Anales PANEL'81/12 JAIIO
Sociedad Argentina de informática
e Investigación Operativa. Buenos Aires, 1981

SOBRE LAS CONDICIONES "ON" DE PL/I Y SU PARAMETRIZACION

Guido Vassallo
Departamento de Computación
Facultad de Ingeniería, Universidad de Buenos Aires

RESUMEN

Se define un fragmento de PL/I, que incluye las construcciones más comunes (asignaciones, bloques, entrada/salida, etc.) y los comandos ON, REVERT, y SIGNAL. Las condiciones consideradas incluyen (algunas de) las predeclaradas y las declaradas por el programador. Se toman en cuenta tanto las acciones establecidas por programa como las del sistema. Se consideran los mecanismos de habilitación e inhabilitación.

Se extiende el minilenguaje con el agregado de parámetros y argumentos para las condiciones, discutiéndose varios modos de pasaje (por valor, por referencia y por valor y resultado), y su especificación tanto en la frase ON como en la SIGNAL.

Finalmente, se sugieren otras posibilidades.

Se utiliza para la definición del significado de las distintas construcciones una versión de la Semántica Matemática de Strachey y Scott.

1. INTRODUCCION

1.1. ANTECEDENTES

Entre las tantas construcciones contenidas en PL/I se destaca el manejo de condiciones.

En general, estas condiciones se asocian a interrupciones originadas durante el proceso, tales como "overflow", fin de archivo, etc. Es posible programar las acciones a efectuar ante cada interrupción, lo que hace más flexible la programación.

Sin embargo, los aspectos lingüísticos esenciales del manejo de condiciones pueden disociarse de los conceptos relativos a interrupciones, utilizando identificadores con el atributo CONDITION - cuyo uso generalizado juega un papel importante en programación concurrente [SCP]. El autor ha encontrado didácticamente oportuno este enfoque, en los cursos regulares dictados en la Facultad de Ingeniería de la Universidad de Buenos Aires.

La descripción precisa de estos mecanismos no resulta fácil, dada la complejidad del lenguaje - y de sus manuales [PL/I]. Consultas casuales hechas a programadores expertos en PL/I han mostrado que la comprensión de ciertos aspectos no es simple, por esa vía.

Una descripción relativamente informal, pero clara y concisa, de los mecanismos mencionados se encuentra en [Wegner] . Ahí figura también el planteo de la posibilidad de extender significativamente la potencia de esas construcciones, mediante su parametrización.

Por otra parte, el Laboratorio de IBM en Viena, abocado a la definición formal de PL/I, inspirándose en ideas de [McCarthy] desarrolló una metodología especial: Vienna Definition Language (VDL) [Lucas y otros] . Este enfoque corresponde a una semántica operacional, en la cual las construcciones lingüísticas son definidas mediante su interpretación en un procesador abstracto.

Simultáneamente, un grupo de investigadores de Oxford, encabezados por Strachey y Scott, han ido desarrollando la semántica matemática [Strachey 66, 73, Milne y Strachey] . La semántica matemática puede ubicarse entre la operacional - de la cual difiere por el menor énfasis dado a cuestiones de tipo implementación - y la semántica axiomática [Hoare 69, 71] , menos constructiva.

La semántica matemática ha resultado particularmente adecuada, y de hecho ha "convertido" al grupo de Viena, que ha desarrollado su propia variante [Bjørner y Jones] . Entre otros resultados, produjo una interpretación del manejo de condiciones parametrizadas, como en la propuesta de [Wegner] .

1.2. PLAN DE TRABAJO

Con el mismo método de un trabajo anterior [Vassallo] , se intenta aquí definir el significado de una familia de mini-sublenguajes de PL/I (con algunas licencias poéticas).

El minilenguaje básico (PL.1), que se describe en 3, incluye, además de las declaraciones de variables y los comandos más comunes, la declaración de identificadores de condición, el establecimiento de condiciones (ON), los comandos SIGNAL y REVERT, las acciones "standard" del sistema, y la habilitación e inhabilitación de condiciones.

Ya saliendo de PL/I, se amplía PL.1 introduciendo varias formas de parametrización para las frases ON y SIGNAL (PL.21 a PL.26).

Se utiliza una variante relativamente primitiva de la semántica matemática, puesto que la omisión de rótulos y transferencias de control en los minilenguajes permite obviar el recurso al concepto de continuación [Strachey y Wadsworth, Milne y Strachey] y expresar el significado de cada comando simplemente como una transición de estados.

En la notación se ha sacrificado algo de concisión a favor de una mejor legibilidad.

2. NOTACIONES BASICAS

No distinguiremos entre tuplas, listas y sucesiones finitas.

Si $x.1,\ldots x.n \in A$, entonces $\langle x.1,\ldots x.n\rangle \in A^n$. Además, $x \neq \langle x\rangle$, es decir $A \neq A^1$.

Por definición la 0-pla o lista nula, nil $= \langle\rangle$, es el único elemento de A°, para cualquier A.

Se usarán $A^+ = A^1 \cup A^2_+ \cup \ldots$
y $A^* = A^\circ \cup A^+$

Se representará por $A \rightarrow B$ el conjunto de todas las funciones parciales con dominio incluído en A y codominio incluído en B. En rigor, en semántica matemática sólo se consideran funciones "buenas" en un cierto sentido.

Al aplicar funciones, a menudo se omitirán paréntesis redundantes; $\ulcorner fx \urcorner$ equivale a $\ulcorner f(x) \urcorner$, y $\ulcorner fhx \urcorner$ a $\ulcorner (f(h))(x) \urcorner$.

"Lo" indefinido se representará por "?". Es, por ejemplo, el "valor" obtenido aplicando una función parcial a un elemento fuera de su dominio. La función totalmente indefinida (vacía) se representará por "$\emptyset$". Las funciones que se usarán serán casi siempre estrictas: $f(?)=?$.

Se usará un operador dom que aplicado a una función produce una permutación de los elementos de su dominio. Ergo, para cualquier A, B, dom $\in (A \rightarrow B) \rightarrow A^*$.

Si $f \in A \rightarrow B$, puede generalizarse a $A^* \rightarrow B^*$ definiendo
$f(\langle x.1,\ldots x.n\rangle) = \langle f(x.1),\ldots f(x.n)\rangle$
En particular, f(nil) = nil.

