XIV Conferencia Latinoamericana de Informática
17avas. Jornadas Argentinas de Informática
e Investigación Operativa
Buenos Aires, Setiembre 1988.

Una representación no tradicional de producto cartesiano

Laura Almada - Gustavo Betarte - Paula Severi

Instituto de Computación
Universidad de la República
Montevideo - URUGUAY

Resumen

Se describe una representación general de producto cartesiano de componentes elementales. Se utiliza como herramienta de especificación a la especificación algebraica de tipos abstractos de datos (E.A.T.A.D.) [Gutagg 78].

1. Introducción

Este trabajo presenta parte de un proyecto desarrollado por el Area de Métodos y Lenguajes, Departamento de Programación, Instituto de Computación, Universidad de la República.

El proyecto surge como una iniciativa de un grupo de docentes de dicha Area, con el objetivo de experimentar nuevas técnicas de diseño, no aplicadas hasta el momento en el mercado uruguayo. Se trata del diseño de un sistema administrativo, a partir de una noción intuitiva de lo que cubre un sistema con estas características, prescindiendo de los detalles particulares de cada empresa. La elección de este tema se debe a que la mayor parte de los sistemas desarrollados en el medio nacional son sistemas administrativos.

Este trabajo se limita a presentar un modelo para los objetos de estructura finita del sistema administrativo.

En la segunda sección, se hace una primera aproximación a la representación general de los objetos del sistema. En la tercera sección se muestra el proceso a través del cual se llega a una representación de los mismos. En la cuarta sección se presentan conclusiones sobre el trabajo. En el apéndice A se presenta la representación usando especificación algebraica. En el apéndice B se muestra como se resuelven posibles requerimientos del usuario. En el apéndice C se ilustran diferencias entre la representación propuesta y la clásica a través de ejemplos.

2. Características generales de los objetos del sistema

Un sistema admistrativo trabaja con un conjunto muy amplio de objetos (cuentas corrientes, empleados, etc.). Sin embargo, si se los analiza con un nivel de abstracción suficientemente elevado, se puede llegar a la conclusión de que todos ellos pueden ser modelados mediante productos cartesianos de componentes elementales. En este trabajo se presenta una representación general del producto cartesiano, de forma tal de uniformizar la estructura de los objetos de un sistema administrativo, a los que se denomina de aquí en más S_objetos.

Como la estructura de todos los S_objetos es la misma (producto cartesiano de componentes elementales), pueden ser manipulados con un conjunto de operaciones comunes a todos ellos. Sin embargo, sería útil que los S_objetos fueran divididos en clases, de acuerdo a las propiedades que cumplen. A un S_objeto perteneciente a una clase se le pueden aplicar además de las operaciones generales, comunes a todos los S_objetos, operaciones particulares de su clase.

Las operaciones que manipulan a los S_objetos deben ser:

- generales - su semántica es común a todos los S_objetos.
- elementales - lo suficientemente sencillas de forma de que no haya superposición de funciones. O sea, deben ser disjuntas.
- fácilmente componibles - las operaciones sobre una clase pueden, en general, ser expresadas como composición de estas operaciones.

3. Fundamentos de la representación de los S objetos

El objetivo de esta sección es definir los S_objetos y como se dijo en la sección anterior éstos no son más que productos cartesianos de componentes elementales, es decir conjuntos de "campos" o atributos.

Estructura e instanciación de atributos

Un atributo puede ser descripto como producto cartesiano de los siguientes conjuntos:

.nombre
.tipo
.largo
.valor

Puede por tanto ser representado por el siguiente Tipo Abstracto de Datos (TAD):

TAD ELEM

creo-elem: ELEM-NOM X ELEM-TIPO X LARGO X VALOR ----> ELEM

Por ejemplo , el atributo Precio de un S_objeto cualquiera podría ser descripto con una cuadrupla de este estilo :

(Precio , Real , 5 , 124.7)

Se considera la componente largo de estas cuadruplas con el objetivo de conocer la cantidad máxima de caracteres que puede tomar el valor de ese atributo. El resto de las componentes tiene un significado bastante claro.

