



# Trabajo Práctico 1

## Especificación de TADs

16 de abril de 2025

AED

### Grupo Almendra

| Integrante                    | LU      | Correo electrónico     |
|-------------------------------|---------|------------------------|
| Puodziunas, Bruno             | 309/23  | puodziunasb@gmail.com  |
| Ozzan Prieto, Luana Constanza | 1444/23 | luanaozzan@gmail.com   |
| Piputto, Lucas Ignacio        | 1345/24 | lucaspiputto@gmail.com |
| Yu, Patricio                  | 1247/24 | yupatricio0@gmail.com  |



**Facultad de Ciencias Exactas y Naturales**  
Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2610 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (+54 11) 4576-3300

<http://www.exactas.uba.ar>

## 1. Renombres de tipo

En lugar de *tuplas* utilizaremos *structs* para una mejor legibilidad de la especificacion a lo largo del documento.

### 1.1. Tipo: *usuario*

El usuario consta de un *id* y una cantidad de *monedas*, ambos enteros positivos, realizaremos el siguiente renombre de tipo:

*usuario* **ES** struct  $\langle id : \mathbb{Z}, monedas : \mathbb{Z} \rangle$

### 1.2. Tipo: *transaccion*

Por definicion del problema sabemos que una transaccion es una cuadru-pla que consta de un *id\_transaccion*, *id\_comprador*, *id\_vendedor*, *monto*, todos enteros positivos, realizaremos el siguiente renombre de tipo:

*transaccion* **ES** struct  $\langle id\_transaccion : \mathbb{Z}, id\_comprador : \mathbb{Z}, id\_vendedor : \mathbb{Z}, monto : \mathbb{Z} \rangle$

### 1.3. Tipo: *bloque*

Un bloque tiene un *id\_bloque* que se representa con un entero positivo y puede tener hasta 50 *transacciones*:

*bloque* **ES** struct  $\langle id\_bloque : \mathbb{Z}, transacciones : seq\langle transaccion \rangle \rangle$

## 2. Consideraciones

### 2.1. Sobre los id de *usuario*, *transaccion* y *bloque*

Por enunciado del problema sabemos que los ids de *transaccion* y *bloque* son consecutivos, tomamos como punto de inicio el  $id = 0$  para ambos.

Para el id del *usuario* consideramos que puede ser cualquier entero positivo, y reservamos el  $id = 0$  para el usuario emisor de monedas que tendrá 3000 monedas de inicio y las usará hasta agotarlas cada primer transacción de un nuevo bloque.

Además, el id de *transaccion* es relativo a cada *bloque*, es decir por ejemplo, el  $id = 0$  de la transaccion del bloque con  $id = 5$  es distinto a la transacción con  $id = 0$  del bloque con  $id = 12$ .

