PRAGMATICA DE *P*

Se utilizará el lenguaje P para escribir programas más complejos que los vistos hasta ahora. Por razones de comodidad, introduciremos nuevas construcciones, "abreviaciones" de secuencias de instrucciones en P. Se mantiene así la semántica que se le ha dado a ***P***.

Lo que haremos será introducir *macros* que se expandirán completamente en el texto del programa.

Se definirán macros para las sentencias, expresiones, condiciones y constantes.

Macros para las sentencias;

SENT ***abreviación*** **MACRO:**
-- declaraciones
<sent>
END

En las macros se utilizan macrovariables Y0 ,Y1 ,... las que en la expansión, serán sustituídas por una variable ***P***.

Ejemplo:

SENT ASSIGN(Y1 ,Y2) MACRO:
-- Y1:Var , Y2:Var
Y1 := SUC(Y2) ;
Y1 := PRED(Y1)
END

Al introducir macros, es un buen hábito probar que se comportan exactamente como queremos.

$$\frac{}{\langle SUC(y),\sigma\rangle \Rightarrow s(\sigma(y))}[E2] \qquad \frac{}{\langle PRED(x),\sigma[x\rightarrow s(\sigma(y))]\rangle \Rightarrow \sigma(y)}[E4]$$

$$\frac{}{\langle X:=SUC(y),\sigma\rangle \Rightarrow \sigma[x\rightarrow s(\sigma(y))]}[A] \qquad \frac{}{\langle X:=PRED(x),\sigma[x\rightarrow s(\sigma(y)]\rangle \Rightarrow \sigma'}[A]$$

$$\frac{}{\langle ASSIGN(x,y),\sigma\rangle \Rightarrow \sigma[x\rightarrow\sigma(y)]}[S]$$

donde $\sigma' = \sigma[x \rightarrow s(\sigma(y)), x \rightarrow\sigma(y)] = \sigma[x \rightarrow\sigma(y)]$

En el texto de la macros se puede usar también variables locales.

Ejemplo.

```
SENT IF  Y1≠ 0  THEN Y2  FI  MACRO
  -- Y1: Var , Y2:Sent
  -- Y3: Var  (variable local)
        ASSIGN(Y3,Y1) ;
        WHILE Y3≠ 0 DO
              Y2 ;
              Y3:= 0
        END
END
```

Las variables locales se cambian a variables con índices mayores que el mayor índice que haya en el programa (o mejor aún, en el programa expandido).

Las macro-definiciones para las expresiones son:

```
EXPR abreviación MACRO:
 --  declaraciones
        <sent>
RESULT (<macro-var>)
```

Ejemplo.

```
EXPR Y1+Y2 MACRO:
 -- Y1,Y2:Var
        WHILE Y2 ≠ 0 DO
              Y1:= SUC(Y1) ;
              Y2:= PRED(Y2)
        END
RESULT(Y1)
```

El tratamiento de las macros para las expresiones, es más complejo que el de las sentencias dado que se deben evitar cambios no deseados en los valores de las variables.

Por lo tanto, trataremos a las variables como variables locales inicializadas con las expresiones dadas en la macro-llamada.

Ejemplo:
Considere el siguiente programa.

```
PROGRAM (X1)
        X2:= X1+X1
RESULT(X2)
```

se expande en :

```
PROGRAM (X1)
  ASSIGN (X3,X1) ;  ---------- X3 := SUC(X1);
  ASSIGN (X4,X1) ;                X3 := PRED(X3);
  WHILE X4 ≠ 0  DO
        X3:= SUC(X3) ;
        X4:= PRED(X4) ;
  END;
  ASSIGN (X2,X3)
RESULT (X2)
```

Definiremos también macros para constantes, como por ejemplo;

```
EXPR  1  MACRO:
        Y1:= 0 ;
        Y1:= SUC(Y1)
RESULT (Y1)
```

```
EXPR  2  MACRO:
        Y1:= 1 ;
        Y1:= SUC(Y1)
RESULT (Y1)
```

etc.

