LENGUAJE *P*

Definimos un lenguaje simple pero con poder expresivo suficiente para ser un modelo de computabilidad.

P es un lenguaje imperativo con cantidad mínima de construcciones. Lo definiremos dando :

1) La sintaxis de ***P***

2) La semántica de ***P***

3) La pragmática de ***P***

SINTAXIS DE *P*

Para presentar la sintaxis de ***P*** utilizaremos BNF (Backus-Naur form)

<prog> ::= PROGRAM (<var>) <sent> RESULT (<var>)

<var> ::= X<número>

<número> ::= <dígito> | <dígito><número>

<dígito> ::= 0 | 1 | ... | 9

<sent> ::= <assign> | <while> | <sec>

<assign> ::= <var> := <expr>

<expr> ::= 0 | SUC(<var>) | PRED(<var>)

<while> ::= WHILE <cond> DO <sent> END

<cond> ::= <var> ≠ 0

<sec> ::= <sent> ; <sent>

Definiremos la siguiente *"sintaxis abstracta" :*

- **Var** $\mathbf{E_1}$: $\dfrac{n \in \mathbf{n}}{var(n) \in Var}$

- **Cond** $\mathbf{E_1}$: $\dfrac{x \in Var}{x \neq 0 \in Cond}$

- **Expr** $\mathbf{E_1}$: $0 \in Expr$

$\mathbf{E_2}$: $\dfrac{x \in Var}{suc(x) \in Expr}$

$\mathbf{E_3}$: $\dfrac{x \in Var}{pred(x) \in Expr}$

- **Sent** $\mathbf{E_1}$: $\dfrac{x \in Var \quad e \in Expr}{assign(x,e) \in Expr}$

$\mathbf{E_2}$: $\dfrac{t \in Cond \quad c \in Sent}{while(t,c) \in Sent}$

$\mathbf{E_3}$: $\dfrac{c_1 \in Sent \quad c_2 \in Sent}{sec(c_1,c_2) \in Sent}$

- **Prog** $\mathbf{E_1}$: $\dfrac{x,y \in Var \quad c \in Sent}{prog(x,c,y) \in Prog}$

Ahora es fácil definir funciones sobre este nuevo conjunto, a los efectos de resolver problemas de caracter sintáctico.

Ejemplo :

Especificar una función que cuente cuantas veces aparece la variable **n** en un programa.

cant-veces : **N** x Prog $\rightarrow$ **N**

cant-veces (n,prog(x,c,y)) = $f_{Var}(n,x) + f_{Sent}(n,c) + f_{Var}(n,y)$

$f_{Var}(n,Var(m)) = \begin{cases} 1 & \text{si } n = m \\ 0 & \text{en otro caso} \end{cases}$

$f_{Sent}(n,while(t,c)) = f_{Cond}(n,t) + f_{Sent}(n,c)$

$f_{Sent}(n,assign(x,e)) = f_{Var}(n,x) + f_{Expr}(n,e)$

$f_{Sent}(n,sec(c,d)) = f_{Sent}(n,c) + f_{Sent}(n,d)$

$f_{Cond}(n,Cond(x)) = f_{Var}(n,x)$

$f_{Expr}(n,0) = 0$

$f_{Expr}(n,suc(x)) = f_{Var}(n,x)$

$f_{Expr}(n,pred(x)) = f_{Var}(n,x)$

Esta función es, entonces, claramente computable.

Similarmente, se pueden especificar funciones como :

max-var-num : Prog $\rightarrow$ N

computa el mayor número de variable utilizada en un programa

cant-sent : Prog $\rightarrow$ N

computa la cantidad de sentencias de un programa

En lo que sigue del curso, utilizaremos tanto la expresión concreta en P como la expresión en sintaxis abstracta en los razonamientos sobre programas en P.

SEMANTICA DE *P*

Presentaremos una semántica operacional para ***P***. Se dará un conjunto de reglas precisas para determinar como es evaluada cada categoría sintáctica.

