



Trabajo Práctico 1

Especificación de TADs

12 de mayo 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 especificación 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*, *id_comprador*, *id_vendedor*, *monto*, todos enteros positivos, realizaremos el siguiente renombre de tipo:

transaccion **ES** struct $\langle id : \mathbb{Z}, id_comprador : \mathbb{Z}, id_vendedor : \mathbb{Z}, monto : \mathbb{Z} \rangle$

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 $\langle id_bloque : \mathbb{Z}, transacciones : seq(transaccion) \rangle$

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 transacción 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  $\{(\forall i : \mathbb{Z}) (0 \leq i < |usuarios| \rightarrow_L (usuarios[i].id > 0 \wedge_L \neg hayRepetidos(usuarios, usuarios[i]) \wedge_L usuarios[i].monedas = 0))\}$ 
    asegura  $\{res.bloques = \langle \rangle\}$ 
    /* 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  $\{res.usuarios = \langle id : 0, monedas : 3000 \rangle ++ usuarios\}$ 
  }

  proc agregarBloque (inout b: Berretacoin, in transacciones: seq<transaccion>) {
    requiere  $\{b = B_0\}$ 
    /* Cantidad limite de transacciones = 50 */
    requiere  $\{|transacciones| \leq 50\}$ 
    /* Los ids de las transacciones son consecutivos arrancando desde el 0, sino no tiene transacciones */
    requiere  $\{transacciones = \langle \rangle \vee_L (transacciones[0].id = 0 \wedge (\forall i : \mathbb{Z}) (0 \leq i < |transacciones| - 1 \rightarrow_L transacciones[i].id + 1 = transacciones[i + 1].id))\}$ 
    /* Un usuario no puede ser comprador y vendedor a la vez en la misma transacción */
    requiere  $\{(\forall i : \mathbb{Z}) (0 \leq i < |transacciones| \rightarrow_L transacciones[i].id_comprador \neq transacciones[i].id_vendedor)\}$ 
    /* Los montos de todas las transacciones son positivos */
    requiere  $\{(\forall i : \mathbb{Z}) (0 \leq i < |transacciones| \rightarrow_L transacciones[i].monto > 0)\}$ 
    /* Si en transacciones aparece un comprador que no era usuario, el mismo debe aparecer como vendedor en alguna transacción previa */
    requiere  $\{(\forall i : \mathbb{Z}) (0 \leq i < |transacciones| \rightarrow_L (\neg perteneceUsuario(b.usuarios, transacciones[i].id_comprador) \rightarrow_L (\exists j : \mathbb{Z}) (0 \leq j < i \wedge_L transacciones[j].id_vendedor = transacciones[i].id_comprador)))\}$ 
    /* Ningún usuario ya existente gasta más de lo que tiene (se verifica transacción a transacción) */
    requiere  $\{(\forall i : \mathbb{Z}) (0 \leq i < |b.usuarios| \rightarrow_L (\forall j : \mathbb{Z}) ((0 \leq j < |transacciones| \wedge_L b.usuarios[i].id = transacciones[j].id_comprador) \rightarrow_L b.usuarios[i].monto + montoRecibido(transacciones, usuarios[i].id, j) - montoGastado(transacciones, usuarios[i].id, j) \geq 0))\}$ 
    /* Verifica que ningún usuario nuevo gastó más de lo que adquirió */
    requiere  $\{(\forall i : \mathbb{Z}) ((0 \leq i < |transacciones| \wedge_L \neg perteneceUsuario(b.usuarios, transacciones[i].id_comprador)) \rightarrow_L montoRecibido(transacciones, transacciones[i].id_comprador, i) - montoGastado(transacciones, transacciones[i].id_comprador, i) \geq 0)\}$ 
    /* Todos los ids de los vendedores son mayores a 0 */
    requiere  $\{(\forall i : \mathbb{Z}) (0 \leq i < |transacciones| \rightarrow_L transacciones[i].vendedor > 0)\}$ 
    /* Todos los ids de los compradores son mayores a 0 excepto el primer comprador que depende de cada caso */
    requiere  $\{(\forall i : \mathbb{Z}) (1 \leq i < |transacciones| \rightarrow_L transacciones[i].comprador > 0)\}$ 
    /* 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  $\{|b.bloques| < 3000 \rightarrow_L |transacciones| > 0 \wedge_L transacciones[0].id_comprador = 0 \wedge_L transacciones[0].monto = 1 \wedge_L noGanoMonedaGratis(b.bloques, transacciones[0].id_vendedor)\}$ 
    /* Si ya se emitieron todas las berretacoin posibles, entonces no se seguirá emitiendo */
    requiere  $\{|b.bloques| \geq 3000 \wedge |transacciones| > 0 \rightarrow_L transacciones[0].id_comprador > 0\}$ 
    /* El bloque nuevo se agrega a la lista de bloques de berretacoin */
    asegura  $\{b.bloques = B_0 ++ \langle id_bloque : |B_0.bloques|, transacciones : transacciones \rangle\}$ 
    /* Todos los usuarios que están en transacciones estan en b.usuarios */
    asegura  $\{(\forall i : \mathbb{Z}) (0 \leq i < |transacciones| \rightarrow_L perteneceUsuario(b.usuarios, transacciones[i].id_vendedor) \wedge_L$ 

```

