Especificación algebraica.

MEET

15 y 20 de ABRIL

2020

Unidad II - Contenidos

Tipos abstractos de datos básicos.

- Especificación algebraica.
- El tipo abstracto de datos Pila. El tipo abstracto de datos Fila. El tipo abstracto de datos Lista y Lista Circular. Aplicaciones.
- Implementación de lo tipos de datos básicos con arreglos y con listas enlazadas. Comparación entre implementaciones con variables estáticas y dinámicas. Limitaciones y ventajas.

Métodos formales

- El lenguaje natural ha sido el mas usado para especificar sistemas pero genera documentos ambiguos, incompletos e inconsistentes.

- Así surgen ***métodos formales*** para especificar el software. En estos métodos los lenguajes usados para describir los sistemas son lenguajes con sintaxis y semántica precisamente definidas llamados ***lenguajes formales.***

Métodos formales

- El termino “método formal” se refiere a las distintas técnicas matemáticas usadas en la especificación formal y desarrollo del software.

- Consiste de un lenguaje de especificación formal y emplea una colección de herramientas que permiten chequear la sintaxis de la especificación así como lo resultados de la especificación.

- Permiten saber sobre que hace el sistema independientemente de su implementación.

Métodos formales

- Usar notación matemática evita la interpretación de las frases del lenguaje natural que es impreciso en la descripción del sistema.

- El lenguaje natural es ambiguo y en la matemática se emplea notación rigurosa y precisa.

- Los métodos formales permiten especificar, desarrollar y verificar un sistema basado en computadora mediante la aplicación de una notación matemática rigurosa.

Métodos formales

- Definición de *Especificación Formal* (Spivey): es el uso de notación matemática para describir de manera precisa las propiedades que un sistema de información debe tener, sin restringir excesivamente la forma en que esas propiedades se alcanzan.

- El termino método formal se usa para describir un lenguaje de especificación formal y un método para diseñar e implementar sistemas de computación. La especificación se escribe en lenguaje matemático y su implementación se deriva de la especificación por refinamientos sucesivos

Métodos formales

Enfoques de los métodos formales

- 1) Orientado a modelos, construyen un modelo matemático del mundo físico o de un sistema. Explican el comportamiento del sistema y permiten hacer predicciones. Ej. Vienna Development Method (VDM), Z
- 2) Axiomático, se enfoca en las propiedades que el sistema debe cumplir y no se produce un modelo abstracto. Se usa notación matemática para establecer las propiedades requeridas y el comportamiento del sistema. Ej. calculus communicating systems (CCS), communicating sequential processes (CSP).

Formalismo o Sistema Formal

Un sistema formado por:

- Un lenguaje simbólico (lenguaje formal) constituido de
  - Un alfabeto
  - Reglas de formación del lenguaje (gramática)
- Un conjunto de axiomas
- Un conjunto finito de reglas de deducción

Una especificación formal es una descripción de una parte de la realidad mediante un sistema formal

Sistemas Formales

Ejemplos de sistemas formales:

- la teoría de conjuntos
- el álgebra de Boole
- el calculo proposicional y de predicados
- los sistemas de Post
- la geometría plana de Euclides (Hilbert)
- la forma norma de Backus
- la aritmética de Peano para los números naturales

Formalismo o Sistema Formal

Algunas áreas de aplicación:

- Ingeniería de Software: Verificación de corrección de programas.
- Lenguajes de Programación: especificación de tipos abstractos de datos, verificación de la correctitud de sus implementaciones
- Comunicación: especificar y verificar protocolos de comunicación.
- Control de Procesos, descripción formal de procesos de fabricación y verificación de sus propiedades.
- Bases de Datos: especificación formal de sistemas

Formalismo o Sistema Formal

Formalismos más usados en Ciencias de la Computación

- Formalismos lógicos
- Formalismos algebraicos

UN SISTEMA AXIOMATICO DE LA LOGICA ELEMENTAL
SINTAXIS

1) El **alfabeto L** del lenguaje es un conjunto formado por tres conjuntos sin elemento común:

i) El conjunto **V** de **variables proposicionales**, V es infinito enumerable.

V = { p1,p2, ...,pn,... }

ii) El conjunto **K** de **conectores proposicionales** o **conectivos primitivos**.

$K = \{\rightarrow\} \cup \{\neg\}$

iii) El conjunto **P** de **símbolos de puntuación**.

P = { (,) }

L es la unión de estos tres conjuntos, $L = V \cup K \cup P$

Disponiendo del alfabeto se pueden formar series finitas o **cadenas de caracteres**. Por ejemplo "(p1 $\rightarrow$ p3)" , "p1 $\neg\neg$"

UN SISTEMA AXIOMATICO DE LA LOGICA ELEMENTAL
LENGUAJE FORMAL

2) El **lenguaje** del cálculo de proposiciones o conjunto **F** de expresiones bien formadas.

De todas las cadenas que se pueden formar con el alfabeto **L** interesa el llamado conjunto de **expresiones bien formadas** o **fórmulas proposicionales.**

El conjunto **F** de todas las expresiones bien formadas que se denomina ebf se define en forma recurrente por las siguientes reglas que se denominan reglas de buena formación:

UN SISTEMA AXIOMATICO DE LA LOGICA ELEMENTAL
LENGUAJE FORMAL

Reglas de buena formación

i) Una variable proposicional es una expresión bien formada.

para todo $p \in V$, $p \in F$, V está contenido en F.

ii) Si A y B son ebf, entonces $\neg A$ y $A \to B$ son ebf.

iii) Regla de clausura: solo las cadenas formadas por aplicación de las reglas i) y ii) son ebf.

