ALGORITMOS Y ESTRUCTURAS DE DATOS II

Ingeniería Técnica en Informática de Gestión

Ingeniería Técnica en Informática de Sistemas

CURSO 2002/03

Documentación complementaria

Tema 1

- Ejemplos de especificaciones algebraicas
  - Polinomio
  - Figuras de trazos
  - Club de Socios
  - Palabras

- Implementación del TAD "conjunto de caracteres"

Ejemplos de especificaciones algebraicas

TAD *polinomio*

especificación tadPolinomio
  usa entero, booleano, cardinal
  género polinomio
  operaciones
    poliCero: → polinomio
    añadir: polinomio entero cardinal → polinomio
    eliminar: polinomio cardinal → polinomio
    sumar: polinomio polinomio → polinomio
    multiplicar: polinomio polinomio → polinomio
    derivada: polinomio → polinomio
    integral: polinomio → polinomio
    evaluar: polinomio entero → entero
    coeficiente: polinomio cardinal → entero
    nulo: polinomio → booleano

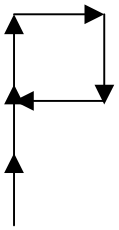
  ecuaciones p: polinomio; c, c1, c2, num: entero; g, g1, g2: cardinal

```
añadir (p, 0, g) = p
añadir (añadir (p, c1, g1), c2, g2) =  si g1 = g2 entonces
                                          añadir (p, c1+c2, g1)
                                       sino
                                          añadir (añadir (p, c2, g2), c1, g1)
                                       fsi


eliminar (poliCero, g) = poliCero
eliminar (añadir (p, c, g1), g2) =  si g1 = g2 entonces
                                          eliminar(p, g)
                                    sino
                                          añadir (eliminar (p, g2), c, g1)
                                    fsi


sumar (poliCero, p) = p
sumar (añadir (p, c, g), q) =  añadir (sumar (p, q), c, g)


multiplicar (poliCero, p) = poliCero
multiplicar (añadir (poliCero, c1, g1), añadir (q, c2, g2)) =
          = añadir (multiplicar (añadir (poliCero, c1, g1), q), c1*c2, g1+g2)
multiplicar (añadir (añadir (p, c1, g1), c2, g2), q) =
          = sumar(multiplicar(añadir(poliCero,c2,g2),q),multiplicar(añadir(p,c1,g1),q))


derivada (poliCero) = poliCero
derivada (añadir (p, c, g)) =  si g = 0 entonces
                                       derivada (p)
                               sino
                                       añadir (derivada (p), c*g, g-1)
                               fsi
```

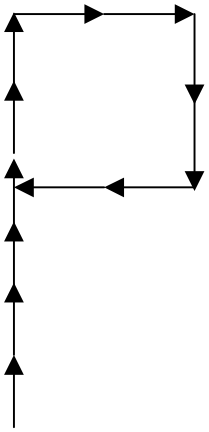
```
integral (poliCero) = poliCero
integral (añadir (p, c, g)) =  si g = 0 entonces
                                    añadir (integral (p), c, 1)
                               sino
                                    añadir (integral (p), c/(e+1), g+1)
                               fsi


evaluar (poliCero, num) =      0
evaluar (añadir (p, c, g), num) = evaluar (p, num) + (c*num^e)


coeficiente (PoliCero, g) = 0
coeficiente (añadir (p, c, g1), g2) =  si g1 = g2 entonces
                                            c + coeficiente (p, g2)
                                       sino
                                            coeficiente (p, g2)
                                       fsi


nulo (poliCero) = verdad
nulo (añadir (p, c, g)) = falso
```

TAD *figura*

Se desea representar símbolos mediante una secuencia de trazos, simulando la escritura manual. A dichos símbolos los denominaremos **figuras**. Para crearlas se dispone de 4 tipos de trazos: **D** (Derecha), **I** (Izquierda), **S** (Subir), **B** (Bajar). Por simplificar, supondremos que dos figuras son iguales si lo son las secuencias de trazos que las representan.



