Una especificación formal de tipos estructurados de datos (Parte primera).

Por: F. Orejas. Facultad de Matemáticas.
Universidad Complutense.

0. Introducción

La utilización del concepto de "Tipo abstracto de Datos", ha surgido desde hace unos años, como una de las herramientas más potentes para el desarrollo de programas fiables. Desde este punto de vista, la concepción de un programa para la resolución de un problema dado, sería la siguiente:

En una 1ª fase se realiza la definición formal del problema. Esta definición tiene dos partes, la primera es la especificación de los tipos de datos del problema, junto con las operaciones definidas sobre ellos; estos tipos de datos tienen que ser, simplemente, una abstracción de lo que son ralmente los datos del problema, en ningún caso una representación de los mismos. Por ejemplo, ante la construcción de un programa para jugar al ajedrez, uno de los tipos de datos a definir sería el tipo tablero, con las operaciones: Jaque?, Jaque mate?, mover peon, movercaballo, etc. Lo que no habría que hacer sería intentar representar el tablero, definiendolo como una matriz, codificando las fichas, los movimientos, etc.

La 2ª parte de la definición, consistiria en la especificación del problema desde un punto de vista formal (por ej. por medio de axiomas), en terminos de los tipos definidos previamente.

En la 2ª fase de la concepción del programa, se desarrolla un algoritmo para resolver el problema, que utilice exclusivamente las operaciones asociadas a los tipos de datos.

En la 3ª fase, se implementan los tipos de datos del problema en terminos de los tipos disponibles en el lenguaje de programación que se utilice. Esta implementación consiste en asociar a los tipos de datos del problema, tipos de datos del lenguaje, y en construir procedimientos asociados a la inicialización y a cada operación existente en los tipos de datos del problema, utilizando las operaciones existentes sobre los tipos de datos del lenguaje. Por ejemplo, si hay que implementar una pila de 100 elementos como máximo, de series de caracteres, en PL/1, se le puede asociar, para representarla, la estructura:

```
1 PILA,
  2 VECTOR (100) CHAR,
  2 PTER DEC FIXED (3);
```

y, por ejemplo, a la operación PUSH (PILA, X) que mete en la pila, la serie de caracteres X, habría que asociarle el procedimiento:

```
PROC  PUSH (PILA, X)
DCL   1 PILA
        2 VECTOR (100) CHAR,
        2 PTER DEC FIXED (3);
      X   CHAR;
IF PTER<100 THEN PTER=PTER+1;
           ELSE ERROR;
  VECTOR (PTER)=X
END
```

Igualmente, tendría que haber procedimientos asociados a la inicialización de la pila y al resto de las operaciones.

A continuación, se codifica el algoritmo en términos del lenguaje de programación utilizado, substituyendo las referencias a las operaciones de los tipos de datos del problema, por llamadas a sus procedimientos asociados, teniendo cuidado de efectuar al principio las llamadas a los procedimientos de inicialización.

En la última fase se verifica el programa, para ello, la concepción que se ha seguido, permite la verificación independiente de la implementación del algoritmo y de los datos.

Como se vé, esta técnica de concepción de programas, permite la obtención de programas muy modulares y faciles de verificar, además, hasta llegar a la fase de implementación, existe una independencia total con respecto al lenguaje que se vaya a utilizar posteriormente. En este momento existen varios lenguajes de programación orientados especificamente hacia la programación con tipos de datos abstractos , como son el SIMULA 67, CLU, ALPHARD, EUCLID y otros. La técnica ha tenido también especial importancia en el diseño de sistemas operativos, un monitor básico en el sentido de Hoare y Brinch-Hansen, es en realidad un tipo abstracto de datos.

El problema fundamental aparece a la hora de establecer cómo deben ser las especificaciones formales de los tipos de datos. De acuerdo con Liskov (Liskov y Zilles 1975, Liskov y Berzins 1977) existen basicamente dos formas de especificación: las especificaciones axiomáticas y las que utilizan un modelo abstracto.

Las especificaciones axiomáticas, definen un tipo de datos a partir de sus propiedades, que son modelizadas en terminos de axiomas. Sus mayores ventajas son: facilitan la verificación de las implementaciones, su grado de abstracción e independencia de representaciones, y su elegancia. Sus mayores inconvenientes: la dificultad de establecer los axiomas correctamente, sin que falten (problema de completitud) y sin que se contradigan (problema de consistencia); la dificultad de comprender el tipo definido, esto es, los axiomas no dan idea de cómo "es" realmente el tipo, y por último su poca constructividad. En este tipo de especificaciones han adquirido especial importancia las llamadas especificaciones algebraicas (ADJ, GUTTAG, ZILLES, etc).

Las especificaciones que utilizan un modelo abstracto, consisten en representar el tipo de datos a definir, en terminos de otro tipo de datos bien conocido, por ejemplo grafos (Earley, Majster, etc.), conjuntos (Hoare), o árboles (Ollongren). La especificación es, entonces, una implementación. Las ventajas que presentan estas especificaciones son: la facilidad de ser establecidas por cualquier programador adecuadamente entrenado, la facilidad para ser comprendidas (siempre que no sean muy complicadas), y la constructividad. Sus inconvenientes: poca utilidad en verificaciones; por otra parte la dependencia del modelo implica, en algunos casos, tener que prestar atención a problemas que no estan en el tipo de datos a definir, sino en el modelo utilizado, esto puede dificultar las especificaciones hasta el punto de quedar incomprensibles.