Ejemplos de ebf: p

$(\neg p)$

$((p \to q) \to r)$

UN SISTEMA AXIOMATICO DE LA LOGICA ELEMENTAL
AXIOMAS

Si A, B, C son ebf de L, es decir pertenecen a F, entonces los siguientes son axiomas de L:

A1. $(A\rightarrow(B\rightarrow A))$

A2. $((A\rightarrow(B\rightarrow C))\rightarrow((A\rightarrow B)\rightarrow(A\rightarrow C))$

A3. $(((\neg B)\rightarrow(\neg A))\rightarrow(((\neg B\rightarrow A)\rightarrow B))$

UN SISTEMA AXIOMATICO DE LA LOGICA ELEMENTAL
REGLA DE DEDUCCION

REGLA DE INFERENCIA

La única regla de inferencia de L es ***modus ponens***,
B es una consecuencia directa de A y de $A \rightarrow B$.
$A, A \rightarrow B \vdash B$

DEFINICIONES

Otros conectivos pueden introducirse como definiciones.

D1. $A \wedge B$ para $\neg(A \rightarrow (\neg B))$

D2. $A \vee B$ para $(\neg A) \rightarrow B$

D3. $A \equiv B$ para $(A \rightarrow B) \wedge (B \rightarrow A)$

ABSTRACCION

- Abstracción: es un mecanismo que permite expresar los detalles relevantes y suprimir los detalles irrelevantes.

- En el caso de programación, el uso de una abstracción es relevante, la forma en que esta implementada esa abstracción es irrelevante.

- Permite distinguir:
  - Que hace un programa y como se usa
  - Como se implementa

ABSTRACCION

- En los lenguajes de programación se ofrece una poderosa manera de abstracción: la función o procedimiento.

- Cuando un programador usa una función, le interesa que hace esa función, no le preocupa el algoritmo que ejecuta para entregar esos resultados.

- Aun mas, las funciones proveen un mecanismo de descomponer un problema en subproblemas y son justamente una herramienta que captura el significado de la abstracción.

TIPOS ABSTRACTOS DE DATOS

*El término Tipo **Abstracto** de Dato (TAD), apareció por primera vez en la literatura como un concepto básico del lenguaje CLU, en el artículo:*

"Programming with abstract data type" (B. Liskov and S. Zilles, 1974). En este articulo se definen los tipos abstractos de datos como:

- *An abstract data type defines a class of abstract objects which is completely characterized by the operations available on those objects. This means that an abstract data type can be defined by defining the characterizing operations for that type.*

.

TIPOS ABSTRACTOS DE DATOS

- *Tipo Abstracto de Datos*: Clase de objetos abstractos que se caracterizan completamente por las operaciones definidas sobre ellos, de modo que quedan definidos por esas operaciones.

- Cuando se usan los tipos abstractos de datos, importa solamente el comportamiento que el objeto tiene caracterizado por sus operaciones, pero no se tiene en cuenta los detalles de la implementación (como por ej. su representación en cuestiones de almacenamiento)

TIPOS ABSTRACTOS DE DATOS

- Los tipos abstractos de datos pretenden ser como los tipos provistos por los lenguajes de programación.

Ej. INTEGER

- El usuario se ocupa solamente de crear objetos de ese tipo y realizar las operaciones permitidas en él. No se preocupa como están representados esos objetos y usa sus operaciones como atómicas, cuando de hecho se requieren muchas instrucciones de máquina para resolverlas.

