DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Especificaciones de TADs

Trabajo Práctico 1

May 15, 2025 — Algoritmos y estructura de datos

Grupo: asdfgh

Integrante	LU	Correo electrónico
Alcaraz Arce, Kayo	1068/24	arcedeveloper@gmail.com
Gonzalez, Moira Agustina	355/23	18agonz@gmail.com
Gurbanov, Manuel	793/24	mgurbanov@dc.uba.ar
Santucho, Agustín	796/24	santucho.ag@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 Consideraciones y presunciones sobre el enunciado

- Presuponemos desde el comienzo, que no consideramos valido un bloque sin ninguna transacción (bloque vacio), ya que no tiene sentido agregarlo.
- Reiterando el item anterior, en el predicado `esCreacionValida(...)` lo volvemos a validar, por que no sabemos si el criterio de correción es que lo valide atómicamente por cada predicado que lo tenga que presuponer, en nuestra especificación de igual forma no llegaría a evaluar `esCreacionValida` con $|bloq| = 0$ porque usamos logica trivaluada en `agregarBloque(...)`
- A partir de la precondición que involucra `esBloqueOrdenado(...)`, podemos asumir que el orden de los identificadores (`id_transaccion`) coincide con el orden en que las transacciones están situadas en la secuencia; analogo para `id_bloque` y el observador bloques.
- Consideramos originalmente esta versión de `agregarBloques(...)`, pero durante las consultas nos aseguraron que nuestra especificación no es la responsable de decidir si el bloque se agrega o no en base al criterio de si el mismo es valido, sino que nuestro procedimiento asume que es valido en un requiere, así que la versión definitiva lo supone en un requiere.

`proc` `agregarBloque`(`inout` bc : BC, `in` bloq : Bloque) {

`requiere` : $\{bc = BC_0\}$

`asegura` : $\{esBloqueValido(BC_0, bloq) \longrightarrow bc.bloques = concat(BC_0, \langle bloq \rangle)\}$

`asegura` : $\{\neg esBloqueValido(BC_0, bloq) \longrightarrow bc.bloques = BC_0.bloques\}$

}

2 Especificacíon completa del TAD $Berretacoin

// Alias de los tipos que vamos a utilizar para mejor legibilidad
Transac **ES** Struct$\langle id_transaccion : \mathbb{Z}, id_comprador : \mathbb{Z}, id_vendedor : \mathbb{Z}, monto : \mathbb{Z}\rangle$
Bloque **ES** Struct$\langle id_bloque : \mathbb{Z}, lista : \text{Seq}\langle\text{Transac}\rangle\rangle$

TAD BC {
obs $bloques$: Seq⟨Bloque⟩

proc nuevoBc() : BC {
requiere : {true}
asegura : $\{res.bloques = \langle\rangle\}$
}

// En esta versión de agregarBloque consideramos que es responsabilidad del usuario la validación del bloque, por eso el predicado está en el requiere. En la sección de "consideraciones" menciones al respecto de por qué tomamos esta decisión.
proc agregarBloque(inout bc : BC, in bloq : Bloque) {
requiere : $\{bc = BC_0\}$
requiere : $\{esBloqueValido(BC_0, bloq)\}$
asegura : $\{bc.bloques = concat(BC_0.bloques, \langle bloq\rangle)\}$
}

proc cotizarAPesos(in bc : BC, in cotizaciones : Seq⟨ℤ⟩) : Seq⟨ℤ⟩ {
requiere : $\{|bc.bloques| = |cotizaciones|\}$
asegura : $\{\cdots$
$|res| = |bc.bloques| \wedge_L \cdots$
$(\forall i : \mathbb{Z})(0 \leq i < |bc.bloques| \longrightarrow_L res[i] = cotizaciones[i] * sumaMontos(bc.Bloques[i])$
}
}