Las reglas van a ser de la forma :

1) $F \Rightarrow F'$

que se lee **F evalúa en F´**

y

2) $$\frac{G_1 \Rightarrow G_1' \quad ... \quad G_n \Rightarrow G_n'}{F \Rightarrow F'}$$

que se lee **F evalúa en F´**

si G_1 evalúa en G_1'
.
.
.
y G_n evalúa en G_n'

Como ***P*** es un lenguaje imperativo, será necesario definir la memoria de trabajo del programa. Dicha memoria contiene el valor de cada variable en un momento dado, o sea, el "estado".

Caracterizamos el estado como una *función parcial* entre las variables y los valores. Se tiene entonces:

$$\sigma \in \text{Estados} : \text{Var} \rightarrow \boldsymbol{N}$$

donde

$$\sigma = \{ <var_1 , valor_1 >,...\}$$

Sobre el conjunto de estados, se van a necesitar dos operaciones :

- valor_de : Var x Estados $\rightarrow \boldsymbol{N}$

- actualizar : Var x $\boldsymbol{N}$ x Estados $\rightarrow$ Estados

Para simplificar, se introducirá el estado Ω , para el cual toda variable tiene el valor 0.

Definiciones:

- valor_de(x,σ) = σ(x) el valor que toma x en el estado σ
- actualizar(x,n,σ) = σ' donde
  $\sigma'(y) = n$ si $x = y$
  $\sigma(y)$ en caso contrario
- $\Omega(x) = 0 \ \forall \ x \in$ Var

Abreviaciones:

- $\sigma[x \rightarrow n] \equiv$ actualizar(x,n,σ)
- $\sigma[x \rightarrow n, y \rightarrow m] \equiv (\sigma[x \rightarrow n])[y \rightarrow m]$

Notar que $\sigma[x \rightarrow n, x \rightarrow m] = \sigma[x \rightarrow m]$

Para describir el valor de una expresión, necesitamos conocer en cual estado estamos. Describimos el valor por medio de una combinación de la expresión y el estado

$$< e, \sigma >$$

De la misma forma, se describe el valor de una sentencia, una condición y un programa por :

$$< c, \sigma > , < u, \sigma > \ y \ < p,n >$$

donde, en el último caso, n es la entrada.

Estamos ahora en condiciones de definir las reglas para computar.

Comenzaremos por los programas.

$$\frac{\langle c,\Omega[x \to n]\rangle \Rightarrow \sigma'}{\langle prog(x,c,y),\, n\rangle \Rightarrow \sigma'(y)}\ [P]$$

donde x, y $\in$ Var , c $\in$ Sent , n $\in \boldsymbol{N}$ y $\Omega, \sigma' \in$ Estados

Para las sentencias, tenemos tres diferentes construcciones y por lo tanto, tres casos:

- Asignación : $$\frac{\langle e,\sigma\rangle \Rightarrow n}{\langle assign(x,e),\sigma\rangle \Rightarrow \sigma[x \to n]}\ [A]$$

donde los valores para las expresiones están definidos por las siguientes cuatro reglas :

- [E_1] $\langle 0,\sigma\rangle \Rightarrow 0$
- [E_2] $\langle suc(x),\sigma\rangle \Rightarrow suc(\sigma(x))$
- [E_3] $\langle pred(x),\sigma\rangle \Rightarrow 0$ si $\sigma(x) = 0$
- [E_4] $\langle pred(x),\sigma\rangle \Rightarrow n$ si $\sigma(x) = suc(n)$

- Secuencia : $$\frac{\langle c,\sigma\rangle \Rightarrow \sigma' \quad \langle d,\sigma'\rangle \Rightarrow \sigma''}{\langle sec(c,d),\sigma\rangle \Rightarrow \sigma''}\ [S]$$

- While : $$\frac{\langle t,\sigma\rangle \Rightarrow Falso}{\langle while(t,c),\sigma\rangle \Rightarrow \sigma}\ [w_1]$$

$$\frac{\langle t,\sigma\rangle \Rightarrow Verd \quad \langle c,\sigma\rangle \Rightarrow \sigma' \quad \langle while(t,c),\sigma'\rangle \Rightarrow \sigma''}{\langle while(t,c),\sigma\rangle \Rightarrow \sigma''}\ [w_2]$$

- Condición : • [V_1] $\langle cond(x),\sigma\rangle \Rightarrow$ Falso si $\sigma(x) = 0$

  • [V_2] $\langle cond(x),\sigma\rangle \Rightarrow$ Verd si $\sigma(x) =;/ 0$

Definición :

Decimos que hemos realizado una prueba para la semántica de un programa ***P*** cuando construimos un árbol de derivación, con la expresión para el valor del programa como consecuencia final (en "la base") y sin premisas por demostrar.