Un operador que se utilizará continuamente, para extender y/o modificar funciones es [Reynolds] :

$\underline{ext} \in (A \rightarrow B) \times A^n \times B^n \rightarrow (A \rightarrow B)$ (para cualquier A,B,n).
En general $\underline{ext}$ $(f, \langle x.1,\ldots x.n\rangle , \langle y.1,\ldots y.n\rangle) = g$, donde

$$g(t) = \begin{cases} y.i \text{ si } t=x.i \text{ para algún } i,\ 1 \leq i \leq n \\ f(t) \text{ si no} \end{cases}$$

Todas las x.i deben ser distintas. Se escribirá $\underline{ext}$ (f,x,y) por $\underline{ext}$ (f, $\langle x\rangle$, $\langle y\rangle$).

Se usará la construcción metalingüística

$\underline{if}$ p $\underline{then}$ u $\underline{else}$ v $\underline{fi}$

tal que: si p es verdadero, vale lo mismo que u; si p es falso, vale lo mismo que v; si p es indefinido, resulta indefinida (aunque esto último podría discutirse, si u y v valen lo mismo). Se entiende que, por ejemplo, $\underline{if}$ 0=0 $\underline{then}$ 0-0 $\underline{else}$ 0/0 $\underline{fi}$ está definido (y vale 0) aunque 0/0 no lo está.

La conjunción lógica se define por p & q = $\underline{if}$ p $\underline{then}$ q $\underline{else}$ $\underline{fa}$ $\underline{fi}$; ergo ? & $\underline{fa}$ = ? , $\underline{fa}$ & ? = $\underline{fa}$, siendo $\underline{ve}$ = verdadero, $\underline{fa}$ = falso.

Se aplicarán a listas las funciones:

cabeza	= $\underline{hd} \in A_+^+ \rightarrow A$	(primer elemento de lista no nula)
cola	= $\underline{tl} \in A^+ \rightarrow A^*$	(sublista obtenida al decapitar otra)
constructor	= $\underline{cons} \in AxA^* \rightarrow A^+$	(lista obtenida yuxtaponiendo un elemento de A y otra lista).

En general, se verifica:

$\underline{hd}$ ($\underline{cons}$ (x,y)) = x
$\underline{tl}$ ($\underline{cons}$ (x,y)) = y
$z \neq \underline{nil} \Rightarrow z = \underline{cons}(\underline{hd}\ z,\ \underline{tl}\ z)$

Nótese que $\underline{cons}$ no es estricta: $\underline{cons}$ (?, $\langle \ldots\rangle$) = $\langle ?,\ldots\rangle \neq ?$.

Otra función es la longitud de una lista, $\underline{lg} \in A \rightarrow \mathbb{N}$, que puede definirse por

$\underline{lg}$ z = $\underline{if}$ z=$\underline{nil}$ $\underline{then}$ 0 $\underline{else}$ 1+$\underline{lg}$($\underline{tl}$ z) $\underline{fi}$.

En algunos casos, se aplicarán índices, o selectores de componentes de tuplas, escritos como sufijos, Así, $\ulcorner x.k \urcorner$ significa x_k, y $\ulcorner z.\underline{hd} \urcorner$ equivale a $\ulcorner \underline{hd}\ z \urcorner$.

3. MINILENGUAJE BASICO (PL.1)

Definiremos un minilenguaje, al cual para futuras referencias denominaremos PL.1.

Este sublenguaje de PL/I incluye:

- algunas de las condiciones, a saber ENDFILE (implícitamente aplicada a SYSIN), CHECK (siempre explícitamente aplicado a una variable), ZERODIVIDE y CONDITION;

- acciones standard del sistema y opción SYSTEM;

- habilitación e inhabilitación de condiciones;

- comando ON, REVERT y SIGNAL; y

- las construcciones ordinarias de cualquier lenguaje imperativo.

3.1. SINTAXIS

Existen las siguientes categorías sintácticas:

Prog	- programas
Cmd	- comandos
Expr	- expresiones
Idf	- identificadores
Cond	- condiciones

Los elementos genéricos de estas categorías se representarán respectivamente por prog, cmd, expr, idf, cond, posiblemente adornados con índices, ápices, etc.

Las producciones de la gramática se expresarán en un BNF informal. Las eventuales ambigüedades e imprecisiones se consideran irrelevantes.

```
prog :: = idf: PROC OPTIONS (MAIN);
          DCL (idf.1,...idf,p)COND,(idf.p+1,...idf.q)VAR;
          cmd.1 ... cmd.n
          END;

cmd  :: = idf = expr ;
        | GET (idf) ;
        | PUT (expr) ;
        | IF expr THEN cmd
        | IF expr THEN cmd.1 ELSE cmd.2
        | DO ; cmd.1 ... cmd.n END ;
        | DO WHILE expr ; cmd.1 ... cmd.n END ;
        | BEGIN ; DCL (idf.1,...idf.h) VAR; cmd.1 ... cmd.n END ;
        | ON cond cmd
        | ON cond SYSTEM;
        | REVERT cond ;
        | SIGNAL cond ;
        | (cond): cmd
        | (NOcond): cmd

cond :: = COND (idf)
        | ENDFILE
        | CHECK (idf)
        | ZDIV
```

```
expr :: = idf
        | expr.1 / expr.2
        | ‹etc›
```

idf a piacere.

Se ha colocado VAR donde correspondería algún otro atributo (p.ej. FLOAT). En ⌜ON cond cmd⌝, no se prohibe, como en PL/I, que cmd sea un grupo DO u otro establecimiento de condición. Tampoco se ha restringido el uso de prefijos a CHECK y ZDIV. Estas restricciones podrían definirse en el nivel sintáctico, distinguiendo comandos simples, bloques y comandos compuestos (IF, DO, ON). Ello complicaría el azúcar sintáctico sin añadir especias semánticas.

Algunas de las restricciones del lenguaje, respecto de PL/I, tales las de no poder omitir o disponer en otro orden las declaraciones, son puramente sintácticas y no disminuyen su potencia. Un simple preprocesamiento permitiría reducir cualquier programa que haga uso de esas licencias a la forma "canónica" aquí definida. En particular, el agrupamiento de las declaraciones de identificadores de CONDición al principio del programa se justifica por ser EXTERNALs.

Una mayor pureza (y menor redundancia) se obtendría usando, en lugar de la sintaxis concreta enunciada, una sintaxis abstracta como las usadas en [McCarthy 66] y [Lucas y otros]. Sin embargo, la presentación resultaría más pesada y no más precisa.

3.2. DOMINIOS Y FUNCIONES SEMANTICAS

3.2.1. VALORES Y DENOTACIONES

Los dominios fundamentales (no ulteriormente analizados) son:

Val: valores "atómicos": lógicos, aritméticos, etc.

Den: denotaciones (que podrán implementarse como ubicaciones de memoria).

Se definirán en base a éstos los demás dominios semánticos.

