

Tema 3

Especificación Algebraica

Introducción



- Aproximación basada en modelos
 - ◆ Tipos básicos (predefinidos)
 - \mathbb{Z} , \mathbb{N} , \mathbb{R}
 - ◆ Tipos nuevos, definidos por el usuario → dando el nombre
 - Cuenta, Saldo, Estudiante
- Diferenciar entre la especificación del tipo de datos y su representación interna e implementación
 - ◆ Tipo Abstracto de Dato (TAD)
 - ◆ Se abstraen las propiedades del comportamiento

Introducción



- TAD consta de:
 - ◆ Especificación clara y concisa del comportamiento de las operaciones sobre los objetos del tipo
 - Contrato entre el creador del TAD y el usuario
 - ◆ Implementación que se encarga de que dicha especificación se satisfaga
- Especificación e implementación están separadas
 - ◆ Cualquier cambio en la representación interna o en la implementación del TAD (que no altere su especificación) no afecta los programas que utiliza

Especificación algebraica



- Diseñada originalmente para la definición de interfaces de tipos abstractos de datos
 - ◆ Especificar el tipo de operaciones → SI
 - ◆ Especificar el tipo de representación → NO
- Define el tipo abstracto de datos en función de las relaciones entre los tipos de operaciones
 - ◆ Utiliza ecuaciones para describir las operaciones del TAD
 - ◆ Las operaciones deben tener nombre y sintaxis
 - Separar los términos bien contruidos de las expresiones contruidas incorrectamente

Especificación algebraica



■ Componentes principales

◆ Signature

- Describe la sintaxis de las expresiones asociadas al TAD
- Para cada operación
 - Nombre o identificador
 - N.º de argumentos y tipo de cada uno de ellos
 - Tipo del resultado

◆ Ecuaciones (Axiomas)


- Describen la semántica de las operaciones
- Tres tipo: Igualdades, Condicionales y error

Especificación algebraica



■ Tipos de operaciones

◆ Constructoras

- Permiten crear todos los objetos del TAD
- Permite crear el universo completo de los objetos del tipo
- Dos tipos 
 - Puras
 - Impuras

◆ No constructoras

- Destructoras
- Consulta

Especificación algebraica



■ Ejemplo: Números naturales

- ◆ Tipo: nat

- ◆ Signature

- $\text{cero}: \rightarrow \text{nat}$ (constructora)
- $\text{sucesor}: \text{nat} \rightarrow \text{nat}$ (constructora)
- $\text{predecesor}: \text{nat} \rightarrow \text{nat}$ (consulta)

- ◆ Ecuaciones (Axiomas)

- (n.1) $\text{predecesor}(\text{cero}) = \text{error}$
- (n.2) $\text{predecesor}(\text{sucesor}(n)) = n$

Especificación algebraica



- Ejemplo: Números naturales
 - ◆ Haciendo uso de las dos operaciones constructoras
 - Definir todos los objetos del TAD
 - `cero, sucesor(cero), sucesor(sucesor(cero)),`
 - Correspondencia biunívoca
 - $0 \rightarrow \text{cero}$
 - $1 \rightarrow \text{sucesor(cero)}$
 - $2 \rightarrow \text{sucesor(sucesor(cero))}$
 -

Especificación algebraica



- Semántica de las operaciones (axiomas)
 - ◆ Operaciones constructoras
 - Determinan el conjunto de los objetos del tipo
 - Definen el universo de los objetos
 - No es necesario definir ecuaciones para describir su comportamiento
 - ◆ Operaciones no constructoras
 - Se definen mediante ecuaciones
 - Las ecuaciones definen cuál es el resultado de aplicar la operación no constructora sobre todos y cada uno de los valores del tipo

Especificación algebraica



■ Definición de axiomas

- ◆ Se escriben usando las operaciones definidas en la signatura
- ◆ Especifican la semántica indicando lo que es siempre cierto acerca del comportamiento de las entidades

■ Regla para construir los axiomas

- ◆ Una ecuación para cada operación no constructora, sobre cada constructor
 - $m \text{ operaciones} * n \text{ constructoras}$ axiomas
- ◆ Se suele usar recursividad