El TAD anterior representa una instancia de un atributo ya que tiene valor asociado. Para definir la estructura de un atributo (sin instanciarlo) alcanzan los tres primeros conjuntos. Se puede formular la definición de la misma con el siguiente TAD:

TAD INFO

creo-info: INFO-NOM X INFO-TIPO X LARGO ----> INFO

En el ejemplo anterior la estructura sería:

(Precio , Real , 5)

Estructura e instanciación de S_objetos

Los tads anteriores definen la estructura e instanciaciones de cualquier atributo de un S_objeto. La estructura de un S_objeto (como conjunto cuyos elementos son estructuras de atributos) puede ser descripto mediante el siguiente TAD:

TAD OBJ-ESTRUC

creo-obj-estruc: LISTA(INFO) X CANT ----> OBJ-ESTRUC

Pero este tipo abstracto presenta una carencia: no es posible distinguir estructuras de clases de S_objetos, es decir esta estructura no tiene nombre. Por tanto, se reformula el tipo anterior como sigue:

TAD OBJ-ESTRUC

creo-obj-estruc: NOM-TIPO X LISTA(INFO) X CANT ---> OBJ-ESTRUC

Este nuevo TAD define la estructura común por medio de la cual se representa cualquier S_objeto, con la siguiente información:

- Un nombre para cada estructura de S_objeto (NOM-TIPO).
- Una definición de la estructura del S_objeto como conjunto de campos (LISTA(INFO)).
- Una componente que representa la cantidad de campos que contiene el S_objeto (CANT).

Los ejemplos siguientes, aunque simplificados muestran la estructura de dos posibles S_objetos.

(a) (CUENTA CORRIENTE, (Numero, entero, 6) (Nombre, lista(char), 30) (Descrip, lista(char), 50) (Saldo, real, 20) , 5)

(b) (EMPLEADO, (Nombre, lista(char), 30) (Seccion, lista(char), 10) (Direccion, lista(char), 40) (Sueldo, real, 20) , 4)

Analizando el tipo abstracto OBJ-ESTRUC y los ejemplos vistos, es posible extraer las siguientes conclusiones:

- La definición de OBJ-ESTRUC es general, en el sentido de que con un único tipo abstracto es posible definir la estructura de cualquier S_objeto. Se puede notar en toda su amplitud esta generalidad si se la compara con la representación tradicional de producto cartesiano. En esta última, para definir una cuenta corriente y un empleado sería necesario definir dos tipos distintos:

TAD CUENTA-CORRIENTE

creo-cta-corr: ENTERO X STRING X STRING X REAL ----> CUENTA-CORRIENTE

TAD EMPLEADO

creo-empleado: STRING X STRING X STRING X REAL ---> EMPLEADO

Con lo cual se necesitaría operaciones distintas para manipular cada uno de los tipos definidos, perdiendo la generalidad deseada.

- Dos elementos distintos del tipo OBJ-ESTRUC definen dos subclases distintas. Entonces, dos S_objetos pertenecen a la misma subclase si su estructura se corresponde con el mismo elemento de OBJ-ESTRUC. Por tanto se ha definido la relación de equivalencia que divide a los S_objetos en subclases. Representar S_objetos distintos de una misma subclase es equivalente a definir instancias diferentes de la subclase, esto es, darle valor a los atributos de la estructura.

Un S_objeto es una instanciación de un elemento de OBJ-ESTRUC. Por tanto, se define el TAD objeto universal como sigue:

TAD OBJ-UNI

creo-obj-uni : NOM-TIPO X LISTA (ELEM) X CANT ----> OBJ-UNI.