3.2.2. AMBIENTES

Se define un conjunto de cuaternas denominadas ambientes ("environments"):

$Amb = (Idf \rightarrow Den) \times (Cond \rightarrow Den) \times (Cond \rightarrow Den) \times (Cond \rightarrow \mathbb{B})$

donde $\mathbb{B} = \{ve, fa\}$ es el conjunto de los valores lógicos.

Un elemento genérico de Amb, se escribirá amb o bien ⟨amb. var, amb. act, amb. lat, amb. hab⟩.

Se dirá que si x es un identificador de VARiable, entonces amb.var(x) es su denotación. Si x es un identificador de CONDición, entonces amb. act(x) y amb.lat(x) son respectivamente sus denotaciones actual y latente, y amb.hab(x) su indicador de habilitación.

Según se verá en las ecuaciones semánticas (3.3), el de ambiente es un concepto estático: en cada punto del texto de un programa es válido un ambiente bien determinado.

3.2.3. ESTADOS

Se define un conjunto de cuaternas denominadas estados:

$$Est = Val \times (Den \rightarrow (Val \cup Cla)) \times Val^* \times Val^*$$

Un elemento genérico de Est, se escribirá est o bien
$\langle$ est.val,est.mem,est.ent,est.sal $\rangle$

Según se verá en las ecuaciones semánticas, el de estado es un concepto dinámico: en cada instante del proceso de ejecución de un programa existe un estado bien determinado.

La interpretación intuitiva de (los componentes de) un estado es la siguiente: est.val es el último valor calculado; est.mem es la función memoria, tal que est.mem(x) es el valor o clausura (v. 3.2.4) "contenido" en x; est.ent y est.sal son respectivamente las corrientes de datos de entrada y salida, concebidas como sucesiones de valores.

3.2.4. CLAUSURAS Y ACCIONES DEL SISTEMA

Se define un conjunto de funciones denominadas clausuras:

$$Cla = Est \rightarrow Est$$

Las clausuras corresponden a transiciones de estado "almacenables".

Comparando las definiciones de Est y Cla se advierte una paradoja, puesto que la cardinalidad de cada uno debería superar a la del otro. Este problema está resuelto en semántica matemática, definiendo cuáles funciones son "buenas".

Las acciones standard del sistema son ciertas clausuras fijas, definidas por una función sis $\in$ Cond $\rightarrow$ Cla. Por ejemplo, podría ser que para todo e $\in$ Est, resulte sis (CHECK (idf)) (e) = $\langle$ e.val, e.mem,e.ent, cons (e.val,cons (idf,e.sal))$\rangle$

En PL/I, las acciones standard pueden incluir terminaciones del proceso, no expresables en esta versión de semántica matemática, aunque sí en la que usa continuaciones [Strachey y Wadsworth].

3.2.5. INTERPRETACION DE PROGRAMAS, COMANDOS Y EXPRESIONES

El significado de un programa, comando o expresión se obtendrá a través de la función $:

$$\$ \in (Prog \cup ((Cmd \cup Expr) \times Amb)) \rightarrow (Est \rightarrow Est)$$

Un programa significa una transición, de un estado "inicial" a otro "final". Lo que interesará normalmente serán el componente est.ent

del estado inicial (datos ingresados al programa) y al componente est.sal del estado final (resultados emitidos por el programa).

Un comando, en general, produce una transformación en la memoria, la entrada y la salida. Una expresión "matemáticamente pura" produce un valor. Una expresión con efectos colaterales, como las que habrá que analizar, produce simultáneamente un valor y una transformación. Por eso se conviene en asimilar los procesos de ejecución de un comando y de evaluación de una expresión: ambos producen un nuevo estado, incluyendo el valor calculado (est.val) y los componentes memoria, entrada y salida. De acuerdo al tipo de $, la transformación de estado denotada por un comando o expresión, depende también del ambiente asociado al punto del programa en que figura dicha frase.

3.2.6. GENERACION DE NUEVAS DENOTACIONES

Se usará además una función generatriz de nuevas denotaciones:

$$\underline{nue} \in \mathbb{N} \to Den^*$$

que definiremos informalmente: para todo natural k, nue k es una k-pla de denotaciones "libres".

3.3. ECUACIONES SEMANTICAS

3.3.1. PROGRAMA

$$\$ (\ \ulcorner \text{idf: PROC OPTIONS(MAIN);}\ \text{DCL(idf.1,...idf.p)COND,(idf.p+1,...idf.q)VAR;}\ \text{cmd.1 ... cmd.n}\ \text{END;}\urcorner\ )\ (est) =$$

$$= \$ (\ \ulcorner \text{BEGIN; DCL(idf.p+1,...idf.q)VAR;cmd.1 ... cmd.n END ;}\urcorner$$
$$, \langle \emptyset\ ,\ \underline{ext}(\emptyset,c,d)\ ,\ \emptyset\ ,\ \underline{ext}(\emptyset,c,\ \langle \underline{ve},...\underline{ve},\underline{ve},\underline{ve}\ \rangle\ )\ \rangle$$
$$(\ \langle \text{est.val},\ \underline{ext}\ (\text{est.mem},d,\text{sis}(c)),\ \text{est.ent},\text{est.sal}\ \rangle\ )$$

donde $c = \langle\ \text{COND(idf.1)},...\text{COND(idf.p)},\text{ENDFILE},\text{ZDIV}\ \rangle$

$d = \underline{nue}\ (p+2)$

El significado de un programa es el mismo que el de un bloque en un ambiente que asocie a cada condición declarada (COND) o predeclarada (ENDFILE, ZDIV) una denotación actual y un indicador de habilitación, y en un estado que se obtiene del inicial inicializando cada denotación de ésas con la correspondiente acción standard del sistema.

3.3.2. ASIGNACION

$$\$ (\ \ulcorner \text{idf = expr ;}\urcorner\ ,\ amb)\ (est) =$$

$$= \underline{\text{if}}\ \ amb.hab(CHECK(idf))\ \underline{\text{then}}\ \text{est}".\ mem(amb.act(CHECK(idf)))(\text{est}")$$
$$\underline{\text{else}}\ \text{est}"\ \underline{\text{fi}}$$

donde est" = ⟨est'.val,ext(est'.mem,amb.var(idf),est'.val),est'.ent,
est'.sal⟩

est' = $(expr,amb) (est)

La interpretación de una asignación se inicia con la evaluación de su parte derecha, produciendo est'; luego se asocian en memoria la denotación de su parte izquierda y el valor calculado, produciendo est"; finalmente, si la condición CHECK está habilitada para tal parte izquierda, se ejecuta la acción asociada en ese momento a dicha condición.