Especificación algebraica



■ Ejemplo: TDA pila enteros

◆ Tipo

- *pila_ent* y *entero*

◆ Signature

- *pila_vacia*: $\rightarrow pila_ent$ (constructora)
- *apilar*: $entero \times pila_ent \rightarrow pila_ent$ (constructora)
- *desapilar*: $pila_ent \rightarrow pila_ent$
- *cima*: $pila_ent \rightarrow entero$
- ◆ Cualquier objeto pila se puede expresar usando únicamente *pila_vacia* y *apilar* (**inducción estructural**)
 - *apilar*(6,*apilar*(4,*apilar*(2,*pila_vacia*)))

Especificación algebraica



■ Ejemplo: TDA pila

- ◆ El funcionamiento de las operaciones *desapilar* y *cima* hay que especificarlo usando propiedades expresadas en forma de ecuaciones.
 - (p.4) $despilar(apilar(i,p)) = p$
 - p es una variable del tipo *pila_ent*
 - i es una variables del tipo *entero*
 - (p.2) $cima(apilar(i,p)) = i$
- ◆ Adicionalmente hay que expresar que no se puede aplicar las operaciones *desapilar* y *cima* a *pila_vacia*
 - (p.3) $desapilar(pila_vacía) = \text{error}$
 - (p.1) $cima(pila_vacía) = \text{error}$

Especificación algebraica



■ Estructura (Somerville)

< SPECIFICATION NAME >

sort < name >

imports < LIST OF SPECIFICATION NAMES >

Informal description of the sort and its operations

Operation signatures setting out the names and the types of the parameters to the operations defined over the sort

Axioms defining the operations over the sort

Especificación algebraica



■ Estructura (Somerville)

◆ Introducción

- Declara la clase de la entidad que se está especificando
- Declaración de *imports*

◆ Descripción

- Descripción informal de las operaciones

◆ Signature

- Define la sintaxis de la interfaz del TAD
- Operaciones, parámetros y resultados

◆ Axiomas

- Define la semántica de las operaciones mediante ecuaciones que caracterizan el comportamiento

Especificación algebraica



■ Ejemplo 1

ARRAY (Elem: [Undefined \rightarrow Elem])

sort Array
imports INTEGER

Arrays are collections of elements of generic type Elem. They have a lower and upper bound (discovered by the operations First and Last) Individual elements are accessed via their numeric index.

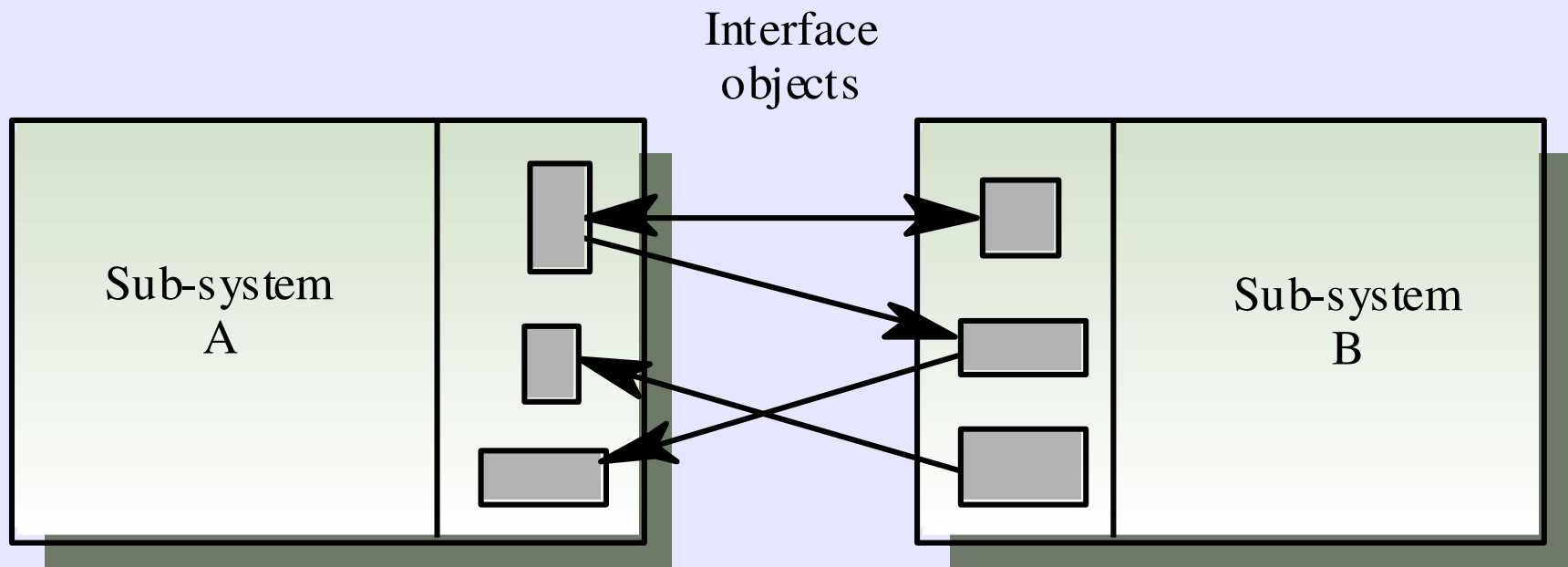
Create takes the array bounds as parameters and creates the array, initialising its values to Undefined. Assign creates a new array which is the same as its input with the specified element assigned the given value. Eval reveals the value of a specified element. If an attempt is made to access a value outside the bounds of the array, the value is undefined.

