"SEMÁNTICA PARA SPL"

RAMOS, I.

En este trabajo se presenta un modelo de semántica calculatoria para el lenguaje SPL. También se presentan varias interpretaciones del lenguaje SPL, tanto en el caso en que se incluyan restricciones como en el caso en que no se incluyan. Finalmente se enumeran las consecuencias y las ventajas deducibles del SPL y de su modelización semántica que se ha descrito.

1. INTRODUCCION

Entre las distintas aproximaciones a la definición de la semántica de un lenguaje de programación: Interpretativa, calculatoria, denotacional, axiomática,... las hay que tienen en cuenta el aspecto dinámico o algorítmico de la ejecución de un programa. Es el caso de la interpretativa y calculatoria. Otras como la denotacional y la axiomática prescinden de esta característica lo que las hace inadecuadas para describir los aspectos "ejecución" de los programas: programas infinitos asociados con programas de resultado indefinido (erróneo),..

En este trabajo se formaliza la semántica de un lenguaje de programación: SPL (Structured Programming Language) /1/, /2/, /3/, /4/ de acuerdo con la aproximación calculatoria /8/ puesto que pensamos que el aspecto algorítmico de un programa es por el momento una realidad. Tendremos por lo tanto en cuenta el aspecto "ejecución" de un programa o cálculos.

En el trabajo usamos pues la notación y orden expositivo de los autores de la aproximación calculatoria a la semántica /10/.

2. CONCEPTO DE PROGRAMA EN SPL

Un programa es en SPL una forma eficaz de especificar el conjunto de ejecuciones (cálculos) que puede generar.

Una ejecución o cálculo es una cadena (o secuencia) de modificaciones (tratamientos) sobre la estructura de la información asociada con el programa.

Cada modificación (también llamada acción) viene descrita en el programa por una instrucción (tratamiento terminal).

Si una ejecución es una cadena de elementos del alfabeto de acciones (pertenecientes al repertorio de un procesador real o virtual), el conjunto de ejecuciones de un programa será un subconjunto de cadenas o sea un lenguaje sobre dicho alfabeto. Una forma cómoda y eficaz de especificar lenguajes es mediante gramáticas y en particular gramáticas algebraicas Tipo 2 de Chomsky (T2) dotadas de cierta función de etiquetaje en las reglas: S - gramáticas /4/.

El aspecto algorítmico viene pues descrito por la "generación" desde el axioma de las frases terminales (cálculos).

La interpretación se realiza a dos niveles:

- Tratamientos terminales.
- Predicados.

La estructura de información asociada con SPL depende en las implementaciones realizadas del lenguaje objeto elegido: ALGOL, COBOL, FORTRAN ...

Sintacticamente el aspecto algorítmico (ESTRUCTURA DE UN PROGRAMA) viene dado por el sis

- I. Ramos Salavert del Centre d'Informàtica de la Universitat de València. Dr. Moliner, s/n. Burjasot (Valencia)
- Article rebut el Gener de 1980.

tema de ecuaciones:

PROGRAMAS = REGLAS .
REGLAS = REGLA ; REGLAS ∪ REGLA
REGLA = IZDA ::= DERECHAS
DERECHAS = DERECHA | DERECHAS ∪ DERECHA
DERECHA = CADENA PREDICADO
CADENA = TRATAMIENTO ∪ TRATAMIENTO CADENA
TRATAMIENTO = TERMINAL ∪ NO TERMINAL
TERMINAL = NIL ∪ a ∪ b ∪ ... ∪ z
NO TERMINAL = ℙ ∪ A ∪ B ∪ ... ∪ Z
IZDA = NO TERMINAL
PREDICADO = DIGITO ∪ DIGITO PREDICADO
DIGITO = 0 ∪ 1 ∪ 2 ∪ ... ∪ 9

Los símbolos subrayados son los que aparecen en las cadenas terminales (símbolos terminales), los no subrayados son símbolos auxiliares usados para definir dichas cadenas (símbolos no terminales).

En las S - Gramáticas descritos por esta metagramática:

ℙ ∈ NO TERMINAL es el axioma

NIL ∈ TERMINAL es la instrucción vacía.

La interpretación de los terminales y predicados se da separadamente y en el lenguaje - de programación escogido.

3. CONCEPTO DE ESTADO DE MEMORIA Y DE MODIFICACIONES

Estructura de información asociada con SPL.

Un programa describe unos tratamientos sobre unos objetos.

El orden de ejecución de los tratamientos -- viene descrito por el aspecto algorítmico, - mientras que en la descripción de los objetos inicialmente se introdujo el concepto de vector de estado y en la actualidad se habla de datos abstractos.

Una de las ventajas de SPL es el que se puede prescindir de una interpretación detallada de los objetos y de los tratamientos para describir su semántica.