Ejemplo :

$$\frac{\dfrac{\dfrac{}{\langle SUC(X),\Omega[X\to 1]\rangle \Rightarrow 2}[E2]}{\langle ASSIGN(X,SUC(X)),\Omega[X\to 1]\rangle \Rightarrow \Omega[X\to 2]}[A]}{\langle PROG(X,ASSIGN(X,SUC(X)),X),1\rangle \Rightarrow 2}[P]$$

Tenemos entonces un método para construir pruebas respecto al resultado (salida) de un programa dada la entrada.

Este método nos permitirá demostrar que un programa computa efectivamente una función dada.

Seremos más precisos en la noción de ***computable***.

Definición : Se dice que un ***P***-Programa **Q** ***computa*** la función parcial

$$\mathbf{P[Q]} : \boldsymbol{N} \to \boldsymbol{N}$$

donde

$\mathbf{P[Q]}(m) = n$ si $\exists\, n \in \boldsymbol{N}$ / $\langle \mathbf{Q}, m\rangle \Rightarrow n$

$\mathbf{P[Q]}(m)$ es indeterminado en caso contrario

Definición : Una función parcial $f : \mathbf{N} \to \boldsymbol{N}$ es ***P-computable***, sii existe un ***P***-programa que la computa.

Aplicaremos la tesis de Türing-Church, y diremos que si algo es computable, entonces es ***P***-computable.

Definición : Una computación de **Q** con una entrada **n** ***converge***, si

$(\exists\, m \in \boldsymbol{N})$ $\langle \mathbf{Q,n} \rangle \Rightarrow m$

lo cual es denotado :

$\langle \overline{\mathbf{Q,n}} \rangle \downarrow$ o simplemente $\langle \mathbf{Q,n} \rangle \downarrow$

En caso contrario, se dice que ***diverge*** , y se anota de la siguiente manera :

$\langle \overline{\mathbf{Q,n}} \rangle \uparrow$ o simplemente $\langle \mathbf{Q,n} \rangle \uparrow$

Ejemplo :

Presentaremos un ejemplo más complejo, a los efectos de mostrar como se trabaja con esta técnica.

¿Qué función computa el siguiente programa ***P***?

```
PROGRAM(X0)
        X1 := 0 ;
        WHILE X0 =;/ 0 DO
            X1 := SUC(X1) ;
           X0 := PRED(X0)
        END
RESULT(X1)
```

Como sólo X0 y X1 son usadas, se introduce la notación

$$\Phi(x,y) \equiv \Omega[X0 \rightarrow x,\ X1 \rightarrow y]$$

Asimismo, llamamos:
c al cuerpo del programa
c' al while
c'' a la sentencia interior al while

y para simplificar las cosas, supondremos que ya hemos demostrado :

Lema 1 :

$$(\forall\, x,y \in \boldsymbol{N}) \quad \langle c'', \Phi(s(x),y) \rangle \Rightarrow \Phi(x,s(y))$$

(se prueba derivando la semántica de c'',
partiendo de un estado $\Omega[X0 \rightarrow x,\ X1 \rightarrow y]$)

Probaremos ahora :

Lema 2 :

$(\forall\ x,y) \in \boldsymbol{N}\ \ < c',\Phi(x,y)> \ \Rightarrow\ \Phi\ (0,\ x+y)$

Probaremos este lema por inducción en x.

