HOJA 2 Ejercicio 2

Ejercicio realizado por **FRANCISCO JAVIER MORENO**

(septiembre 2008)

Se considera una instrucción de iteración con el siguiente formato:
hasta que *Var* **se anule hacer**
se pasa => I_0
no llega => I_1
reduce *Var*
fin hasta
donde *Var* es una variable entera, e I_0 e I_1 son instrucciones. Su semántica operacional informal es como sigue:

1. Si *Var* = 0, terminar la iteración
2. Si *Var* > 0, ejecutar I_0
3. Si *Var* < 0, ejecutar I_1
4. Si *Var* > 0, *Var* *Var* -1
5. Si *Var* < 0, *Var* *Var* +1
6. ir a 1

Se pide:

a) **[1 punto]** Traduce a código-p las siguientes instrucciones comentando el código generado:

```
j:= 100;
i:= -7 ;
hasta que i se anule hacer
se pasa => j := j - i
no llega => j := j + i
reduce i
fin hasta;
```

Para traducir considera que las variables `i y j` han sido declaradas en el programa principal, al que también pertenece este fragmento de código, y que se utiliza un modelo de memoria que asigna direcciones absolutas a las variables*. Utiliza las instrucciones apila-dir x , desapila-dir x, donde el argumento x es una dirección absoluta de la memoria de datos.*

b) **[2 puntos]** Formaliza mediante una gramática de atributos la traducción de esta instrucción al lenguaje de la máquina P sin etiquetas simbólicas. Haz primero un esquema que muestre cómo se organiza la traducción.

Semántica operacional:

1. Si *Var* = 0, terminar la iteración
2. Si *Var* > 0, ejecutar I_0
3. Si *Var* < 0, ejecutar I_1
4. Si *Var* > 0, *Var* ← *Var* - 1
5. Si *Var* < 0, *Var* ← *Var* + 1
6. ir a 1

Diagrama de bloques:

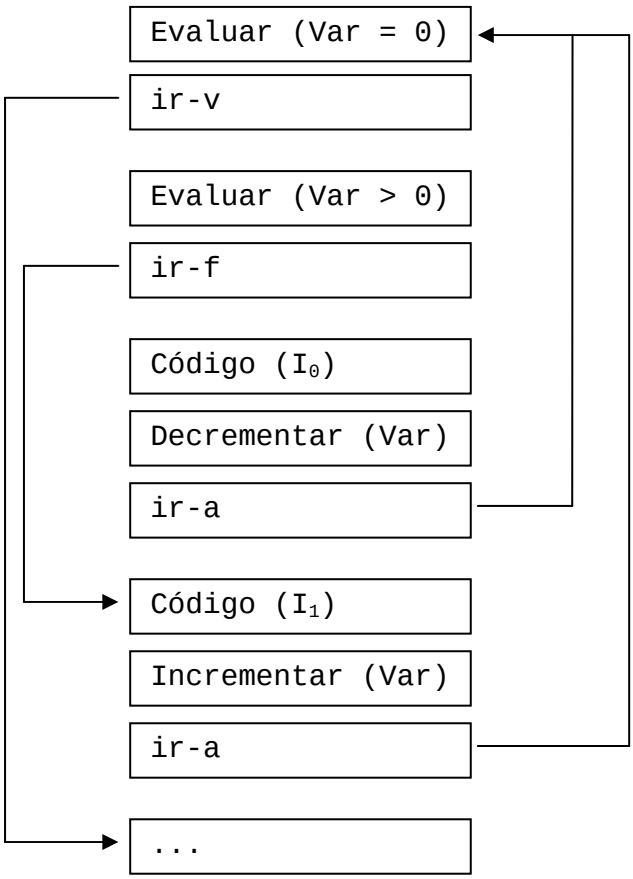
```
Evaluar (Var = 0):
   apila-dir(Var.dir)
   apila(0)
   igual

Evaluar (Var > 0):
   apila-dir(Var.dir)
   apila(0)
   mayor



Decrementar (Var):
   apila-dir(Var.dir)
   apila(1)
   resta
   desapila-dir(Var.dir)


Incrementar (Var):
   apila-dir(Var.dir)
   apila(1)
   suma
   desapila-dir(Var.dir)
```



<u>Código-p de las siguientes instrucciones:</u>

```
j:= 100;
i:= -7 ;
hasta que i se anule hacer
se pasa => j := j - i
no llega => j := j + i
reduce i
fin hasta;
```

```
1) apila(100)
2) desapila-dir(1)
3) apila(-7)
4) desapila-dir(0)
5) apila-dir(0)
6) apila(0)
7) igual
8) ir-v(31)
9) apila-dir(0)
10) apila(0)
11) mayor
12) ir-f(22)
13) apila-dir(1)
14) apila-dir(0)
15) resta
16) desapila-dir(1)
17) apila-dir(0)
18) apila(1)
19) resta
20) desapila-dir(0)
21) ir-a(5)
22) apila-dir(1)
23) apila-dir(0)
24) suma
```

```
25) desapila-dir(1)
26) apila-dir(0)
27) apila(1)
28) suma
29) desapila-dir(0)
30) ir-a(5)
31) ...
```

Consideraciones:

1. El código-p se genera siguiendo el orden de la sintaxis de la instrucción.

2. Las variables i y j ya han sido declaradas en el programa principal. La variable i está en la posición 0 de memoria y j está en la posición 2.

