DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 1 - Especificación de TADs

Criptomonedas

Algoritmos y Estructura de Datos 2

Grupo Pythonisbetter

Integrante	LU	Correo electrónico
Perez Marzo, Jordan Alexis	738/24	jordanpm30@gmail.com
Nuñez Geronimo, Sebastian Javier	986/24	sebustos2394@gmail.com
Camacho Gomez, Lucero Belen	667/23	camacholu14@gmail.com
Suarez, Ricardo Javier	127/20	ricardojaviersuarezz@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. Definición de Tipos

type *Bloque* ES $tupla\langle \mathbb{Z}, seq\langle Transaccion\rangle\rangle$
type *idBloque* ES $\mathbb{Z}$
type *Transaccion* ES $tupla\langle idTransaccion, idComprador, idVendedor, monto\rangle$
type *idTransaccion, idComprador, idVendedor, monto* ES $\mathbb{Z}$
type *Usuario, Saldo* ES $\mathbb{Z}$
type *Cotizacion* ES $\mathbb{Z}$

Si hacían que esto fuera struct en vez de tupla, podían escribir T.monto en vez de t_3 y se entendería un poco mejor.

2. Especificación

TAD $BerretaCoin {

obs *bloques* = $seq\langle Bloque\rangle$
obs *dineroUsuariosTotal* = $dict\langle Usuario, Saldo\rangle$

2.1. Ejercicio 1

```
proc $BerretaCoinVacio () : $ Berretacoin {
    asegura {res.bloque =<>}
    asegura {res.dineroUsuarioTotal = {}}
}

proc agregarBloque (inout BCoin : $ Berretacoin , in b : Bloque) : {
    requiere {BCoin = BCoin_0}
    requiere {0 < |b_1| ≤ 50}
    requiere {|BCoin_0.bloques| ≤ 2999 ⟶ (bloqueValidoConTransaccionEspecial(b) ∧
    vendedorDeCreacionSiempreDistinto(BCoin_0.bloques, b))}
    requiere {|BCoin_0.bloques| > 2999 ⟶ bloqueValidoSinTransaccionEspecial(b)}
    requiere {nadieGastaMasDeLoQueTiene(BCoin_0.dineroUsuarioTotal, b)}
    requiere {esBloqueConsecutivoConLosDemasPorId(BCoin_0.bloques, b)}
    asegura {BCoin.bloques = BCoin_0.bloques + + < b >}
    asegura {(∀j : Z)((esUsuarioDelBloque(j, b)) ⟶_L
    BCoin.dineroUsuariosTotal[j] = IfThenElse(j ∉ BCoin_0.dineroUsuarioTotal,
    montoComprado(b_1, j) − montoVendido(b_1, j),
    BCoin_0.dineroUsuarioTotal[j] + montoComprado(b_1, j) − montoVendido(b_1, j)) )}
    asegura {(∀j : Z)((j ∈ BCoin_0.dineroUsuarioTotal ∧ ¬esUsuarioDelBloque(j, b)) ⟶_L
    BCoin.dineroUsuariosTotal[j] = BCoin_0.dineroUsuarioTotal[j]))}
    asegura {(∀j : Z)((j ∉ BCoin_0.dineroUsuarioTotal ∧ ¬esUsuarioDelBloque(j, b)) ⟶_L
    j ∉ BCoin.dineroUsuariosTotal )}
}

pred bloqueValidoConTransaccionEspecial (b: Bloque) {
    (b_1[0]_1 = 0 ∧ b_1[0]_2 > 0 ∧ b_1[0]_3 = 1 ∧ b_1[0]_0 > 0) ∧ (∀i : Z)(1 ≤ i < |b_1| ⟶_L
    (idTransaccionYUsuariosYMontosValidos(b_1[i]))) ∧ tieneTransaccionesConsecutivas(b)
}
pred bloqueValidoSinTransaccionEspecial (b: Bloque) {
    (∀i : Z)(0 ≤ i < |b_1| ⟶_L (idTransaccionYUsuariosYMontosValidos(b_1[i]))) ∧ tieneTransaccionesConsecutivas(b)
}
pred idTransaccionYUsuariosYMontosValidos (t: Transaccion) {
    t_1 > 0 ∧_L t_2 > 0 ∧_L t_1 ≠ t_2 ∧_L t_3 > 0 ∧_L t_0 > 0)
}
pred tieneTransaccionesConsecutivas (b: Bloque) {
    (∀j : Z)(0 ≤ j < |b_1| − 1) ⟶_L b_1[j]_0 = b_1[j + 1]_0 − 1)
}
pred vendedorDeCreacionSiempreDistinto (s:seq⟨Bloque⟩,b: Bloque) {
    (∀i : Z)(0 ≤ i < |s| ⟶_L BCoin_0.bloques[i]_1[0]_2 ≠ b_1[0]_2)
}
pred nadieGastaMasDeLoQueTiene (d: dict ⟨Usuario, Saldo⟩, b : Bloque){
```

Si esto estuviera dentro de un aux, sabría qué hace con solo leer el título!

OJO, la asignación en diccionarios con "=" No existe. Revisen el apunte de TADs

otra interpretación podría ser que $b_1[j]_0 < b_1[j+1]_0$

Recuerden que el comprador es el que gasta la plata (comma un objeto a cambio de $ BC). Osea comprando perdés plata y vendiendo hacés. Está al revés la resta.