Macros de Condición.

```
COND  "abreviación"  MACRO:
  -- declaraciones
        <sent>
RESULT (<macro-var>)
```

Ejemplo.

```
COND (Y1≠ 0 ) & (Y2≠ 0 ) MACRO
  --  Y1,Y2:Var;
        Y3:= 0 ;
        IF Y1≠ 0 THEN
                IF Y2≠ 0 THEN
                        Y3:= SUC(Y3)
                FI
        FI
RESULT (Y3)
```

Más adelante se van a usar:

- ES_PRIMO(n) → da como resultado 1 si **n** es primo y 0 si no es
- PRIMO(n) → para obtener el n-ésimo número primo
- PRIMEXP(m,n)→ devuelve el exponente del m-ésimo número primo en la descomposición en factores primos del número **n**

```
EXPR ES_PRIMO (Y0)  MACRO:
        Y1:= 1 ; Y2:= 1 ;
        WHILE Y2 ≠ 0 DO
                Y1:= SUC(Y1) ;
                Y2:= Y0 MOD Y1
        END ;
        Y3:= Y0 -Y1 ;
        IF Y3 ≠ 0 THEN
                Y4:= 0
        ELSE
                Y4:= 1
        FI
RESULT (Y4)
```

CODIFICACION DE PROGRAMAS

Consideraremos computaciones con objetos distintos de los números naturales. Supongamos que tenemos una función

$g : A \rightarrow B$

La podremos computar si encontramos una función de codificación

$c : A \rightarrow N$

una función de decodificación

$d : N \rightarrow B$

y una función computable

$f : N \rightarrow N$

tal que

$g = d \circ f \circ c$

Los programas en ***P*** están restringidos a una sola entrada y una salida. Para manejar entradas o salidas múltiples codificaremos enuplas de naturales por naturales. Para lograr esto bastará con codificar pares.

$\langle x,y \rangle \in A \times B$ si $x \in A$, $y \in B$

fst (<a,b>) = a

snd (<a,b>) = b

Para codificar A x B definimos :

cod (<a,b>) = $2^{\{a\}}3^{\{b\}}$

(Usaremos a veces la notación <a,b> para referirnos a la codificación del par).

En lenguaje ***P*** tendremos las siguientes macros :

Para codificar :

```
EXPR PAR(Y1,Y2) MACRO
        Y3 := (2**Y1)*(3**Y2)
RESULT(Y3)
```

y para decodificar :

```
EXPR FST(Y1) MACRO
        Y2 := PRIMEXP(1,Y1)
RESULT(Y2)

EXPR SND(Y1) MACRO
        Y2 := PRIMEXP(2,Y1)
RESULT(Y2)
```

Construiremos programas en P cuyos datos de entrada y de salida son otros programas en P, eventualmente con sus datos.

Para eso necesitamos codificar y decodificar programas escritos en *P*. El conjunto de los programas *P* es un conjunto definido inductivamente; daremos funciones de codificación y decodificación para cada uno de los elemento sintácticos de *P*.

Las funciones de codificación tienen como dominio el conjunto del que se trate y como codominio los naturales. Las funciones de decodificación son funciones de naturales en el conjunto sintáctico del que se trate ; son , en general, funciones parciales.

Daremos a continuación los nombres de las funciones de codificación y decodificación y una expresión para las funciones de codificación.

VARIABLE :
cod-var (Xi) = i + 1
decod-var

CONDICION :
cod-cond (Xi /= 0) = i + 1
decod-cond

EXPRESION :
cod-expr(0) = 2
cod-expr(SUC(Xi)) = $3^{\{Xi\}}$
cod-expr(PRED(Xi)) = $5^{\{Xi\}}$
decod-expr

ASSIGN :
cod-ass (ass(x,e)) = $2^{\{x\}}\ 3^{\{e\}}$
decod-ass-var
decod-ass-expr

WHILE :
cod-wh(while(c,s)) = $5^{\{c\}}\ 7^{\{s\}}$
decod-wh-cond
decod-wh-sent

SECUENCIA :
cod-sec(sec(s1,s2)) = $11^{\{s1\}}\ 13^{\{s2\}}$
decod-sec-1
decod-sec-2

Dado que (s1;s2);s3 es, de acuerdo a la semántica, equivalente a s1;(s2;s3) se implementa cod-sec de modo que en ambos casos la codificación sea :

$$11^{\{s1\}}\ 13^{\{s2;s3\}}$$

donde s1 no es una secuencia

PROGRAMA :
cod-prog(prog(x,c,y)) = $2^{\{x\}}\ 3^{\{c\}}\ 5^{\{y\}}$
decod-prog-vare
decod-prog-sent
decod-prog-vars

Por ejemplo, al siguiente programa *R*

```
PROGRAM(X0)
        X0 := 0
RESULT(X0)
```

le corresponde el número

$$2.3^{2.9}.5 = 10.3^{18}$$

Las funciones de codificación y decodificación se implementan mediante macros en ***P*** (de tipo expresión).