El valor resultante de la asignación (est".val,si el CHECK no produce efectos extraños) es el mismo que fue transmitido. Esto sugiere inmediatamente la ecuación semántica correspondiente a una asignación múltiple

⌜ idf.1,...idf.k = expr; ⌝

donde en definitiva se estaría considerando la asignación como una expresión, como en [APL], o [Algol 68] .

3.3.3 LECTURA

$ (⌜GET (idf) ;⌝, amb) (est) =

= if est.ent=nil then est.mem(amb.act(ENDFILE))(est)
else if amb.hab(CHECK(idf))
then est'.mem(amb.act(CHECK(idf)))(est')
else est' fi fi

donde est' = ⟨v,ext(est.mem,amb.var(idf),v),tl(est.ent),est.sal⟩
v = hd(est.ent)

Si la corriente de datos no está agotada, idf pasa a poseer el valor del primer valor de dicha corriente, y ésta pierde la cabeza (el primer valor de est.ent es en cada momento el próximo valor a leer); finalmente, si CHECK está habilitado para idf, se ejecuta la acción correspondiente. Si la corriente de entrada está agotada, se ejecuta la acción asociada a ENDFILE.

3.3.4. EMISION

$ (⌜PUT (expr) ;⌝ , amb) (est) =

= ⟨est'.val,est'.mem,est'.ent,cons (est'.val,est'.sal)⟩

donde est' = $ (expr,amb) (est)

Se evalúa expr y se agrega a la cabeza de la lista de salida el valor calculado. En cada instante hd(est.sal) es el último valor emitido. El valor resultante del comando PUT es el valor emitido.

3.3.5. COMANDOS ALTERNATIVOS

$ (⌜IF expr THEN cmd⌝ , amb) (est) =
= if est'. val then $(cmd,amb)(est') else est' fi

$ (⌜IF expr THEN cmd. 1 ELSE cmd. 2⌝ , amb) (est) =
= if est'.val then $(cmd.1,amb)(est') else $(cmd.2,amb)(est') fi

donde est' = $(expr,amb)(est).

Se evalúa la expresión y se ejecuta luego el comando seleccionado según el valor hallado.

3.3.6. COMANDO COMPUESTO

$ (⌜DO; cmd.1 ... cmd.n END;⌝ , amb) (est) =
= $(cmd.n,amb)(...$(cmd.1,amb)(est)...)
o de otra forma,

$ (⌜DO; cmd.1 ... cmd.n END;⌝ , amb)
= $(cmd.n,amb)° ...° $(cmd.1,amb)

El efecto es el de componer las transiciones de estado correspondientes a cmd.1,...cmd.n. En particular (si n=0):

$(⌜DO; END;⌝ , amb)(est)=est

3.3.7. COMANDO ITERATIVO

$ (⌜DO WHILE expr; cmd.1 ... cmd.n END;⌝ , amb) = W
donde W ∈ Est → Est es tal que para todo est ∈ Est,
W(est)=if est'.val then W($(⌜DO;cmd.1...cmd.nEND;⌝ ,amb)(est'))
else est' fi
donde est' = $(expr,amb)(est).

La ecuación semántica refleja la equivalencia entre

⌜DO WHILE expr; ---⌝ y
⌜IF expr THEN DO; DO; --- DO WHILE expr; --- END;⌝

Se demuestra en semántica matemática que tal ecuación semántica recursiva tiene efectivamente una solución única.

3.3.8. BLOQUE

$ (⌜BEGIN;DCL(idf.1,...idf.h)VAR; cmd.1 ... cmd.n END;⌝ ,amb)
(est) =
= $ (⌜DO; cmd.1 ... cmd.n END;⌝
, ⟨ ext(amb.var, ⟨ idf.1,...idf.h ⟩ , nue h)
,ext(ext(Ø,x,d),z,b),amb.act
,ext(amb.hab,z, ⟨ fa,...fa ⟩) ⟩)
(⟨ est.val
,ext(ext(est.mem,d,est.mem(amb.act(x))),b,sis(z))
,est.ent , est.sal ⟩)

donde x = dom(amb.act)
d = nue(lg x)
z = ⟨ CHECK(idf.1),...CHECK(idf.h) ⟩
b = nue h

Se ejecuta el comando compuesto constituído por cmd.1,... cmd.n, en un nuevo ambiente (local) y en un nuevo estado. El ambiente local se obtiene del ambiente global amb mediante: la creación de denotaciones para las variables locales; la creación de un ambiente actual local "isomorfo" al global y extendido con denotaciones para las condiciones CHECK aplicadas a las variables locales; la adopción del ambiente actual global como ambiente latente local; y la extensión del ambiente de habilitación global teniendo en cuenta que las condiciones CHECK están inhabilitadas mientras no se las habilite expresamente. En el nuevo estado, a las denotaciones actuales creadas para las condiciones heredadas del bloque externo y para las CHECK de las variables locales, se asocian las clausuras heredadas y las de sistema, respectivamente.

3.3.9. ESTABLECIMIENTO DE CONDICION

$ (⌜ON cond cmd⌝ , amb) (est) =
= ⟨ est.val,ext(est.mem,amb.act(cond),$(cmd,amb)),est.ent,est.sal ⟩

El efecto es comparable al de una asignación en la cual la denotación actual de cond se asocia con la clausura consistente en el significado de cmd.En PL/I se toma $(⌜BEGIN;cmd END;⌝,amb), algo restrictivo.

Esta clausura incluye amb, lo que implica que al ejecutarse, por ej., ⌜SIGNAL cond;⌝ , las eventuales variables que figuren en cmd se interpretarán en amb (el ambiente del establecimiento de condición), y no en el ambiente de la frase SIGNAL. En cambio, el estado al que se aplicará la clausura será el estado en el momento del señalamiento de la condición.

Se notará la similitud con una declaración de procedimiento: se trataría de una "declaración dinámica" [Wegner].

$ (⌜ON cond SYSTEM ;⌝ , amb) (est) =
= ⟨ est.val,ext(est.mem,amb.act(cond),sis(cond)) ,est.ent,est.sal ⟩

En este caso, la clausura asociada es la acción del sistema.

3.3.10. REVERSION DE CONDICION

$ (⌜REVERT cond ;⌝ , amb) (est) =
= ⟨ est.val,ext(est.mem,amb.act(cond),est.mem(amb.lat(cond)))
,est.ent,est.sal ⟩

El efecto es comparable al de una asignación, en la cual a la denotación actual de cond se le asocia la misma clausura que a la denotación latente.

En la ejecución de 2 o más REVERTs referidos a la misma condición, sin que entre ellos se ejecuten ON o activaciones/desactivaciones de bloques, sólo el primer REVERT puede tener un efecto no nulo. Al activarse un bloque todo ocurre como si se ejecutara un REVERT generalizado a todas las condiciones declaradas.