### 3. Definicion del TAD

```

TAD Berretacoin {
  obs bloques: seq⟨bloque⟩
  obs usuarios: seq⟨usuario⟩

  proc nuevoBerretacoin (in usuarios: seq⟨usuario⟩) : Berretacoin {
    requiere { |usuarios| ≥ 1 }
    requiere { (∀i : Z) ((0 ≤ i < |usuarios|) →L ((usuarios[i].id >
0) ∧L ¬(hayRepetidos(usuarios, usuarios[i])) ∧L (usuarios[i].monedas = 0))) }
    asegura { res.bloques = ⟨⟩ }
    /* Se crea el usuario de id 0 en la primera posición de la lista de usuarios y se le concatena la lista de
    usuarios recibida */
    asegura { res.usuarios = ⟨⟨id : 0, monedas : 3000⟩⟩ ++ usuarios }
  }

  proc agregarBloque (inout b: Berretacoin, in transacciones: seq⟨transaccion⟩) {
    /* Cantidad limite de transacciones = 50 */
    requiere { b = B0 }
    /* Los ids de las transacciones son consecutivos arrancando desde el 0, sino no tiene transacciones */
    requiere { |transacciones| ≤ 50 }
    requiere { ((transacciones[0].id = 0) ∧ (∀i : Z) ((0 ≤ i < |transacciones| - 1) →L
(transacciones[i].id + 1 = transacciones[i + 1].id))) ∨ (transacciones = ⟨⟩) }
    requiere { (∀i : Z) ((0 ≤ i < |transacciones|) →L (transacciones[i].id_comprador ≠
transacciones[i].id_vendedor)) }
    /* Si en transacciones aparece un comprador que en B0 no está como usuario, el mismo debe aparecer
    como vendedor primero */
    requiere
    { (∀i : Z) ((0 ≤ i < |transacciones|) →L ((¬perteneceUsuario(B0, transacciones[i].comprador)) →L
((∃j : Z) ((0 ≤ j < i) →L (transacciones[i].comprador = transacciones[j].vendedor)))) ) }
    /* Ningún usuario gasta más de lo que tiene (se verifica transacción a transacción) */
    requiere { (∀i : Z) ((0 ≤ i < |b.usuarios|) →L ((∀j : Z) (((0 ≤ j < |transacciones|) ∧ ((b.usuarios[i].id =
transacciones[j].id_comprador) ∨ (b.usuarios[i].id = transacciones[j].id_vendedor)) →L
((ifThenElse(perteneceUsuario(B0.usuarios, b.usuarios[i].id), B0.usuarios[i].monto, 0) +
montoRecibido(transacciones, usuarios[i], j) - montoGastado(transacciones, usuarios[i], j)) ≥ 0)))) ) }
    /* El vendedor es algún usuario arbitrario siempre distinto para todas las operaciones de creación de
    berretacoin y además se emite una moneda en la primera transacción de cada bloque para los primeros
    3000 bloques, y luego no se emitirán más monedas */
    requiere { (|B0.bloques| ≤ 3000) →L (
      (¬transacciones = ⟨⟩) ∧L
      (transacciones[0].id_comprador = 0) ∧L
      (transacciones[0].monto = 1) ∧L
      (noGanoMonedaGratis(B0.bloques, transacciones[0].id_vendedor)) ∧L
      ((∀i : Z) ((1 ≤ i < |transacciones|) →L ((transacciones[i].comprador_id ≠ 0) ∧L
(transacciones[i].vendedor_id ≠ 0)))) )
    }

    /* Si ya se emitieron todas las berretacoin posibles, entonces no se seguirá emitiendo */
    requiere { (|B0.bloques| > 3000) →L ((∀i : Z) ((0 ≤ i < |transacciones|) →L
((transacciones[i].comprador_id ≠ 0) ∧L (transacciones[i].vendedor_id ≠ 0)))) }
    /* El bloque nuevo se agrega a la lista de bloques de berretacoin */
    asegura { b.bloques = B0 ++ ⟨id_bloque : |B0.bloques|, transacciones : transacciones⟩ }
    /* Todos los vendedores que están en transacciones y no están en B0.usuarios, están en b.usuarios */
    asegura
    { (∀i : Z) (((0 ≤ i < |transacciones|) ∧L ¬(perteneceUsuario(B0.usuarios, transacciones[i].vendedor)))
→L (perteneceUsuario(b.usuarios, transacciones[i].vendedor))) }
    /* No van a haber ids repetidos en b.usuarios */
    asegura { (∀i : Z) ((0 ≤ i < |b.usuarios|) →L ¬(hayRepetidos(b.usuarios, b.usuarios[i]))) }
    /* Actualizamos de ser necesario la cantidad de monedas de los usuarios */
    asegura { (∀i : Z) ((0 ≤ i < |b.usuarios|) →L (
      /* Caso: nuevo usuario y fue vendedor (previamente ya verificamos que sea vendedor antes que
      comprador, es decir que no gaste antes de tener) */