En algún caso nos interesará chequear la pertenencia de un elemento de nuestro lenguaje a determinada categoría sintáctica. Esto se traduce en macros de condición :
es-0, es-pred, es-suc
es-ass, es-while, es-sec

Llamaremos índice de un programa al número que le corresponde en la codificación.

Dado el índice **q**

Ix(q) - programa de índice q Ix : N $\rightarrow$ Prog

ϕq - función computada por el programa Ix(q)

$\phi q(m) = n$ sii <Ix(q),m> $\Rightarrow$ n

Dado que no todo natural es el índice de algún programa, para que Ix sea una función total definimos :

Ix(q) = (PROGRAM(X0) X0 := 0 RESULT(X0)) si q no es el índice de ningún programa.

UN INTERPRETE PARA *P* EN *P*

Construiremos en *P* un programa IP que , recibiendo como entrada un programa en *P* Q (su codificación) y un natural m, realiza la ejecución de Q con entrada m, devolviendo el resultado correspondiente.

IP utiliza las reglas de evaluación de las instrucciones de *P*, o sea, la semántica operacional que hemos definido para el lenguaje *P*.

Definiremos previamente un método para codificación del estado de un programa y las correspondientes funciones de acceso y actualización

El estado es una función de variables en naturales

$\sigma : \{Var\} \rightarrow N$

cod-estado$(\sigma) = 2^{\sigma(X0)}.3^{\sigma(X1)}.5^{\sigma(X2)}. ...$

esto implica cod-estado$(\Omega) = 1$

valor-de (x,σ) se implementa mediante PRIMEXP y

actualizar(x,n,σ) mediante PRIMEXP y operaciones aritméticas.

Notar que si bien un estado es un conjunto infinito (cantidad infinita de variables), logramos codificar un estado mediante un natural, aprovechando el hecho de que sólo una cantidad finita de variables puede tener valor distinto de 0.

Las siguientes macros , además de las ya mencionadas,serán utilizadas :

eval-ass(sent-ass,estado) - modifica el estado
eval-cond(cond,estado) - evalúa en un booleano
es-sec(sent) - chequea si una sentencia es una secuencia
es-ass(sent) - chequea si una sentencia es una asignación
cod-sec(s1,s2) - codifica una secuencia a partir de las sentencias s1 y s2

La entrada a IP es la codificación de un par <q,n> ; se cumple:

$< IP,\{<q,n>\} > \Rightarrow m$ si $< Ix(q),n > \Rightarrow m$ y

$< IP,\{<q,n>\} > \uparrow$ si $< Ix(q),n > \uparrow$

IP :

```
PROGRAM(X0)
     X1 := FST(X0);        X1 - programa Q   prog(x,c,y)
     X2 := SND(X0);        X2 - entrada n a Q
     X3 := DECOD-PROG-VARE(X1);
     X4 := DECOD-PROG-SENT(X1);
     X5 := DECOD-PROG-VARS(X1);
     X6 := OMEGA;                        X6 - estado
     X6 := ACTUALIZAR(X3,X2,X6);
     WHILE X4 ≠ 0 DO
          IF ES-SEC(X4)
          THEN X7 := DECOD-SEC-1(X4);
               X4 := DECOD-SEC-2(X4)
          ELSE X7 := X4;
               X4 := 0
          FI;
          IF ES-ASS(X7)
          THEN X6 := EVAL-ASS(X7,X6)
          ELSE X8 := DECOD-WH-COND(X7);
               X8 := EVAL-COND(X8,X6);
               IF X8 = 1
               THEN X9 := DECOD-WH-SENT(X7);
                    IF X4 ≠ 0
                    THEN X4 := COD-SEC(X7,X4)
                    ELSE X4 := X7
                    FI;
                    X4 := COD-SEC(X9,X4)
               FI
          FI
     END;
     X10 := VALOR-DE(X5,X6)
RESULT(X10)
```