En este trabajo se presenta una nueva técnica de especificación de tipos de datos, tan constructiva como las especificaciones que utilizan algún modelo fijo, pero eliminando precisamente la sujeción a ninguno. En este sentido el nuevo método comparte todas las ventajas de dichas especificaciones pero sin algunos de sus inconvenientes, puede por tanto ser considerado superior.

Con respecto a la técnica axiomática, el nuevo método puede considerarse complementario con ella, es decir las ventajas de uno son los inconvenientes de la otra y viceversa.

Todo el tratamiento que se hace en este trabajo es algebráico, de hecho, las estructuras de datos tal como estan definidas constituirían categorias de algebras heterogeneas (many-sorted), sin embargo la palabra algebra no aparece en el trabajo, el motivo es doble: en 1er lugar el tratamiento se complica a partir de la definición de los tipos estructurados de datos, éstos parecen, a primera vista, ser algebras de algebras, pero al establecer la nueva relación de equivalencia (en la 2^{a} parte del artículo a aparecer proximamente), aparece un nuevo concepto de tipo de datos no claramente modelizable en terminos algebraicos. Qué es exactamente, forma parte de la investigación en curso. En 2º lugar, una algebraización conduciría probablemente a una perdida de comprensibilidad.

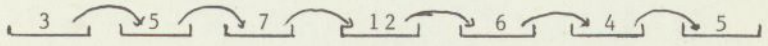
El artículo está dividido en dos partes, ésta es la primera, en ella se definen los conceptos de estructura de datos, y de tipo estructurado de datos, y aplicando el método descrito se definen como ejemplo dos tipos de datos. En la 2ª parte que aparecerá proximamente, se define un nuevo concepto de equivalencia entre tipos de datos, estableciendo también los conceptos de representación y de representación limitada.

1. Estructuras de datos

Un tipo estructurado de datos es, desde un punto de vista informal un conjunto de estructuras de datos, junto con unas operaciones (modificaciones) que nos permiten obtener unas estructuras a partir de otras. Por ejemplo, el tipo "lista de enteros", puede ser considerado como el conjunto de todas las listas de enteros sobre las que se pueden aplicar las operaciones "inserción", "supresión" y "modificación". Será pues preciso comenzar a explicar que se entiende por "estructura de datos".

Una estructura de datos viene caracterizada, por la información que "almacena" y por la forma en que se puede acceder a dicha información, en este sentido, se considera definida una estructura de datos a partir de una familia de conjuntos de objetos y de una familia de funciones de acceso sobre dichos conjuntos. Entre los conjuntos de objetos estarán los conjuntos de datos, pero a menudo existirán otros objetos, que sin ser datos, serán necesarios para definir las estructuras, serán conjuntos que aprecerán con bastante naturalidad en las definiciones, y de los que se hará abstracción, en lo posible, más adelante. Las funciones de acceso, serán funciones parciales 0-arias y 1-arias. Una función de 0 argumentos representa un acceso directo a un objeto, mientras que una función de 1 argumento representa los accesos entre distintos objetos.

Ejemplo 1: Sea la lista de enteros, representada graficamente por



Esta estructura puede ser definida como:

Objetos = {Nodos, Enteros}

Funciones de acceso = {cabeza, sucesor, valor}

Siendo Nodos = {n1, n2, n3, n4, n5, n6, n7}

Enteros = $\mathbb{Z}$

y las funciones de acceso:

- cabeza: ⟶ Nodos

cabeza = n1

(cabeza es una función de 0 argumentos que permite acceder al nodo n1, considerado cabeza de la lista)

- sucesor: Nodos ⟶ Nodos

sucesor (n1) = n2
sucesor (n2) = n3
sucesor (n3) = n4
sucesor (n4) = n5
sucesor (n5) = n6
sucesor (n6) = n7
sucesor (n7) = indefinido

- valor: Nodos ⟶ Enteros

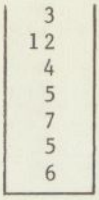
valor (n1) = 3
valor (n2) = 5
valor (n3) = 7
valor (n4) = 12
valor (n5) = 6
valor (n6) = 4
valor (n7) = 5

En esta estructura, nodos sería un conjunto de objetos que no son datos.

En general el acceso a un dato se realizará: o bien mediante la aplicación sucesiva de funciones 1-arias sobre un objeto obtenido a partir de la utilización de un acceso 0-ario, por ejemplo en la lista definida anteriormente para acceder al 4º valor habría que acceder al nodo que ocupa la cabeza (acceso 0-ario) y a continuación aplicar tres veces la función sucesor y una la función valor, es decir, se accedería a traves de valor (sucesor (sucesor (sucesor (cabeza)))). O bien mediante la aplicación sucesiva de funciones 1-arias sobre otro dato, tal sería el caso de una matriz. En este sentido, se considera que los datos son en cierta forma, directamente accesibles.

