

Tabla — TDA de tabla 2D de enteros

Tabla con dimensiones máximas fijas que almacena valores enteros para cálculos numéricos.

Validación: 0 errores, 1 warning. El único warning es un falso positivo de V1: `celdaValida` está definida con LHS de variable libre (no pattern-matchea los constructores). Esta versión `IMPORTS Natural, Integer` y movió `max` al tipo `Integer`, lo que eliminó el segundo warning que tenía la versión anterior.

Cambio respecto a la versión anterior — `IMPORTS Natural, Integer`:

- Tabla depende de dos tipos externos: `Natural` (índices y dimensiones) e `Integer` (valores almacenados). Ahora se importan explícitamente.
- `max(a, b)` — máximo de dos enteros — es una operación del tipo `Integer`, no de `Tabla`. Se movió fuera: ya no se declara ni axiomatiza en `Tabla`; se usa en `mayor` como operación importada (igual que potencia en `Polinomio`).
- Resultado: `Tabla` conoce solo SUS operaciones; la aritmética entera vive en `Integer`.

Diseño algebraico: dos constructores básicos y la funcionalidad pedida como operaciones derivadas.

- `crearTabla(maxFilas, maxColumnas)` — tabla vacía con dimensiones fijadas.
- `agregar(t, f, c, v)` — coloca el entero `v` en la celda `(f, c)`.
- `existe, valor, mayor, sumaColumna` — operaciones derivadas.

Toda tabla se representa como `agregar(agregar(...agregar(crearTabla(mf,mc), ...)))`. Las celdas se indexan desde 1.

Decisión clave — SIN sobrescritura: la precondition de `agregar` exige `not existe(t, f, c)`, así cada celda se escribe a lo sumo una vez. Esto hace que `mayor` y `sumaColumna` se definan por recursión simple: no hay que detectar valores "tapados" por escrituras posteriores. Si se quisiera permitir sobrescritura, `mayor` se complicaría bastante (habría que excluir el valor viejo de la celda reescrita).

1. Especificación

```
CLASS Tabla
IMPORTS Natural, Integer
BASIC CONSTRUCTORS crearTabla, agregar
EFFECTIVE
TYPE Tabla

OPERATIONS

-- — Constructores básicos —————

-- crearTabla(maxFilas, maxColumnas): tabla vacía.
crearTabla: Natural * Natural -> Tabla;

-- agregar(t, f, c, v): coloca v en la celda (f, c).
-- Precondición: La celda es válida y todavía no tiene valor.
agregar: Tabla(t) * Natural(f) * Natural(c) * Integer(v) -> Tabla
```

```

    pre: celdaValida(t, f, c) and not existe(t, f, c);

-- — Dimensiones —————
cantidadFilas:  Tabla -> Natural;
cantidadColumnas: Tabla -> Natural;

-- celdaValida(t, f, c): ¿(f, c) está dentro de las
-- dimensiones de t? (índices desde 1).
celdaValida: Tabla * Natural * Natural -> Boolean;

-- — Consultas sobre celdas —————

-- existe(t, f, c): ¿hay un entero almacenado en (f, c)?
existe: Tabla(t) * Natural(f) * Natural(c) -> Boolean
    pre: celdaValida(t, f, c);

-- valor(t, f, c): valor del entero en (f, c).
valor: Tabla(t) * Natural(f) * Natural(c) -> Integer
    pre: existe(t, f, c);

-- — Consultas agregadas —————

-- tieneValores(t): ¿la tabla tiene al menos un valor?
tieneValores: Tabla -> Boolean;

-- mayor(t): el mayor entero almacenado en la tabla.
mayor: Tabla(t) -> Integer
    pre: tieneValores(t);

-- sumaColumna(t, c): suma de los enteros de la columna c.
sumaColumna: Tabla(t) * Natural(c) -> Integer
    pre: (c >= 1) and (c <= cantidadColumnas(t));

-- Nota: `max: Integer * Integer -> Integer` (máximo de dos
-- enteros) es una operación del tipo Integer, importada vía
-- IMPORTS Integer, así que NO se declara aquí.

AXIOMS  t: Tabla;  f, f1, c, c1, mf, mc: Natural;  v: Integer;

-- —————
-- Dimensiones: fijadas por crearTabla, invariantes ante agregar.
-- —————

cantidadFilas(crearTabla(mf, mc))    = mf;
cantidadFilas(agregar(t, f, c, v))    = cantidadFilas(t);

cantidadColumnas(crearTabla(mf, mc)) = mc;
cantidadColumnas(agregar(t, f, c, v)) = cantidadColumnas(t);

-- —————
-- celdaValida: (f, c) en [1..maxFilas] x [1..maxColumnas].
-- —————

celdaValida(t, f, c) =
    (f >= 1) and (f <= cantidadFilas(t))
    and (c >= 1) and (c <= cantidadColumnas(t));

-- —————
-- existe: ¿la celda (f, c) tiene un valor?
-- —————

existe(crearTabla(mf, mc), f, c) = False;

((f1 = f) and (c1 = c)) implies