Asociada al programa IP podemos construir la macro MIP, para manejar IP como una expresión. Utilizaremos a menudo la macro eval-prog(p,n) que codifica el par <p,n> y evalúa Ix(p) con entrada n mediante la macro MIP.

```
EXPR EVAL-PROG (Y0,Y1) MACRO
     Y2 := COD-PAR(Y0,Y1);
     Y3 := MIP(Y2)
RESULT(Y3)
```

En algunas situaciones necesitaremos utilizar la expresión :

eval-prog-step(p,n,m) = <1,w> si <Ix(p),n> $\Rightarrow$ w en a lo sumo **m** pasos
= <0,0> en otro caso

Para poder construir una macro en *P* debemos modificar el intérprete de modo que evalúe el programa que se interpreta una cantidad dada de pasos.

Intérprete modificado - IPS

```
PROGRAM(X0)                X0 - <p,<n,m>>
     X1 := FST(X0);        X1 - programa Q   prog(x,c,y)
     X2 := SND(X0);        X2 - <n,m>
     X11 := SND(X2);       X11 - m
     X2 := FST(X2);        X2 - n
     X3 := DECOD-PROG-VARE(X1);
     X4 := DECOD-PROG-SENT(X1);
     X5 := DECOD-PROG-VARS(X1);
     X6 := OMEGA;                          X6 - estado
     X6 := ACTUALIZAR(X3,X2,X6);
     WHILE X4 ≠ 0  & X11 ≠ 0  DO
          IF ES-SEC(X4)
          THEN X7 := DECOD-SEC-1(X4);
               X4 := DECOD-SEC-2(X4)
          ELSE X7 := X4;
               X4 := 0
          FI;
          IF ES-ASS(X7)
          THEN X6 := EVAL-ASS(X7,X6)
          ELSE X8 := DECOD-WH-COND(X7);
               X8 := EVAL-COND(X8,X6);
               IF X8 = 1
               THEN X9 := DECOD-WH-SENT(X7);
                    IF X4 ≠ 0
                    THEN X4 := COD-SEC(X7,X4)
                    ELSE X4 := X7
                    FI;
                    X4 := COD-SEC(X9,X4)
               FI
          FI;
     X11 := PRED(X11)
     END;
     IF X4 ≠ 0
     THEN X10 := COD-PAR(0,0)
     ELSE X10 := VALOR-DE(X5,X6);
          X10 := COD-PAR(1,X10)
     FI
RESULT(X10)
```

Mediante la macro MIPS asociada a IPS podemos construir la macro eval-prog-step en modo análogo a eval-prog

Ejercicio 1.-

Escribir las siguientes macros en P :

- IF <cond> THEN <sent> ELSE <sent> FI
- $X \; 5 \; Y = X - Y \begin{cases} X - Y & \text{si } X \geq Y \\ 0 & \text{en caso contrario} \end{cases}$
- X = Y
- X < Y
- X MOD Y
- PRIMO(n) $\rightarrow$ el resultado es el enésimo número primo
- PRIMEXP(m,n) $\rightarrow$ el resultado es el exponente del enésimo número primo en la descomposición en factores primos del número **m**
- ES_PRIMO(n) $\rightarrow$ el resultado es **verdadero** si **n** es primo, **falso** en caso contrario
- DECOD-ASSIGN-VAR(X)
- DECOD-ASSIGN-EXPR(X)
- VALOR-DE (X,σ) - Valor de la variable X en el estado σ
- ACTUALIZAR (X,Y,σ) - Nuevo estado σ [X $\leftarrow$ Y]
- EVAL-ASSIGN(X,σ) - Estado resultante de evaluar la asignación X en σ
- COD-SEC (S1,S2) - Dados los códigos de sentencias S1 y S2 calcula el código de la secuencia. Recordar que S1 puede ser el código de una secuencia.
- LOOP - Ciclo infinito
- SKIP - Ningún efecto

Ejercicio 2.-
Sean los programas :

P ≡ PROGRAM (Xi)
C
RESULT (Xj)

Q ≡ PROGRAM(Xi)
C
Xmax+1 := 0
RESULT(Xj)

Construir una macro en P que dado el código de un programa (P) **p** devuelva el de un par **<p',n>** donde p' es el código de Q y n el número de la sentencia "Xmax+1 := 0".