Create (Integer, Integer) \rightarrow Array
Assign (Array, Integer, Elem) \rightarrow Array
First (Array) \rightarrow Integer
Last (Array) \rightarrow Integer
Eval (Array, Integer) \rightarrow Elem

First (Create (x, y)) = x
First (Assign (a, n, v)) = First (a)
Last (Create (x, y)) = y
Last (Assign (a, n, v)) = Last (a)
Eval (Create (x, y), n) = Undefined
Eval (Assign (a, n, v), m) =
 if m < First (a) **or** m > Last (a) **then** Undefined **else**
 if m = n **then** v **else** Eval (a, m)



- Especificaciones de interfaces de subsistemas



Especificación algebraica



- Definición de una especificación algebraica
 - ◆ Seis fases
 - No es necesario que se realicen secuencialmente
 - Los resultados se pueden ir refinando



Especificación algebraica



- Fases en el proceso de desarrollo
 - ◆ Estructura de la especificación
 - Qué necesidad hay. Qué clases se necesitan
 - Definición informal de las operaciones de cada clase
 - ◆ Nombrado de la especificación
 - Nombre que se le dará a la especificación y a las clases
 - Determinar si requiere parámetros genéricos
 - ◆ Selección de las operaciones
 - Basadas en la funcionalidad de la interfaz
 - Tipo → creación, modificación e inspección

Especificación algebraica



- Fases en el proceso de desarrollo
 - ◆ Especificación informal de las operaciones
 - Escribir una especificación informal de cada operación
 - Como las operaciones afectan a la clase definida
 - ◆ Definición de la signature
 - Sintaxis de las operaciones y sus parámetros
 - A lo mejor es necesario cambiar la especificación informal
 - ◆ Definición de axiomas
 - Semántica de las operaciones mediante ecuaciones

Especificación algebraica



- Ejemplo: Lista enlazada
 - ◆ Nombrado de la especificación
 - Especificación → **LIST**
 - Tipo → **List**
 - Tiene un parámetro genérico (**Elem**)
 - La lista es una colección de elementos de cualquier tipo
 - ◆ Selección de las operaciones + especificación informal
 - **ListaVacía** → crea una lista vacía
 - **Add** → añade un elemento al final de una lista
 - **Head** → obtener el primer elemento de una lista
 - **Tail** → devuelve la lista sin el primer elemento
 - **Length** → longitud de la lista

Especificación algebraica



■ Ejemplo: Lista enlazada

◆ Sintaxis de las operaciones

- Parámetros entrada y /o resultados

- Elementos de la clase definida (List)
- Elementos de clases genéricas (Integer or Boolean) → imports

- Operaciones

- *ListaVacía*: $\rightarrow List (Elem)$ (constructora)
- *Add* : $List (Elem) \times Elem \rightarrow List (Elem)$ (constructora)
- *Tail*: $List (Elem) \rightarrow List (Elem)$
- *Head*: $List (Elem) \rightarrow Elem$
- *Length*: $List (Elem) \rightarrow Integer$