En algunas estructuras de datos, se puede considerar que hay almacenada información no accesible, por ejemplo, considerar que en una pila sólo hay guardado el valor que ocupa la cumbre, porque es el único accesible, supondría renunciar a información que de alguna forma está en la pila. Por otra parte, permitir el acceso, al resto de los valores almacenados en la pila, supondría definir una estructura que no sería una pila. Para resolver el problema se considera que en algunas estructuras existen "funciones escondidas" funciones que representan accesos que de alguna forma estan en la estructura, pero que esta 'prohibido' utilizarlos.

Ejemplo 2: Sea la pila de enteros representada graficamente por:



Esta estructura puede ser definida como:

Objetos = {Nodos, Enteros}

funciones = {cumbre, antecesor, valor}
antecesor es escondida.

Nodos = {n1, n2, n3, n4, n5, n6, n7}

Enteros = $\mathbb{Z}$

Cumbre: ——> Nodos

cumbre = n7

Antecesor: Nodos ——> Nodos

antecesor (n7) = n6
.....
antecesor (n2) = n1
antecesor (n1) = indef.

Valor: Nodos ——> Enteros

valor (n7) = 3
valor (n6) = 12
.....
valor (n1) = 6

En esta estructura se considerará entonces que el único dato accesible es 3 (accesible por medio de valor (cumbre)), ya que es el único dato que se puede obtener por composición de funciones de acceso (no escondidas).

Definición

Una estructura de datos E es un par E = (O,F), donde O es una familia de conjuntos $\{O_i\}_{i \in I}$ indexada por un conjunto I, llamados conjuntos de objetos de la estructura.Y donde F es una familia de funciones parciales, $\{f_j\}_{j \in J}$ indexadas por un conjunto J, y finitamente definidas sobre los conjuntos de Objetos. Una función n determinará para cada indice j la aridad de f_j (0 ó 1) mientras que una función a determinará para cada j el dominio y la imagen de la función f_j. Es decir

$n: J \longrightarrow \{0,1\}$

$a: \{1,2\} \times J \longrightarrow I$

siendo $f_j: \longrightarrow {}^{O}a(2,j)$ si $n(j)=0$

$f_j: {}^{O}a(1,j) \longrightarrow {}^{O}a(2,j)$ si $n(j)=1$

Además en I y en J hay dos subconjuntos distinguidos $D \subset I$ y $A \subset J$. Siendo D el conjunto de los indices de los conjuntos de datos y siendo A el conjunto de los indices de las funciones de Acceso (no escondidas).

Ejemplo 2^1: De acuerdo con esta definición, la pila del ejemplo 2 vendría formalizada como un par (O,F) siendo

$O = \{ {}^{O}_{nodos}, {}^{O}_{enteros} \}$ es decir I={nodos, enteros}

$F = \{ f_{cumbre}, f_{antecesor}, f_{valor} \}$ es decir J={cumbre, antecesor, valor }

Además:

D = {enteros}	a(2, cumbre) = nodos
A = {cumbre, valor}	a(1, valor) = nodos
n(cumbre) = 0	a(2, valor) = enteros
n(valor) = 1	a(1, antecesor) = nodos
n(antecesor) = 1	a(2, antecesor) = nodos

Hasta aquí tendríamos lo que podría ser llamado la especificación sintáctica de la estructura, que sería, en cierta forma, común a la de todas las pilas, para acabar de definir la estructura habrá que definir exactamente cuales son los conjuntos de objetos y las funciones de la estructura, es decir:

$${}^{O}_{nodos} = \{n1,....n7\}$$

$${}^{O}_{enteros} = \mathbb{Z}$$

$$f_{cumbre} \longrightarrow {}^{O}_{nodos}$$

también

$$f_{cumbre} = n7$$

etc.

Definición

Dados dos conjuntos I y J, un subconjunto de I, $D \subset I$ y dos funciones $n: J \longrightarrow \{0,1\}$

y $a: \{1,2\} \times J \longrightarrow I$

llamaremos $\mathcal{H}(J)$ al lenguaje $\mathcal{H}(J) \subset J^*$ definido como:

1) $\forall j \in J \Rightarrow j \in \mathcal{H}(J)$

2) $\forall x \in \mathcal{H}(J) \forall j \in J [x=j_1 \ldots j_n \wedge n(j)=1 \wedge a(1,j)=a(2,j_1) \Rightarrow jx \in \mathcal{H}(J)]$

3) Nada más pertenece a $\mathcal{H}(J)$

llamaremos $\mathcal{H}_D(J)=\{ x \in \mathcal{H}(J) / x=j_1 \ldots j_n \wedge a(2,j_1) \in D \wedge (n(j_n)=0 \vee \vee (n(j_n)=1 \wedge a(1,j_n) \in D))\}$