Necesitamos suponer una memoria (estructura de la información) /9/:

M = {O , L , M}

O = Conjunto de objetos
L = Conjunto de funciones de acceso
M = Conjunto de modificaciones

susceptible de estar en un conjunto de estados (valor en un momento dado de las funciones de acceso):

E = {eo, e, ..., en, Ω},

siendo Ω el estado indefinido que cumple

$\forall m \in M \quad m(\Omega) = \Omega$

suponemos además un conjunto de modificaciones

M = {mo, ..., mk, NIL}

tales que:

$\forall m \in M ; \quad m: E \to E$

Las modificaciones se describen en un programa mediante instrucciones (TERMINALES). La - modificación identidad se representa mediante NIL ∈ TERMINALES y la hemos notado NIL -- cumpliendo:

e ∈ E igual (NIL(e),e)

Siendo el igual la igualdad entre estados de memoria que suponemos una función primitiva definida.

4. SEMANTICA DE SPL

No vamos a describir paso a paso la ejecución de un programa sino que globalmente especificaremos el conjunto de cálculos con él asociados, siendo éste conjunto el valor semántico del programa considerado (semántica calculatoria).

Un cálculo no lo definimos pues como una sucesión de estados de memoria (semántica interpretativa) sino como una sucesión de modificaciones elementales (representadas por -- instrucciones) de la estructura de información asociada al lenguaje (memoria).

4.1 NOCIÓN DE CÁLCULO

Intuitivamente un cálculo es la ejecución en secuencia de un conjunto de acciones elementales.

Formalmente un cálculo es una secuencia finita (o no) de elementos del conjunto (M) de modificaciones elementales de la estructura de información asociada al lenguaje.

Toda modificación $m \in M$ se describe mediante una instrucción a ∈ TERMINAL.

Llamaremos M^* al monoide libre sobre M y M^ω al conjunto de cálculos infinitos. Por M^∞ notaremos

$$M^\infty = M^* \cup M^\omega$$

(es también un monoide al que se le extiende la operación de concatenación).

Todo cálculo finito

$$C = m_1 . m \ldots m_n$$

se interpreta como la modificación:

$$m_n . m_{n-1} \cdots m_1$$

de la estructura de la información, interpretando la concatenación como la composición según el orden de escritura de las modificaciones elementales:

$$m_i \in M , \quad i=1,n.$$

Usaremos pues indistintamente las dos notaciones.

4.2 COMPOSICIÓN SECUENCIAL DE CÁLCULOS

La concatenación en M permite expresar la ejecución secuencial de los cálculos.

4.3 COMPOSICIÓN CONDICIONAL E ITERATIVA

Con estas formas de composición se asocian un conjunto de cálculos. Por ejemplo a la regla:

A ::= a 1 | b 2

le asociamos el conjunto de cálculos:

{igual (Eval'(1),verdad).Eval(a)} ∪

∪ {(Eval'(2),verdad).Eval(b)}

ya dijimos que igual(u,v) es la modificación elemental, restricción de la aplicación identidad al conjunto de estados de memoria para los que u es igual a v. Una definición más precisa sería:

$$\text{igual}(u,v)(e) = \begin{cases} e \text{ si } u=v \text{ se verifica en el estado } e \\ \Omega \text{ sino} \end{cases}$$

Por otra parte, las funciones Eval se definen como:

Eval: TERMINALES → M

Eval': PREDICADO → {verdad,falso}

Materializan la Interpretación que se realiza.

Un cálculo es pues un elemento de $P(M^\infty)$: conjunto de las partes de M^∞. Todos los conjuntos de cálculos se construyen partiendo de las modificaciones elementales por aplicación de dos leyes de composición internas en $P(M^\infty)$:

- Concatenación "." extensión natural de la de M^∞ y definida por:

$$C_1 . C_2 = \{\alpha.\beta \mid \alpha \in C_1, \beta \in C_2\} \quad C_1 \in P(M^\infty), \; C_2 \in P(M^\infty)$$

- Reunión conjuntista "∪"

Cálculos asociados con una regla

El conjunto de cálculos asociados con una regla depende de los cálculos asociados a las partes derechas que las componen. Podemos definirlo por las siguientes ecuaciones:

a) Condicional

A la regla

P ::= α 1 | β 2

siendo: V = TERMINAL ∪ NO TERMINAL
P ∈ NO TERMINAL
α,β ∈ V^+
1,2 ∈ PREDICADO

le corresponde el sistema

A(P):

νP = igual(Eval'(1),verdad).$\nu(\alpha)$ ∪

∪ igual(Eval'(2),verdad).$\nu(\beta)$

Donde A(P) es el sistema de ecuaciones -- asociadas con P∈N de acuerdo con las reglas de las que es parte izquierda y $\nu(\alpha)$ es el conjunto de modificaciones asocia-- das con la cadena de instrucciones α.