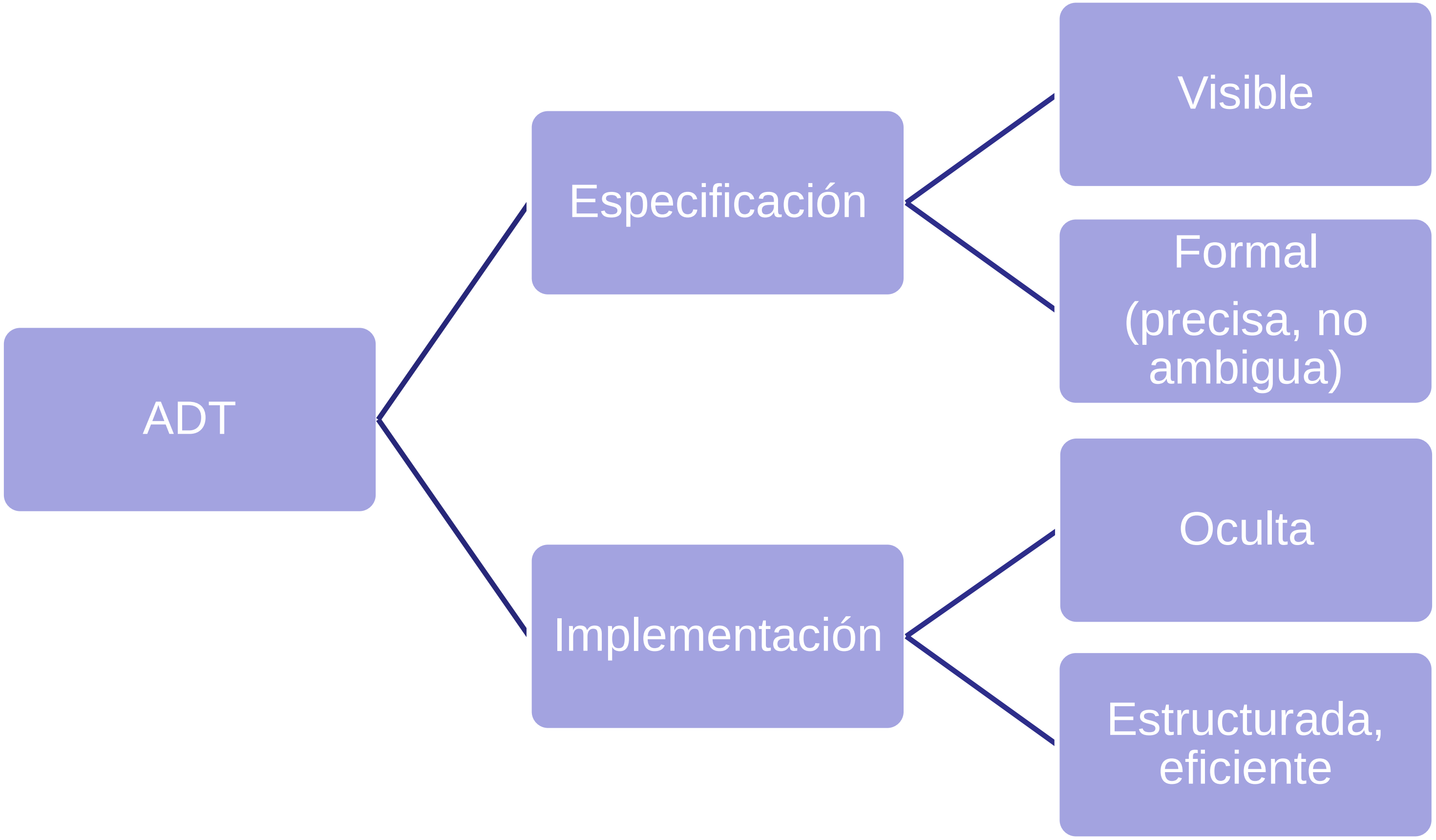
TIPOS ABSTRACTOS DE DATOS

La teoría de los tipos abstractos de datos considera la definición de ***nuevos tipos de datos en dos etapas*** ***<u>separadas</u>* *e* *<u>independientes</u>***:

- La ***especificación*** de las propiedades que definen al tipo de dato.
- La ***implementación*** del tipo de dato.

La importancia reside en la esencia de la **<u>abstracción</u>** que separa la forma en que se *definen* y de cómo se pueden ***usar*** los objetos de la manera en que se ***representan***.

TIPOS ABSTRACTOS DE DATOS



TIPOS ABSTRACTOS DE DATOS

1) Etapa de diseño de especificaciones:

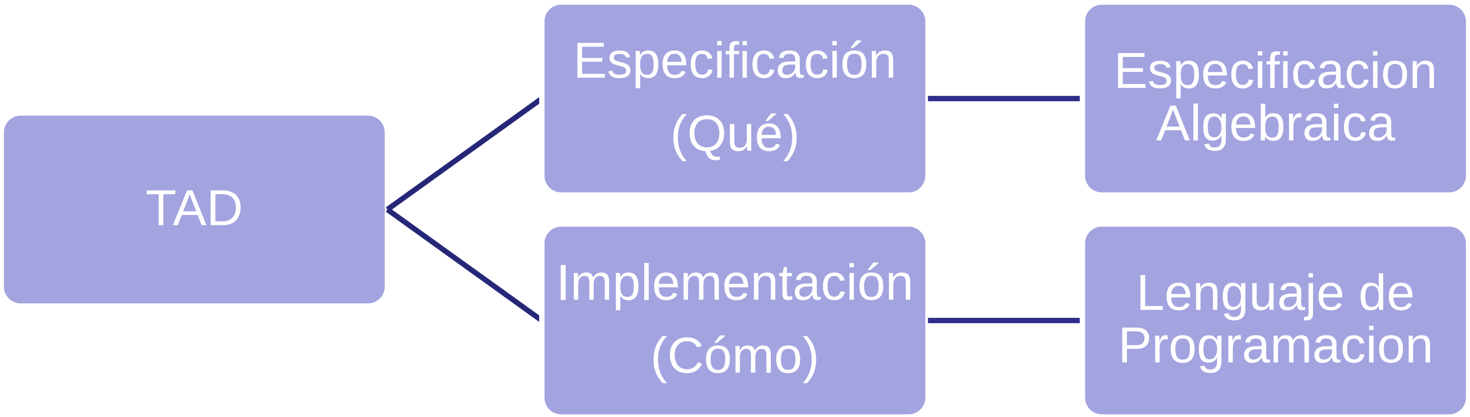
- Establece las propiedades que definen el tipo.
- Consiste en describir los objetos y las operaciones con un modelo matemático.
- El resultado es una especificación independiente de su futura implementación.
- La especificación debe ser: general, precisa, legible y no ambigua.
- De la gran variedad de métodos de especificación, en este curso se usará la ***Especificación Algebraica***.

TIPOS ABSTRACTOS DE DATOS

***2)* Etapa de implementación:**

- Elegir una estructura de datos adecuada del lenguaje de programación para representar el tipo
- Escribir la codificación de cada operación mediante un procedimiento o función.
- La implementación debe ser correcta y eficiente.
- La representación de los datos y la implementación de las operaciones se mantienen encapsuladas y protegidas del usuario.
- Existe gran variedad de lenguajes de programación para implementar: ***Lenguaje C, C++, Java, Python*, etc.**

TIPOS ABSTRACTOS DE DATOS



ESPECIFICACIONES ALGEBRAICAS

Una especificación algebraica es un algebra de tipos definida por un par: (Σ, E),

donde Σ es una signatura, un conjunto de conjuntos

E un conjunto de ecuaciones con respecto a Σ.

<u>Signatura</u>

- Una signatura Σ es un par (S,F), donde S es un conjunto de dominios y F es un conjunto de funciones sobre los dominios de S. La signatura especifica cómo se construyen los términos.