Especificación algebraica



■ Ejemplo: Lista enlazada

◆ Definición de axiomas

• Conjunto de ecuaciones

- (I.1) $\text{Head}(\text{ListaVacia}) = \text{Error} \rightarrow \text{Error al evaluar una lista vacía}$
- (I.2) $\text{Head}(\text{Add}(L, v)) = \text{if } L = \text{ListaVacia} \text{ then } v \text{ else } \text{Head}(L)$
- (I.3) $\text{Lenght}(\text{ListaVacia}) = 0$
- (I.4) $\text{Lenght}(\text{Add}(L, v)) = 1 + \text{Lenght}(L)$
- (I.5) $\text{Tail}(\text{ListaVacia}) = \text{ListaVacia}$
- (I.6) $\text{Tail}(\text{Add}(L, v)) = \text{if } L = \text{ListaVacia} \text{ then } \text{ListaVacia} \text{ else } \text{Add}(\text{Tail}(L), v)$

Especificación algebraica



■ Ejemplo 2

LIST (Elem)

sort List
imports INTEGER

Defines a list where elements are added at the end and removed from the front. The operations are Create, which brings an empty list into existence, Cons, which creates a new list with an added member, Length, which evaluates the list size, Head, which evaluates the front element of the list, and Tail, which creates a list by removing the head from its input list. Undefined represents an undefined value of type Elem.

Create \rightarrow List
Cons (List, Elem) \rightarrow List
Head (List) \rightarrow Elem
Length (List) \rightarrow Integer
Tail (List) \rightarrow List

Head (Create) = Undefined **exception** (empty list)
Head (Cons (L, v)) = **if** L = Create **then** v **else** Head (L)
Length (Create) = 0
Length (Cons (L, v)) = Length (L) + 1
Tail (Create) = Create
Tail (Cons (L, v)) = **if** L = Create **then** Create **else** Cons (Tail (L), v)

Especificación algebraica



■ Especificación incremental

- ◆ Reutilizar especificaciones desarrolladas
 - Minimizar el esfuerzo necesario para desarrollar una especificación formal
- ◆ Especificaciones simples se usan como “bloques de construcción” para especificaciones más complejas
- ◆ 3 formas de reutilizar especificaciones
 - Instanciación de especificaciones genéricas
 - Desarrollo incremental de especificaciones
 - Enriquecimiento de especificaciones

Especificación algebraica



■ Especificación incremental (*Instanciación*)

- ◆ Coger una especificación previa especificada con un parámetro genérico y darle valor (instanciarlo)

```
ARRAY ( Elem: [Undefined → Elem] )  
  
sort Array  
imports INTEGER  
  
Arrays are collections of elements of generic type Elem. They have a  
lower and upper bound (discovered by the operations First and Last)  
Individual elements are accessed via their numeric index.  
Create takes the array bounds as parameters and creates the array,  
initialising its values to Undefined. Assign creates a new array which  
is the same as its input with the specified element assigned the given  
value. Eval reveals the value of a specified element. If an attempt is  
made to access a value outside the bounds of the array, the value is  
undefined.  
  
Create (Integer, Integer) → Array  
Assign (Array, Integer, Elem) → Array  
First (Array) → Integer  
Last (Array) → Integer  
Eval (Array, Integer) → Elem  
  
First (Create (x, y)) = x  
First (Assign (a, n, v)) = First (a)  
Last (Create (x, y)) = y  
Last (Assign (a, n, v)) = Last (a)  
Eval (Create (x, y), n) = Undefined  
Eval (Assign (a, n, v), m) =  
    if m < First (a) or m > Last (a) then Undefined else  
        if m = n then v else Eval (a, m)
```

```
CHAR_ARRAY : ARRAY  
  
sort Char_array instantiates Array (Elem:=Char)  
imports INTEGER
```

Especificación algebraica



- Especificación incremental (***Desarrollo incremental***)
 - ◆ Desarrollar especificaciones simples y usarlas para construir otras más complejas
 - ◆ Las especificaciones simples son importadas en las más complejas
 - ◆ Las operaciones definidas en las especificaciones simples se pueden usar en las complejas
 - ***Directamente***, si el nombre de la operación no coincide con el de alguna operación de la especificación compleja
 - ***Anteponiendo el nombre de la especificación*** si hay otra operación que se llame igual

Especificación algebraica



- Especificación incremental (***Desarrollo incremental***)
 - ◆ Tipo: contador
 - ◆ Usa/Import:nat
 - ◆ Signature
 - inicial: \rightarrow contador (constructora)
 - incrementar: contador \rightarrow contador (constructora)
 - decrementar: contador \rightarrow contador
 - reiniciar: contador \rightarrow contador
 - valor: contador \rightarrow nat