Proposición

$\mathcal{H}(J)$ y $\mathcal{H}_D(J)$ son lenguajes regulares

Demostración

Trivial. Bastaría ver que $\mathcal{H}(J)$ es reconocido por el autómata:

$$(I\ \{q_0,q_1\},J,\delta,q_0,I\ \{q_1\})$$

siendo $\forall j$: $\delta(q_0,j)=i$ si $n(j)=1 \wedge a(1,j)=i$

$\delta(q_0,j)=q_1$ si $n(j)=0$

$\forall i\ \forall j$: $\delta(i,j)=i'$ si $n(j)=1 \wedge a(1,j)=i' \wedge a(2,j)=i$

$\delta(i,j)=q_1$ si $n(j)=0 \wedge a(2,j)=i$

Y $\mathcal{H}_D(J)$ es reconocido por el autómata

$$(I\ \{q_0,q_1\},J,\ \delta,q_0,D\ \{q_1\})$$

siendo $\forall j$: $\delta(q_0,j)=i$ si $n(j)=1 \wedge a(2,j) \in D \wedge a(1,j)=i$

$\delta(q_0,j)=q_1$ si $n(j)=0 \wedge a(2,j) \in D$

$\forall i\ \forall j$: $\delta(i,j)=i'$ si $n(j)=1 \wedge a(2,j)=i \wedge a(1,j)=i'$

$\delta(i,j)=q_1$ si $n(j)=0 \wedge a(2,j)=i$

A continuación se definirá lo que se entiende por objeto accesible y objeto alcanzable. Intuitivamente un objeto será accesible si se puede obtener por medio de la composición de funciones de acceso (no escondidas), mientras que será alcanzable cuando se pueda obtener por medio de la composición de funciones de la estructura (escondidas o no).

Definición

Dada una estructura $E=(\{O_i\}_{i \in I},\{f_j\}_{j \in J})$ se dice que un objeto $o \in O_i$ para algún $i \in I$, es alcanzable(respectivamente accesible)

en E, desde un objeto $o' \in O_{i'}$, para algún $i' \in I$, a traves de un camino $X = j_1 \ldots j_n$ sii

1) $x \varepsilon \mathcal{H}(J)$ (resp. $x \varepsilon \mathcal{H}(A)$)

2) $a(2,j_1)=i \wedge n(j_n)=1 \wedge a(i,j_1)=i'$

3) $f_{j_1} \circ \ldots \circ f_{j_n} (o')=o$

Se dice que un objeto $o \varepsilon O_i$, para algun $i \varepsilon I$, es alcanzable (respectivamente accesible), a través de un camino $x = j_1 \ldots j_n$, sii:

1) $x \varepsilon \mathcal{H}(J)$ (respectivamente $x \varepsilon \mathcal{H}(A)$)

2) $a(2,j_1)=i \wedge n(j_n)=0$

3) $f_{j_1} \circ \ldots \circ f_{j_n} = o$

Ejemplo 3

En la pila de enteros de los ejemplos 2 y 2^1: el nodo n5 es alcanzable, pero no accesible, desde el nodo n7 a traves del camino antecesor antecesor.

El entero 5 es alcanzable, pero no accesible, a traves de los caminos valor antecesor antecesor antecesor cumbre y valor antecesor antecesor antecesor antecesor antecesor cumbre.

El entero 3 es accesible y alcanzable desde el nodo n7 a traves del camino valor.

El entero 3 es accesible y alcanzable a traves del camino valor cumbre.

Notación: Cuando sea necesario distinguir entre los elementos (objetos, funciones, etc....) de dos estructuras distintas, se superindexaran con el nombre de las estructuras a las que pertenezcan. Por ejemplo f_j^E denotara la función f_j de la estructura E.

Cuando dos estructuras tengan los mismos conjuntos I,J de indices se entenderá también que tienen los mismos D, A, a, n.

Definición

Dos estructuras $E = (\{O_i\}_{i\in I} \{f_j\}_{j\in J})$ y $E' = (\{O'_i\}_{i\in I} \{f'_j\}_{j\in J})$ con los mismos conjuntos de indices I y J, se dice que son equivalentes $E \equiv E'$ sii.

1) $\forall i\varepsilon I \quad \exists \phi_i : O_i^E \longrightarrow O_i^{E'}$ biyectiva y tal que si $i\varepsilon D$ entonces ϕ_i es la identidad

2) $\forall i\varepsilon D \; \forall o\varepsilon O_i \; \forall x\varepsilon \mathcal{H}(J)$, o es alcanzable en E a traves del camino x sii o es alcanzable en E' a traves del camino x.

3) $\forall i_1, i_2 \varepsilon D \; \forall o_1 \varepsilon O_{i_1} \; \forall o_2 \varepsilon O_{i_2} \; \forall x\varepsilon \mathcal{H}(J)$, se verifica que o_1 es alcanzable en E, desde o_2 a través de x si y solo si o_1 es alcanzable en E', desde o_2 a traves de x.

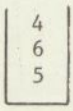
Proposición

$\equiv$ es una relación de equivalencia.

Lo que intuitivamente se dice en la definición anterior es que dos estructuras son equivalentes, si y sólo si, sus conjuntos de datos son los mismos, sus conjuntos de objetos, que no son datos tienen el mismo cardinal (esto lo exigirá la definición posterior de tipo de datos) y si en las dos estructuras se pueden alcanzar los mismos valores utilizando los mismos caminos.