3.3.11. SEÑALAMIENTO DE CONDICION

$ (⌜SIGNAL cond;⌝ ,amb) (est) =
=if amb.hab(cond)then est.mem(amb.act(cond))(est) else est fi

El efecto, si la condición está habilitada, es aplicar la clausura asociada a la denotación actual de cond. Nótese la analogía con una invocación de rutina.

3.3.12. COMANDOS PREFIJADOS

$ (⌜(cond):cmd⌝ , amb)
=$(cmd, < amb.var,amb.act,amb.lat,ext(amb.hab,cond,ve) >)

$ (⌜(NOcond):cmd⌝ ,amb)
=$(cmd, < amb.var,amb.act,amb.lat,ext(amb.hab,cond,fa) >)

Un prefijo ("statement prefix") afecta el indicador de habilitación del correspondiente comando. Tratándose de un componente del ambiente, el efecto del prefijo es válido sólo durante la ejecución del comando al cual se ha prefijado.

No hemos seguido estrictamente las reglas de PL/I, según las cuales en ⌜(cond): IF expr THEN cmd⌝ , por ejemplo, el prefijo valdría en expr pero no en cmd.

3.3.13. VALOR DE UN IDENTIFICADOR

$ (idf , amb) (est) = est.mem(amb.var(idf))

El valor poseído por un identificador es el asociado a (o contenido en) su denotación.

3.3.14. VALOR DE UN COCIENTE

$(⌜expr.1/expr.2⌝ , amb) (est) =
= if amb.hab (ZDIV)
  then if num(est'.val) & est'.val ≠ 0
       then if num(est".val)
            then < est".val/est'.val,est".mem,est".ent,est".sal >
            else est".mem(amb.act(ZDIV)) (est") fi
       else est'.mem(amb.act(ZDIV)) (est') fi
  else < est".val/est'.val,est".mem,est".ent,est".sal > fi
donde est' = $ (expr.2,amb)(est)
     est" = $ (expr.1,amb) (est')
     num ∈ Val → 𝔹 es el predicado característico del subconjunto de valores numéricos.

En PL/I, la verificación aquí realizada con num, se efectúa parcialmente en compilación, y eventualmente por la condición CONVERSION.

Se deduce de la ecuación dada que si ZDIV no está habilitada puede producirse una situación de error no ulteriormente analizada aquí.

3.4. CONSTRUCCIONES ADICIONALES

Se indicarán algunas construcciones no pertenecientes a PL/I, que pueden ser fácilmente definidas en el formalismo utilizado para PL.1.

3.4.1. CONSTRUCCION "AT END"

Se agrega la producción

```
    cmd :: = GET (idf) AT END cmd
```

Su significado se define por

```
    $(⌜GET(idf) AT END cmd⌝ , amb) (est) =
= if est.ent=nil then $(cmd,amb)(est)
  else if amb.hab(CHECK(idf))
       then est'.mem(amb.act(CHECK(idf)))(est')
       else est'   fi   fi

donde est' = ⟨v,ext(est.mem,amb.var(idf),v),tl(est.ent),est.sal⟩
      v    =    hd (est.ent)
```

Comparando con las ecuaciones anteriores, se advierte que esta construcción

```
no equivale a    ⌜ON ENDFILE cmd GET (idf);⌝
ni a             ⌜ON ENDFILE cmd GET (idf); REVERT ENDFILE;⌝
sino a           ⌜BEGIN; ON ENDFILE cmd GET (idf); END;⌝
```

(Este tipo de discusión se haría más difícil con las usuales definiciones de manual).

3.4.2. CONTROL DINAMICO DE ASERCIONES

En Algol W existe un comando de la forma

assert ⟨expresión lógica⟩ .

Este comando, inspirado en las aserciones de [Floyd], [Hoare 69], etc., difiere de éstas por tratarse de expresiones (del lenguaje de programación) que son verificadas dinámicamente (en ejecución) y no estáticamente. Por lo tanto no sirven para demostrar que un programa es correcto, aunque sí para corroborarlo, como herramienta de depuración.

El significado de este comando puede encuadrarse en lo ya visto, considerando un nuevo tipo de condición, inhabilitada si no se la habilita expresamente, y que se invocaría por una frase ASSERT más bien que con SIGNAL.

Se agregan las producciones:

```
cmd   :: = ASSERT expr;
cond  :: = ASSERTION
```

En la interpretación de un programa (3.3.1.) deberá tomarse en cuenta la inicialización con sis (ASSERTION), y fa.

La única ecuación semántica a agregar sería

```
    $ ( ⌜ASSERT expr ;⌝ , amb) (est) =
= if amb.hab(ASSERTION)
  then if est'.val then est'
       else est'.mem(amb.act(ASSERTION)) (est')  fi
  else est   fi
donde est' = $(expr,amb)(est)
```

Comparando con las ecuaciones anteriores se advierte que esta construcción no equivale en general a

⌜IF ¬(expr) THEN SIGNAL ASSERTION;⌝

a menos que esté en un contexto en el cual ASSERTION esté habilitada.

3.4.3. EXPRESIONES BLOQUE Y COMANDO RETURN

En [Algol W] , [Algol 68] y [BCPL] , existen expresiones precedidas por bloques, que al estilo PL/I se escribirían

```
⌜DO  ; cmd.1 ... cmd.n-1 RETURN(expr); END;⌝
⌜BEGIN;DCL (idf.1,...idf.h)VAR;cmd.1...cmd.n-1 RETURN(expr);END;⌝
```

Su interpretación estaría dada exactamente por las ecuaciones semánticas 3.3.6 y 3.3.8, definiendo

```
    $ ( ⌜RETURN(expr);⌝   ,amb) = $(expr , amb)
```

3.4.4. EXPRESION VALOR LEIDO

En [BCPL] , p. ej., se tienen expresiones como readnum (), cuyo valor es el del primer número de la corriente de entrada.

Esta construcción puede agregarse a PL.1 con la producción

```
    expr :: = READVAL
```

y la ecuación semántica

```
    $ (READVAL , amb) (est) =
= if est.ent=nil then est.mem(amb.act(ENDFILE))(est)
  else < hd(est.ent),est.mem,tl(est.ent),est.sal>   fi
```

3.5. NOTA SOBRE INVOCACION MULTIPLE Y RECURSIVIDAD

La ejecución del comando asociado a una condición puede incluir el señalamiento de otra. En tal caso se dice [PL/I] que se están ejecutando "unidades ON descendentes". Las ecuaciones semánticas dadas muestran que la última unidad ON activada es la primera en completar su ejecución.