- Especificación incremental (***Desarrollo incremental***)
 - ◆ Tipo: contador
 - ◆ Axiomas
 - (c.1) decrementar (inicial) = error
 - (c.2) decrementar (incrementar(x)) = x
 - (c.3) reiniciar (inicial) = inicial
 - (c.4) reiniciar (incrementar(x)) = inicial
 - (c.5) valor (inicial) = cero
 - (c.6) valor (incrementar(x)) = sucesor(valor(x))

Especificación algebraica



■ Especificación incremental (*Desarrollo incremental*)

COORD
sort Coord imports INTEGER, BOOLEAN
Defines a sort representing a Cartesian coordinate. The operations defined on Coord are X and Y which evaluate the x and y attributes of an entity of this sort and Eq which compares two entities of sort Coord for equality.
Create (Integer, Integer) → Coord ; X (Coord) → Integer ; Y (Coord) → Integer ; Eq (Coord, Coord) → Boolean ;
X (Create (x, y)) = x Y (Create (x, y)) = y Eq (Create (x1, y1), Create (x2, y2)) = ((x1 = x2) and (y1 = y2))

CURSOR
sort Cursor imports INTEGER, COORD, BITMAP
A cursor is a representation of a screen position. Defined operations are Create which associates an icon with the cursor at a screen position, Position which returns the current coordinate of the cursor, Translate which moves the cursor a given amount in the x and y directions and Change_Icon which causes the cursor icon to be switched.
The Display operation is not defined formally. Informally, it causes the icon associated with the cursor to be displayed so that the top-left corner of the icon represents the cursor's position. When displayed, the 'clear' parts of the cursor bitmap should not obscure the underlying objects.
Create (Coord, Bitmap) → Cursor Translate (Cursor, Integer, Integer) → Cursor Position (Cursor) → Coord Change_Icon (Cursor, Bitmap) → Cursor Display (Cursor) → Cursor
Translate (Create (C, Icon), xd, yd) = Create (COORD.Create (X(C)+xd, Y(C)+yd), Icon) Position (Create (C, Icon)) = C Position (Translate (C, xd, yd)) = COORD.Create (X(C)+xd, Y(C)+yd) Change_Icon (Create (C, Icon), Icon2) = Create (C, Icon2)

Especificación algebraica



- Especificación incremental (***Enriquecimiento***)
 - ◆ Similar a la herencia en la orientación a objetos
 - ◆ Las operaciones y axiomas de la especificación base pasan a formar parte de la nueva especificación
 - Las nuevas operaciones pueden sobrescribir operaciones base con el mismo nombre
 - Se pueden añadir nuevas operaciones o eliminar
 - ◆ No es lo mismo que importar una especificación
 - Las operaciones o axiomas son accesibles, pero no forman parte de la nueva especificación

Especificación algebraica



- Especificación incremental (***Enriquecimiento***)
 - ◆ Ejemplo → Enriquecer el tipo pila
 - ◆ Signature
 - $\text{es_vacía: pila_ent} \rightarrow \text{bool}$
 - Comprueba si una pila esta vacía
 - $\text{altura: pila_ent} \rightarrow \text{nat}$
 - Devuelve el n.º de elementos de una pila
 - $\text{coger: nat} \times \text{pila_ent} \rightarrow \text{pila_ent}$
 - Devuelve la pila formada por los n primeros elementos
 - $\text{eliminar: nat} \times \text{pila_ent} \rightarrow \text{pila_ent}$
 - Devuelve la pila tras eliminar los n primeros elementos

Especificación algebraica



■ Especificación incremental (*Enriquecimiento*)

- ◆ Ejemplo → Enriquecer el tipo pila

- ◆ Axiomas



- (p.5) $\text{es_vacía}(\text{pila_vacía}) = \text{T}$
- (p.6) $\text{es_vacía}(\text{apilar}(i, p)) = \text{F}$
- (p.7) $\text{altura}(\text{pila_vacía}) = 0$
- (p.8) $\text{altura}(\text{apilar}(i, p)) = 1 + \text{altura}(p)$
- (p.9) $\text{coger}(n, \text{pila_vacía}) = \text{if } n=0 \text{ then pila_vacía else error}$
- (p.10) $\text{coger}(n, \text{apilar}(i, p)) = \text{if } n=0 \text{ then pila_vacía else apilar}(i, \text{coger}(n-1, p))$