En el ejemplo:

```
                    (Numero, entero, 6, 102430)
        (I)         (Nombre, lista(char), 30, "INCO")
(CUENTA CORRIENTE,  (Descrip, lista(char), 50, "Fac.Ing.") , 5 )
                    (Saldo, real, 20, 2340000.00)

                    (Numero, entero, 6, 102431)
        (II)        (Nombre, lista(char), 30, "IME")
(CUENTA CORRIENTE,  (Descrip, lista(char), 50, "Fac.Ing.") , 5 )
                    (Saldo, real, 20, 140000.00)
```

Operaciones sobre los S_objetos

Hasta ahora se ha centrado la discusión en la definición de los S_objetos. A continuación, se hace un análisis de las ventajas que se obtienen en la manipulacion de distintos S_objetos con un mismo conjunto de operaciones.

En primer lugar, como todos los S_objetos que se manipulan son instancias diferentes del mismo tipo, se necesita una única operación constructora.

El conjunto de operaciones selectoras (una para cada campo), típicas del producto cartesiano tradicional, se transforman en una única operación de selección, común a todos los S_objetos, independientemente de la subclase a la que pertenecen.

Con la representación propuesta es posible definir operaciones de selección más sofisticadas que las clásicas, como por ejemplo la selección de los tipos de los S_objetos y de los atributos correspondientes.

En función de estas operaciones (constructoras y selectoras de componentes) es posible construir nuevas operaciones generales sobre los S_objetos (por ejemplo actualizar valores de componentes), que le agregan poder de manipulación al tipo abstracto.

Subclases, estructuras e instanciaciones

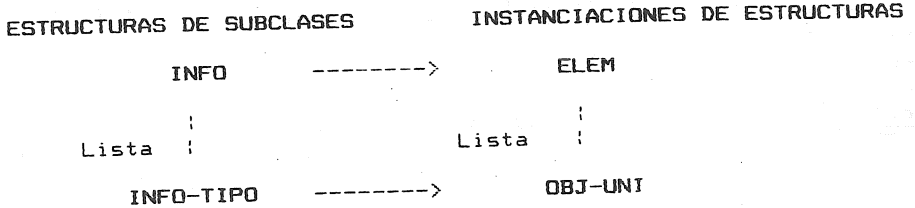
El problema original era representar todos los S_objetos con una estructura común, o sea definir una sola clase que los contenga. Esta clase tiene asociada operaciones propias que pueden ser aplicadas a todos los S_objetos. Pero además de éstas, existen operaciones que van a ser particulares de algún subconjunto de los objetos, por ejemplo habrán operaciones que interesará aplicar sobre objetos de tipo CUENTA CORRIENTE que no serán las mismas que se aplicarán sobre los objetos de tipo EMPLEADO. Cada subconjunto con operaciones propias asociadas es una subclase, y su estructura podrá ser definida como una instancia del TAD OBJ-ESTRUC.

Entonces, un S_objeto cualquiera puede verse como una instancia del objeto universal (TAD OBJ-UNI) y se construye asociándole valores al conjunto de atributos correspondientes a la subclase a la cual pertenece. Por lo tanto, existe una correspondencia entre los elementos de los tipos abstractos que definen las estructuras de los S_objetos y los elementos de los tipos que definen instancias de los mismos.

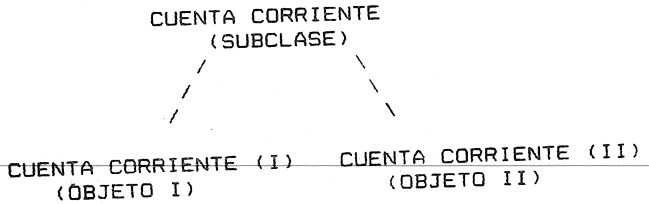
Una primera correspondencia se establece entre los elementos del TAD INFO (estructura de atributos) y los del TAD ELEM (instancias de los mismos).