Ejemplo 4

Dada la pila:



y las definiciones:

$E1 = (\{0_{enteros}, 0_{nodos}\}, \{f_{cumbre}, f_{antecesor}, f_{valor}\})$

$0_{enteros} = \mathbb{Z}$

$0_{nodos} = \{n1, n2, n3, n4, n5\}$

$f_{cumbre} = \longrightarrow 0_{nodos}$

$f_{cumbre} = n1$

$f_{antec.} = 0_{nodos} \longrightarrow 0_{nodos}$ (Escondida)

$f_{antec.}(n1) = n2,\ f_{antec.}(n2) = n3,$

$f_{antec.}(n3) = \ldots = f_{antec.}(n5) = \text{indefinida}$

$f_{valor} = 0_{nodos} \longrightarrow 0_{enteros}$

$f_{valor}(n1) = 4\ f_{valor}(n2) = 6$

$f_{valor}(n3) = 5\ f_{valor}(n4) = \text{indef.}\ f_{valor}(n5) \neq 8$

$E2 = (\{0_{enteros}, 0_{nodos}\}, \{f_{cumbre}, f_{antec.}, f_{valor}\})$

$0_{enteros} = \mathbb{Z}$

$0_{nodos} = (n1, n3, n5, n7, n9)$

$f_{cumbre} = \longrightarrow 0_{nodos}$

$f_{cumbre} = n3$

$f_{antec.} = 0_{nodos} \longrightarrow 0_{nodos}$

$f_{antec}(n3) = n1,\ f_{antec.}(n1) = n5,\ f_{antec.}(n5) = n7$

$f_{antec.}(n7) = n9,\ f_{antec.}(n9) = n7$

$f_{valor} = 0_{nodos} \longrightarrow 0_{enteros}$

$f_{valor}(n3) = 4,\ f_{valor}(n1) = 6,\ f_{valor}(n5),$

$$f_{valor}(n7) = f_{valor}(n9) = \text{indefinido}.$$

Se tiene que E1 ≡ E2 ya que los únicos datos alcanzables en las dos estructuras son 4,6 y 5.

Si definieramos una estructura E3 con sólo 3 nodos, no se consideraria equivalente a E1 ni a E2 ya que, intuitivamente, en E1 y E2 podríamos "meter" 2 nuevos datos, mientras que E3 estaria completa.

Definición

Una estructura $E=(\{O_i\}_{i\in I}, \{f_j\}_{j\in J})$ esta en forma normal sii $\forall i\in I \quad \forall o\in O_i$ si no existe $x\in H(J)$ tal que o es alcanzable a traves del camino x, ó no existen $x\in H(J)$, $i'\in D$ y $o'\in O_{i'}$, tales que o es alcanzable desde o', a través de x, ó no existen $x\in H(J)$ $i'\in D$ y $o'\in O_{i'}$, tales que o' es alcanzable desde o, a través de x. Entonces se tiene que para todo j, que verifique que a(1,j)=i, $f_j(o)$ está indefinida.

En el ejemplo anterior ni E1, ni E2 estan en forma normal, E1 estaría en forma normal si $f_{valor}(n5)$ estuviera indefinido.

2. Tipos estructurados de datos

Con el formalismo visto en el párrafo anterior se pueden definir estructuras de datos como objetos estáticos, que no pueden ser modificados, sin embargo ésto, por si sólo, tendrá poco interes; cuando uno utiliza una pila no quiere, sólo consultar sus valores, sino que querrá meter nuevos valores, o sacar alguno. En este sentido se entenderá por un tipo estructurado de datos a un conjunto de estructuras, junto con una serie de operaciones (modificaciones) entre ellas.

Para poder ver entonces qué es un tipo estructurado, habrá primero que definir qué es una modificación.

Definición

Dos estructuras $E = (\{O_i\}_{i\in I}, \{f_j\}_{j\in J})$ y

$E' = (\{O'_i\}_{i\in I}, \{f'_j\}_{j\in J})$, se dice que son similares

$E \approx E$ sii 1) $\forall i\in D \quad O_i^E = O_i^E$

2) $\forall i\in I-D \quad \text{Card}(O_i^E) = \text{Card}(O_i^{E'})$

Proposicion

$\approx$ es una relación de equivalencia

Definición

Sea $\mathcal{E}$ la clase de todas las estructuras de datos, y sea $\mathcal{E}/\approx$ el conjunto de las clases de equivalencia de $\approx$. A las clases $T \in \mathcal{E}/\approx$ se les llmará tipos generalizados de estructuras de datos. Dada una estructura E se denotará por $|E|$ al tipo generalizado al que pertenece.

Un tipo generalizado vendrá caracterizado por el par $(\{0_i\}_{i\in I}, J)$, donde $\{0_i\}_{i\in I}$ será la familia de objetos de las estructuras del tipo (salvo biyección, para los conjuntos que no sean datos) y J es el conjunto de indices de las funciones de las estructuras, como de costumbre a I y J estarán asociados A, D, a y h.