Especificación algebraica



- Especificación incremental (***Enriquecimiento***)
 - ◆ Ejemplo → Enriquecer el tipo pila
 - ◆ Axiomas
 - (p.11) eliminar (n,pila_vacia) = if n=0 then pila_vacia
else error
 - (p.12) eliminar (n, apilar(i,p)) = if n=0 then apilar(i,p)
else eliminar(n-1,p)

Especificación algebraica



■ Especificación incremental (*Enriquecimiento*)

Operation	Description
Create	Brings a list into existence.
Cons (New_list, Elem)	Adds an element to the end of the list.
Add (New_list, Elem)	Adds an element to the front of the list.
Head (New_list)	Returns the first element in the list.
Tail (New_list)	Returns the list with the first element removed.
Member (New_list, Elem)	Returns true if an element of the list matches Elem
Length (New_list)	Returns the number of elements in the list

```
NEW_LIST ( Elem: [Undefined → Elem; .=. → Boolean] )
sort New_List enrich List
imports INTEGER, BOOLEAN

Defines an extended form of list which inherits the operations
and properties of the simpler specification of List and which adds
new operations (Add and Member) to these.
See Figure 10.10 for a description of the list operations.

Add (New_List, Elem) → New_List
Member (New_List, Elem) → Boolean

Add (Create, v) = Cons (Create, v)
Member (Create, v) = false
Member (Add (L, v), v1) = ((v = v1) or Member (L, v1))
Member (Cons (L, v), v1) = ((v = v1) or Member (L, v1))
Head (Add (L, v)) = v
Tail (Add (L, v)) = L
Length (Add (L, v)) = Length (L) + 1
```

Especificación algebraica



- ¿Cómo manejar situaciones de error?
 - ◆ Usar una operación constante especial que indique que no está definido → Undefined /Error
 - Ejemplo del array
 - ◆ La operación devuelve una tupla, donde uno de los componentes indican si la operación tuvo éxito o no
 - Ejemplo de una cola
 - ◆ Incluir en la especificación una sección de excepciones
 - Define condiciones bajo las que los axiomas no se cumplen

Especificación algebraica



■ ¿Cómo manejar situaciones de error?

QUEUE (Elem: [Undefined \rightarrow Elem])

sort Queue **enrich** List
imports INTEGER

This specification defines a queue which is a first-in, first-out data structure. It can therefore be specified as a List where the insert operation adds a member to the end of the queue.
See Figure 10.12 for a description of queue operations.

Get (Queue) \rightarrow (Elem, Queue)

Get (Create) = (Undefined, Create)
Get (Cons (Q, v)) = (Head (Q), Tail (Cons (Q, v)))

LIST (Elem)

sort List
imports INTEGER

See Figure 10.4

Create \rightarrow List
Cons (List, Elem) \rightarrow List
Tail (List) \rightarrow List
Head (List) \rightarrow Elem
Length (List) \rightarrow Integer

Head (Cons (L, v)) = **if** L = Create **then** v **else** Head (L)
Length (Create) = 0
Length (Cons (L, v)) = Length (L) + 1
Tail (Create) = Create
Tail (Cons (L, v)) = **if** L = Create **then** Create **else** Cons (Tail (L), v)

exceptions

Length (L) = 0 \Rightarrow **failure** (Head (L))

Especificación algebraica



- Demostración de propiedades
 - ◆ La especificación algebraica establece una base formal que permite evaluar expresiones y demostrar propiedades de una manera razonada
 - Demostración por deducción
 - Demostración por inducción
- Demostración por deducción
 - ◆ Demostrar que dos expresiones son equivalentes
 - ◆ Se basa en la reescritura de expresiones
 - En cada paso se ha de aplicar una ecuación de la especificación o una propiedad previamente demostrada

Especificación algebraica



■ Demostración por deducción

- ◆ $\text{altura}(\text{desapilar}(\text{apilar}(x, \text{apilar}(y, \text{pila_vacía})))) = 1$
 - $\text{altura}(\text{desapilar}(\text{apilar}(x, \text{apilar}(y, \text{pila_vacía}))))$
 - $= (p.4) \text{ altura } (\text{apilar } (y, \text{pila_vacía}))$
 - $= (p.8) 1 + \text{altura}(\text{pila_vacía})$
 - $= (p.7) 1 + 0$
 - $= 1$