Definimos $\nu(\alpha)$, $\nu(\beta)$ por recurrencia sobre la longitud α y β (notada respectivamente como $||\alpha||$ y $||\beta||$) en la forma:

$\nu(\alpha)$ = Si $||\alpha|| \leq 1$ entonces si α ∈ TERMINAL entonces Eval (α)

sino (* α ∈ NO TERMINAL *) A(α)

f si

sino (* α será: α = σ.α', con α, α' ∈ V*, σ ∈ V y $||\alpha'|| \leq ||\alpha||$ *)$\nu(\sigma).\nu(\alpha')$

f si

Las cadenas comprendidas entre (* y *) son comentarios y se incluyen en la ecuación solo a efectos aclarativos.

El sistema A(P) aparece formado por una - ecuación que liga la incógnita $\nu(P)$ aso-- ciada a P, a sus partes derechas: $\nu(\alpha)$ y $\nu(\beta)$.

En la definición de $\nu(\alpha)$ aceptamos que si $||\alpha|| < 1$ entonces Eval(α)=NIL.

Como $||\alpha||$ es finito, obtenemos una sucesión monótona decreciente que asegura que el - proceso de sustitución termina.

Solo en el caso en que existe:

A ∈ NO TERMINAL tal que A ::= A

o de una descripción explícita de un cálculo infinito como:

A ::= a A

el proceso de sustitución no termina.

b) Iteración vendrá dada en SPL por:

P ::= σ P 1 | NIL 2

donde P ∈ NO TERMINAL, σ∈V^+
1,2 ∈ PREDICADO
NIL ∈ TERMINAL

le corresponde el sistema:

$\nu(P)$ = igual(Eval'(1),verdad).$\nu(\sigma).\nu(P)$ ∪

∪ igual(Eval(2).verdad).Eval(NIL)

que traduce la equivalencia (equivalencia entendida como igualdad de las cadenas de modificaciones asociadas y usando una notación de lenguaje de programación)

$\nu(P) \equiv$ si Eval'(1) entonces $\nu(\sigma).\nu(P)$

sino Eval(NIL)

A toda regla y en general a todo programa ℙ se le asocia un sistema de cálculo A(ℙ) sobre $P(M^{\infty})$ en la forma:

$$A(\mathbb{P}): \begin{cases} \nu(\mathbb{P}) = f(\nu(p_1),\ldots,\nu(p_k)) \\ A(p_i) \ ; \quad i=1,2,\ldots,1 \end{cases}$$

siendo las p_i las partes derechas de ℙ (p_i∈V^+, i=1,k).

La función f se compone de las "funciones de base"

- Concatenación "."
- Reunión "∪"

Para este sistema hay siempre una solu--

ción mínima /10/.

La componente relativa a $\mathbb{P}$ en esta solución mínima es el conjunto de cálculos que llamamos valor semántico de $\mathbb{P}$.

5. RESTRICCION E INTERPRETACIONES DE SPL

Con la finalidad de hacer de SPL un lenguaje de programación se han realizado:

- Una restricción
- Varias interpretaciones

Restricción

Dada una regla, sean 1, 2, ... K los predicados de sus partes derechas. Se cumple que:

$$\{\bigcup_{i=1}^{k} Eval'(i) = verdad;$$

siendo $\cup$ el "o" lógico, y

$$\forall i,j \begin{cases} 1 \leq i \leq k \\ \\ 1 \leq j \leq k \end{cases}$$

$Eval'(i)$ Y $Eval'(j) = falso \}$

con lo cual obtenemos SPL - secuencial /1/, /2/, /3/, /4/, /7/.

La instrucción condicional

(si ... entonces ... sino ...)

se obtiene haciendo en:

P ::= α 1 | β 2 que

Eval'(2) = NO Eval'(1)

La instrucción iterativa se obtiene haciendo que

P ::= σ . P 1 | NIL 2 que

Eval'(2) = NO Eval'(1)...

Sin restricción

Si no incluimos la restricción, varias interpretaciones son posibles:

1) SPL con vuelta atrás

El hecho de que no se cumpla la restricción en el proceso de obtención por sustitución del cálculo partiendo del sistema de ecuaciones se debe a que en ese momento de la ejecución esta información no está disponible, lo que origina varios cálculos posibles. De ellos solo uno es correcto lo que se averigua al final del proceso de sustitución recursivamente: (Descenso recursivo desde el axioma) /6/.

2) SPL no determinista

No se cumple la restricción y varios cálculos pueden originarse en cada sustitución. Esta línea está siendo desarrollada con propósito de estudiar la equivalencia semántica entre las ejecuciones de programas no interpretados, con fines de optimización...