Una segunda correspondencia se establece entre los elementos que define el TAD OBJ-ESTRUC (estructuras de subclases) y los que define el TAD OBJ-UNI (instancias de los anteriores).

Dichas correspondencias pueden ser esquematizadas como sigue:



Por ejemplo, la estructura de la subclase CUENTA CORRIENTE definida mediante OBJ-ESTRUC en el ejemplo (a) se corresponde con varias instancias de ésta como se vio en los ejemplos (I) y (II). Esto se puede ver con el siguiente esquema:



Una estructura para las estructuras

Las definiciones dadas no son suficientes para construir un S_objeto perteneciente a una subclase dado el nombre de la misma. Es necesario disponer de un conjunto de elementos de tipo OBJ-ESTRUC. La representación de este conjunto es una lista.

Con esto aún no es suficiente. Es necesario disponer de alguna operación que permita efectivamente crear un S_objeto. Esta operación, recibe el nombre de una subclase, selecciona de la lista de elementos de tipo OBJ-ESTRUC aquel cuyo nombre coincida con el dado, y aplica la operación cr-obj-uni a la componente del elemento de OBJ-ESTRUC hallado.

Evidentemente, esto no podría haber sido resuelto si no se tuviera una lista de todas las subclases. Esta última operación no es igual al resto ya que puede ser aplicada sin conocer la estructura del objeto universal.

Se debe hacer notar que se ha logrado la independencia deseada relativa a los tipos que se definen ya que no es necesario definir tipos distintos para cada caso particular como en el producto cartesiano clásico. Para ello, se pasan los tipos al dominio de los valores y se le asocian operaciones a éstos.

Se podría generalizar aún más y decir que es posible definir un conjunto de operaciones sobre el objeto universal, escondiendo en su aplicación no sólo la estructura del objeto universal sino también la existencia del resto de las estructuras como OBJ-ESTRUC. A este conjunto de operaciones que definen un lenguaje de distinto nivel de abstracción que el del objeto universal, lo llamamos lenguaje del usuario.

4. Conclusiones

Se ha presentado una representación general de producto cartesiano de componentes elementales. Esta representación, única para todos aquellos objetos que puedan ser modelados por medio de producto cartesiano, permite que se definan operaciones comunes a todos ellos.

Los dominios de las componentes del producto cartesiano pasan a ser variables, y por tanto también se hace variable el nombre que identifica al conjunto de estos dominios (su tipo). Esto hace que el nombre del tipo sea tratado como una componente más del producto cartesiano.

La definición propuesta permite la creación de nuevos tipos mediante operaciones. Para ello es necesario poder manipular el conjunto de definiciones de las subclases. Este conjunto puede ser visto como generador de los S_objetos (elementos del espacio del sistema).

Una instancia posterior a este trabajo sería definir un conjunto de S_objetos, y operaciones que lo manipulen.

Bibliografía

1. Meyer, Bertrand, "Reusability: The case for Object-Oriented Design", IEEE. 1987.

2. Stroustrup, Bjarne, "What is "Object-Oriented Programming"?, BIGRE (INRIA). 1987.

3. Liskov, Barbara y Guttag, John, "Abstraction and specification in program development", MIT PRESS. 1986

4. Cardelli, Luca y Wegner, Peter, "On understanding types data abstraction and polimorphism", ACM COMPUTING SURVEYS. 1985

Agradecimientos

Agradecemos especialmente la colaboración de las siguientes personas cuyos aportes contribuyeron a la elaboración de este trabajo:

- Msc. Juan José Cabezas
- Ing. Alfredo Viola
- Prof. Alvaro Tasistro
- Ing. Hermann Stephen
- Ing. Patricia Peratto

Y a todos los compañeros del Departamento de Programación, por el continuo apoyo recibido.

APENDICE A

Este apéndice presenta la especificación algebraica de los tipos abstractos utilizados para la definición de los S_objetos.

Representación de la estructura de un atributo

TAD INFO.