Definición

Dado un tipo generalizado $T = (\{0_i\}_{i\in I}, J)$ y una función parcial

$$M : T \times 0_{i1} x \ldots x 0_{ik} \longrightarrow T$$
$$i1, \ldots, ik \in D$$

Se dice que M es una modificación sobre T sii

1) M es computable

2) $\forall E1, E2 \in T(E1 \equiv E2 \Rightarrow \forall o_1 \in 0_{i1}, \ldots, \forall o_k \in 0_{ik}$, se verifica $M(E1, o_1, \ldots, o_k) \equiv M(E2, o_1, \ldots, o_k)$ ó ambos están indefinidos).

Definición

Dada una estructura de datos $E_o = (\{0_i\}_{i\in I}, \{f_j\}_{j\in J})$ y un conjunto indexado de modificaciones $\{M_k\}_{k\in k}$

$$M_k : |E_o| x 0_{1(k,1)} x \dots x 0_{1(k,m(k))} \longrightarrow |E_o|$$

donde 1 es una aplicación parcial

$$1 : K \times \mathbb{N} \longrightarrow D$$

y m es una aplicación

$$m : K \longrightarrow \mathbb{N}$$

Se denomina tipo estructurado de datos generado por E_o y $\{M_k\}_{k\in K}$, $\mathcal{E}(E_o, \{M_k\}_{k\in K})$ al menor suboonjunto de $|E_o|$ cerrado bajo $\{M_k\}_{k\in K}$ y que contiene a E_o . E_o será llamada estructura inicial del tipo.

Teorema

Sea $\mathcal{E}_o(E_o, \{M_k\}_{k\in K}) = \{E_o\}$, y sea para $n>o$

$$\mathcal{E}_n(E_o, \{M_k\}_{k\in K}) = \{E / \exists E' \in \mathcal{E}_{n-1}(E_o, \{M_k\}_{k\in K}) \; k\in K \; \exists o_1 \in O_{1(k,1)} \cdots \exists o_{n(k)} \in O_{1(k,n(k))} \; [E = M_k(E', o_1, \dots, o_{n(k)})]\}.$$

Entonces $\forall E \in |E_o| \; (E \in \mathcal{E}(E_o, \{M_k\}_{k\in K}) \Leftrightarrow \exists_n (E \in \mathcal{E}_n(E_o, \{M_k\}_{k\in K})))$.

Es decir $\mathcal{E}(E_o, \{M_k\}_{k\in K}) = \bigcup_{n\in\mathbb{N}} \mathcal{E}_n(E_o, \{M_k\}_{k\in K})$

Demostración

Será necesario probar que $\mathcal{E} = \bigcup_{n\in\mathbb{N}} \mathcal{E}_n(E_o, \{M_k\}_{k\in K})$ verifica:

a) Es cerrado bajo las operaciones de modificación (trivial)
b) Contiene a E_o (trivial)
c) Si $\mathcal{E}'$ es cerrado bajo las operaciones de nodificación y $E_o \in \mathcal{E}'$ entonces $\mathcal{E} \subset \mathcal{E}'$

Vamos a demostrar c) Para ello se demostrará por inducción, que para todo n:

$$\mathcal{C}_n(E_o\ \{M_k\}_{k\in K}) \subset \mathcal{C}'$$

1) $\mathcal{C}_o(E_o, \{M_k\}_{k\in K}) \subset \mathcal{C}'$ pues $\mathcal{C}_o = \{E_o\}$

2) Supuesto que $\mathcal{C}_n \subset \mathcal{C}'$ vamos a ver que $\mathcal{C}_{n+1} \subset \mathcal{C}'$

$E \in \mathcal{C}_{n+1} \Rightarrow \exists k\in K\ \exists o_1 \in O_{1(k,1)} \cdots \exists o_{n(k)} \in O_{1(k,n(k))}\ \exists E' \in \mathcal{C}_n (E = M_k(E', o_1, \ldots, o_{n(k)})) \Rightarrow E \in \mathcal{C}'$

por hipotesis de inducción y por ser $\mathcal{C}'$ cerrado bajo $\{M_k\}_{k\in K}$.

Es decir se ha definido un tipo estructurado de datos como el conjunto de estructuras de datos, que se pueden obtener por modificaciones sucesivas de una estructura inicial.

Intuitivamente esta estructura inicial, sería la que se obtendría al realizar la declaración del tipo, es decir, ante la declaración:

var v: $\mathcal{C}$ se supondría que v tomaría como valor el de la estructura inicial de $\mathcal{C}$. En general esta estructura, será una estructura con casi todas sus funciones indefinidas.

Una especificación formal de un tipo estructurado de datos consistirá pues en 1) Definir la estructura inicial del tipo y 2) Definir las modificaciones. En seguida se verán algunos ejemños de especificaciones formales, pero antes se va a definir la equivalencia de tipos estructurados.

