Semántica: principales enfoques

- Semántica Operacional

  Se define el significado mediante una *máquina abstracta* (con *estados*) y *secuencias de cómputos* sobre dicha máquina

- Semántica Denotacional

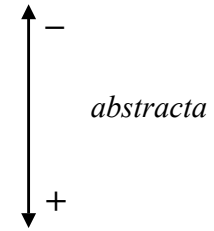
  El significado viene dado por un *objeto matemático* (en general una *función*), estando aún la noción de *estado* presente.

- Semántica Axiomática

  El significado se describe mediante *sentencias lógicas* sobre el *efecto* de la ejecución de un programa.

Semántica: principales usos

- Semántica Operacional
  - Implementadores del lenguaje
- Semántica Denotacional
  - Diseñadores del lenguaje
- Semántica Axiomática
  - Usuarios del lenguaje



- Las tres técnicas de especificación semántica asociarán significado a unas *representaciones abstractas* de los constructores sintácticos del L.P. a describir ➔ ***sintaxis abstracta del lenguaje sujeto***

Sintaxis abstracta de un lenguaje (sujeto)

- Conjunto de sentencias que identifican los hechos sintácticos de un L.P. que son relevantes semánticamente.
- La ***sintaxis abstracta*** se puede expresar en BNF eliminando el "azúcar sintáctico". Ejemplo:

```
if a=b then <statement> fi      (en Pam)   y
eq a,b : <statement>            (en Eva)
```

son sintácticamente distintas, pero su ***sintaxis abstracta*** es:

```
<cond-statement> ::= <comp> <statement>
```

y el árbol correspondiente expresa su ***forma fundamental***.

Semántica Operacional

VDL (Lenguaje de Definición de Viena)

Metalenguaje para la definición semántica operacional de un L.P. en el que puede definirse:

- *Máquina abstracta* ➔ interpreta programas (abstractos) del L.S. a través de una *secuencia de estados*
- *Sintaxis abstracta del L.S.*
- Conj. de *definiciones de instrucción* ➔ indican las transiciones entre estados

Notación de objetos VDL

Notación general para describir y manipular estructuras jerárquicas (los objetos VDL se representan mediante árboles).

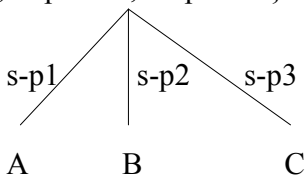
Tipos de objetos VDL

Objeto elemental: objeto sin estructura (con letras mayúsculas o caracteres entre comillas)

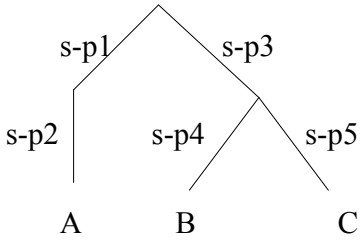
Objeto estructurado: conjunto de pares <s,a> con s selector y a objeto VDL (nombre de selector con letras minúsculas, usualmente comienza con s-)

Ejemplos

{<s-p1: A>,<s-p2: B>, <s-p3: C>} se representa como



t = {<s-p1: {<s-p2: A>}>,<s-p3: {<s-p4:B>, <s-p5: C>}>}



Objeto nulo: Ω {<s-p1:A>, <s-p2:Ω>}= {<sp-1:A>}

Consideraremos **selector : objeto -> objeto** (función)

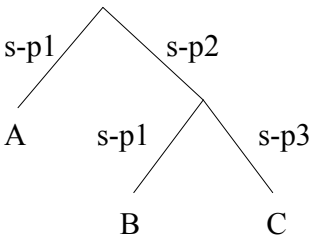
s-p3(**t**) = {<s-p4:B>, <s-p5: C>} s-p6(**t**) = Ω

(s-p4 • s-p3)(**t**) = s-p4(s-p3(**t**)) = B (composición de funciones)

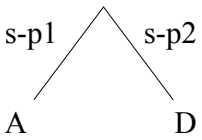
Operación "mutación" μ

μ(a; <s:b>) denota el objeto obtenido al reemplazar en el objeto **a** la componente seleccionada por **s** por el objeto **b**.