3. Debemos comparar dos veces la variable i con 0: primero, para ver si i es igual a 0 (en ese caso, salimos del bucle); y después, si i es mayor que 0 (en caso de no serlo, se salta a la instrucción 22).

Gramática de atributos para la traducción a código-p.

Atributo etiq

I_0.etiqh = IHasta.etiqh + 8
I_1.etiqh = I_0.etiq + 5
IHasta.etiq = I_1.etiq + 5

Atributo cod

I_0.codh = IHasta.codh || eval-igual0(Var) || ir-v(IHasta.etiq) ||
eval-mayor0(Var)||ir-f(I1.etiqh)
I_1.codh = I_0.cod || decrementar(Var) || ir-a(IHasta.etiqh)
IHasta.cod = I_1.cod || incrementar(Var) || ir-a(IHasta.etiqh)

eval-igual0(Var) = apila-dir(Var.dir) || apila(0) || igual
eval-mayor0(Var) = apila-dir(Var.dir) || apila(0) || mayor
decrementar(Var) = apila-dir(Var.dir) || apila(1) || resta ||
desapila-dir(Var.dir)
incrementar(Var) = apila-dir(Var.dir) || apila(1) || suma ||
desapila-dir(Var.dir)
Var.dir = obtenerDir(Var.lex,IHasta.tsh)

Acondicionamiento de la gramática:

Atributo etiq

I_0.etiqh = IHasta.etiqh + 8
I_1.etiqh = I_0.etiq + 5
IHasta.etiq = I_1.etiq + 5

Atributos auxiliares

IHasta.etiqParcheoSalida = IHasta.etiqh + 3
IHasta.etiqParcheoMenor = IHasta.etiqh + 7
IHasta.etiqMenor = I_0.etiq + 5
IHasta.etiqSalida = I_1.etiq + 5

Atributo cod

I_0.codh = IHasta.codh || eval-igual0(Var) || ir-v(?) ||
eval-mayor0(Var) || ir-f(?)
I_1.codh = parchea(IHasta.etiqParcheoMenor, IHasta.etiqMenor,
I_0.cod || decrementar(Var) || ir-a(IHasta.etiqh))

```
IHasta.codh = parchea(IHasta.etiqParcheoSalida, IHasta.etiqSalida,
I1.cod || incrementar(Var) ||
ir-a(IHasta.etiqh)

eval-igual0(Var) = apila-dir(Var.dir) || apila(0) || igual
eval-mayor0(Var) = apila-dir(Var.dir) || apila(0) || mayor
decrementar(Var) = apila-dir(Var.dir) || apila(1) || resta ||
desapila-dir(Var.dir)
incrementar(Var) = apila-dir(Var.dir) || apila(1) || suma ||
desapila-dir(Var.dir)
Var.dir = obtenerDir(Var.lex,IHasta.tsh)
```

Esquema de traducción:

```
IHasta ->      hasta que Var {
                  dirVar = obtenerDir(Var.lex, IHasta.tsh)
               }
               se anule hacer {
                  dirVuelta = contProg;
                  emite(apila-dir(dirVar));
                  emite(apila(1));
                  emite(igual);
                  dirParcheoIgual = contProg;
                  emite(ir-v(?));
               }
               si se pasa {
                  emite(apila-dir(dirVar));
                  emite(apila(1));
                  emite(mayor);
                  dirParcheoMayor = contProg;
                  emite(ir-f(?));
               }
               I0 {
                  emite(apila-dir(dirVar));
                  emite(apila(1));
                  emite(resta);
                  emite(desapila-dir(dirVar));
                  emite(ir-a(dirVuelta));
               }
               si no llega I1 {
                  dirMenor = contProg;
                  emite(apila-dir(dirVar));
                  emite(apila(1));
                  emite(suma);
                  emite(desapila-dir(dirVar));
                  emite(ir-a(dirVuelta));
                  dirSalida = contProg;
                  parchea(dirParcheoMayor, dirMenor);
               }
               fin hasta {
                  parchea(dirParcheoIgual, dirSalida);
               }
```