Paso Base: x = 0

Entonces por V1 y W1 se cumple que :

$$\frac{}{<X0\neq 0,\Phi(0,y)> \ \Rightarrow\ \text{Falso}}\ [V1]$$

$$\frac{}{<c',\Phi(0,y)> \ \Rightarrow\ \Phi\ (0,y)}\ [W1]$$

$\Phi(0,y) = \Phi(0,x+y)$, por lo tanto el lema es válido para x=0.

Paso Inductivo: Sea x = s(n).

Hipótesis inductiva: $(\forall y\ \in \boldsymbol{N})\ <c',\Phi(n,y)> \Rightarrow \Phi\ (0,n+y)$

Por V2 se tiene (A) :

$$\frac{}{<X0\neq 0,\Phi(s(n),y)> \ \Rightarrow\ \text{Verd.}}\ [V2]$$

Por el Lema 1 (B) :

$$\frac{}{<c'',\Phi(s(n),y)> \ \Rightarrow\ \Phi(n,s(y))}\ [\text{Lema1}]$$

Por lo tanto;

$$\frac{\dfrac{}{<c',\Phi(n,s(y)) \ \Rightarrow \Phi(0,n+s(y))}\ [\text{HI}]\quad (A)\ (B)}{<c',\Phi(s(n),y)> \ \Rightarrow\ \Phi(0,n+s(y))}\ [W2]$$

Pero $\Phi(0,n+s(y)) = \Phi(0,s(n)+y)$, entonces el lema se cumple para todo n.

El resto es simple :

$$\frac{\dfrac{}{\langle 0,\Omega[X0\to x]\rangle \Rightarrow 0}[E1] \qquad \langle c',\Phi(x,0)\rangle \Rightarrow \Phi(0,x)}{\langle X1:=0;\, c',\Omega[X0\to x]\rangle \Rightarrow \Phi(0,x)}[S]$$

$$\frac{\langle X1=0,\Omega[X0\to x]\rangle \Rightarrow \Phi(x,0) \qquad \langle X1:=0;\, c',\Omega[X0\to x]\rangle \Rightarrow \Phi(0,x)}{\langle c,x\rangle \Rightarrow \Phi(0,x)(X1) = x}[P]$$

Por lo que se concluye que c es la función identidad.

Ejercicio 1.-

Sea NxN = {(0,0) , (0,1) , (1,0) , (0,2) , (1,1) , (2,0) , ... } el conjunto de pares de naturales,

y la función $J : N^2 \to N$ definida por : $J(m,n) = 1/2(m+n)(m+n+1) + m$

a) Es J inyectiva ? Sobreyectiva ? Demostrar

b) Construir 2 funciones $K:N\to N$ y $L:N \to N$, tales que $J^{-1}(n) = (K(n),L(n))$

Ejercicio 2.-

Definir las siguientes funciones sobre programas P :

mav-var-num : $Prog \to N$
computa el máximo número de variable utilizada en un programa

cant-sent : $Prog \to N$
computa la cantidad de sentencias de un programa

Ejercicio 3.-

Indicar qué función computa el siguiente programa. Demostrar.

PROGRAM (X0)

$$
c\left\{\begin{array}{l}
X1 := SUC(X0); \\
X1 := PRED(X1); \\
X2 := 0; \\
c'\left\{\begin{array}{l}
\text{WHILE } X1 \neq 0 \text{ DO} \\
\quad c''\left\{\begin{array}{l} X2 := X0 + X2; \\ X1 := PRED(X1); \end{array}\right. \\
\textit{END}
\end{array}\right.
\end{array}\right\}
$$

RESULT (X2)

Se sugiere utilizar la siguiente abreviación :
$\Phi\ (x,y,z) = \Omega\ [X0 \rightarrow x,\ X1 \rightarrow y,\ X2 \rightarrow z]$

Suponer conocido el siguiente resultado :
$\forall\ x,y,z \quad <c'', \Phi\ (x,s(y),z)> \Rightarrow \Phi\ (x,y,z+x)$

Ejercicio 4.-

Escribir programas en P que computen las siguientes funciones:

i. $f(x) = 2x$

ii. $f(x) = x - 4$ si $x > 4$,
$f(x) = 0$ en caso contrario.