En particular pueden coexistir varias activaciones de una misma unidad ON. Es decir, pueden establecerse condiciones recursivas. Así, el programa

```
Q: PROC OPTIONS (MAN); DCL(C)COND,(V)VAR;
   ON COND(C) BEGIN;
     PUT(V); IF V THEN DO; V = ¬ V; SIGNAL COND (C); END;
     ELSE  PUT ('UF'); END;
   V = '1' B; SIGNAL COND (C);
   END;
```

produciría como salida ⟨"UF", fa, ve ...⟩ .

Puede recordarse que, en PL/I, los procedimientos son recursivos sólo si se especifica tal propiedad, y (en ese caso) la semántica de la correspondiente declaración lo refleja [Vassallo] definiendo circularmente el nuevo ambiente creado. No ocurre lo mismo en el caso de establecimientos de condición, debido a que éstos no son declaraciones que alteren el ambiente, sino comandos que alteran el estado.

4. PARAMETRIZACION DE LAS CONDICIONES (PL.2k)

[Wegner] señala la similitud de las construcciones ON y SIGNAL, respectivamente, con "declaraciones dinámicas" e invocaciones de rutinas sin parámetros, y sugiere su parametrización.

Esto resultaría ventajoso si se considera, por ejemplo, que en muchos casos se desea reaccionar ante la ocurrencia de una condición emitiendo un mensaje variable.

La parametrización puede describirse con los mismos esquemas de la definición semántica de procedimientos ([Milne y Strachey] , [Vassallo] , [Donahue]). En efecto, las clausuras que se utilizan en este trabajo coinciden con las que se usan para evaluar procedimientos en la descripción de lenguajes del tipo de Algol.

Sin embargo, en esos casos, la clausura ("código puro" más lista de variables libres, o equivalente) es mantenida en el ambiente y no en el estado, por tratarse de declaraciones estáticas, de modo que los identificadores de procedimientos mantienen su significado en todo su alcance textual. En cambio, en lenguajes como [BCPL] , en el cual las clausuras son "almacenadas", sería posible asimilar la frase ON a una asignación y la SIGNAL a una invocación de rutina.

En la familia de minilenguajes que se definirán en esta sección, se agregarán parámetros formales en las frases ON y argumentos efectivos en la SIGNAL. Entre los varios modos de pasaje se discutirán tres: por valor, por referencia y por valor/resultado. Se examinará luego la posibilidad de determinar el mecanismo de pasaje en la ON y en la SIGNAL, la inclusión de parámetros de tipo condición y otras cuestiones.

Se definirán solamente las construcciones lingüísticas que en virtud de la parametrización se modifiquen respecto de PL.1.

4.1. PASAJE POR VALOR (PL.21)

El mecanismo por valor ("by value") es el que usa PL/I para transmitir argumentos que no sean identificadores.

4.1.1. SINTAXIS

Se agregan las producciones

cmd :: = ON cond(idf.1,...idf.k) cmd
| SIGNAL cond(expr.1,...expr.k);

Sintácticamente, los parámetros formales (idf.1,...idf.k) de la ON se usarán como identificadores de VARiables en cmd.

4.1.2. DOMINIOS SEMANTICOS

Se modifica la definición de clausura:

Cla = Val* x Est → Est

Una clausura representa una transición de estado dependiente de (0 o más) valores. Podría también haberse definido Cla = Val* → (Est → Est).

4.1.3. ECUACIONES SEMANTICAS

$(⌜ON cond(idf.1,...idf.k) cmd⌝ , amb) (est) =
= ⟨ est.val, ext(est.mem,amb.act(cond),f), est.ent, est.sal ⟩
donde f ∈ Cla es tal que para todo v ∈ Val^k, e ∈ Est:
f(v,e) = $(cmd, ext(amb.var, ⟨idf.1,...idf.k⟩ ,d), amb.act, amb.lat, amb.hab)
(⟨e.val, ext(e.mem,d,v), e.ent, e.sal⟩)
donde d = nue k.

$(⌜SIGNAL cond(expr.1,...expr.k);⌝ , amb) (est) =
= if amb.hab(cond)
then e^k. mem(amb.act(cond))(⟨e^1.val,...e^k.val⟩ , e^k)
else est fi
donde e^i = $ (expr.i,amb) (e^{i-1})
e° = est

Al señalarse una condición, se evalúan los argumentos (de izquierda a derecha) y se aplica a los valores y estado resultantes la clausura correspondiente.

4.2. PASAJE POR REFERENCIA (PL.22)

El pasaje por referencia ("by reference", "by location") es el mecanismo usual en PL/I para transmitir argumentos efectivos que sean identificadores, y en Fortran para transmitir arreglos. Permite alterar los valores de los argumentos.

La versión que aquí se presenta tiene la desventaja de admitir únicamente identificadores, como argumentos efectivos. Una versión más general está incluída en PL.25.

4.2.1. SINTAXIS

cmd :: = ON cond (idf.1,...idf.k) cmd
| SIGNAL cond(idf'.1,...idf'.k);

4.2.2. DOMINIOS SEMANTICOS

Cla = Den* x Est → Est

Una clausura representa aquí una transición de estado dependiente de 0 o más denotaciones.

4.2.3. ECUACIONES SEMANTICAS

$ (⌜ON cond(idf.1,...idf.k) cmd⌝ , amb) (est) =
= ⟨est.val,ext(est.mem,amb.act(cond),f),est.ent,est.sal⟩
donde f ∈ Cla es tal que para todo d ∈ Den^k, e ∈ Est, f(d,e) =
=$(cmd, ⟨ext(amb.var, ⟨idf.1,...idf.k⟩ ,d),amb.act,amb.lat,amb.hab⟩) (e)

$ (⌜SIGNAL cond(idf'.1,...idf'.k);⌝ ,amb) (est) =
= if amb.hab(cond)
then est.mem(amb.act(cond))(amb.var ⟨ idf'.1,...idf'.k ⟩,est)
else est fi

Al señalar la condición se ejecutará el comando asociado a la misma en un ambiente,ampliación del del establecimiento, en el que son sinónimos (tienen iguales denotaciones) los parámetros formales y los argumentos efectivos correspondientes

4.3. PASAJE POR VALOR/RESULTADO (PL.23)

El pasaje por valor/resultado ("by value result") es el usual en Fortran para transmitir argumentos correspondientes a parámetros escalares (no es exactamente el mismo que se define en [Algol W]).

Este mecanismo y el pasaje por referencia producen efectos coincidentes en muchos casos, pero no siempre.

En esta versión, como en PL.22,los argumentos deben ser identificadores. Una generalización puede hacerse utilizando denotaciones anónimas, como en PL.25.

