TEMA 3: SEMÁNTICA AXIOMÁTICA

1. Semántica axiomática de un lenguaje de programación
2. Tipos abstractos de datos

2. Tipos abstractos de datos

- **Para poder utilizar tipos estructurados en los programas, será necesario tenerlos definidos como Tipos Abstractos de Datos (TAD).**
- **Un TAD nos indica las operaciones que se pueden realizar y la semántica de dichas operaciones. Es independiente de la implementación.**
- **La especificación de los TAD que vamos a utilizar se denomina especificación algebraica o ecuacional.**
- **La especificación consta de dos partes: signatura y ecuaciones.**

2. Tipos abstractos de datos

- **Signatura:** Sección donde se declaran los identificadores y operaciones que se van a utilizar.

```
Tipo Nombre
    género: Tipos que se utilizan
Op:
    Id_op : Param_entrada -> Param_salida
    ....
```

  - En *género* se especifican los tipos de datos que se van a utilizar en la declaración de las operaciones. Supondremos que booleanos, enteros, naturales y reales ya están incluidos.
  - En parámetros de entrada y parámetros de salida se pone únicamente el tipo de los parámetros.

2. Tipos abstractos de datos

- Ejemplo:

```
Tipo contador
    género: contador
Op:
    cero : -> contador
    incr : contador -> contador
    decr : contador -> contador
    si_cero : contador -> booleano
```

Generadoras

- Las operaciones que nos permiten generar todos los valores posibles del tipo se denominan *generadoras*.

2. Tipos abstractos de datos

- **Ecuaciones:** Describen el comportamiento de las operaciones para cualquier posible valor de entrada.
  - Normalmente se utilizan para describir el funcionamiento de las operaciones no generadoras cuando se aplican a operaciones generadoras.

    Ejemplo:

    `decr(incr(x) ) ≡ x`

  - Cuando las operaciones generadoras producen más valores de los correctos para el TAD, hace falta escribir también ecuaciones que relacionen entre sí las generadoras (Ej. conjuntos).

2. Tipos abstractos de datos

- **Ecuaciones condicionales:** Ecuaciones de la forma:

  `t1 ≡ t2 => t3 ≡ t4`

  - Si t1 y t2 son equivalentes, entonces se puede aplicar la ecuación t3≡t4.

- **Ecuaciones de error:** indican cuando una operación parcial no se puede realizar.

  Ejemplo:

  `decr(cero) ≡ error`

2. Tipos abstractos de datos

- **Parametrización:** Uno o varios de los tipos utilizados en la especificación del TAD no se concreta. A este tipo se le denomina parámetro formal.
  - Lo indicaremos mediante las siglas PF delante del género.

```
PF. género: elemento
```

- **Operaciones ocultas:** operaciones útiles para especificar el tipo pero que no se pueden utilizar en el programa, aunque sí en los asertos. Normalmente no se implementan.

  Ejemplo: altura de una pila.

2.1. Especificación algebraica de varios TAD

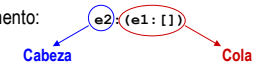
- **Lista:**
  - Secuencia de elementos.
  - El tipo estructurado recursivo más simple posible
  - Corresponde con las listas de Prolog

Ejemplo: Notación de listas

Lista vacía: `[]`

Lista con 1 elemento: `e1:[]` `[e1]`

Lista con 2 elemento: `e2:(e1:[])` `[e1,e2]`



2.1. Especificación algebraica de varios TAD

```
Tipo Lista
   P.F. género: elem
        género: lista

Operaciones:
    []   : → lista
    _:_  : elem, lista → lista
    _++_ : lista, lista → lista

Ecuaciones:    ∀ l1, l2: lista; ∀ e: elem
L1) [] ++ l1 = l1
L2) (e:l1) ++ l2 = e:(l1++l2)


Operación:     sobre lista y nat
    long: lista → nat
Ecuaciones:
    long([]) = 0
    long(e:l) = 1 + long(l)
```


2.1. Especificación algebraica de varios TAD

- **Pila:**
  - Es una lista con una operación extra para sacar el **último** elemento introducido.

Ejercicio 1: Decir cual será la pila resultante de las siguientes operaciones:

cima(desapilar(apilar(3, apilar(1, apilar(2, p_nula)))))

Ejercicio 2: Ecuaciones de *Igual(p1, p2)*.