Cr-info: INFO-TIPO X INFO-NOM X LARGO ----> INFO.

Info-tipo: INFO ----> INFO-TIPO.
Info-nom: INFO ----> INFO-NOM.
Info-largo: INFO ----> LARGO.

Representación de la instanciación de un atributo

TAD ELEM.

Cr-elem: ELEM-TIPO X ELEM-NOM X LARGO X VALOR ----> ELEM.

Elem-tipo: ELEM -----> ELEM-TIPO.
Elem-nom : ELEM -----> ELEM-NOM.
Elem-largo: ELEM -----> LARGO.
Elem-val : ELEM -----> VALOR.

Representación de la estructura de un S_objeto

TAD OBJ-ESTRUC.

Cr-obj-estruc: NOM-TIPO X LISTA(INFO) X CANT ----> OBJ-ESTRUC.

Obj-estruc-nom: OBJ-ESTRUC ----> NOM-TIPO.
Obj-estruc-list: OBJ-ESTRUC ----> LISTA(INFO).
Obj-estruc-cant: OBJ-ESTRUC ----> CANT.

Representación de la instanciación de un S_objeto

TAD OBJ-UNI.

Cr-obj-uni: NOM-TIPO X LISTA(ELEM) X CANT ----> OBJ-UNI.

Obj-uni-nom: OBJ-UNI ----> NOM-TIPO.
Obj-uni-list: OBJ-UNI ----> LISTA(ELEM).
Obj-uni-cant: OBJ-UNI ----> CANT.

Las operaciones de todos los tads definidos son selectoras, por tanto su especificación se omite.

Tad's primitivos o predefinidos

- LISTA(T).
- INTEGER.
- CHAR.
- BOOLEAN.
- REAL.

Otros Tad's

Como se planteaba en la tercera sección, para poder representar y manipular el conjunto de subclases que definen los S_objetos, se define un TAD LISTA(OBJ-ESTRUC), para lo que se utiliza el TAD primitivo LISTA(T).

APENDICE B

Especificación de los requerimientos del usuario

En este apéndice se resuelven algunos de los posibles requerimientos del usuario sobre los S_objetos, trabajando con los TADs previamente definidos.

Es importante señalar, que la comunicación del usuario con el sistema, es resuelta por un módulo de interfase, pero dicho tratamiento escapa a los objetivos de este trabajo.

Primero se describen los requerimientos en lenguaje natural, y luego se especifican algebraicamente.

Descripción en lenguaje natural de los requerimientos

1- Crear un objeto de una subclase.
Dado un nombre de subclase, se obtiene un S_objeto perteneciente a la misma. El valor de la cuarta componente es nulo.

2- Asignar valor a un atributo de un S objeto.
Dado un S_objeto, el nombre de un atributo y un valor, se obtiene el S_objeto con el valor de ese atributo modificado.

3- Seleccionar un atributo de un S objeto.
Dado un S_objeto y un nombre de atributo, se obtiene toda la información asociada al mismo.

4- Seleccionar información de todos los atributos de un S objeto.
Dado un S_objeto se obtiene la lista de atributos del mismo.

5- Seleccionar el tipo de un atributo de un S objeto.
Dado un S_objeto y el nombre de un atributo, se obtiene el tipo asociado al atributo.

Especificación algebraica de los requerimientos

1 - Crear un objeto de una subclase.

Sintaxis

```
Dev-obj : NOM-TIPO ------> OBJ-UNI

Operaciones auxiliares

Crea-obj : OBJ-ESTRUC  ----> OBJ-UNI
Asig-val : LISTA(INFO) ----> LISTA(ELEM)
Subclase : LISTA(OBJ-ESTRUC) X NOM-TIPO ----> OBJ-ESTRUC
```

Semántica

```
Dev-obj(n)=Crea-obj(Subclase(l,n))

Crea-obj(l) =
  Cr-obj-uni(Obj-estruc-nom(l),Asig-val(Obj-estruc-list(l)),
                                        Obj-estruc-cant(l))

Asig-val(Cr-lista()) = Cr-lista()
Asig-val(Add(l,i)) =
     Add(Asig-val(l),Cr-elem(Info-tipo(i),Info-nom(i),
                                        Info-largo(i),"vn")

Subclase(Cr-lista(),n) = indef
Subclase(Add(l,i),n)   = if Obj-estruc-nom(i) = n then i
                         else  Subclase(l,n)
```

2 - Asignar valor a un atributo de un S objeto.