SSSDBI (Fig. a)



SSSSSSDDBBII (Fig. b)

espec figuras
  usa booleanos, naturales
  géneros trazo, figura
  operaciones
        D, I, S, B: → trazo
        fVacía: → figura
        añadir: figura trazo → figura
        vacia?: figura → booleano
        iguales?: figura figura → booleano
        giro: figura → figura /* Gira una figura 90 grados a la derecha */
        zoom2x: figura → figura /* Aumenta por 2 una figura (figura b.) */
        trazosH: figura → natural /* Número de trazos horizontales de una figura */

ecuaciones f, f1, f2: figura; t, t1, t2: trazo

        vacia? (fVacía) = verdad
        vacia? (añadir (f, t)) = falso

        iguales? (fVacía, f) = vacia? (f)
        iguales? (añadir (f1, t1), añadir (f2, t2)) = iguales? (f1, f2) **&** (t1 = t2)
        iguales? (f1, f2) = iguales? (f2, f1) /* conmutatividad */

        giro (fVacía) = fVacía
        giro (añadir (f, t)) = **caso**
                t = D: añadir (giro (f), B)
                t = I: añadir (giro (f), S)
                t = S: añadir (giro (f), D)
                t = B: añadir (giro (f), I)
            fcaso

        zoom2x (fVacía) = fVacía
        zoom2x (añadir (f, t)) = añadir (añadir (zoom2x (f), t), t)

        trazosH (fvacía) = 0;
        trazosH (añadir (f, t)) = **caso**
                (t = I) $\veebar$ (t = D): trazosH (f) + 1
                t = S: trazosH (f)
                t = B: trazosH (f)
            fcaso

fespec

TAD *Club de Socios*

Un estudiante de informática pertenece a un club de socio de su localidad y el presidente le ha pedido que diseñe una aplicación para el mantenimiento de los socios y del club. El estudiante ha decidido usar los conceptos adquiridos en la asignatura de *Algoritmos y Estructuras de Datos II* para diseñar dicha aplicación, así que, usando la metodología correcta, ha pensado definir un TAD llamado **socio** para almacenar la información de cada uno de los socios, y otro TAD llamado **club**, que será el que realice el mantenimiento de los socios.

La especificación algebraica del TAD **club** es la siguiente:

especificación Club de Socios
  usa booleanos, enteros, cadenas, socios, listaSocios
  género club
  operaciones

creaClub: → club	*{crea el club vacío}*
altaSocio: club socio → club	*{añade un nuevo socio}*
bajaSocio: club entero → club	*{elimina un socio, dado su nº de socio}*
juveniles: club → listaSocios	*{devuelve una lista con los socios del club que son juveniles}*
parcial primero: club →socio	*{devuelve el primer socio en orden alfabético}*
listado: club → listaSocios	*{devuelve una lista con todos los socios ordenados alfabéticamente}*

fespec

(1) Especificar algebraicamente el TAD **socio** con una operación generadora de socios y cuatro operaciones observadoras para extraer la información del socio. Para crear este TAD es necesario usar el tipo *tipoSocio*, que consiste en cuatro constantes (I: Infantil, J: Juvenil, S: Senior, A:Adulto). Cada socio estará identificado por 4 campos: un entero, una cadena, un *tipoSocio* y una fecha, que corresponden con el nº de socio (campo identificativo), nombre, tipo y fecha de nacimiento. Suponer que tenemos disponible el tipo *fecha*.

(2) Completar la especificación algebraica del TAD club con el dominio de definición y las ecuaciones de cada una de las operaciones. Suponer que tenemos disponible el tipo *listaSocios*, con las operaciones típicas de las listas vistas en clase (*[]*, *+izq*, *_&_*, etc.)

(1)

especificación Socios
  usa enteros, cadenas, fechas, tipoSocio
  géneros socio
  operaciones
    creaSocio: entero cadena tipoSocio fecha → socio
    numSocio: socio → entero
    nombreSocio: socio → cadena
    tipo: socio → tipoSocio
    fechaN: socio → fecha

  ecuaciones n: entero; N:cadena; t: tipoSocio; f: fecha
    numSocio (creaSocio (n,N,t,f)) = n
    nombreSocio (creaSocio (n,N,t,f)) = N
    tipo (creaSocio (n,N,t,f)) = t
    fechaN (creaSocio (n,N,t,f)) = f

fespec

(2)

dominio de definición c: club; s: socio
primero (altaSocio (c,s))

ecuaciones c: club; s,s1,s2: socio; n: entero;

altaSocio (altaSocio (c, s1), s2)) = **si** numSocio(s1) = numSocio(s2) **entonces** altaSocio(c, s1)
sino altaSocio (altaSocio (c, s2), s1))
fsi