2.1. Especificación algebraica de varios TAD

```
Tipo Pila
   P.F. género: elem
        género: pila
Operaciones:
    p_nula: → pila
    apilar: elem, pila → pila
    desapilar: pila → pila
    cima: pila → elem
    nula: pila → bool

Ecuaciones:          ∀ p: pila; ∀ x: elem
P1) desapilar(apilar(x,p)) = p
P2) cima(apilar(x,p)) = x
P3) nula(apilar(x,p)) = falso
P4) nula(p_nula) = cierto

Ec. Error:
P5) desapilar(p_nula) = error
P6) cima(p-nula) = error
```


2.1. Especificación algebraica de varios TAD

```
Operación:     sobre pila y nat
    altura: pila → nat

Ecuaciones:
PA1) altura(p_nula) = 0
PA2) altura(apilar(x, p)) = 1 + altura(p)
```

2.1. Especificación algebraica de varios TAD

- **Cola:**
  - Es una lista con una operación extra para sacar el **primer** elemento introducido.

Ejercicio 1: Decir cual será la cola resultante de las siguientes operaciones:

primero(pide_turno(4, avance(pide_turno(1, pide_turno(2, c_nula)))))

2.1. Especificación algebraica de varios TAD

Tipo Cola
P.F. género: elem
género: cola

Operaciones:
c_nula: → cola
pide_turno: elem, cola → cola
primero: cola → elem
avance: cola → cola
nula: cola → boolean

Ecuaciones: ∀ c: cola; ∀ x, y: elem
C1) primero(pide_turno(x, pide_turno(y, c))) ≡ primero(pide_turno(y, c))
C2) primero(pide_turno(x, c_nula)) ≡ x
C3) avance(pide_turno(x, pide_turno(y, c))) ≡ pide_turno(x,avance(pide_turno(y,c)))
C4) avance(pide_turno(x, c_nula)) ≡ c_nula
C5) nula(pide_turno(x, c)) ≡ falso
C6) nula(c_nula) ≡ cierto

2.1. Especificación algebraica de varios TAD

Ec. Error
C7) avance(c_nula) ≡ error
C8) primero(c_nula) ≡ error

Operación: sobre cola y nat
long: cola → nat

Ecuaciones:
CL1) long(c_nula) ≡ 0
CL2) long(pide_turno(x, c)) ≡ 1 + long(c)

2.1. Especificación algebraica de varios TAD

- **Árbol:**
  - Existen diversas variantes. Aquí definiremos únicamente árboles binarios y consideraremos que el árbol más simple posible es el árbol nulo (sin elementos).

Ejemplo: Árbol con un solo elemento:
a = plantar(x, a_nulo, a_nulo)

Ejercicio: Representar el siguiente árbol utilizando las constructoras del tipo.

2.1. Especificación algebraica de varios TAD

Tipo Árbol
P.F. género: elem
género: árbol

Operaciones:
a_nulo: → árbol
plantar: elem, árbol, árbol → árbol
hi, hd: árbol → árbol
raíz: árbol → elem
nulo: árbol → boolean

Ecuaciones:
A1) hi(plantar(x, a1, a2)) ≡ a1
A2) hd(plantar(x, a1, a2)) ≡ a2
A3) raiz(plantar(x, a1, a2)) ≡ x
A4) nulo(plantar(x, a1, a2)) ≡ falso
A5) nulo(a_nulo) ≡ cierto

Ec. Error
A6) raiz(a_nulo) ≡ error
A7) hi(a_nulo) ≡ error
A8) hd(a_nulo) ≡ error

2.1. Especificación algebraica de varios TAD

Operación: sobre árbol y enteros
altura: árbol → entero
Ecuaciones:
AA1) altura(plantar(x, a1, a2)) ≡ 1 + max(altura(a1), altura(a2))
AA2) altura(a_nulo) ≡ 0

Operación: sobre árbol y enteros
tam: árbol → entero
Ecuaciones:
AT1) tam(plantar(x, a1, a2)) ≡ 1 + tam(a1) + tam(a2)
AT2) tam(a_nulo) ≡ 0

2.1. Especificación algebraica de varios TAD

- **Tablas y vectores:**
  - Representan una función parcial.
  - Tienen un dominio llamado índice y una imagen llamada valor.
  - Llamamos **tabla** a una función parcial donde no existe ninguna restricción sobre los índices.
  - Llamamos **vector** a una especialización de la tabla donde los índices son una subsecuencia de enteros o un tipo enumerado. Habrá por tanto un índice máximo y mínimo.