proc maximosTenedores(in bc : BC) : Seq⟨ℤ⟩ {
requiere : {true}
// esUsuario(bc,u) descarta los casos bordes explicados en esPotencialMaximoTenedor(...)
asegura : $\{(\forall u : \mathbb{Z})(u \in res \longleftrightarrow (\text{esUsuario}(bc, u) \wedge \text{esPotencialMaximoTenedor}(bc, u)))\}$
// Nos aseguramos que aparezca en la secuencia como maximo una sola vez
asegura : $\{(\forall i : \mathbb{Z})(\forall j : \mathbb{Z})(0 \leq i,\ j < |res| \longrightarrow_L (i \neq j \longrightarrow res[i] \neq res[j]))\}$
}

proc montoMedio(in bc : BC) : ℝ {
requiere : {true}
asegura : $\{\text{cantidadDeTransaccionesEntreUsuarios}(bc) = 0 \longrightarrow res = 0\}$
asegura : {
$\text{cantidadDeTransaccionesEntreUsuarios}(bc) \neq 0 \longrightarrow_L \cdots$

$$res = \sum_{i=0}^{|bc.bloques|-1} \left(\sum_{j=0}^{|bc.bloques[i].lista|-1} \frac{\text{if } esTransacEntreUsuarios(bc,\ i,\ j) \text{ then } bc.bloques[i].lista[j].monto \text{ else } 0 \text{ fi}}{\text{cantidadDeTransaccionesEntreUsuarios}(bc)} \right)$$

}
}

// Es bloque valido hace la conjunción de los distintos criterios de validación. Lo utilizamos para poder referenciarlos a todos en un solo predicado `esBloqueValido(...)`

`pred esBloqueValido`(bc : BC, bloq : Bloque) {

$1 \leq |bloq.lista| \leq 50 \ \wedge_L$
$bloq.id = |bc.bloques|$
$\cdots \wedge$ esCreacionValida$(bc, bloq)$
$\cdots \wedge$ nadieGastaDeMas$(bc, bloq)$
$\cdots \wedge$ esBloqueOrdenado$(bloq)$
$\cdots \wedge$ tieneMontosPositivos$(bloq)$
$\cdots \wedge$ noSeVendenASiMismos$(bloq)$]

}

`pred nadieGastaDeMas`(bc : BC, bloq : Bloque) {

$(\forall i : \mathbb{Z})(0 \leq i < |bloq.lista| \longrightarrow_L \cdots$
$(bloq.lista[i].id\_comprador = 0 \ \vee$ usuarioNoGastoDeMas$(bc, \ bloq, \ bloq.lista[i].id_comprador)$
$)$

}

// El pred asume que todos los bloques ya puestos en la lista de bloques se consideran validos. El cuantificador nos garantiza que no importa (en orden) cuantas transacciones tomemos, siempre el monto va a ser positivo.

`pred usuarioNoGastoDeMas`(bc : BC, bloq : Bloque, id_usr : $\mathbb{Z}$) {

$(\forall k : \mathbb{Z})(0 \leq k \leq |bloq.lista| \longrightarrow_L \cdots$
(gastosTotalesUsuario$(bc, id_usr) \ +$ balanceDeUsuarioEnBloque(subseq$(bloq.lista, 0, k), id_usr) \geq 0$)
$)$

}

`pred esCreacionValida`(bc : BC, bloq : Bloque) {

$(|bloq| > 0)$
$\vdots$
$\wedge_L$
$\vdots$
// Una transac. de creación solo puede crear una unidad de BC.
$(bloq[0].id\_comprador = 0 \longrightarrow bloq[0].monto = 1)$
// Evalua true si el usuario nunca minó (fue el vendedor) de una transac.
$\cdots \wedge$ usuarioNuncaCreó$(bc, bloq.lista[0].id_vendedor)$
// El id del comprador o vendedor de las transac. tienen id≠0 (solo puede comprar en la primera)
$\cdots \wedge (\forall i : \mathbb{Z})(1 \leq i < |bloq.lista| \longrightarrow_L (bloq.lista[i].id\_comprador \neq 0 \ \wedge \ bloq.lista[i].id\_vendedor \neq 0))$