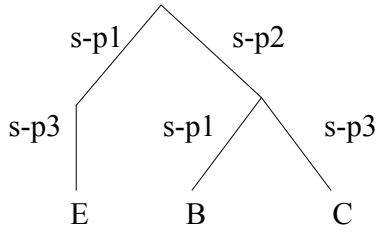
Sea **t** =



μ(**t** ; <s-p2:D>) =

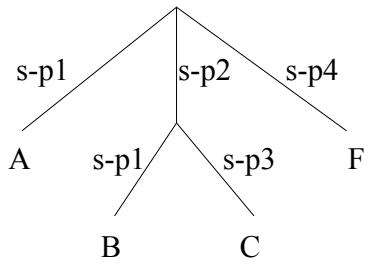


μ(**t** ; <s-p1:{<s-p3:E>}>) =



μ(**t** ; <s-p2: Ω>) =

μ(**t** ; <s-p4: F>) =

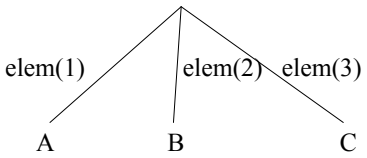


Objeto VDL "lista"

objeto cuyos selectores se nombran por elem(i) para i=1,2,...

Ejemplo:

t = {<elem(1):A>,<elem(2):B>,<elem(3):C>} = <A,B,C>



Operaciones sobre objetos VDL listas:

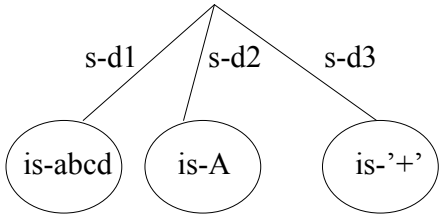
head = elem(1)	head(**t**) = A
tail	tail(**t**) = {<elem(1):B>,<elem(2):C>} = <B,C>
long	long(**t**) = 3
^ (concatenación)	**t** ^ tail(**t**) = <A,B,C,B,C>

Predicados VDL

son funciones de la forma f : Objeto -> {T, F} cuyo nombre comienza usualmente por is-

- *Predicados que preguntan por objetos elementales:*
  - is-'+' (op) ¿es op el objeto '+'?
  - is-Ω (z) ¿es z el objeto vacío?
- *Predicados que preguntan por objetos estructurados:*

is-triple = (<s-d1:is-abcd>, <s-d2:is-A>, <s-d3:is-'+'>)



is-asmt-st = (<s-lhs:is-var>, <s-rhs:is-expr>)

- *Predicados definidos por disyunción:*

  is-abcd = is-A ∨ is-B ∨ is-C ∨ is-D

  is-expr = is-infix-expr ∨ is-var ∨ is-intg
- *Predicados sobre listas: terminan por -list*

  is-expr-list predicado cierto para un objeto "lista" cuyos componentes verifican el predicado is-expr

La especificación VDL de la semántica de un LP incluye *definiciones de predicado* para:

- describir la estructura de un estado ε de la máquina abstracta.
- describir la sintaxis abstracta del Lenguaje Sujeto

Ejemplo: **VDL para la semántica de Pam** (tabla 4.1)

- ***Predicados para describir los estados de la máquina abstracta:***

  is-ε = (<s-c: is-c>, <s-stg :is-value-list>, <s-input: is-intg-list>, <s-output:is-intg-list>)

  is-value = is-intg ∨ is-UNDEFINED

Indicaciones:

s-stg(ε) es el objeto que representa la memoria del estado **ε**

elem(i) • s-stg (ε) indica el valor en memoria de la variable **i**-ésima (en el estado ε)

s-input(ε) es el objeto que representa el fich.entrada del estado **ε**

s-output(ε) es el objeto que representa el fich.salida del estado **ε**

s-c(ε) es "el árbol de control" del estado **ε**

- ***Predicados para describir la sintaxis abstracta de Pam:***

  is-series = is-st-list

  is-st = is-read-st ∨ is-write-st ∨ is-asmt-st ∨ is-cond-st ∨
  is-def-loop ∨ is-indef-loop

  -

  is-def-loop = (<s-limit: is-expr>, <s-body: is-series>)

  -

  is_var = (<s-addr: is-intg>)
  donde el entero seleccionado por s-addr será mayor que 0

  is-EQ, is-GT, ... (en s-rel de is-comp) y is-PLUS, is-MINUS, ... (en s-opr de is-infix-expr) preguntan por los objetos elementales que representan (en la sintaxis abstracta) a los op. relacionales y aritméticos (respectivamente) de Pam.

Ejercicio

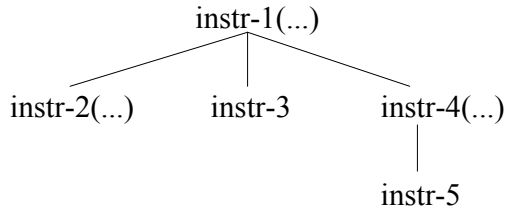
Dibujar (como árbol) el objeto VDL que satisface is-series, correspondiente al siguiente programa Pam:

```
read a;
if a = 0  then read b; write b
          else write a
fi
```

