DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 1

Especificación de TADs

17 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 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 $\langle id\_bloque : \mathbb{Z}, transacciones : seq\langle transaccion \rangle \rangle$

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 emitidor 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: $seq\langle bloque\rangle$

obs usuarios: $seq\langle usuario\rangle$

`proc nuevoBerretacoin` (in usuarios: $seq\langle usuario\rangle$) : Berretacoin {

`requiere` $\{(\forall i : \mathbb{Z})\ ((0 \leq i < |usuarios|) \longrightarrow_L ((usuarios[i].id > 0) \wedge_L \neg(hayRepetidos(usuarios,\ usuarios[i])) \wedge_L (usuarios[i].monedas = 0)))\}$

`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` $\{res.usuarios = \langle\langle id : 0,\ monedas : 3000\rangle\rangle\ ++\ usuarios\}$

}

`proc agregarBloque` (inout b: Berretacoin, in transacciones: $seq\langle transaccion\rangle$) {

`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` $\{((transacciones[0].id = 0) \wedge ((\forall i : \mathbb{Z})\ ((0 \leq i < |transacciones| - 1) \longrightarrow_L (transacciones[i].id + 1 = transacciones[i + 1].id)))) \vee (transacciones = \langle\rangle)\}$

/* Un usuario no puede ser comprador y vendedor a la vez en la misma transacción */

`requiere` $\{(\forall i : \mathbb{Z})\ ((0 \leq i < |transacciones|) \longrightarrow_L (transacciones[i].id\_comprador \neq transacciones[i].id\_vendedor))\}$

/* Si en transacciones aparece un comprador que no era usuario, el mismo debe aparecer como vendedor en alguna transacción previa*/

`requiere` $\{(\forall i : \mathbb{Z})\ ((0 \leq i < |transacciones|) \longrightarrow_L ((\neg perteneceUsuario(b.usuarios,\ transacciones[i].id\_comprador)) \longrightarrow_L ((\exists j : \mathbb{Z})\ ((0 \leq j < i) \longrightarrow_L (transacciones[i].id\_comprador = transacciones[j].id\_vendedor)))))\}$

/* Ningún usuario ya existente gasta más de lo que tiene (se verifica transacción a transacción) */

`requiere` $\{(\forall i : \mathbb{Z})\ ((0 \leq i < |b.usuario|) \longrightarrow_L ((\forall j : \mathbb{Z})\ (((0 \leq j < |transacciones|) \wedge (b.usuarios[i].id = transacciones[j].id\_comprador)) \longrightarrow_L (b.usuarios[i].monto + montoRecibido(transacciones,\ usuarios[i].id,\ j) - montoGastado(transacciones,\ usuarios[i].id,\ j)) \geq 0)))))\}$

/* Verifica que ningún usuario nuevo gastó más de lo que adquirió */

`requiere` $\{(\forall i : \mathbb{Z})\ (((0 \leq i < |transacciones|) \wedge_L (\neg perteneceUsuario(b.usuarios, transacciones[i].id\_comprador))) \longrightarrow_L (montoRecibido(transacciones,\ transacciones[i].id\_comprador,\ i) - montoGastado(transacciones,\ transacciones[i].id\_comprador,\ i)))\}$

/* 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` $\{(|b.bloques| \leq 3000) \longrightarrow_L$ (

$(|transacciones| > 0) \wedge_L$

(transacciones[0].id_comprador = 0) $\wedge_L$

(transacciones[0].monto = 1) $\wedge_L$

(noGanoMonedaGratis(b.bloques, transacciones[0].id_vendedor)) $\wedge_L$

(transacciones[0].id_vendedor > 0) $\wedge_L$

$((\forall i : \mathbb{Z})\ ((1 \leq i < |transacciones|) \longrightarrow_L ((transacciones[i].id\_comprador \neq 0) \wedge_L$

(transacciones[i].id_vendedor $\neq 0$))))

)

}

/* Si ya se emitieron todas las berretacoin posibles, entonces no se seguirá emitiendo */

`requiere` $\{(|b.bloques| > 3000) \longrightarrow_L ((\forall i : \mathbb{Z})\ ((0 \leq i < |transacciones|) \longrightarrow_L ((transacciones[i].id\_comprador \neq 0) \wedge_L (transacciones[i].id\_vendedor \neq 0))))\}$

/* 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 vendedores que están en transacciones y no están en $B_0.usuarios$, están en b.usuarios */

`asegura`

$\{(\forall i : \mathbb{Z})\ (((0 \leq i < |transacciones|) \wedge_L \neg(perteneceUsuario(B_0.usuarios, transacciones[i].id\_vendedor))) \longrightarrow_L (perteneceUsuario(b.usuarios, transacciones[i].id\_vendedor)))\}$