Definición

Dado un tipo generalizado $T = (\{O_i\}_{i\in J}, J)$ y dos modificaciones sobre el tipo

$$M_1 : T \times O_{i1} \times \ldots \times O_{im} \longrightarrow T$$

$$M_2 : T \times O_{i1} \times \ldots \times O_{im} \longrightarrow T$$

son equivalentes $M_1 \equiv M_2 \Leftrightarrow \forall E \in T\ (M_1(E) \equiv M_2(E))$

Proposición

$\equiv$ es una relación de equivalencia.

NOTA: Aunque usar el mismo simbolo $\equiv$ para la equivalencia de Estructuras, modificaciones y, a continuación, de tipos estructurados, sea un abuso de notación, por el contexto en que aparezca el simbolo se deducirá de qué equivalencia se trata.

Notación: Cuando sea necesario distinguir entre los elementos (estructuras, modificaciones, etc.) de varios tipos estructurados, se superindexaran con el nombre del tipo.

Definición

Dados dos tipos estructurados $\mathcal{E}=(E_0,\{M_k\}_{k\in K})$ y $\mathcal{E}'=(E_0',\{M_k'\}_{k\in K})$ se dice que $\mathcal{E}$ es un subtipo de $\mathcal{E}'$, $\mathcal{E}\leq\mathcal{E}'$ sii:

1) $\forall k\in K(\ M_k^{\mathcal{E}}\equiv M_k^{\mathcal{E}'}\ )$

2) $\forall E\in\mathcal{E}\ \exists E'\in\mathcal{E}'(E\equiv E')$

y se dice que son equivalentes, $\mathcal{E}\equiv\mathcal{E}'$ sii $\mathcal{E}\leq\mathcal{E}'\wedge\mathcal{E}'\leq\mathcal{E}$

Proposición

$\equiv$ es una relación de equivalencia.

Teorema

Sea $\mathcal{E}=(E_0,\ \{M_k\}_{k\in K})\ \wedge\ \mathcal{E}'=(E_0',\{M_k'\}_{k\in K})$

$\mathcal{E}\leq\mathcal{E}'\ \Leftrightarrow$ 1) $\forall k\in K(M_k^{\mathcal{E}}\equiv M_k^{\mathcal{E}'})$

2) $\exists E'\in\mathcal{E}'(E_0\equiv E')$

Demostración

$\Rightarrow$) Trivial

$\Leftarrow$) Por una parte es evidente que

$$\forall n\in\mathbb{N}\ \forall E_1^{\mathcal{E}}\in\mathcal{E}_n(E_0,\{M_k^{\mathcal{E}}\}_{k\in K})\ \exists E_2\in\mathcal{E}_n(E',\{M_k^{\mathcal{E}'}\}_{k\in K})(E_1^{\mathcal{E}}\equiv E_2^{\mathcal{E}'})$$

Por otra parte es claro que

$$\forall n\in\mathbb{N}(\mathcal{E}_n(E',\{M_k^{\mathcal{E}}\}_{k\in K})\subseteq\mathcal{E}')$$

De ahí se deduciría facilmente que

$$\mathcal{E} = \bigcup_{n \in \mathbb{N}} \mathcal{E}_n (E_0, \{M_k^{\mathcal{E}}\}_{k \in K}) \leq \mathcal{E}'$$

Corolario

Sean $\mathcal{E} = (E_0, \{M_k\}_{k \in K})$ y $\mathcal{E}' = (E_0', \{M_k'\}_{k \in K})$, tales que $\forall k \in K$, se verifica que: $M_k^{\mathcal{E}} \equiv M_k^{\mathcal{E}'}$. Entonces:

$$E_0 \equiv E_0' \Rightarrow \mathcal{E} \equiv \mathcal{E}'$$

Ejemplo 5

Vamos a definir formalmente lo que es el tipo lista de enteros .

La estructura inicial del tipo L_o será la lista vacia que se definirá como:

$L_o = (\{0_{nodos}, 0_{enteros}, 0_{Bool}\}, \{f_{cabeza}, f_{sucesor}, f_{valor}, f_{vacia}, f_{lugar}\}$

$D = \{enteros, Bool\}$

$A = \{cabeza, sucesor, valor, vacia\}$

$0_{nodos} = \{ n_o, n_1, \ldots, n_m \ldots\}$

$0_{enteros} = \mathbb{Z}$

$0_{Bool} = \{verdadero, falso\}$

$f_{cabeza} = \longrightarrow 0_{nodos}$

$f_{cabeza} = indef$ (en la lista vacia no hay cabeza)

$f_{sucesor} = 0_{nodos} \longrightarrow 0_{nodos}$

$f_{sucesor}(n) = indef. \ \forall n \in 0_{nodos}$

$f_{valor} = 0_{nodos} \longrightarrow 0_{enteros}$

$f_{valor}(n) = indef. \ \forall n \in 0_{nodos}$

$f_{vacia} = \longrightarrow 0_{Bool}$

$f_{vacia} = verdadero$

$f_{lugar} = 0_{enteros} \longrightarrow 0_{nodos}$

$f_{lugar}(a) = indef. \ \forall a \in \mathbb{N}$

Con esto estaría definida la lista inicial. Para acabar de

definir el tipo lista de enteros, habrá que definir las modificaciones permitidas sobre el tipo generalizado de la lista inicial.