ESPECIFICACIONES ALGEBRAICAS

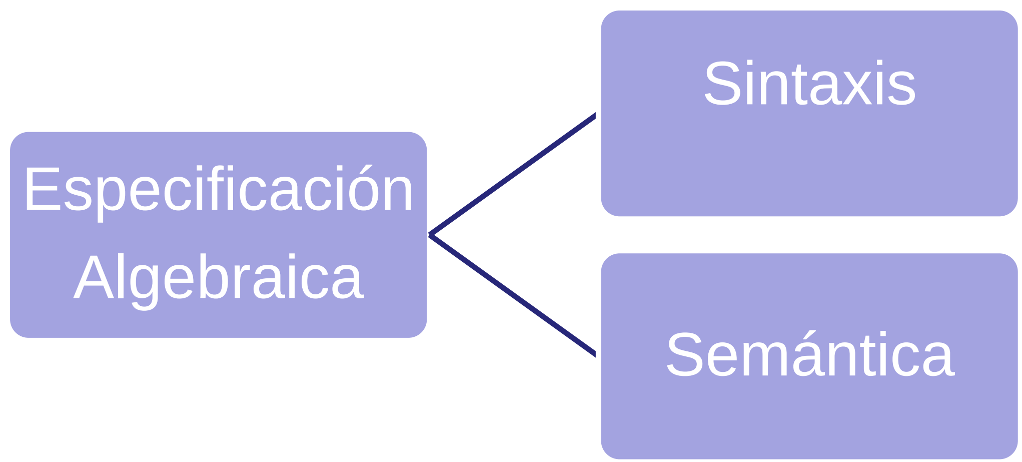
Ventajas de una especificación algebraica:

- es declarativa, evitando los detalles de programación, es independiente del lenguaje
- es razonablemente intuitiva
- es suficientemente rigurosa
- a partir de los axiomas se puede construir las pruebas de corrección de la especificación
- es fácil de leer y comprender, facilitando la verificación informal.

ESPECIFICACIONES ALGEBRAICAS

Una especificación algebraica es un algebra de tipos que define conjuntos y relaciones (operaciones).
Consta de 2 partes:

- **Parte sintáctica**
- **Parte semántica**



ESPECIFICACIONES ALGEBRAICAS

- **Parte sintáctica:** un conjunto de funciones (operaciones) definidas para el objeto, determina: nombres, dominios y rangos.

nombre : dominio → rango

Las operaciones pueden ser de distintos tipos:

- constructoras primitivas (construyen objetos del tipo),
- constructoras (los modifican),
- selectoras (separan distintas partes del objeto),
- test (deciden sobre alguna propiedad de objetos del tipo)
- Otras operaciones

ESPECIFICACIONES ALGEBRAICAS

- **Parte semántica:** un conjunto finito de axiomas algebraicos que demuestran los efectos de las operaciones sobre los objetos.

- Estos axiomas describen **QUE** hace la operación.

- Existen operaciones para las cuales no se dan axiomas, son las constructoras primitivas del tipo.

- A partir de esas operaciones constructoras primitivas se definen todas las otras operaciones del tipo.

ESPECIFICACIONES ALGEBRAICAS

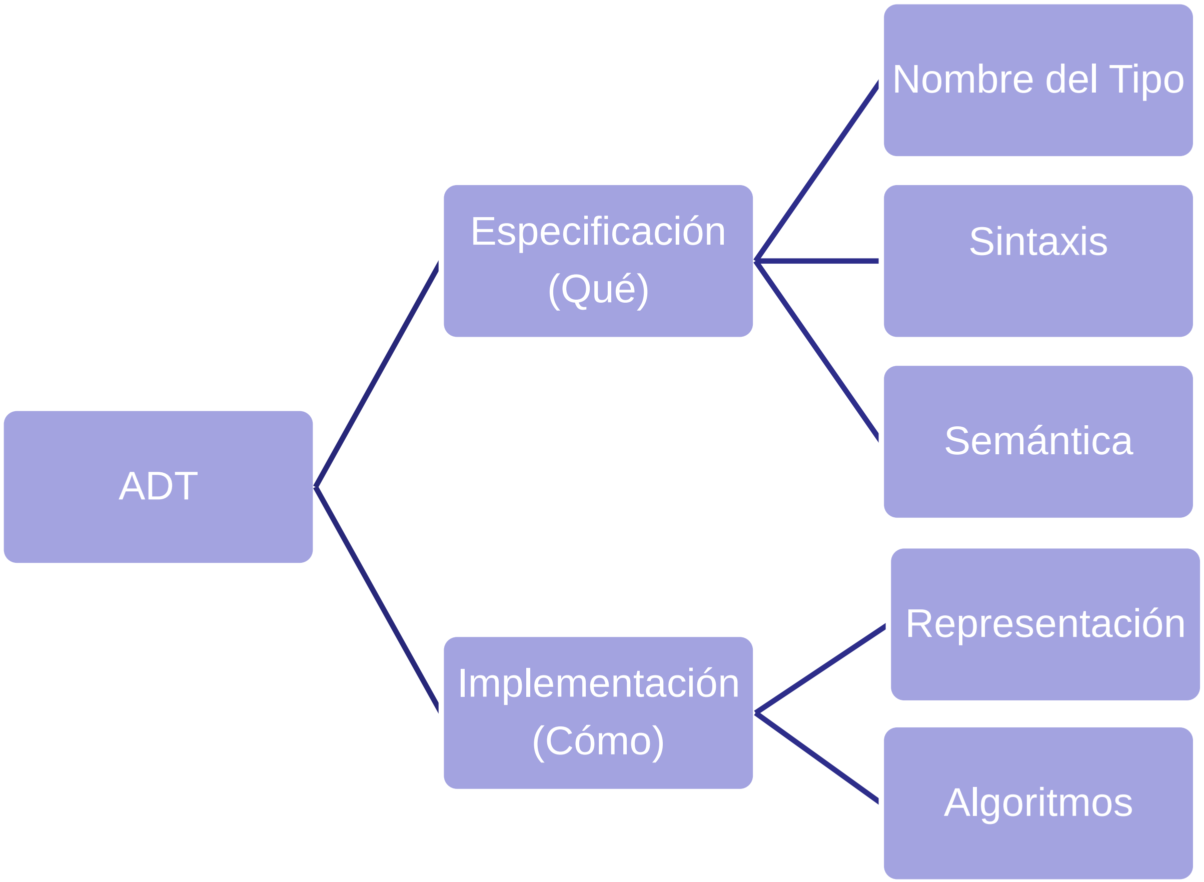
- **Parte semántica:** Los axiomas son de la forma:

operación (constructora primitiva) ≡ definición

Deben contener :

- **del lado izquierdo:** la operación aplicada a las constructoras primitivas.
- **del lado derecho:** operaciones ya definidas, o la misma operación que se esta definiendo, variables libres cuantificadas universalmente y test condicionales.

TIPOS ABSTRACTOS DE DATOS



ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

TIPO: BOOL

OPERACIONES

Sintaxis:

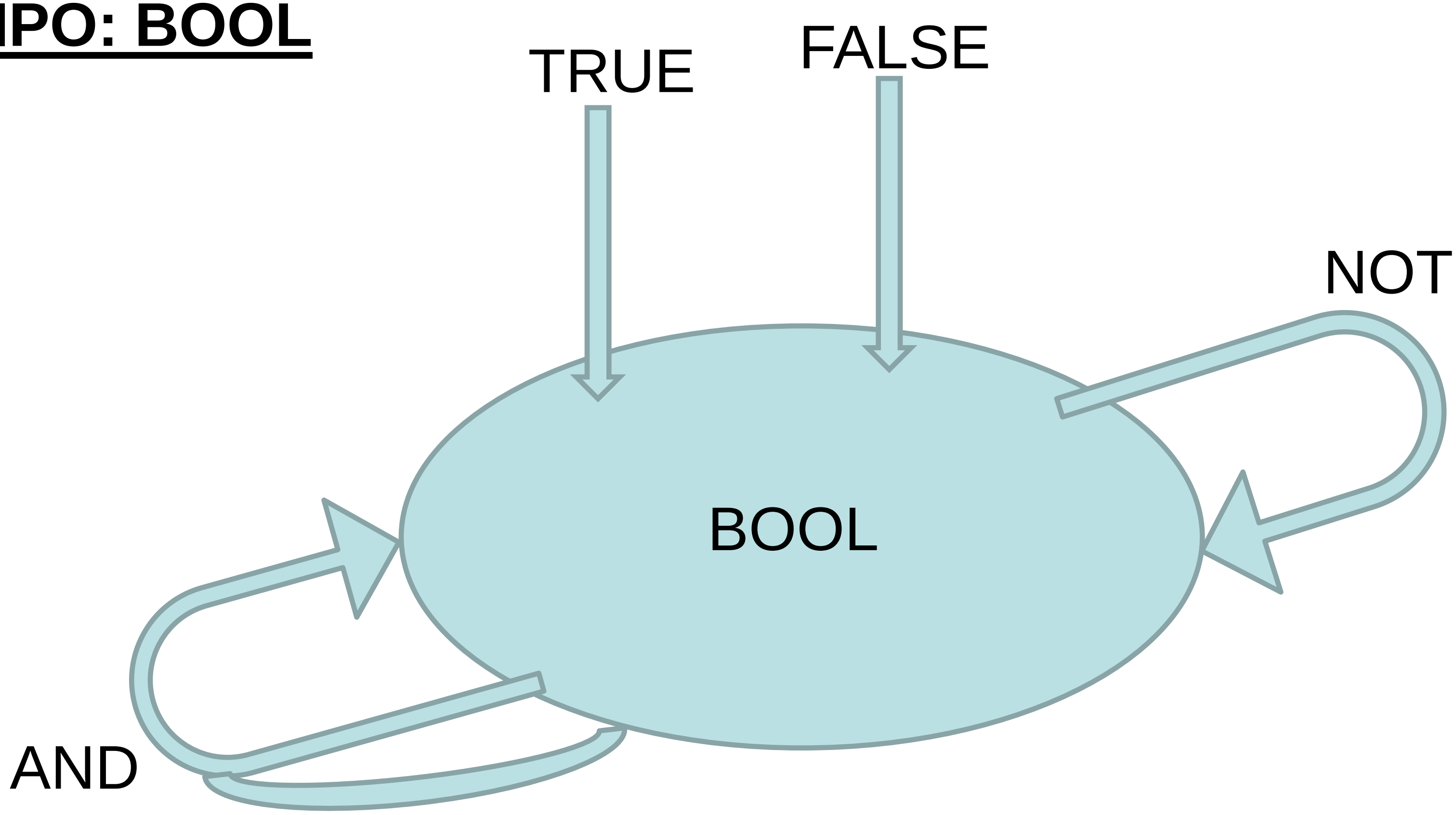
TRUE **:** → BOOL
FALSE **:** → BOOL
NOT **:** BOOL→ BOOL
AND **:** BOOL **x** BOOL→ BOOL

Semántica:

NOT(TRUE) ≡ FALSE
NOT(FALSE) ≡ TRUE
AND(TRUE,TRUE) ≡ TRUE
AND(FALSE,TRUE) ≡ FALSE
AND(TRUE,FALSE) ≡ FALSE
AND(FALSE,FALSE) ≡ FALSE

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

TIPO: BOOL



ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

<u>TIPO: BOOL</u>

Otras definiciones:

Sintaxis:

OR **:** BOOL **x** BOOL→ BOOL
IMP **:** BOOL **x** BOOL→ BOOL
SSI **:** BOOL **x** BOOL→ BOOL

Semántica: para todo a, b $\in$ BOOL

OR(a,b) $\equiv$ NOT(AND(NOT(a),NOT(b)))
IMP(a,b) $\equiv$ OR(NOT(a),b)
SSI(a,b) $\equiv$ AND(IMP(a,b),IMP(b,a))

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

<u>TIPO: BOOL</u> OPERACIONES

Sintaxis:

TRUE **:** → BOOL
FALSE **:** → BOOL
NOT **:** BOOL→ BOOL
AND **:** BOOL **x** BOOL→ BOOL
OR **:** BOOL **x** BOOL→ BOOL
IMP **:** BOOL **x** BOOL→ BOOL
SSI **:** BOOL **x** BOOL→ BOOL

Semántica: para todo a, b $\in$ BOOL

NOT(TRUE) $\equiv$ FALSE
NOT(FALSE) $\equiv$ TRUE
AND(TRUE,TRUE) $\equiv$ TRUE
AND(FALSE,TRUE) $\equiv$ FALSE
AND(TRUE,FALSE) $\equiv$ FALSE
AND(FALSE,FALSE) $\equiv$ FALSE

OR(a,b) $\equiv$ NOT(AND(NOT(a),NOT(b)))
IMP(a,b) $\equiv$ OR(NOT(a),b)
SSI(a,b) $\equiv$ AND(IMP(a,b),IMP(b,a))

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

TIPO: NAT

OPERACIONES

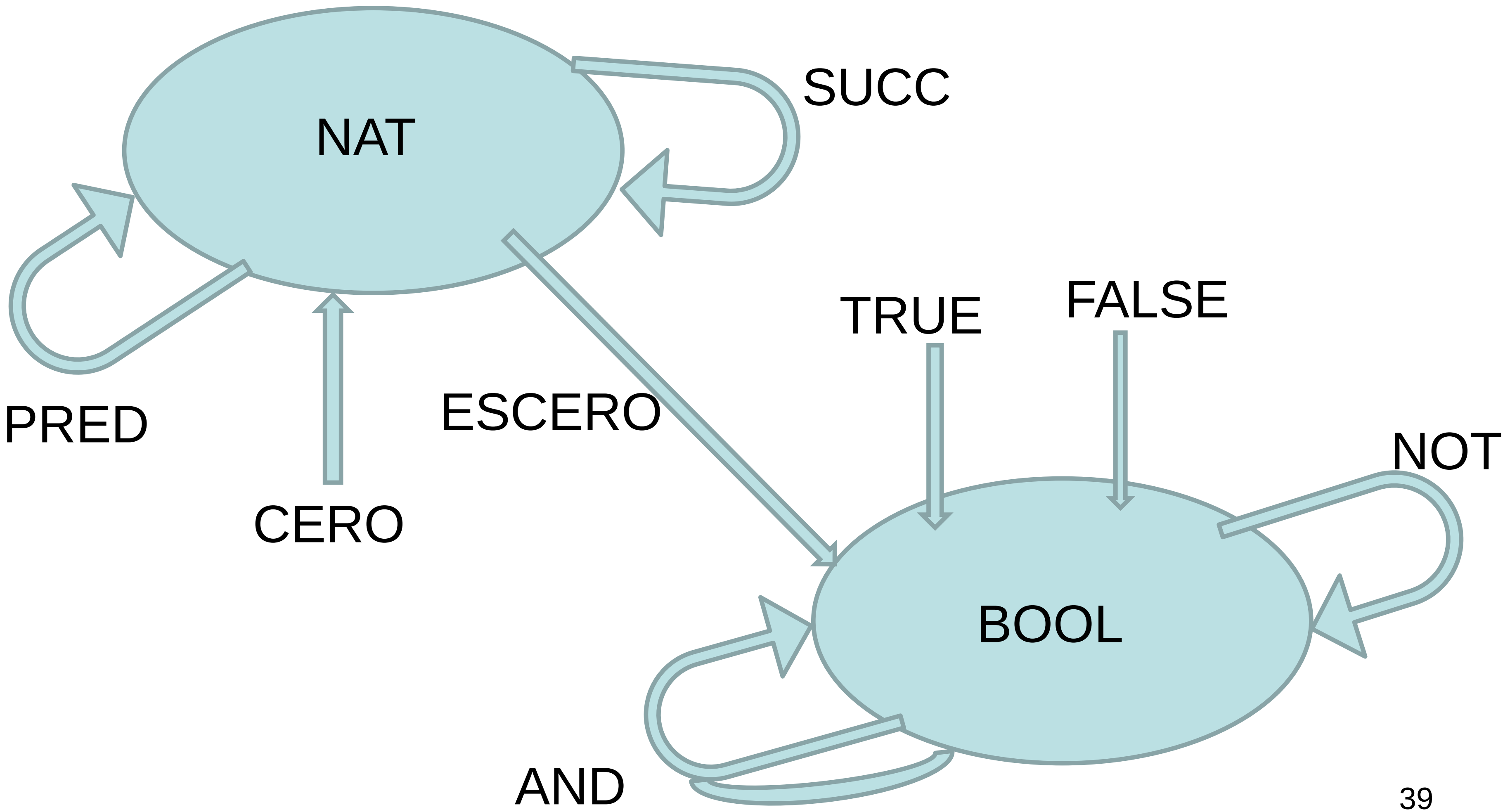
Sintaxis:

CERO : $\rightarrow$ NAT
SUCC : NAT$\rightarrow$ NAT
IGUALCERO : NAT $\rightarrow$ BOOL
PRED : NAT - {CERO} $\rightarrow$ NAT parcial

Semántica: Para todo x $\in$ NAT

IGUALCERO(CERO) $\equiv$ TRUE
IGUALCERO(SUCC(x)) $\equiv$ FALSE
PRED(SUCC(x)) $\equiv$ x

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS



ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

<u>TIPO: NAT</u>

OPERACIÓN ESPAR

Sintaxis:

ESPAR **:** NAT → BOOL

Semántica: Para todo x $\in$ NAT

ESPAR(CERO) $\equiv$ TRUE
ESPAR (SUCC(CERO)) $\equiv$ FALSE
ESPAR (SUCC (SUCC(x))) $\equiv$ ESPAR(x)

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

TIPO: NAT

OPERACIÓN IGUAL

Sintaxis:

IGUAL **:** NAT x NAT → BOOL

Semántica: Para todo x , y $\in$ NAT

IGUAL(CERO,CERO) $\equiv$ TRUE
IGUAL (CERO,SUCC(x)) $\equiv$ FALSE
IGUAL (SUCC(x),CERO) $\equiv$ FALSE
IGUAL (SUCC(x),SUCC(y)) $\equiv$ IGUAL (x,y)

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

TIPO: NAT

OPERACIÓN MAX

Sintaxis:

MAX **:** NAT x NAT → NAT

Semántica: Para todo x , y $\in$ NAT

MAX (CERO,CERO) $\equiv$ CERO
MAX (CERO,SUCC(x)) $\equiv$ SUCC(x)
MAX (SUCC(x),CERO) $\equiv$ SUCC(x)
MAX (SUCC(x),SUCC(y)) $\equiv$ SUCC (MAX(x,y))

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

TIPO: NAT

OPERACIÓN SUMA

Sintaxis:

SUMA : NAT x NAT → NAT

Semántica: Para todo x , y $\in$ NAT

SUMA (CERO,CERO) $\equiv$ CERO
SUMA (CERO,SUCC(x)) $\equiv$ SUCC(x)
SUMA (SUCC(x),CERO) $\equiv$ SUCC(x)
SUMA (SUCC(x),SUCC(y)) $\equiv$ SUCC(SUCC(SUMA(x,y)))

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

TIPO: NAT

OPERACIÓN SUMA en 2 axiomas

Sintaxis:

SUMA **:** NAT x NAT → NAT

Semántica: Para todo x , y $\in$ NAT

SUMA (CERO, x) $\equiv$ x

SUMA (SUCC(x), y) $\equiv$ SUCC(SUMA(x,y))

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

TIPO: NAT

OPERACIÓN MULT

Sintaxis:

MULT : NAT x NAT $\rightarrow$ NAT

Semántica: Para todo x , y $\in$ NAT

MULT (CERO,CERO) $\equiv$ CERO
MULT (CERO,SUCC(x)) $\equiv$ CERO
MULT (SUCC(x),CERO) $\equiv$ CERO
MULT (SUCC(x),SUCC(y)) $\equiv$
SUCC(SUMA(SUMA(MULT(x,y),x),y))

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

<u>TIPO: NAT</u>

OPERACIÓN MULT en 2 axiomas

Sintaxis:

MULT **:** NAT x NAT → NAT

Semántica: Para todo x , y $\in$ NAT

MULT (x, CERO) $\equiv$ CERO

MULT (x, SUCC(y)) $\equiv$ SUMA(MULT(x,y),x)

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

<u>TIPO: CADENA</u>

OPERACIONES

Sintaxis:

NULA **:** → CADENA

ESNULA **:** CADENA → BOOL

LARGO **:** CADENA → ENTERO ≥ 0

AGREGAR **:** CADENA X CHAR → CADENA

CONCAT **:** CADENA X CADENA → CADENA

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

Semántica: Para todo s,t $\in$ CADENA, $\forall$c $\in$ CHAR,

ESNULA(NULA) $\equiv$ TRUE

ESNULA(AGREGAR(s,c)) $\equiv$ FALSE

LARGO(NULA) $\equiv$ 0

LARGO(AGREGAR(s,c)) $\equiv$ LARGO(s) + 1

CONCAT(s,NULA) $\equiv$ s

CONCAT(s, AGREGAR(t,c)) $\equiv$ AGREGAR(CONCAT(s,t),c)

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

<u>TIPO: COMPLEJO</u>

OPERACIONES

Sintaxis:

ARMAR **:** REAL x REAL→ COMPLEJO

SUMA **:** COMPLEJO x COMPLEJO → COMPLEJO

RESTA **:** COMPLEJO x COMPLEJO → COMPLEJO

MULTIPLICA **:** COMPLEJO x COMPLEJO → COMPLEJO

DIVIDE **:** COMPLEJO x COMPLEJO → COMPLEJO U {indefinido}

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

INVERSO : COMPLEJO → COMPLEJO U {indefinido}

OPUESTO : COMPLEJO → COMPLEJO

PREAL : COMPLEJO → REAL

PIMAG : COMPLEJO → REAL

ESREAL: COMPLEJO → BOOL

ESIMAG : COMPLEJO → BOOL

CONJUGADO : COMPLEJO → COMPLEJO

IGUAL : COMPLEJO x COMPLEJO → BOOL

NORMA : COMPLEJO → REAL

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

Semántica: Para todo a, b, c, d $\in$ REAL,

SUMA(ARMAR(a,b), ARMAR(c,d)) $\equiv$ ARMAR (a+c,b+d)