```

```

    existe(agregar(t, f1, c1, v), f, c) = True;

not ((f1 = f) and (c1 = c)) implies
    existe(agregar(t, f1, c1, v), f, c) = existe(t, f, c);

-- -----
-- valor: entero almacenado en (f, c).
-- La precondition existe(t, f, c) excluye el caso crearTabla.
-- -----
((f1 = f) and (c1 = c)) implies
    valor(agregar(t, f1, c1, v), f, c) = v;

not ((f1 = f) and (c1 = c)) implies
    valor(agregar(t, f1, c1, v), f, c) = valor(t, f, c);

-- -----
-- tieneValores: ¿hay al menos un valor almacenado?
-- -----
tieneValores(crearTabla(mf, mc)) = False;
tieneValores(agregar(t, f, c, v)) = True;

-- -----
-- mayor: el mayor entero almacenado.
-- La precondition tieneValores(t) excluye el caso crearTabla.
-- Sin sobrescritura, cada agregar aporta un valor distinto.
-- -----
not tieneValores(t) implies
    mayor(agregar(t, f, c, v)) = v;

tieneValores(t) implies
    mayor(agregar(t, f, c, v)) = max(v, mayor(t));

-- -----
-- sumaColumna: suma de los enteros de la columna c.
-- -----
sumaColumna(crearTabla(mf, mc), c) = 0;

(c1 = c) implies
    sumaColumna(agregar(t, f, c1, v), c) = v + sumaColumna(t, c);

not (c1 = c) implies
    sumaColumna(agregar(t, f, c1, v), c) = sumaColumna(t, c);

```

END-CLASS

2. Operaciones

Operación	Tipo	Axiomas
crearTabla	constructor básico	—
agregar	constructor básico <i>pre: celda válida ∧ no ocupada</i>	—
cantidadFilas / cantidadColumnas	derivada	2 c/u — base sobre crearTabla + invariante ante agregar

celdaValida	auxiliar	1 — definición extensional vía las dimensiones
existe	derivada <i>pre: celda válida</i>	3 — crearTabla $\rightarrow$ False + 2 implies sobre agregar
valor	derivada <i>pre: existe(t,f,c)</i>	2 — implies sobre agregar según coincida la celda
tieneValores	auxiliar	2 — crearTabla $\rightarrow$ False, agregar $\rightarrow$ True
mayor	derivada <i>pre: tieneValores(t)</i>	2 — implies según sea el primer valor o no
sumaColumna	derivada <i>pre: columna válida</i>	3 — crearTabla $\rightarrow$ 0 + 2 implies según coincida la columna
max	importada — operación del tipo Integer (IMPORTS Integer)	— (definida y axiomatizada en Integer, no aquí)

3. Ejemplo: tabla 3×3

Construimos una tabla 3×3 con valores en (1,1)=8, (2,1)=5, (1,2)=20:

```
t = agregar(agregar(agregar(crearTabla(3, 3), 1, 1, 8), 2, 1, 5), 1, 2, 20)
```

	col 1	col 2	col 3
fila 1	8	20	.
fila 2	5	.	.
fila 3	.	.	.