/* No van a haber ids repetidos en b.usuarios */
asegura {$(\forall i : \mathbb{Z})\ ((0 \leq i < |b.usuarios|) \longrightarrow_L \neg(hayRepetidos(b.usuarios, b.usuarios[i])))$}
/* Actualizamos de ser necesario la cantidad de monedas de los usuarios */
asegura {$(\forall i : \mathbb{Z})\ ((0 \leq i < |b.usuarios|) \longrightarrow_L$ (
/* Caso: nuevo usuario y fue vendedor (previamente ya verificamos que sea vendedor antes que comprador, es decir que no gaste antes de tener) */
$\big((\neg(perteneceUsuario(B_0.usuarios, b.usuarios[i].id))) \land$
$(vendedorPerteneceATransaccion(b.usuarios[i], transacciones)) \longrightarrow_L (b.usuarios[i].monedas =$
$montoRecibido(transacciones, b.usuarios[i].id) - montoGastado(transacciones, b.usuario[i].id))\big)$
$\lor_L$
/* Caso: era usuario y no recibio más monedas, es decir a lo sumo gastó */
$\big((\text{perteneceUsuario}(B_0.usuarios, b.usuarios[i].id)) \land$
$((\neg(vendedorPerteneceATransaccion(b.usuarios[i], transaccion))) \longrightarrow_L (b.usuarios[i].id =$
$B_0.usuarios[i].id \land b.usuarios[i].monedas =$
$B_0.usuarios[i].monto - montoGastado(transacciones, b.usuarios[i].id)))\big)$
$\lor_L$
/* Caso: era usuario y recibió más monedas */
$\big((\text{perteneceUsuario}(B_0.usuarios, b.usuarios[i].id)) \land$
$(vendedorPerteneceATransaccion(b.usuarios[i], transaccion)) \longrightarrow_L (b.usuarios[i].id =$
$B_0.usuarios[i].id \land b.usuarios[i].monedas = B_0.usuarios[i].monedas +$
$montoRecibido(transacciones, b.usuarios[i].id) - montoGastado(transacciones, b.usuarios[i].id))\big)))$
/* 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) : $seq\langle usuario\rangle$ {
asegura
{$(\forall i : \mathbb{Z})\ ((0 \leq i < |b|) \longrightarrow_L ((b.usuarios[i] \in res) \iff (esMaximoTenedor(b.usuarios, b.usuarios[i]))))$}
}

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

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

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

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

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

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

pred noGanoMonedaGratis (bloques: $seq\langle bloque\rangle$, id_vendedor: $\mathbb{Z}$) {

$(\forall i : \mathbb{Z})\ ((0 \leq i < |bloques|) \longrightarrow_L (bloques[i].transacciones[0].id\_vendedor \neq id\_vendedor))$

}

pred vendedorPerteneceATransaccion (usuario: usuario, transacciones: $seq\langle transaccion\rangle$) {

$(\exists i : \mathbb{Z})\ ((0 \leq i < |transacciones|) \wedge_L (usuario.id = transacciones[i].id\_vendedor))$

}

aux montoRecibido (transacciones: $seq\langle transaccion\rangle$, id_u: $\mathbb{Z}$, idx: $\mathbb{Z}$) : $\mathbb{Z}$ =

$\left(\sum_{k=0}^{idx} ifThenElse(transacciones[k].id\_vendedor = id\_u,\ transacciones[j].monto,\ 0)\right)$;

aux montoGastado (transacciones: $seq\langle transaccion\rangle$, id_u: $\mathbb{Z}$, idx: $\mathbb{Z}$) : $\mathbb{Z}$ =

$\left(\sum_{k=0}^{idx} ifThenElse(transacciones[k].id\_comprador = id\_u,\ transacciones[j].monto,\ 0)\right)$;

aux montoTotalOperado (transacciones: $seq\langle transaccion\rangle$) : $\mathbb{Z}$ =

$\left(\sum_{k=0}^{|transacciones|-1} transacciones[k].monto\right)$;

aux montoTotalOperadoBloques (bloques: $seq\langle bloque\rangle$) : $\mathbb{Z}$ =

$\left(\sum_{k=0}^{|bloques|-1} montoTotalOperado(transaccionesDesde(ifThenElse(b.bloques[k].id\_bloque \leq 3000, 1, 0),\right.$

$\left.\text{b.bloques[k].transacciones}))\right)$;

aux cantTotalDeOperacionesBloques (bloques: $seq\langle bloque\rangle$) : $\mathbb{Z}$ =

$\left(\sum_{k=0}^{|bloques|-1} |b.bloques[k].transacciones| - ifThenElse(b.bloques[k].id\_bloque \leq 3000, 1, 0)\right)$;

aux transaccionesDesde (indice: $\mathbb{Z}$, transacciones: $seq\langle transaccion\rangle$) : $seq\langle transaccion\rangle$ =

$(subseq(transacciones, indice, |transacciones|))$;

}