Sean las siguientes modificaciones:

$M_{act}: |L_0| \times 0_{enteros} \times 0_{enteros} \longrightarrow |L_0|$

$M_{act}(L,a,b)$ será la lista obtenida a partir de la lista L, haciendo que el nodo que ocupa el lugar a pase a valer b, en vez de su antiguo valor. Es decir:

$M_{act}(L,a,b) = indef.$ si $f^L_{valor}(f^L_{lugar}(a)) = indef.$

Sino la nueva estructura obtenida $L'=M_{act}(L,a,b)$, será aquella en la que las funciones esten definidas como:

$$f^{L'}_{cabeza} = f^L_{cabeza}$$

$$f^{L'}_{sucesor}(n) = f^L_{sucesor}(n) \qquad \forall n \in 0_{nodos}$$

$$f^{L'}_{vacia} = f^L_{vacia}$$

$$f^{L'}_{lugar}(c) = f^L_{lugar}(n) \qquad \forall c \in 0_{enteros}$$

$$f^{L'}_{valor}(n) = \begin{cases} f^L_{valor}(n) & \text{si } f^L_{lugar}(a) \neq n \\ b & \text{si } f^L_{lugar}(a) = n \end{cases}$$

$M_{meter}: |L_o| \times 0_{enteros} \times 0_{enteros} \longrightarrow |L_o|$. $M_{meter}(L, a, b)$ será la lista resultante de insertar en la posición a de la lista L el valor b.

$M_{meter}(L, a, b) = indef.$ si $a<0$ ó $(a>0$ y $f_{lugar}(a-1)=indef)$

en otro caso la lista resultante L', será aquella cuyas funciones sean:

Sea n_1 un nodo tal que $f^{L}_{valor}(n_1)$ = indef. y $f^{L}_{lugar}(a-1) \neq n_1$

$$f^{L'}_{cabeza} = \begin{cases} n_1 & \text{si} \quad a=o \\ f^{L}_{cabeza} & \text{si} \quad a>o \end{cases}$$

$$f^{L'}_{sucesor}(n) = \begin{cases} f^{L}_{sucesor}(n) & \text{si} \quad n \neq f^{L}_{lugar}(a-1) \quad \text{y} \quad n \neq n_1 \\ n_1 & \text{si} \quad n = f^{L}_{lugar}(a-1) \\ f^{L}_{sucesor}\, f^{L}_{lugar}(a-1) & \text{si} \quad n = n_1 \end{cases}$$

$$f^{L'}_{vacia} = \text{falso}$$

$$f^{L'}_{lugar}(c) = \begin{cases} f^{L}_{valor}(n) & \text{si} \quad c<a \\ n_1 & \text{si} \quad c=a \\ f^{L}_{lugar}(c+1) & \text{si} \quad c>a \end{cases}$$

$$f^{L'}_{valor}(n) = \begin{cases} f^{L}_{valor}(n) & \text{si} \quad n \neq n_1 \\ b & \text{si} \quad n = n_1 \end{cases}$$

Analogamente se podría definir

$$M_{supr} \quad |L_o| \times \mathbb{Z} \longrightarrow |L_o|$$

siendo $M_{supr}(L,a)$ la lista obtenida después de suprimir en la lista L el nodo de lugar a.

El tipo de lista de enteros sería entonces:

$$\mathcal{E}(L_o, \{M_{act}, M_{meter}, M_{supr}\})$$

Ejemplo 6

Vamos ahora a definir el tipo: array [1..n] de enteros

La estructura inicial sería:

$$A_o = (\{0_{enteros}\} \{f_{valor}\})$$

$$0_{enteros} = \mathbb{Z}$$

$$f_{valor} = 0_{enteros} \longrightarrow 0_{enteros}$$

$$f_{valor}(a) = \text{indef.} \qquad \forall a \in 0_{enteros}$$

Se definirá una única modificación sobre $|A_o|$

$$M_{:=} : |A_o| \times 0_{enteros} \times 0_{enteros} \longrightarrow |A_o|$$

La estructura $M_{:=}$ (A, a, b) sería la que intuitivamente, se obtendría después de realizar a operación A[a]:=b, es decir:

$$M_{:=}(A, a, b) = \text{indef.} \quad \text{si} \quad a>n$$

En otro caso será la estructura cuya función de acceso esta definida como:

$$f_{valor}^{M_{:=}(A,a,b)}(c) = \begin{cases} f_{valor}^{A}(c) & \text{si } c \neq a \\ b & \text{si } c = a \end{cases}$$

El tipo estructurado array [1..n] de enteros se definiria entonces como $\mathcal{E}(A_o, \{M_{:=}\})$

Resumiendo, para dar la especificación formal de un tipo estructurado de datos, habrá que:

1.- Definir la estructura inicial.
2.- Definir cada una de las modificaciones, especificando;
   a) Cuando estan indefinidas.
   b) En caso de no estar indefinidas, como son las funciones de la estructura modificada, a partir de las funciones de la estructura antes de modificar.