```

```

    ((¬(perteneceUsuario( $B_0.usuarios, b.usuarios[i].id$ ))) ∧
    (vendedorPerteneceATransaccion( $b.usuarios[i], transacciones$ ))  $\rightarrow_L (b.usuarios[i].monedas =$ 
    montoRecibido( $transacciones, b.usuarios[i]$ ) − montoGastado( $transacciones, b.usuario[i]$ )))
     $\vee_L$ 
    /* Caso: era usuario y no recibio más monedas, es decir a lo sumo gastó */
    ((perteneceUsuario( $B_0.usuarios, b.usuarios[i].id$ )) ∧
    ((¬(vendedorPerteneceATransaccion( $b.usuarios[i], transaccion$ )))  $\rightarrow_L (b.usuarios[i].id =$ 
     $B_0.usuarios[i].id \wedge b.usuarios[i].monedas =$ 
     $B_0.usuarios[i].monto - montoGastado(transacciones, b.usuarios[i])$ )))
     $\vee_L$ 
    /* Caso: era usuario y recibió más monedas */
    ((perteneceUsuario( $B_0.usuarios, b.usuarios[i].id$ )) ∧
    (vendedorPerteneceATransaccion( $b.usuarios[i], transaccion$ ))  $\rightarrow_L (b.usuarios[i].id =$ 
     $B_0.usuarios[i].id \wedge b.usuarios[i].monedas = B_0.usuarios[i].monedas +$ 
    montoRecibido( $transacciones, b.usuarios[i]$ ) − montoGastado( $transacciones, b.usuarios[i]$ )))
    /* El caso de nuevo usuario y no fue vendedor no se contempla en el asegura pues o no sería un
    nuevo usuario o no cumple con los requiere */
  }
}

proc maximosTenedores (in b: Berretacoin) : seq<usuario> {
  asegura
  { $(\forall i : \mathbb{Z}) ((0 \leq i < |b|) \rightarrow_L ((b.usuarios[i] \in res) \iff (esMaximoTenedor(b.usuarios, b.usuarios[i]))))$ }
}

proc montoMedio (in b: Berretacoin) :  $\mathbb{R}$  {
  requiere {True}
  asegura { $|b.bloques| = 0 \rightarrow_L res = 0$ }
  asegura {cantTotalDeOperacionesBloques( $b.bloques$ ) = 0  $\rightarrow_L res = 0$ }
  asegura
  { $res = (montoTotalOperadoBloques(b.bloques)) / (cantTotalDeOperacionesBloques(b.bloques))$ }
}

proc cotizacionAPesos (in cotizaciones: seq< $\mathbb{Z}$ >) : seq< $\mathbb{Z}$ > {
  requiere { $|cotizaciones| = |b.bloques|$ }
  requiere { $(\forall i : \mathbb{Z}) ((0 \leq i < |cotizaciones|) \rightarrow_L (cotizaciones[i] > 0))$ }
  asegura { $|res| = |cotizaciones|$ }
  asegura { $(\forall i : \mathbb{Z}) ((0 \leq i < |cotizaciones|) \rightarrow_L (res[i] =$ 
   $cotizaciones[i] * montoTotalOperado(bloques[i].transacciones)))$ }
}

pred hayRepetidos (s: seq<usuario>, u: usuario) {
   $\left( \sum_{i=0}^{|s|-1} ifThenElseFi(s[i].id = u.id, 1, 0) \right) \geq 2$ 
}

pred nuevoUsuarioValido (s: seq<usuario>, u: usuario) {
   $(u.monedas = 0) \wedge ((\forall i : \mathbb{Z}) ((0 \leq i < |s|) \rightarrow_L (s[i].id \neq u.id)))$ 
}

pred esMaximoTenedor (s: seq<usuario>, u: usuario) {
   $(\forall i : \mathbb{Z}) ((0 \leq i < |s|) \rightarrow_L (u.monedas \geq s[i].monedas))$ 
}

pred perteneceUsuario (s: seq<usuario>, id_u:  $\mathbb{Z}$ ) {
   $(\exists i : \mathbb{Z}) ((0 \leq i < |s|) \wedge_L (s[i].id = id_u))$ 
}

pred noGanoMonedaGratis (bloques: seq<bloque>, id_vendedor:  $\mathbb{Z}$ ) {
   $(\exists i : \mathbb{Z}) ((0 \leq i < |bloques|) \rightarrow_L (bloques[i].transacciones[0].vendedor \neq id_vendedor))$ 
}

pred vendedorPerteneceATransaccion (usuario: usuario, transacciones: seq<transaccion>) {
   $(\exists i : \mathbb{Z}) ((0 \leq i < |transacciones|) \wedge_L (usuario.id = transacciones[i].vendedor))$ 
}

```

```

aux montoRecibido (transacciones: seq⟨transaccion⟩, u: usuario, idx: ℤ) : ℤ =
  (∑k=0idx ifThenElse(transacciones[k].id_vendedor = u.id, transacciones[j].monto, 0));
aux montoGastado (transacciones: seq⟨transaccion⟩, u: usuario, idx: ℤ) : ℤ =
  (∑k=0idx ifThenElse(transacciones[k].id_comprador = u.id, transacciones[j].monto, 0));
aux montoTotalOperado (transacciones: seq⟨transaccion⟩) : ℤ =
  (∑k=0|transacciones|-1 transacciones[k].monto);
aux montoTotalOperadoBloques (bloques: seq⟨bloque⟩) : ℤ =
  (∑k=0|bloques|-1 montoTotalOperado(transaccionesDesde(ifThenElse(b.bloques[k].id_bloque ≤ 3000, 1, 0),
    b.bloques[k].transacciones)));
aux cantTotalDeOperacionesBloques (bloques: seq⟨bloque⟩) : ℤ =
  (∑k=0|bloques|-1 |b.bloques[k].transacciones| - ifThenElse(b.bloques[k].id_bloque ≤ 3000, 1, 0));
aux transaccionesDesde (indice: ℤ, transacciones: seq⟨transaccion⟩) : seq⟨transaccion⟩ =
  (subseq(transacciones, indice, |transacciones|));
}

```