4.3.1. SINTAXIS

cmd :: = ON cond (idf.1,...idf.k) cmd
| SIGNAL cond (idf'.1,...idf'.k) ;

4.3.2. DOMINIOS SEMANTICOS

Cla = (Den* x Est) → Est

4.3.3. ECUACIONES SEMANTICAS

$ (⌜ON cond (idf.1,...idf.k) cmd⌝ , amb) (est)=
⟨est.val,ext(est.mem,amb.act(cond),f), est.ent,est.sal⟩
donde f ∈ Cla es tal que para todo d ∈ Den^k , e ∈ Est,
f(d,e) = ⟨ e'.val,ext(e'.mem,d,e'.mem(d')),e'.ent,e'.sal⟩
donde d' = nue k
e' = $ (cmd
, ⟨ ext(amb.var,⟨idf.1,...idf.k⟩ ,d')
,amb.act,amb.lat,amb.hab ⟩)
(⟨ e.val,ext(e.mem,d',e.mem(d)),e.ent,e.sal⟩)

```
      $ ( ⌜ SIGNAL cond (idf'.1,...idf'.n);⌝  ,amb) (est) =
= if amb.hab(cond)
  then est.mem(amb.act(cond))(amb.var < idf'.1,...idf'.k > ,est)
  else est      fi
```

Al señalarse la condición se ejecutará el comando asociado a la condición, en un ambiente, ampliación del del establecimiento, en el cual se dota a los parámetros formales con nuevas denotaciones, y en un estado en el cual a dichas denotaciones se asocian los valores poseídos por los argumentos efectivos. Finalmente, se copian "en" las denotaciones de los argumentos efectivos los resultados que quedaron asociados a las nuevas.

4.4. MODO DE PASAJE ESPECIFICADO EN EL ESTABLECIMIENTO DE CONDICION (PL.24)

En algunos casos será preferible la transmisión por valor (que no impone restricciones a la forma de los argumentos efectivos) y en otros la transmisión por referencia o por valor/resultado (que permiten modificar argumentos).

Para disponer de ambas posibilidades se especificará el modo de pasaje al establecer una condición, en la tradición de [Algol] . Otra solución se verá como PL.25.

4.4.1. SINTAXIS

```
cmd   :: = ON cond((idf.1,...idf.k)VAL,(idf.k+1,...idf.m)REF)cmd
         | SIGNAL cond(expr.1,...expr.k;idf'.k+1,...idf'.m);
```

También podrían tenerse especificaciones VAL_RES (o RES).

4.4.2. DOMINIOS SEMANTICOS

```
Cla = (Val* x Den* x Est) → Est
```

4.4.3. ECUACIONES SEMANTICAS

```
      $ ( ⌜ ON cond((idf.1,...idf.k)VAL,(idf.k+1,...idf.m)REF)cmd⌝
        ,amb)  (est) =
= <est.val,ext(est.mem,amb.act(cond),f),est.ent,est.sal>
donde f ∈ Cla es tal que para todo v ∈ Val^k, d ∈ Den^(m-k), e ∈ Est,
f(v,d,e) = $ (cmd
        , < ext(ext(amb.var, < idf.1,...idf.k > ,d')
                , < idf.k+1,...idf.m > , d)
          ,amb.act,amb.lat,amb.hab > )
        ( <e.val,ext(e.mem,d',v),e.ent,e.sal> )
donde d' = nue k
      $ ( ⌜ SIGNAL cond(expr.1,...expr.k;idf'.k+1,...idf'.m);⌝
        , amb)  (est) =
= if amb.hab(cond)
  then e^k.mem(amb.act(cond))
       ( < e^1.val,...e^k.val> ,amb.var < idf'.k+1,...idf'.m > , e^k)
  else est      fi
donde e^i = $(expr.i,amb) (e^(i-1))
      e^0 = est
```

4.5. MODO DE PASAJE ESPECIFICADO EN EL SEÑALAMIENTO DE CONDICION (PL.25)

El problema planteado en 4.4 puede también resolverse "a la PL/I", reservando la decisión acerca del modo de pasaje a la invocación (SIGNAL) y no al establecimiento (ON) de la condición.

La transmisión será por referencia en el caso de identificadores, por ejemplo en ⌜SIGNAL COND(C)(X);⌝ ; y será equivalente a una transmisión por valor para otros argumentos efectivos, por ejemplo en ⌜SIGNAL COND(C)(4);⌝ .

Se utilizará en todos los casos transmisión por referencia, pero creando, para los argumentos efectivos que no sean identificadores, denotaciones "anónimas" (fuera del dominio de su ambiente).

4.5.1. SINTAXIS

cmd :: = ON cond (idf.1,...idf.k) cmd
| SIGNAL cond (expr.1,...expr.k);

4.5.2. DOMINIOS SEMANTICOS

Cla = (Den* x Est) → Est

4.5.3. ECUACIONES SEMANTICAS

$(⌜ ON cond (idf.1,...idf.k) cmd ⌝ , amb) (est) =
= ⟨est.val, ext(est.mem, amb.act(cond), f), est.ent, est.sal⟩
donde f ∈ Cla es tal que para todo d ∈ Den^k, e ∈ Est,
f(d,e) = $ (cmd
, ⟨ ext(amb.var, ⟨idf.1,...idf.k⟩, d), amb.act, amb.lat, amb.hab ⟩)
(e)

$ (⌜SIGNAL cond (expr.1,...expr.k);⌝ , amb) (est) =
= if amb.hab(cond)
then est.mem(amb.act(cond)) (⟨ d^1,...d^k ⟩ , e^k)
else est fi
donde d^i = if expr.i ∈ Idf then amb.var(expr.i) else nue 1 fi
e^i = if expr.i ∈ Idf then e^{i-1}
else ⟨e'^i.val, ext(e'^i.mem, d^i, e'^i.val), e'^i.ent, e'^i.sal⟩ fi
e'^i = $ (expr.i, amb) (e^{i-1})
e^0 = est

Al señalarse la condición, se ejecuta la clausura asociada a su denotación actual en un ambiente en el cual a cada parámetro formal corresponde una denotación (la del argumento efectivo si éste es un identificador, o una "anónima"), y en un estado en el cual cada una de estas denotaciones contiene el valor del argumento efectivo correspondiente. En particular, si todos los argumentos efectivos son identificadores, entonces dicho estado (e^k) coincide con el del señalamiento (est).

4.6. PARAMETROS DE TIPO CONDICION (PL.26).

Las construcciones anteriores pueden generalizarse, admitiendo que como argumentos a transmitir figuren no sólo expresiones que posean valores en Val, sino también condiciones que posean clausuras en Cla.