// Esta transacción es de creación si está antes del bloque 3000.
// Esta transacción NO es de creación si está después del bloque 3000.
$\cdots \wedge \cdots$
$\Big(\Big(|bc.bloque| < 3000 \ \wedge bloq.lista[0].id\_comprador = 0 \Big) \vee \cdots$
$\Big(|bc.bloques| \geq 3000 \ \wedge \ bloq.lista[0].id\_comprador \neq 0 \Big) \Big)$

}

`pred usuarioNuncaCreó`(bc : BC, id_usr : $\mathbb{Z}$) {

$(\forall i : \mathbb{Z})(0 \leq i < |bc.bloques| \longrightarrow_L \cdots$
$[bc.bloques[i].lista[0].id\_comprador \neq 0 \ \vee \ bc.bloques[i].lista[0].id\_vendedor \neq id\_usr])$

}

`pred noSeVendenASiMismos`(bloq : Bloque) {

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

}

```
pred esBloqueOrdenado(bloq : Bloque) {
    (∀i : ℤ)(1 ≤ i < |bloq.lista| ⟶_L bloq.lista[i].id_transaccion = bloq.lista[i − 1].id_transaccion + 1)
}

pred tieneMontosPositos(bloq : Bloque) {
    (∀i : ℤ)(0 ≤ i < |bloq.lista| ⟶_L bloq.lista[i].monto > 0)
}

// Decimos que es potencial ya que si no hubiera ningún bloque, o todos los usuarios quedaran en monto=0, entonces todos los numeros enteros serán potencial máximo; estos casos borde los descartamos en el proc maximosTenedores(...)
pred esPotencialMaximoTenedor(bc : BC, id_usr ℤ) {
    (∀u' : ℤ)(gastosTotalesUsuario(bc, id_usr) ≥ gastosTotalesUsuario(bc, u'))
}

pred esBloqueQueSoloCrea(bloque : Bloque) {
    |bloq.lista| = 1 ∧_L bloq.lista[0].id_comprador = 0
}

pred esUsuario(bc : BC, id_usr : ℤ) {
    (id_usr ≠ 0) ∧ ···
    (∃i : ℤ)(0 ≤ i < |bc.bloques| ∧_L usuarioEstaEnBloque(bc.bloques[i], id_usr))
}

pred usuarioEstaEnBloque(bloq : Bloque, id_usr : ℤ) {
    (∃j : ℤ)(0 ≤ j < |bloq.lista| ∧_L (bloq.lista[j].id_comprador = id_usr ∨ bloq.lista[j].id_vendedor = id_usr))
}
pred esTransacEntreUsuarios(bc : BC, i : ℤ, j : ℤ) {
    bc.bloques[i].lista[j].id_comprador ≠ 0
}
```

aux `gastosTotalesUsuario`(bc : BC, id_usr : $\mathbb{Z}$) : $\mathbb{R} = \cdots$

$$\sum_{i=0}^{|bc.bloques|-1} \text{balanceDeUsuarioEnBloque}(bc.bloques[i].lista, id\_usr)$$

aux `balanceDeUsuarioEnBloque`(lista : Seq$\langle Transac \rangle$, id_usr : $\mathbb{Z}$) : $\mathbb{R} = \cdots$

$$\sum_{i=0}^{|lista|-1} ifThenElse(lista[i].id\_vendedor = id\_usr, lista[i].monto, 0) \; - \; \sum_{i=0}^{|lista|-1} ifThenElse(lista[i].id\_comprador = id\_usr, lista[i].monto, 0)$$

aux `sumaMontos`(bloq : Bloque) : $\mathbb{Z} = \cdots$

$$\sum_{i=0}^{|bloq.lista|-1} bloq.lista[i].monto$$

aux `cantidadDeTransaccionesEntreUsuarios`(bc : BC) : $\mathbb{Z} = \cdots$

$$\sum_{i=0}^{|bc.bloques|-1} \left(\sum_{j=0}^{|bc.bloques[i].lista|-1} \text{if esTransacEntreUsuarios}(bc,\ i,\ j) \text{ then } 1 \text{ else } 0 \text{ fi} \right)$$

}