Sintaxis

```
Ins-val: OBJ-UNI X ELEM-NOM X VALOR ----> OBJ-UNI

Operacion auxiliar

Cambiar-Valor: LISTA(ELEM) X ELEM-NOM X VALOR ---->
                                              LISTA(ELEM)
```

Semántica

```
Ins-val(o,n,v) =
    Cr-obj-uni(Obj-uni-nom(o),
        Cambiar-valor(Obj-uni-list(o),n,v), Obj-uni-cant(o))
```

```
Cambiar-valor(Cr-lista(),n,v) = indef
Cambiar-valor(Add(l,e),n,v) =
            if Elem-nom(e) = n then
                Add(l,Cr-elem(Elem-tipo(e),n,Elem-largo(e),v)
            else
                Add(Cambiar-valor(l,n,v),e)
```

3 - Seleccionar un atributo de un S objeto.

Sintaxis

Obt-elem: OBJ-UNI X ELEM-NOM ----> ELEM

Operación auxiliar.

Get-elem: LISTA(ELEM) X ELEM-NOM ---->ELEM.

Semántica

```
Obt-elem(o,ne) = Get-elem(Obj-uni-list(o),ne)

Get-elem(Cr-lista(),n) = indef
Get-elem(Add(l,e),n)   = if Elem-nom(e) = n then e
                         else  Get-elem(l,n)
```

4 - Seleccionar información de todos los atributos de un S objeto.

Sintaxis

Obj-uni-list: OBJ-UNI ----> LISTA(ELEM)

Semántica

```
Obj-uni-list(Cr-obj-uni(n,l(e),c) = l(e)
```

5 - Seleccionar el tipo de un atributo de un S objeto.

Sintaxis

Tip-atr: OBJ-UNI X ELEM-NOM ----> ELEM-TIPO

Semántica

```
Tip-atr(o,ne)  =  Elem-tipo(Obt-elem(o,ne))
```

APENDICE C

Este apéndice ilustra a través de ejemplos el alcance que tienen las operaciones definidas sobre las estructuras propuestas.

Ejemplo1.

Dados dos S_objetos, en caso de que sean del mismo tipo devolver una lista de reales donde cada real sea la suma de las componentes reales respectivas de cada S_objeto.

En este ejemplo, se muestra la posibilidad de manejar los tipos de los objetos y sus componentes como variables.

Sintaxis

```
Suma-dos-objetos: OBJ-UNI X OBJ-UNI ----> LISTA(REAL)

Operaciones auxiliares

Selec-valor: LISTA(ELEM) X ELEM-TIPO ----> LISTA(VALOR)
Suma-lista: LISTA(VALOR) X LISTA(VALOR) ----> LISTA(VALOR)
```

Semántica

```
Suma-dos-objetos(Cr-obj-uni(n,l,c), Cr-obj-uni(n',l',c')) =
    if  n = n' then
       Suma-lista(Selec-valor(l,REAL), Selec-valor(l',REAL))
    else
       indefinido
    endif

Selec-valor(Cr-lista(), n) = Cr-lista()
Selec-valor(Add(l,e), n) =
                  if Tipo-elem(e) = n then
                        Add(Selec-valor(l,n), Elem-val(e))
                  else
                        Selec-valor(l,n)
                  endif

Suma-lista(Cr-lista(), Cr-lista()) = Cr-lista()
Suma-lista(Cr-lista(), Add(l,e)) = Add(l,e)
Suma-lista(Add(l,e), Cr-lista()) = Add(l,e)
Suma-lista(Add(l,e), Add(l',e')) =
                     Add(Suma-lista(l,l'),Suma-nros(e,e'))
```

Donde Suma-nros es una operación que suma 2 números (reales o enteros).

