada letra, ordenada por frecuencia y a igual cantidad de apariciones ordenadas alfabéticamente.

Usar como estructura de datos un diccionario con las siguientes características:

1. Diccionario declarativo, seguro y desempquetado.
2. Diccionario con estado, inseguro y empaquetado. ¿Para qué puede ser útil un tipo de datos con esas características? Comparar con las clases en Python.

Por ejemplo dada la lista de entrada:

```
[e l q u e s a b e s a b e y e l q u e n o e s j e f e]
```

Debería mostrar:

```
e: 9  
s: 3  
a: 2  
b: 2  
l: 2  
q: 2  
u: 2  
f: 1  
j: 1  
n: 1  
o: 1  
y: 1
```

El TAD Diccionario debe soportar las siguientes primitivas:

{NewDicc}: devuelve un diccionario vacío

{Put Dicc Clave Valor}: inserta el par Clave:Valor en el diccionario

{Get Dicc Clave}: devuelve el Valor asociado a la Clave

{Domain Dicc}: devuelve una lista con todas las claves presentes en el diccionario Dicc.

4.2. Resolución

Implementación de un Diccionario con estado, inseguro y empaquetado en Go

```
1 package main
2
3 import (
4     "fmt"
5     "sort"
6 )
7
8 /* Creo una estructura de mapa ordenado */
9 type mapaOrdenado struct {
10     m map[string]int
11     s []string
12 }
13
14 func (sm *mapaOrdenado) Len() int {
15     return len(sm.m)
16 }
17
18 func (sm *mapaOrdenado) Less(i, j int) bool {
19     return sm.m[sm.s[i]] > sm.m[sm.s[j]]
20 }
21
22 func (sm *mapaOrdenado) Swap(i, j int) {
23     sm.s[i], sm.s[j] = sm.s[j], sm.s[i]
24 }
25
26 func ordenarClaves(m map[string]int) []string {
27     sm := new(mapaOrdenado)
28     sm.m = m
29     sm.s = make([]string, len(m))
30     i := 0
31     for key, _ := range m {
32         sm.s[i] = key
33         i++
34     }
35
36     sort.Sort(sm)
37     return sm.s
38 }
39
40 /* Estructura de Diccionario */
41 type Diccionario map[string]int
42
43 /* Creo un Diccionario vacío y lo devuelvo. */
44 func NuevoDicc() (nuevoDicc Diccionario) {
45     nuevoDicc = make(map[string]int)
```

```

46
47     fmt.Println("Se creo el dicc")
48
49     return nuevoDicc
50 }
51
52 /* {Put Dicc Clave Valor}: inserta el par Clave:Valor en el
53 diccionario */
54 func (unDicc *Diccionario) Put(clave string, valor int) {
55     (*unDicc)[clave] = valor
56 }
57
58 /* {Get Dicc Clave}: devuelve el Valor asociado a la Clave */
59 func (unDicc *Diccionario) Get(clave string) (valor int, existe
60 bool) {
61     if (*unDicc)[clave] != 0 {
62         return (*unDicc)[clave], true
63     }
64
65     return 0, false
66 }
67
68 /* {Domain Dicc}: devuelve una lista con todas las claves
69 presentes en el diccionario Dicc */
70 func (unDicc *Diccionario) Domain() (unaLista []string) {
71     unaLista = ordenarClaves((*unDicc))
72
73     return unaLista
74 }
75
76 func main(){
77     /* Creo una Cadena y un Diccionario */
78     unaCadena := "elquesabesabeyelqueno es jefe"
79     unDiccionario := NuevoDicc()
80
81     /* Cargo el Diccionario con los caracteres de la Cadena */
82     for _, vC := range(unaCadena) {
83
84         vD, ok := unDiccionario.Get(string(vC))
85
86         if ok {
87             /* El valor vC existe y se aumenta a vC+1 */
88             unDiccionario.Put(string(vC), vD+1)
89         } else {
90             /* El valor vC NO existe y se aumenta a 1 */
91             unDiccionario.Put(string(vC), 1)
92         }
93     }
94
95     dominio := unDiccionario.Domain()
96
97

```

98	/* Recorre e imprime el diccionario ordenado */
99	for _, v := range (dominio){
100	cant, _ := unDiccionario.Get(v)
101	fmt.Println(v, ":", cant)
102	}
	}

Se implementó el diccionario del punto 4.2, ya que Go utiliza un modelo de estados. También es del tipo empaquetado ya que las funciones están incorporadas a los tipos. Y por último, es inseguro ya que el encapsulamiento puede ser definido por el programador debido a que el lenguaje no fuerza a hacerlo.