bajaSocio (creaClub, n) = creaClub
bajaSocio (altaSocio (c,s), n) = **si** numSocio(s) = n **entonces** bajaSocio(c,n)
sino altaSocio (bajSocio (c,n), s)
fsi

juveniles (creaClub) = listaVacía
juveniles (altaSocio (c,s)) = **si** tipo(s) = "J" **entonces** +izq (s, juveniles (c))
sino juveniles(c)
fsi

primero (altaSocio (creaClub, s)) = s
primero (altaSocio (altaSocio (c, s1), s2)) = **si** nombre (s1) < nombre (s2) **entonces** primero (altaSocio (c, s1))
sino primero (altaSocio (c, s2))
fsi

listado (creaClub) = listaVacía
listado (altaSocio (c, s)) = +izq (primero (altaSocio (c, s)),
listado (bajaSocio (altaSocio (c, s), numSocio (primero (altaSocio (c, s))))))

fespec

TAD *Palabras*

Se desea crear el TAD *Palabras* con un conjunto de operaciones. Para ello se dispone de la siguiente signatura:

espec PALABRAS
usa cadenas, naturales, booleanos
género palabra
operaciones

pVacía: → palabra	*{crea la palabra vacía}*
pon-letra: palabra carácter → palabra	*{añade una letra al final de una palabra}*
longitud: palabra → natural	*{devuelve la longitud de una palabra}*
pon-primera: carácter palabra → palabra	*{añade una letra al principio de una palabra}*
parcial letra: natural palabra → carácter	*{devuelve la letra i-ésima}*
parcial primera: palabra → carácter	*{devuelve la primera letra de una palabra}*
parcial última: palabra → carácter	*{devuelve la última letra de una palabra}*
parcial elimPrim: palabra → palabra	*{devuelve la palabra sin la primera letra}*
vacía?: palabra → booleano	*{indica si una palabra está vacía}*
igual?: palabra palabra → booleano	*{indica si una palabra es igual a otra}*
inversa?: palabra palabra → booleano	*{indica si una palabra es la inversa de otra}*
capicúa?: palabra → booleano	*{indica si una palabra es capicúa}*
pon-final: palabra → palabra	*{desplaza la primera letra al final de la palabra}*
pon-principio: palabra → palabra	*{desplaza la última letra al principio de la palabra}*
ordenada?: palabra → booleano	*{devuelve verdadero si las letras de la palabra están ordenadas alfabéticamente}*

Ejemplos: pon-final ($a_1\ a_2\ ...\ a_n$) = $a_2\ a_3\ ...\ a_n\ a_1$
pon-principio ($a_1\ a_2\ ...\ a_n$) = $a_n\ a_1\ ...\ a_{n-1}$

Clasificar las operaciones y completar la especificación algebraica del TAD PALABRAS con los dominios de definición de las operaciones parciales y las ecuaciones, sabiendo que las operaciones generadoras son *pVacía* y *pon-letra*.

dominio de definición i: natural; p: palabra; c: caracter
letra (i, p) está definido sólo si $0 < i \leq$ longitud (p)
primera (pon-letra (p, c))
última (pon-letra (p, c))
elimPrim (pon-letra (p, c))
ecuaciones c, c1, c2: carácter; p,p1,p2: palabra; i: natural
longitud (pVacia) = 0
longitud (pon-letra (p, c)) = suc (longitud (p))

pon-primera (c, pVacia) = pon-letra (pVacia, c)
pon-primera (c1, pon-letra (p, c2)) = pon-letra (pon-primera (c1, p), c2)

letra (i, pon-letra (p, c)) = **si** i = suc (longitud (p)) **entonces** c
sino letra (i, p)
fsi

/ versión 1 */*
primera (pon-letra (p, c)) = **si** vacía? (p) **entonces** c
sino primera (c)
fsi

/ versión 2 */*
primera (p) = **si** ¬ vacía?(p) **entonces** letra (1, p)

