Hoja II.5

SOLUCIÓN DEL EJERCICIO 3

Programa while P tal que $\varphi_P = C_A$ con A = { x: $|x|_a > 0$ }.

La función característica de A devuelve **a** si $x \in A$ (es decir si la entrada contiene alguna **a**) y ε en caso contrario. Por tanto, un programa posible es el siguiente (evitamos instrucciones supérfluas para facilitar luego la codificación):

```
while nonem?(X1) loop
      if car_a?(X1) then X1:= ε; X0:= cons_a(X0); end if;
      X1:=cdr(X1);
end loop;
```

Un índice correspondiente a φ_P con el alfabeto $\sum$ = {a, b}.

Tenemos en cuenta que el alfabeto tiene dos símbolos, los códigos reservados para cada tipo de programa-while son:

Asignación vacía XI:= ε;	$cod^2(0, I)$
Asignación $cons_a$ XI:= $cons_a$(XJ);	$cod^3(1, I, J)$
Asignación cdr XI:= cdr(XJ);	$cod^3(3, I, J)$
Composición P Q	$cod^3(4, \Re_W(P), \Re_W(Q))$
Condicional car_a **if** car_a?(XI) **then** P **end if**;	$cod^3(5, I, \Re_W(P))$
Iteración **while** nonem?(XI) **loop** P **end loop**;	$cod^2(7+I, \Re_W(P))$

Por tanto la codificación del programa la hallaríamos calculando la siguiente expresión:

$$cod^2(8,\ cod^3(4,\ cod^3(5, 1, cod^3(4, cod^2(0, 1), cod^3(1, 0, 0)\ )\ ), cod^3(3, 1, 1)\ )\ )$$

teniendo en cuenta que $cod^2(i, j) = (i+j)*(i+j+1)/2+j$ y $cod^3(i, j, k) = cod^2(i, cod(i, k))$.

Si consideramos el alfabeto $\sum$ **= {a}** serviría el mismo programa, ya que en las llamadas a *cons* y *car* que incorpora solo figura el símbolo **a** (otra cosa es que probablemente preferiríamos escribir otro más sencillo que no incluyera la condicional). La codificación, empero, no sería la misma, ya que los códigos de operación reservados a cada tipo de programa-while dependen de la cardinalidad del alfabeto, con lo que sólo habría un código (el 1) para las asignaciones *cons*, y otro (el 4) para las condicionales. Es decir, que, por ejemplo, el código de nuestro programa sería de la forma $cod^2(6, \ldots)$, en lugar de $cod^2(8, \ldots)$, por lo que ya no podría ser el mismo que antes.

Hoja II.5

SOLUCIÓN DE ALGUNOS APARTADOS DEL EJERCICIO 4

Hoja II.5- 4 c – Nivel de anidamiento

Iremos descomponiendo el programa para analizarlo. Tendremos una pila en la que guardaremos pares (**P**, **n**), donde **P** es un subprograma de X1 y **n** es el nivel de anidamiento a que se ha encontrado. Inicialmente la pila contendrá únicamente el par (X1, 0). En cada iteración procesamos uno de estos pares, actualizando el máximo provisional (en X0) y devolviendo en su caso a la pila los subprogramas de P producidos con sus respectivos niveles.

```
PILA := empilar(cod_2(0, X1), <]);
X0 := 0;
while  not P_vacía?(PILA)  loop
      NIVEL := decod_2_1(cima(PILA));
      PROG := decod_2_2(cima(PILA));
      PILA := desempilar(PILA);
      if  asig_vacía?(PROG) or asig_cons?(PROG) or
            asig_cdr?(PROG)  then
                  if NIVEL > X0 then
                        X0 := NIVEL;
                  end if;
      elsif  composicion?(PROG)  then
            PILA := empilar(cod_2(NIVEL, inst_int_2(PROG)), PILA);
            PILA := empilar(cod_2(NIVEL, inst_int(PROG)), PILA);
      else
            PILA := empilar(cod_2(NIVEL+1, inst_int(PROG)), PILA);
      end if;
end loop;
```