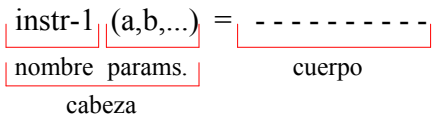
(suponer que la variable **a** se asocia a la dirección 1 y la variable **b** a la dirección 2)

VDL: control y notación para definiciones de instrucción

- La parte de control **s-c(ε)** de un estado **ε** de la máquina abstracta se puede representar mediante un árbol de instrucciones:



- Las definiciones de instrucción son de la forma:



- El cuerpo de una definición de instrucción es una expresión condicional de la forma:

  condición-1 –> grupo-1
  condición-2 –> grupo-2
  - - - - - - - - - - - - - - - -
  condición-n –> grupo-n

  que indica el primer grupo tal que su condición sea cierta, o bien el número de condiciones es 0 (cuerpo = grupo)

- Las condiciones son expresiones booleanas (la última suele ser T)
- Un grupo puede ser de dos tipos:
  - de reemplazamiento o
  - de devolución de valor y/o cambio de estado

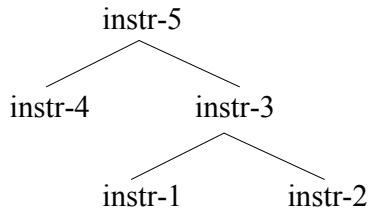
Grupo de reemplazamiento

- Función a realizar: *"el grupo de instrucciones reemplaza a la instrucción que se está interpretando en el árbol de control"*
- Es una expresión para un subárbol de instrucciones indicada por tabulación y puntuación

  Ejemplo:

```
instr-5 ;          ó      instr-5 ;
  instr-4 ,                   instr-3 ;
  instr-3 ;                     instr-2 ,
    instr-1 ,                   instr-1
    instr-2                   instr-4
```

  representan ambos al objeto (subárbol de control) siguiente:



Ejemplo en Pam:

```
repeat-series(ser,n) =
    n≤0   –>   null
    T     –>   repeat-series(ser,n-1) ;
               exec-series(ser)
```

grupo de reemplazamiento

Grupo de devolución de valor y/o cambio de estado

- Función a realizar: *"especifican un valor que se pasará como argumento a una o más instrucciones del árbol y/o producen un cambio en alguna componente del estado"*
- Consiste en:

0 ó 1 línea del tipo	PASS : expresión
0,1 ó más líneas del tipo	selector : expresión

Ejemplo en Pam:

```
input-val =
  is-<> • s-input (ε)  –>   error
  T                    –>   PASS : head • s-input (ε)
                            s-input : tail • s-input (ε)
```

grupo de devolución/cambio

- Especificación de un valor pasado
  - se prefija un *nombre* a la instrucción que lo produce (*nombre :*)
  - dicho *nombre* aparece como argumento en instrucciones "más altas" en el árbol de control

  Ejemplo:

```
instr-3 =  instr-2(..., a, ...);
             a: instr-1
```

```
instr-2(..., a, ...)
        |
a : instr-1
```

  instr-1 = PASS : exp

  al ser interpretada instr-1 se evaluará exp y su valor será pasado a un argumento de instr-2 (el indicado con el nombre **a**)

- La instrucción nula se define null = PASS : Ω

Ciclo de operación básico de una máquina abstracta VDL

1. En un estado dado, elegir para ejecutar una instrucción asociada a un nodo hoja del árbol de control (cualquiera si más de una)
2. Evaluar el cuerpo de la definición de instrucción correspondiente, obteniendo el grupo que se ha de ejecutar

   2.1. Si es un grupo de reemplazamiento:

   - sustituir los parámetros formales por los argumentos, en el grupo dado
   - reemplazar la instrucción hoja por el subárbol representado por el grupo (modificado)

   2.2. Si es un grupo de devolución de valor y/o cambio de estado

   - realizar el paso de valor y/o las acciones de cambio de estado
   - borrar la instrucción hoja del árbol de control

Repetición del ciclo de operación básico

- Repetir todo el ciclo básico hasta que el árbol de control quede vacío, lo que indica que la máquina está en un estado final
- Se parte de un estado inicial ε_1 consistente en el caso de Pam:

  s-input (ε_1) = lista de enteros (fichero de entrada)

  s-output(ε_1) = < > (fichero de salida vacío)

  s-stg(ε_1) = <UNDEFINED, UNDEFINED,...> (memoria)

  s-c(ε_1) = árbol de control formado por una sola hoja:

  exec-series(**t**)

  donde **t** es el objeto VDL que representa al programa (abstracto) Pam que va a ser interpretado.

Ejercicio

Hallar la secuencia de estados de la máquina VDL producidos al interpretar el programa Pam:

```
read a;
if a = 0  then read b; write b
          else write a
fi
```

sobre el fichero de entrada **<2, 3, 4>**