Especificación algebraica



■ Demostración por deducción

- ♦ $\text{Tail}([5,7,9]) = [7,9]$

- (I.6) $\text{Tail}(\text{Cons}(L,v)) = \text{if } L = \text{Create} \text{ then Create else } \text{Cons}(\text{Tail}(L),v)$

- $[5,7,9] = \text{Cons}([5,7],9)$

- ♦ Proceso

- $\text{Tail}([5,7,9]) = \text{Tail}(\text{Cons}([5,7],9)) = \text{Cons}(\text{Tail}([5,7]),9) =$

- $\text{Cons}(\text{Tail}(\text{Cons}([5],7)),9) = \text{Cons}(\text{Cons}(\text{Tail}([5]),7),9) =$

- $\text{Cons}(\text{Cons}(\text{Tail}(\text{Cons}([],5)),7),9) =$

- $\text{Cons}(\text{Cons}([\text{Create}],7),9) = \text{Cons}([7],9) = [7,9]$

Especificación algebraica



■ Ejemplo: Sistema crítico

- ◆ Sistema de control de tráfico aéreo → sector controlado del espacio aéreo.
- ◆ Cada sector puede incluir un número de avión (identificador)
- ◆ Por razones de seguridad los aviones se separan 300m
- ◆ El sistema avisa al controlador si un avión intenta posicionarse incumpliendo esta restricción

Especificación algebraica



- Operaciones críticas sobre el objeto sector
 - ♦ Enter
 - Añade un avión al espacio aéreo a una altura específica
 - ♦ Leave
 - Elimina el avión especificado del sector controlado
 - ♦ Move
 - Mueve un avión de una altura a otra
 - ♦ Lookup
 - Dado un identificador devuelve la altura del avión

Especificación algebraica



- En ocasiones es necesario definir a priori otras operaciones
 - ◆ Simplificar la especificación
- Las otras operaciones se definen en términos de estas (operaciones primitivas)
- Operaciones primitivas
 - ◆ *Create* → crea un sector sin aviones
 - ◆ *Put* → añade un avión sin comprobar restricciones
 - ◆ *In-space* → determina si un avión está en el sector
 - ◆ *Occupied* → dada una altura, comprueba si hay un avión a menos de 300 m de esa altura.

Especificación algebraica



SECTOR

sort Sector

imports INTEGER, BOOLEAN

Enter - adds an aircraft to the sector if safety conditions are satisfied

Leave - removes an aircraft from the sector

Move - moves an aircraft from one height to another if safe to do so

Lookup - Finds the height of an aircraft in the sector

Create - creates an empty sector

Put - adds an aircraft to a sector with no constraint checks

In-space - checks if an aircraft is already in a sector

Occupied - checks if a specified height is available

Enter (Sector, Call-sign, Height) → Sector

Leave (Sector, Call-sign) → Sector

Move (Sector, Call-sign, Height) → Sector

Lookup (Sector, Call-sign) → Height

Create → Sector

Put (Sector, Call-sign, Height) → Sector

In-space (Sector, Call-sign) → Boolean

Occupied (Sector, Height) → Boolean

Especificación algebraica



```
Enter (S, CS, H) =  
  if In-space (S, CS) then S exception (Aircraft already in sector)  
  elseif Occupied (S, H) then S exception (Height conflict)  
  else Put (S, CS, H)  
  
Leave (Create, CS) = Create exception (Aircraft not in sector)  
Leave (Put (S, CS1, H1), CS) =  
  if CS = CS1 then S else Put (Leave (S, CS), CS1, H1)  
  
Move (S, CS, H) =  
  if S = Create then Create exception (No aircraft in sector)  
  elseif not In-space (S, CS) then S exception (Aircraft not in sector)  
  elseif Occupied (S, H) then S exception (Height conflict)  
  else Put (Leave (S, CS), CS, H)  
  
-- NO-HEIGHT is a constant indicating that a valid height cannot be returned  
  
Lookup (Create, CS) = NO-HEIGHT exception (Aircraft not in sector)  
Lookup (Put (S, CS1, H1), CS) =  
  if CS = CS1 then H1 else Lookup (S, CS)  
  
Occupied (Create, H) = false  
Occupied (Put (S, CS1, H1), H) =  
  if (H1 > H and H1 - H ≤ 300) or (H > H1 and H - H1 ≤ 300) then true  
  else Occupied (S, H)  
  
In-space (Create, CS) = false  
In-space (Put (S, CS1, H1), CS) =  
  if CS = CS1 then true else In-space (S, CS)
```