$(\forall i : Z)(0 \leq i < |b_1| \longrightarrow_L$
$(\forall j : Z)(j \notin d \wedge_L [(montoComprado(subseq(b_1, 0, i+1), j) - (montoVendido(subseq(b_1, 0, i+1), j)))] \geq 0)$
$\wedge$
$(\forall n : Z)(0 \leq n < |b_1| \longrightarrow_L$
$(\forall m : Z)(m \in d \wedge_L d[m] + [(montoComprado(subseq(b_1, 0, n+1), m) - (montoVendido(subseq(b_1, 0, n+1), m)))] \geq 0)$
$\}$

Igualmente este razonamiento está bien.

Está raro esto. Para todos los elementos del bloque ningún j ∈ al dic de saldos y pasan cosas con los saldos y para todos los elems del bloque, todos pertenecen al dic y pasan cosas con los saldos. Esas 2 cosas no pueden suceder a la vez. Aún si hubiera un 'V' entre los ∀i no estaría bien, porque están predicando sobre todos los elems del bloque a la vez.

```
pred esBloqueConsecutivoConLosDemasPorId (s:seq⟨Bloque⟩,b: Bloque) {
    |s| > 0 ∧ s[|s| − 1]_0 = b_0 − 1
}
aux montoComprado (t:seq⟨Transaccion⟩, u: Usuario) : Z =
Σ_{i=0}^{|t|−1} IfThenElse(t[i]_1 = u, t[i]_3, 0);
aux montoVendido (t:seq⟨Transaccion⟩, u: Usuario) : Z =
Σ_{i=0}^{|t|−1} IfThenElse(t[i]_2 = u, t[i]_3, 0);
pred esUsuarioDelBloque  (u: Usuario , b: Bloque) {
    (∃i : Z)(0 ≤ i < |b_n| ∧_L (b_{1[i]_1} = u ∨_L b_{1[i]_2} = u))
}
```

no va el L, b1[i] no se puede indefinir.

2.2. Ejercicio 2

```
proc maximoTenedores (in BCoin : $BerretaCoin) : seq⟨ℤ⟩{
    requiere {True}
    asegura {(∀i : Usuario) (0 ≤ i < |res| ⟶_L usuarioConMasBCoin(res[i], BCoin.dineroUsuarioTotal))}
    asegura {noHayRepetidos(res)}
    asegura {(∀j : Usuario)(j ∈ Bcoin.dineroUsuarioTotal
    ∧_L usuarioConMasBCoin(j, BCoin.dinerousuarioTotal)) ⟶_L j ∈ res}
}
pred usuarioConMasBCoin (u : Usuario,    d : dict ⟨Usuario, Saldo⟩){
    (∀j : Usuario)((j ∈ d ∧ u ∈ d ∧ j ≠ u) ⟶_L d[j] ≤ d[u])
    }
pred noHayRepetidos (s:seq⟨T⟩) {
    (∀i : Z)(0 ≤ i < |s| ⟶_L ¬(∃j : Z)((0 ≤ j < |s| ∧ j ≠ i) ∧_L s[i] = s[j])
}
```

Está bien la lógica. Sugerencia de prolijidad:
si es máximo → está en res.
si no es máximo → no está en res.
Se puede escribir como
es máximo ↔ está en res.
Lo simplificaría su lógica.

→ les faltaría aseguar que los observadores no cambian luego de las operaciós.

2.3. Ejercicio 3

```
proc montoMedio (in BCoin : $ BerretaCoin ) : R {
    asegura {|BCoin.bloques| = 0 ⟶ res = 0}
    asegura {|BCoin.bloques| > 0 ⟶ res = montoTotalTransacciones(BCoin.bloques) / cantidadTransacciones(BCoin.bloques)}
    }
    aux montoTotalTransacciones (s:seq⟨bloques⟩) : R =
    Σ_{i=0}^{|s|−1} montoPorBloque(s[i]);
    aux montoPorBloque (s:bloque) : R =
    Σ_{i=0}^{|b|−1} IfThenElse(b_1[j]_1 ≠ 0, b_1[j]_3, 0);
    aux cantidadTransacciones (s:seq⟨bloque⟩) : R =
    Σ_{i=0}^{|s|−1} IfThenElse(s[i]_{1_{0_1}}, |s[i]_1| − 1, |s[i]_1|);
```

esto no es un valor de verdad.

2.4. Ejercicio 4

proc cotizacionAPesos (in BCoin : $ BerretaCoin , in c : $seq\langle Cotizacion\rangle$) : $seq\langle Z\rangle$ {

requiere $\{|BCoin.bloques| = |c|\}$ ✓

requiere $\{(\forall i : Z)(0 \leq i < |c| \longrightarrow_L c[i] > 0)\}$

asegura $\{|res| = |BCoin.bloques| \wedge |res| = |c|\}$ → es redundante escribir los 2, ya arriba dicen que miden lo mismo.

asegura $\{(\forall i : Z)(0 \leq i < |res|) \longrightarrow_L res[i] = montoPorBloque(BCoin.bloques[i]) * c[i]\}$ ✓

}

aclaren que no va a ser una auxiliar global!

→ Para la reentrega, fijense que el único observador que necesitan realmente es el de los bloques.

Notes también que cada vez que usan los otros observadores, podrían reemplazarlo por una auxiliar que haga la cuenta en base a lo que ya tienen.

Es decir, pueden dejar sus observadores y arreglar los errores, o reescribirlos sin, lo cual sería más correcto.