(. = celda sin valor)

Evaluación de las operaciones

Expresión	Resultado	Por qué
cantidadFilas(t)	3	Fijado por crearTabla(3, 3), invariante ante agregar.
existe(t, 1, 1)	True	Se hizo agregar(..., 1, 1, 8).
existe(t, 3, 3)	False	Ningún agregar tocó (3, 3); se llega hasta crearTabla $\rightarrow$ False.
valor(t, 2, 1)	5	El agregar(..., 2, 1, 5) coincide con la celda consultada.
valor(t, 1, 2)	20	Idem con (1, 2).
mayor(t)	20	$\max(20, \max(5, 8)) = \max(20, 8) = 20$.

<code>sumaColumna(t, 1)</code>	13	Celdas de columna 1: (1,1)=8 y (2,1)=5 $\rightarrow 5 + 8 = 13$.
<code>sumaColumna(t, 2)</code>	20	Solo (1,2)=20.
<code>sumaColumna(t, 3)</code>	0	Columna 3 vacía; se llega a <code>crearTabla</code> $\rightarrow 0$.

Traza de mayor(t)

Paso	Reescritura
1	<code>mayor(agregar(t2, 1, 2, 20))</code> con <code>t2 = agregar(agregar(crearTabla(3,3),1,1,8),2,1,5)</code> . <code>tieneValores(t2) = True</code> $\rightarrow \max(20, \text{mayor}(t2))$
2	<code>mayor(agregar(t1, 2, 1, 5))</code> con <code>t1 = agregar(crearTabla(3,3),1,1,8)</code> . <code>tieneValores(t1) = True</code> $\rightarrow \max(5, \text{mayor}(t1))$
3	<code>mayor(agregar(crearTabla(3,3), 1, 1, 8))</code> . <code>tieneValores(crearTabla(3,3)) = False</code> $\rightarrow 8$
4	Sustituyendo: $\max(20, \max(5, 8)) = \max(20, 8) = 20$

4. Propiedades garantizadas

Sin sobrescritura $\Rightarrow$ conteo único: como `agregar` exige `not existe(t, f, c)`, cada celda aparece a lo sumo una vez en la cadena de `agregar`. Por eso `sumaColumna` suma cada celda exactamente una vez, y `mayor` compara cada valor exactamente una vez — ambas recursiones son correctas sin lógica extra de "des-duplicación".

Parcialidad bien delimitada: `valor` y `mayor` son operaciones parciales. `valor(t, f, c)` exige `existe(t, f, c)` — por eso no tiene axioma para `crearTabla` (en una tabla recién creada ninguna celda existe). `mayor(t)` exige `tieneValores(t)` — no tiene sentido el mayor de una tabla vacía. Las precondiciones documentan exactamente el dominio de cada operación.

Dimensiones inmutables: `cantidadFilas` y `cantidadColumnas` quedan fijadas por `crearTabla` y son invariantes ante cualquier `agregar` (axioma `cantidadFilas(agregar(t,...)) = cantidadFilas(t)`). Una tabla nunca cambia de tamaño una vez creada.

Separación de responsabilidades vía IMPORTS: `max` no figura en OPERATIONS ni en AXIOMS de `Tabla`. Es una operación del tipo `Integer`, importada. `Tabla` la usa en el axioma de `mayor` pero su definición y axiomas (`max(a,b)=a` si $a \geq b$, si no b) son responsabilidad de `Integer`. Cada TDA conoce solo sus propias operaciones; `Natural` e `Integer` aportan la aritmética.

Spec validada con el analyzer del workspace: 0 errores, 1 warning (V1 false positive sobre `celdaValida`, operación definida extensionalmente que no pattern-matchea los constructores de `Tabla`).