Notar que esta operación se ha definido sobre cualquier par de S_objeto independientemente de la clase a la que pertenezcan. Esto es posible ya que existe una estructura común para todos los S_objetos.

Si se definieran los S_objetos como en el producto cartesiano clásico, no existiría una representación única para todos los S_objetos, y sería necesario definir operaciones distintas para cada clase, aunque la semántica de ellas sea idéntica excepto por los tipos involucrados.

La distinción de tipos es necesaria en ambos casos. Cuando se define una estructura común alcanza con comparar los tipos ya que es una componente más. Cuando se usa el producto cartesiano clásico la distinción de tipos está implícita en su estructura por lo que es necesario definir una operación distinta para cada tipo.

Por otra parte, en el primer caso el tipo de los atributos de un S_objeto es variable, por lo que se puede realizar una operación que dado un S_objeto y un tipo de atributo seleccione todos los atributos que sean de ese tipo. La estructura definida para los S_objetos permite definir operaciones generales para las clases de los S_objetos y sus atributos. Esto no es posible en el producto cartesiano clásico.

Ejemplo 2.

Dado un elemento de OBJ-ESTRUC y una lista de INFO-NOM obtener un nuevo elemento de OBJ-ESTRUC que esté compuesto únicamente por aquellos atributos cuyos nombres se corresponden con los de la lista.

Sintaxis

Proyectar: OBJ-ESTRUC X LISTA[INFO-NOM] X NOM-TIPO ---> OBJ-ESTRUC

Operaciones auxiliares

Cr-lista-proy: LISTA[INFO] X LISTA[INFO-NOM] ----> LISTA[INFO]

Semántica

```
Proyectar(Cr-obj-estruc(n,l,c), l', n') =
    Cr-obj-estruc(n',Cr-lista-proy(l,l'),
                                Cantidad(Cr-lista-proy(l,l')))

Cr-lista-proy(Cr-lista(), Cr-lista()) = Cr-lista()
Cr-lista-proy(Cr-lista(), Add(l,e)) = Cr-lista()
Cr-lista-proy(Add(l,e), Cr-lista()) = Cr-lista()
Cr-lista-proy(Add(l,e), Add(l',e')) =
    if Pert-nom(Add(l,e), e') then
       Add(Cr-lista-proy(Add(l,e),l'), Obt-nom(Add(l,e),e')
    else
       Cr-lista-proy(Add(l,e), l')
    endif
```

Pert-nom es una función que dados una lista[info] y un nombre de atributo verifica si éste pertenece a la misma. Obt-nom es una función que dados un lista[info] y un nombre de atributo, selecciona el atributo de la lista correspondiente al mismo.

En este caso, se construye una nueva subclase por medio de una operación. Notar que se esta definiendo una forma de construir nuevos tipos a partir de otros dados.

En el producto cartesiano clásico no solo es necesario conocer las componentes que definen al tipo que se quiere proyectar sino que además debe realizarse una nueva especificación en la que se define la estructura del tipo obtenido de la proyección.

Ejemplo:
Si a partir del tipo CUENTA CORRIENTE (ver ejemplo de parte 3) se quisiera obtener un nuevo tipo que solo tuviera el número y el nombre de esta, entonces lo que se debería hacer en cada caso es:

i) Aplicar la siguiente operación:

```
proyectar( (CUENTACORRIENTE, l, c), {Numero, nombre}, DESCUE )
```

ii) Definir un nuevo tipo:

```
TAD DESCUE

cr_descue:  REAL X STRING ----> DESCUE.
```