Así, podría escribirse, por ejemplo, el programa

```
P : PROC OPTIONS (MAIN) ; DCL (D,E) COND;
    ON COND(D)((U)COND,(W)VAR) DO;SIGNAL U;ON U PUT(W);END;
    ON COND(E) PUT('AH') ;
    SIGNAL COND(D) (COND(E),'OH') ;
    SIGNAL COND(E) ;
    END ;
```

obteniendo sucesivamente la emisión de "AH" y de "OH".

Se ampliará al efecto el lenguaje PL.25, transmitiendo las denotaciones de los argumentos variables, y las denotaciones actuales y latentes e indicadores de habilitación de los argumentos que sean condiciones.

4.6.1. SINTAXIS

cmd :: = ON cond((idf.1,...idf.p)COND,(idf.p+1,...idf.q)VAR)cmd
| SIGNAL cond.0 (cond.1,...cond.p,expr.p+1,...expr.q);

4.6.2. DOMINIOS SEMANTICOS

Cla = (Den* x Den* x $\mathbb{B}$ * x Den* x Est) → Est

4.6.3. ECUACIONES SEMANTICAS

$(⌜ON cond((idf.1,...idf.p)COND,(idf.p+1,...idf.q)VAR)cmd⌝ , amb) (est) =
= ⟨est.val,ext(est.mem,amb.act(cond),f),est.ent,est.sal⟩
donde f ∈ Cla es tal que para todo a,l ∈ Den^p, h ∈ $\mathbb{B}^p$,
d ∈ Den^{q-p}, e ∈ Est,
f (a,l,h,d,e) =
=$(cmd, ⟨ ext(amb.var, ⟨idf.p+1,...idf.q⟩ ,d),ext(amb.act,c,a)
,ext(amb.lat,c,l),ext(amb.hab,c,h)⟩) (e)
donde c = ⟨idf.1,...idf.p⟩

$(⌜SIGNAL cond.0(cond.1,...cond.p,expr.p+1,...expr.q);⌝ , amb) (est) =
= if amb.hab (cond.0)
then e^{q-p}. mem(amb.act(cond.0))
(amb.act(x),amb.lat(x),amb.hab(x), ⟨d^{p+1},...dq⟩ ,e^q)
else est fi
donde x = ⟨cond.1,...cond.p⟩
d^i = if expr.i ∈ Idf then amb.var(expr.i) else nue 1 fi
e^i = if expr.i ∈ Idf then e^{i-1}
else ⟨e'^{i}.val,ext(e'^{i}.mem,d^i,e'^{i}.val),e'^{i}. ent,e'^{i}.sal⟩ fi
e'^{i}= $ (expr.i,amb) (e^{i-1})
e^p = est

4.7. NOTA SOBRE PASAJE POR NOMBRE

El mecanismo de transmisión por nombre ("by name"), con su potencia (y su opacidad referencial) puede describirse, siguiendo el informe [Algol], mediante una substitución textual (no trivial) de los parámetros formales por los argumentos efectivos en el comando correspondiente a la condi-

ción. Esto sugeriría "almacenar" no ya la clausura $(cmd,amb) sino el propio texto cmd, y el ambiente amb.

Otra alternativa es el uso de "thunks" (Ingermann): ello implicaría la introducción de clausuras para procedimientos funcionales, que produzcan valores (además de eventuales cambios de estado).

Una de las aplicaciones que darían interés al pasaje por nombre sería la posibilidad de transmitir un comando. Por otra parte, esto puede probablemente simularse en PL.26, transmitiendo una condición establecida con ese comando.

BIBLIOGRAFIA

[Algol] P. Naur (Comp): Revised Report on the Algorithmic Language Algol 60 - reimpreso en [Rosen] .

[Algol W] R. Sites: Algol W. Reference Manual - Stanford Univ., 1972

[Algol 68] C.H. Lindsey, S.G. van der Meulen: Informal Introduction to Algol 68 North-Holland, 1973.

[APL] K. Iverson: A Programming Language - Wiley, 1962.

[BCPL] M. Richards, C. Withby-Strevens: BCPL, the Language and its Compiler - Cambridge Univ. Press, 1979.

[Bjørner y Jones] D.Bjørner, C.B.Jones (Comp.): The Vienna Development Method: The Meta-Language - Springer-Verlag, 1978

[Braffort y Hershberg] P. Braffort, D.Hershberg (Comp.): Computer Programming and Formal Systems - North-Holland, 1963

[Donahue] J.E. Donahue: Complementary Definitions of Programming Language Semantics - Springer-Verlag, 1976.

[Engeler] E.Engeler (Comp.): Symposium on Semantics of Algorithmic Languages- Springer-Verlag, 1971.

[Gries] D.Gries: Compiler Construction for Digital Computers - Wiley, 1971.

[Hoare 69] C.A.R. Hoare: The Axiomatic Basis of Computer Programming - Comm. ACM 12 10, 1969.

[Hoare 71] C.A.R.Hoare: Procedures and Parametres: an Axiomatic Approach - en [Engeler]

[Lucas y otros] Lucas, Lauer,Stigleitner: Method and Notation for the Formal Definition of Programming Languages - IBM TR 25087.

[McCarthy 63] J. McCarthy: A Basis for a Mathematical Theory of Computation - en [Braffort y Hershberg]

[McCarthy 66] J.McCarthy: A Formal Description of a Subset of Algol - en [Steel].

[Milne y Strachey] R.E.Milne, C. Strachey: A Theory of Programming Language Semantics - Chapman and Hall, 1976.

[NCC] :Standard Fortran Programming Manual - National Computing Centre, 1972.

[PL/I] : OS PL/I Checkout and Optimizing Compilers: Language Reference Manual- IBM GC 33-0009-4, 1976.

[Pratt] T.W.Pratt: Programming Languages: Design and Implementation - Prentice-Hall, 1975.

[Reynolds] J.C.Reynolds: Definitional Interpreters for Higher-Order Programming Languages - Proc. ACM 25th Nat.Conf., 1972.

[Rosen] S.Rosen(Comp.): Programming Systems and Languages - Mc.Graw Hill,1967.

[SCP] R.C.Holt,G.S.Graham, E.D.Lazowska, M.A. Scott: Structured Concurrent Programming With Operating Systems Applications - Addison Wesley, 1978.

[Steel] T.B.Steel(Comp.):Formal Language Description Languages - North-Holland 1966.

[Strachey]66 C.Strachey: Towards a Formal Semantics - en [Steel]

[Strachey]73 C.Strachey: The Varieties of Programming Languages - Oxford Univ. Press, 1974.

[Vassallo] G.Vassallo: Mathematical Semantics: a Tool to Survey, Relate and Combine Programming Languages - Univ. of Essex, 1974.

[Wegner] P. Wegner: Programming Languages, Information Structures and Machine Organization - McGraw Hill, 1968.