RESTA(ARMAR(a,b), ARMAR(c,d)) $\equiv$ ARMAR (a-c,b-d)

MULTIPLICA(ARMAR(a,b), ARMAR(c,d)) $\equiv$ ARMAR (a*c-b*d, a*d+b*c)

DIVIDE(ARMAR(a,b), ARMAR(c,d)) $\equiv$ si c*c+d*d = 0 entonces

indefinido

sino

ARMAR ((a*c+b*d)/(c*c+d*d), (-a*d+b*c)/ (c*c+d*d))

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

INVERSO (ARMAR(a,b)) ≡ si a=0 AND b=0 entonces indefinido
sino ARMAR (a/(a*a+b*b),-b/(a*a+b*b))
OPUESTO (ARMAR(a,b)) ≡ ARMAR (-a,-b)
PREAL (ARMAR(a,b)) ≡ a
PIMAG (ARMAR(a,b)) ≡ b
ESREAL (ARMAR(a,b)) ≡ b=0
ESIMAG (ARMAR(a,b)) ≡ si a=0 AND b≠0 entonces TRUE sino FALSE

CONJUGADO(ARMAR(a,b)) ≡ ARMAR (a,-b)

IGUAL(ARMAR(a,b), ARMAR(c,d)) ≡ si a=c AND b=d entonces TRUE
sino FALSE

NORMA(ARMAR(a,b)) ≡ a*a + b*b

TIPOS ABSTRACTOS DE DATOS GENERICOS

- Los TADs genéricos representan colecciones de elementos todos del mismo tipo.
- Estos TADs definen un cierto comportamiento independiente del tipo de sus elementos.
- Para poder expresar genéricamente el tipo de los elementos se utilizan parámetros.
- De esta forma, se pueden construir ejemplares del TAD genérico utilizando otros TADs que cumplan con las restricciones del parámetro indicado en su especificación.

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

TIPO: VECTOR (ITEM)

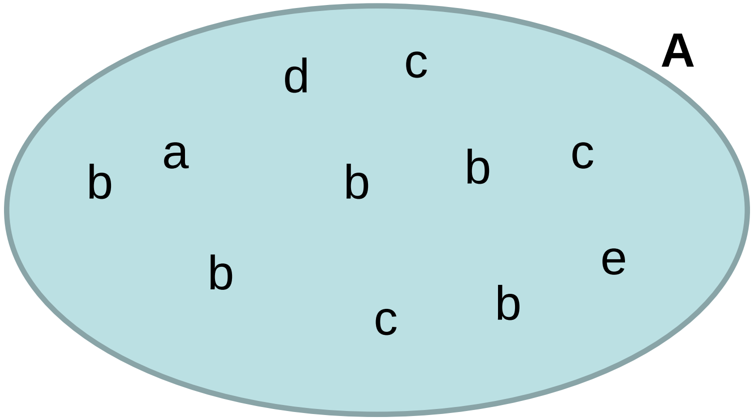
OPERACIONES

Sintaxis:
VECTORVACIO **:** → VECTOR
ALMACENAR **:** VECTOR **x** ENTERO **x** ITEM → VECTOR
OBTENER **:** VECTOR **x** ENTERO → ITEM U { indefinido}

Semántica: Para todo A $\in$ VECTOR, $\forall$ i,j $\in$ ENTERO, $\forall$x $\in$ ITEM
OBTENER(VECTORVACIO,i) $\equiv$ indefinido
OBTENER(ALMACENAR(A,i,x), j) $\equiv$ si i=j entonces
x
sino
OBTENER(A,j)

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

<u>TIPO: MULTICONJUNTO(ITEM)</u>



OPERACIONES

Sintaxis:

MULTICONJUNTOVACIO **:** → MULTICONJUNTO

ESVACIO **:** MULTICONJUNTO → BOOL

PERTENECE **:** MULTICONJUNTO **x** ITEM → BOOL

INSERTAR **:** MULTICONJUNTO **x** ITEM → MULTICONJUNTO

BORRAR **:** MULTICONJUNTO **x** ITEM → MULTICONJUNTO

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

Semántica: Para todo A $\in$ MULTICONJUNTO , $\forall$ i, j $\in$ ITEM.

ESVACIO(MULTICONJUNTOVACIO) $\equiv$ TRUE
ESVACIO(INSERTAR(A,i)) $\equiv$ FALSE

PERTENECE(MULTICONJUNTOVACIO,i) $\equiv$ FALSE
PERTENECE(INSERTAR(A,i),j) $\equiv$ si i**=**j entonces
TRUE
Sino
PERTENECE(A,j)

Otra manera de definir PERTENECE:

PERTENECE(MULTICONJUNTOVACIO,i) $\equiv$ FALSE
PERTENECE(INSERTAR(A,i),j) $\equiv$ (i**=**j) OR PERTENECE(A,j)

= representa la operación IGUALITEM

ESPECIFICACIONES ALGEBRAICAS - EJEMPLOS

Semántica: Para todo A $\in$ MULTICONJUNTO , $\forall$ i, j $\in$ ITEM.

BORRAR(MULTICONJUNTOVACIO,i) $\equiv$ MULTICONJUNTOVACIO
BORRAR(INSERTAR(A,i),j) $\equiv$ si i=j entonces
BORRAR (A,j)
sino
INSERTAR(BORRAR(A,j),i)

= *representa la operación IGUALITEM*
BORRAR borra todas las ocurrencias de un ITEM