Hoja II.5- 4-d posible_ciclo

Iremos descomponiendo el programa para analizar cada bucle. Tendremos una pila adicional en la que analizaremos los bucles, uno a uno, a medida que los encontremos. En la pila de análisis de los bucles estudiamos cada asignación viendo si la variable del lado izquierdo coincide con la del bucle. Si en algún bucle no encontramos una asignación de esta forma hay un posible ciclo.

```
PILA := empilar(X1, <]);
X0 := false;
while  not P_vacía?(PILA) and not X0 loop
      PROG := (cima(PILA);
      PILA := desempilar(PILA);
      if  composicion?(PROG)  then
            PILA := empilar(inst_int_2(PROG), PILA);
            PILA := empilar(inst_int(PROG), PILA);
      elsif condicion?(PROG)  then
            PILA := empilar(inst_int(PROG), PILA);
      elsif iteracion?(PROG)  then
            PILA := empilar(inst_int(PROG), PILA);
            PILA_BUCLE := empilar(inst_int(PROG), <]);
            VAR_BUCLE := ind_var(PROG);
            MODVAR := false;
            while  not P_vacía?(PILA_BUCLE) and not MODVAR loop
                  INSTR:= cima(PILA_BUCLE);
                  PILA_BUCLE:= desempilar(PILA_BUCLE);
                  if asig_vacía?( INSTR) or asig_cons?( INSTR) or
                        asig_cdr?( INSTR)  then
                        MODVAR := ind_var(INSTR) = VAR_BUCLE;
                  elsif composicion?(PROG)  then
                        PILA_BUCLE:= empilar(inst_int_2(INSTR), PILA_BUCLE);
                        PILA_BUCLE:= empilar(inst_int(INSTR), PILA_BUCLE);
                  else
                        PILA_BUCLE:= empilar(inst_int(INSTR), PILA_BUCLE);
                  end if;
            end loop;
            X0:= not MODVAR;
      end if;
      end loop;
```

Hoja II.5- 4-g Quita y pon

Para distinguir si las instrucciones estaban seguidas en el texto del programa necesitamos comprobar que se encuentran en el mismo nivel, por lo que al ir descomponiendo el programa para analizarlo guardamos además de la instrucción el nivel. Tendremos una pila con pares (**n**, **P**), donde **P** es un subprograma de X1 y **n** nos indica el **nivel de anidamiento** del subprograma.

Una variable **ALARMA** nos informa si ha aparecido una instrucción de la forma XI:= $cons_a$(XI); y a qué nivel. Como informa de dos cuestiones también la representamos como un par (i+1, n). El primer elemento del par nos indica que hay alarma (puede haber dos instrucciones superfluas si es mayor que cero) y cuál es el número de la variable. El segundo elemento del par nos informa del nivel del subprogarma.

Inicialmente la pila contendrá únicamente el par (0, X1) y ALARMA será el para (0,0). En cada iteración procesamos uno de los pares de la pila devolviendo, en su caso, a la pila los subprogramas de P producidos con sus respectivos niveles y comprobando si es una instrucción de alarma o si se han detectado dos instrucciones "superfluas".

```
ALARMA: = cod_2(0,0); NIVEL := 0;
PILA := empilar(cod_2(NIVEL, X1), <]);
X0 := false;
while  not P_vacía?(PILA) and not X0 loop
      NIVEL.= decod_2_1(cima(PILA));
      PROG := decod_2_2(cima(PILA));
      PILA := desempilar(PILA);
      if  composicion?(PROG)  then
            PILA := empilar(cod_2(NIVEL, inst_int_2(PROG)), PILA);
            PILA := empilar(cod_2(NIVEL, inst_int(PROG)), PILA);
      elsif condicion?(PROG) or iteración?(PROG) then
            ALARMA:= 0;
            PILA := empilar(cod_2(NIVEL+1, inst_int(PROG)), PILA);
      elsif asig_cons?(PROG) and ind_var?(PROG) = ind_var_2?(PROG) then
                  ALARMA := cod_2(ind_var(PROG) + 1, NIVEL);
      elsif asig_cdr?(PROG) and ALARMA > 0
                  and NIVEL = decod_2_2(ALARMA) and
                  decod_2_1(ALARMA) –1 = ind_var?(PROG) and
                   decod_2_1(ALARMA) –1 = ind_var_2?(PROG) then
                  X0:= true;
      else ALARMA := 0;
      end_if;
end loop;
```