3) SPL concurrente

No se cumple la restricción y varios cálculos pueden originarse en una sustitución. Estos cálculos son procesos concurrentes en la realización de la tarea especificada en el programa /5/.

Interpretaciones realizadas

Las interpretaciones del SPL se han realizado especificando las funciones Eval y Eval' en distintos lenguajes de programación

- ALGOL 60 /6/
- FORTRAN /7/
- COBOL ANSI74 /7/
- PL/I /7/

Dando origen a los correspondientes sistemas de programación operativos sobre UNIVAC-1180.

Las interpretaciones a nivel objetos se realizan siempre en el lenguaje escogido.

La introducción de datos Abstractos en SPL - /5/ se está realizando al resolver un problema planteado por C. Pair (comunicación personal) a SPL:

¿Si queremos realizar la misma modificación m sobre objetos diferentes, tenemos que dar

un nombre distinto a cada instrucción que se evalúa a esa modificación o aceptamos parámetros en las instrucciones TERMINALES?.

La introducción de Datos abstractos resuelve el problema mediante el uso de parámetros -- que son objetos:

"Una modificación es la ejecución de un procedimiento: P sobre un objeto O del tipo abstracto T". Obviamente el procedimiento es -- uno de los del tipo abstracto. Por lo tanto la notación será:

.uso : P (O)

.declaración (en un T.dato abstracto)

procedure P(O:T);<cuerpo proc.>;

Un programa en SPL aparece entonces como:

```
Program   PRO     (input, output);
const...;
Abstract Data Type T1; T2;...Tn;
Objets O1 : T1  ;  O2 : T2  ; ......;
Procedure P1(O:T;O':T';...;O":T");
          . <cuerpo proc.>;
          .
          .  (* inter. objetos  *)
          .  Pn...,
          .
begin     S - Gramática   end.
```

En este caso la estructura de la información asociada con SPL vendrá dada por la reunión de los objetos definidos en cada programa y las funciones de acceso y modificaciones declaradas en los tipos abstractos a los que pertenecen.

6. CONCLUSIONES

Como conclusiones a este trabajo y en general a SPL podemos enunciar las siguientes:

- La separación clara entre el aspecto esquema de control e interpretación de un programa permite dedicarse especificamente a cada uno de ellos aumentando la legibilidad de los programas y también su "manipulabilidad" formal.

- La proximidad a la Teoría de lenguajes del esquema de control de SPL permite la reducibilidad de algunos problemas a otros en este campo tan bien definido de la Informática (y con tantos resultados).

- Tal y como está pensado, SPL es un "mecano" desde el punto de vista de los lenguajes y la programación: programación con vuelta - atrás, no determinista, concurrente...

- La metodología de la programación que permite se encuentra entre los "niveles de -- abstracción" de Dijkstra y los "refinamientos sucesivos" de Wirth.

- Permite una programación bastante "natural" ya que pensamos que:
  . el aspecto estático y dinámico se mantienen lo más próximo posible,
  . la mente humana está acostumbrada al pensamiento secuencial (pensar en varias cosas a la vez es difícil) para lo que SPL está perfectamente adaptado.

7. BIBLIOGRAFIA

/1/ RAMOS, I., MOYA, R.: "Metaprogramación - estructurada". Congreso Nacional de Informática y Automática. Madrid 1975.

/2/ RAMOS, I., MOYA, R.: "SPL (Structured -- Programming Language): A Language for -- structured programming". Annual IFIP - III Conference. Lorient 1977.

/3/ MOYA, R., RAMOS, I.: "EPL: Un lenguaje - en español para la programación estructurada". Proceso de Datos, n.70, Julio 1977, pp. 7-12.

/4/ MOYA, R., RAMOS, I.: "SP-Grammars (Con-- text-Free Grammars for Structured Pro-- gramming)". ACM-Sigplan, v.14, n.1, 1979, pp. 69-76.

/5/ RAMOS, I.: "Un modelo de esquema de control para programas concurrentes". Conferencia Centro de Cálculo de la Univ. Politécnica. Barcelona 1979.

/6/ AGUIRRE, J.: "Sistema de programación estructurada con vuelta atrás". Tesina Universidad de Bilbao. 1978.

/7/ MOYA, R.: "Diseño e implementación de un sistema de Programación Estructurada". -

Universidad de Bilbao. 1977. Tesis Doctoral.

/8/ FINANCE, J.P.: "Contribution à la Formalisation d'un Langage de Programmation, Application à ALGOL 68". Thèse de Spécialité. Univ. de Nancy I, 1974.

/9/ PAIR, C.: "Les structures d'information et leur representation en memoire". Ecole d'été d'informatique de l'AFCET. Ales 1971.

/10/ LIVERCY, C.: "Theorie des programmes". Dunod. 1978.