```

perteneceUsuario(b.usuarios, transacciones[i].id_comprador))}
/* El orden de los usuarios sigue siendo el mismo orden en la lista */
asegura {(∀i : ℤ) (0 ≤ i < |B₀.usuarios| →L b.usuarios[i].id = B₀.usuarios[i].id)}
/* 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(B₀.usuarios, b.usuarios[i].id) ∧
  vendedorPerteneceATransaccion(b.usuarios[i], transacciones)) →L b.usuarios[i].monedas =
  montoRecibido(transacciones, b.usuarios[i].id, |transacciones| - 1) -
  montoGastado(transacciones, b.usuario[i].id, |transacciones| - 1))
  ∨L
  /* Caso: era usuario (previamente en los requiere ya verificamos que no gaste mas de lo que tenia
  transaccion a transaccion) */
  (perteneceUsuario(B₀.usuarios, b.usuarios[i].id) →L b.usuarios[i].monedas =
  B₀.usuarios[i].monedas + montoRecibido(transacciones, b.usuarios[i].id, |transacciones| - 1) -
  montoGastado(transacciones, b.usuarios[i].id, |transacciones| - 1)))
  /* 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
  {(∀i : ℤ) (0 ≤ i < |b| →L (b.usuarios[i] ∈ res ⇔ esMaximoTenedor(b.usuarios, b.usuarios[i])))}
  asegura {(∀i : ℤ) (0 ≤ i < |res| →L ¬hayRepetidos(res, res[i]))}
}

proc montoMedio (in b: Berretacoin) : ℝ {
  requiere {True}
  asegura {|b.bloques| = 0 →L res = 0}
  asegura {cantTotalDeOperacionesBloques(b.bloques) = 0 →L res = 0}
  asegura {res = montoTotalOperadoBloques(b.bloques)/cantTotalDeOperacionesBloques(b.bloques)}
}

proc cotizacionAPesos (in b: Berretacoin, in cotizaciones: seq<ℤ>) : seq<ℤ> {
  requiere {|cotizaciones| = |b.bloques|}
  requiere {(∀i : ℤ) (0 ≤ i < |cotizaciones| →L cotizaciones[i] > 0)}
  asegura {|res| = |cotizaciones|}
  asegura {(∀i : ℤ) (0 ≤ i < |cotizaciones| →L res[i] =
  cotizaciones[i] * montoTotalOperado(b.bloques[i].transacciones))}
}

pred hayRepetidos (s: seq<usuario>, u: usuario) {
  (∑i=0|s|-1 ifThenElseFi(s[i].id = u.id, 1, 0)) ≥ 2
}

pred nuevoUsuarioValido (s: seq<usuario>, u: usuario) {
  u.monedas = 0 ∧ (∀i : ℤ) (0 ≤ i < |s| →L s[i].id ≠ u.id)
}

pred esMaximoTenedor (s: seq<usuario>, u: usuario) {
  (∀i : ℤ) (0 ≤ i < |s| →L u.monedas ≥ s[i].monedas)
}

pred perteneceUsuario (s: seq<usuario>, id_u: ℤ) {
  (∃i : ℤ) (0 ≤ i < |s| ∧L s[i].id = id_u)
}

pred noGanoMonedaGratis (bloques: seq<bloque>, id_vendedor: ℤ) {
  (∀i : ℤ) (0 ≤ i < |bloques| →L bloques[i].transacciones[0].id_vendedor ≠ id_vendedor)
}

```

```

pred vendedorPerteneceATransaccion (usuario: usuario, transacciones: seq⟨transaccion⟩) {
  (∃i : ℤ) (0 ≤ i < |transacciones| ∧  $L$  usuario.id = transacciones[i].id_vendedor)
}

aux montoRecibido (transacciones: seq⟨transaccion⟩, id_u: ℤ, idx: ℤ) : ℤ =
  ∑k=0idx ifThenElse(transacciones[k].id_vendedor = id_u, transacciones[j].monto, 0);

aux montoGastado (transacciones: seq⟨transaccion⟩, id_u: ℤ, idx: ℤ) : ℤ =
  ∑k=0idx ifThenElse(transacciones[k].id_comprador = id_u, 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|);
}

```