Ejercicio: Decir cual será la tabla resultante de las siguientes operaciones:

val(asig(asig(crear, 1, 10), 2, 20), 1)

2.1. Especificación algebraica de varios TAD

Tipo Tabla
P.F. género: índice, valor

Operaciones:
= : índice, índice → boolean

Ecuaciones:
(i = i) ≡ cierto
(i = j) ≡ cierto ≡ > (j = i) ≡ cierto
(i = j ∧ j = k) ≡ cierto ≡ > (i = k) ≡ cierto

Consta además de:
género: tabla

Operaciones:
crear: → tabla
asig_t: tabla, índice, valor → tabla
val_t: tabla, índice → valor
def_t: tabla, índice → boolean

2.1. Especificación algebraica de varios TAD

Ecuaciones: ∀ t: tabla; ∀ i, j: índice; ∀ x, y: valor
T1) i = j ≡ cierto ≡ > asig_t(asig_t(t, i, x), j, y) ≡ asig_t(t, j, y)
T2) i = j ≡ falso ≡ > asig_t(asig_t(t, i, x), j, y) ≡ asig_t(asig_t(t, j, y), i, x)
T3) i = j ≡ cierto ≡ > val_t(asig_t(t, i, x), j) ≡ x
T4) i = j ≡ falso ≡ > val_t(asig_t(t, i, x), j) ≡ val_t(t, j)
T5) i = j ≡ cierto ≡ > def_t(asig_t(t, i, x), j) ≡ cierto
T6) i=j ≡ falso ≡ > def_t(asig_t(t, i, x),j) ≡ def_t(t,j)
T7) def_t(crear, i) ≡ falso

Ec. de error:
T8) val_t(crear, i) ≡ error

2.1. Especificación algebraica de varios TAD

Tipo Vector (sobre boolean, nat, tabla)
P.F. género: valor
renombro: "tabla de nat y valor" como "vector de valor"

Operaciones:
min, max: → nat

c: nat → nat
asig: vector, índice, valor → vector
val: vector, índice → valor
def: vector, índice → boolean

Ecuaciones: ∀ a: vector; ∀ i: índice; ∀ x: valor
V1) (min ≤ i) ∧ (max ≥ i) ≡ cierto ≡ > c(i) ≡ I
V2) asig(a, i, x) ≡ asig_t(a, c(i), x)
V3) val(a, i) ≡ val_t(a, c(i))
V4) def(a, i) ≡ def_t(a, c(i))

Ec. de error:
V5) (i < min) ≡ c(i) ≡ error
V6) (i > max) ≡ c(i) ≡ error

2.1. Especificación algebraica de varios TAD

- **Listas con punto de interés:**
  - También se las conoce como estructuras con acceso secuencial (o ficheros).
  - Son secuencias de elementos, al igual que las listas, pilas o colas, pero en estas estructuras se permite recorrer la estructura elemento a elemento sin necesidad de extraerlos.

Ejercicio: Decir cual será la lista resultante de las siguientes operaciones:

actual(avanzar(iniciar(insertar(2, insertar(1, crear)))))

2.1. Especificación algebraica de varios TAD

Tipo Lista_PI (sobre boolean, lista)
P.F. género: elem

Operaciones:
<_,_> : lista, lista → lista_PI
crear : → lista_PI
iniciar : lista_PI → lista_PI
avanzar : lista_PI → lista_PI
actual : lista_PI → elem
final : lista_PI → bool
vacia : lista_PI → bool
insertar : elem, lista_PI → lista_PI
eliminar : lista_PI → lista_PI

2.1. Especificación algebraica de varios TAD

Ecuaciones: ∀ l1, l2: lista; ∀ e: elem

LPI1) *crear ≡ <[],[]>*
LPI2) *iniciar(<[],l1>) ≡ <[],l1>*
LPI3) *iniciar(<e: l1, l2>) ≡ iniciar(<l1, e: l2>)*
LPI4) *avanzar(<l1, e: l2>) ≡ <e: l1, l2>*
LPI5) *actual(<l1, e: l2>) ≡ e*
LPI6) *final(<l1, e: l2>) ≡ falso*
LPI7) *final(<l1, []>) ≡ cierto*
LPI8) *insertar(e1, <l1, l2>) ≡ <e: l1, l2>*
LPI9) *eliminar(<l1, e: l2>) ≡ <l1, l2>*

Ec. de Error:

LPI10) *avanzar(<l1, []>) ≡ error*
LPI11) *actual(<l1, []>) ≡ error*
LPI12) *eliminar(<l1, []>) ≡ error*

2.2. Deducción ecuacional

- **Conociendo las ecuaciones de un TAD, ya se puede utilizar dentro de un programa y verificarlo.**
- **Dentro del programa o de los asertos se deberán utilizar únicamente las operaciones definidas en el TAD.**

Ejercicio 1:

```
{Pre: ??}
        p := apilar(x, p)
{Post: cima(p) = 1}
```

2.2. Deducción ecuacional

- **En el tipo vector no hemos utilizado hasta ahora las operaciones ni las ecuaciones del TAD. ¿No se podría seguir así?**

Ejercicio 2:

```
{Pre: ??}
        a[a[2]] := 1
{Post: a[a[2]] = 1}
```

Calcular la precondición. Comprobar con el vector a[1] = 3 y a[2] = 2 si la precondición es correcta.