última (pon-letra (p, c)) = c

elimPrim (pon-letra (p, c)) = **si** vacía? (p) **entonces** pVacía
sino pon-letra (elimPrim (p), c)
fsi

vacía? (pVacía) = verdad
vacía? (pon-letra (p, c)) = falso

igual? (pVacía, pVacía) = verdad
igual? (pVacía, pon-letra (p, c)) = falso
igual? (pon-letra (p, c), pVacía) = falso
igual? (pon-letra (p1, c1), pon-letra (p2, c2)) = c1 = c2 $\wedge$ igual? (p1, p2)

inversa? (pVacía, pVacía) = verdad
inversa? (pVacía, pon-letra (p, c)) = falso
inversa? (pon-letra (p, c), pVacía) = falso
inversa? (pon-letra (p1, c1), pon-letra (p2, c2)) = primera (pon-letra (p1, c1)) = c2 $\wedge$
inversa? (elimPrim (pon-letra (p1, c1)), p2)

capicúa? (p) = inversa? (p, p)

pon-final (pVacía) = pVacía
pon-final (pon-letra (p, c)) = **si** vacía? (p) **entonces** pon-letra (p,c)
sino pon-letra (elimPrim (pon-letra (p, c), primera (p)))
fsi

pon-principio (pVacía) = pVacía
pon-principio (pon-letra (p, c)) = pon-primera (c, p)

ordenada? (pVacía) = verdad
ordenada? (pon-letra (p, c)) = **si** vacía?(p) **entonces** verdad
sino ordenada (p) $\wedge$ última (p) $\leq$ c
fsi

fespec

Definición e Implementación del TAD "conjunto de caracteres"

```
módulo conjuntos
importa booleanos, caracteres, cardinales fimporta
exporta  (* INTERFAZ. La documentación completa del tipo y de las operaciones se deja como ejercicio para el alumno *)
         tipo       conjcar;
         acción vacío (var A: conjcar);
                  (* inicializa a Falso el conjunto A *)
         función esVacío (A: conjcar) dev (booleano);
                  (* devuelve si A es o no vacío *)
         acción poner (c: carácter; var A: conjcar);
                  (* inserta el carácter c en el conjunto A *)
         acción quitar (c: carácter; var A: conjcar);
                  (* elimina el carácter c del conjunto A *)
         función pertenece (c: carácter; A: conjcar) dev (booleano);
                  (* ¿c ∈ A? *)
         acción unión (A,B: conjcar; var C: conjcar);
                  (*  realiza la unión entre A y B, almacenándola en C *)
         acción intersección (A,B: conjcar; var C: conjcar);
                  (*  realiza la intersección entre A y B, almacenándola en C *)
         función cardinal (A: conjcar) dev (cardinal);
                  (* devuelve el número de elementos del conjunto A *)
fexporta

implementación
         tipo conjcar = registro
                              elem: array [0..255] de booleano;
                              card: cardinal;
                        fregistro
         ftipo


         acción vacío (var A: conjcar);
         var  c: cardinal;  fvar
         inicio
          A.card:= 0;
          para c:= 0 hasta 255 hacer
                  A.elem[c]:= falso;
          fpara
         facción
         función esVacío (A: conjcar) dev (booleano);
         inicio
          devuelve (A.card = 0)
         ffunción


         función pertenece (c:carácter; A: conjcar) dev (booleano);
         inicio
          devuelve (A.elem[c])
         ffunción
```

```
acción poner (c:carácter; var A: conjcar);
inicio
  si ¬pertenece (c, A) entonces
            A.elem[c]:= verdad;
            A.card := A.card +1;
  fsi
facción


acción quitar (c:carácter; var A: conjcar);
inicio
  si pertenece (c, A) entonces
            A.elem[c]:= falso;
            A.card := A.card -1;
  fsi
facción


acción unión (A,B: conjcar; var C: conjcar);
var  x: cardinal;  fvar
inicio
  C.card:= 0;
  para x:= 0 hasta 255 hacer
            C.elem [x]:= A.elem[x] or B.elem[x];
            si C.elem[x] entonces
              C.card:= C.card +1;
            fsi
  fpara
facción


acción intersección (A,B: conjcar; var C: conjcar);
var   x: cardinal;  fvar
inicio
  C.card:= 0;
  para x:= 0 hasta 255 hacer
            C.elem [x]:= A.elem[x] and B.elem[x];
            si C.elem[x] entonces
              C.card:= C.card +1;
            fsi
  fpara
facción



función cardinal (A: conjcar) dev (cardinal);
inicio
  devuelve (A.card);
ffunción


fmódulo conjcar
```