

1. Especificación del TAD \$Berretacoin

1.1. Introducción:

Para la creación de la cripto \$Berretacoin especificamos un TAD que modela su funcionamiento. Nuestra cripto tiene usuarios que identificamos con un número entero positivo. Cada transacción involucra exactamente a dos usuarios. Las transacciones se agrupan en bloques, como en Blockchain, y cada bloque contiene a lo sumo 50 transacciones, cuyos identificadores están ordenados dentro del bloque.

La primera transacción de cada bloque “crea” una nueva unidad de \$Berretacoin, hasta alcanzar el límite de emisión, que está fijado en 3000 unidades. A estas transacciones especiales las denominamos “transacciones de creación”, y tienen la característica de que el comprador tiene identificador 0 y el vendedor es algún usuario arbitrario siempre distinto.

A su vez, cada bloque tiene un identificador que es un entero no negativo. Los bloques se encadenan de forma consecutiva de acuerdo al identificador.

1.2. Definición de Tipos de Datos

Definimos ciertos atajos para referirnos a los tipos de datos utilizados.

`transaccion` = tupla $\langle id_transaccion : \mathbb{Z}, id_comprador : \mathbb{Z}, id_vendedor : \mathbb{Z}, monto : \mathbb{Z} \rangle$
`id_bloque` : \mathbb{Z}

1.3. TAD \$Berretacoin

obs blockchain: $seq \langle bloque \rangle$

```
proc agregarBloque (inout B : $Berretacoin, in ts : seq<transaccion>)  
  requiere {B = B0}  
  requiere {ts = ts0}  
  requiere {0 < |ts| ≤ 50}  
  requiere {( |B.blockchain| < 3000 ∧L (∀i : ℤ) (  
0 < i < |ts| →L ¬transaccionDeCreacion(ts[i]) ∧L transaccionBienDefinida(ts[i]) ∧L transaccionDeCreacion(ts[0])  
)) ∨L ( |B.blockchain| ≥ 3000 ∧L (∀j : ℤ) (  
0 ≤ j < |ts| →L ¬transaccionDeCreacion(ts[j]) ∧L transaccionBienDefinida(ts[j])  
))) }  
  asegura {id_bloque > 0 ∧L (∀i : ℤ) (  
0 ≤ i < |B0.blockchain| →L B0.blockchain[i][0] ≠ id_bloque  
) ∧L B.blockchain = B0.blockchain + (id_bloque, ts0) }
```

```
proc maximosTenedores (in B : $Berretacoin) : seq<transaccion>  
  requiere {|B.blockchain| > 0}  
  asegura {(∀i : ℤ) (  
0 ≤ i < |res| →L (∃j : ℤ) (  
0 ≤ j < |B.blockchain| ∧L (∃k : ℤ) (  
0 ≤ k < |B.blockchain[j][1]| ∧L res[i] = B.blockchain[j][1][k][2]  
))  
))  
}  
  asegura {(∀i : ℤ) (  
0 ≤ i < |res| →L (∀k, j : ℤ) (  
0 ≤ k < |B.blockchain| ∧L 0 ≤ j < |B.blockchain[k][1]| →L saldo(res[i], B) ≥ saldo(B.blockchain[k][1][j][2], B)  
))  
})
```

```

proc montoMedio (in B : $Berretacoin) :  $\mathbb{N}$ 
  requiere  $\{|B.blockchain| > 0\}$ 
  asegura  $\{(|B.blockchain| < 3000 \wedge_L res = \frac{sumaMontoTotal(B.blockchain) - |B.blockchain|}{cantTransacciones(B.blockchain) - |B.blockchain|}) \vee_L (|B.blockchain| \geq 3000 \wedge_L$ 
   $res = \frac{sumaMontoTotal(B.blockchain) - 3000}{cantTransacciones(B.blockchain) - 3000})\}$ 

proc cotizacionAPesos (in B : $Berretacoin, in s : seq( $\mathbb{Z}$ )) : seq( $\mathbb{Z}$ )
  requiere  $\{|B.blockchain| = |s|\}$ 
  asegura  $\{|res| = |s|\}$ 
  asegura  $\{(\forall i : \mathbb{Z}) ($ 
     $0 \leq i < |s| \longrightarrow_L s[i] * sumaMontoBloque(B.blockchain, i) = res[i]$ 
   $)\}$ 

aux compraVenta (B : $Berretacoin, id_persona :  $\mathbb{Z}$ , n :  $\mathbb{Z}$ ) :  $\mathbb{Z}$  =
 $\sum_{i=0}^{|B.blockchain|-1} \sum_{j=0}^{|B.blockchain[i][1]|-1} \text{ifthenelse}(B.blockchain[i][1][j][n] = id\_persona, B.blockchain[i][1][j][3], 0);$ 

aux saldo (B : $Berretacoin, id_persona :  $\mathbb{Z}$ ) :  $\mathbb{Z}$  = compraVenta(B, id_persona, 2) - compraVenta(B, id_persona, 1);

aux sumaMontoTotal (B : $Berretacoin) :  $\mathbb{Z}$  =
 $\sum_{i=0}^{|B.blockchain|-1} \sum_{j=0}^{|B.blockchain[i][1]|-1} B[i][1][j][3];$ 

aux sumaMontoBloque (B : $Berretacoin, n :  $\mathbb{Z}$ ) :  $\mathbb{Z}$  =
 $\sum_{i=0}^{|B.blockchain[n][1]|-1} B[n][1][i][3];$ 

aux cantTransacciones (B : $Berretacoin) :  $\mathbb{Z}$  =
 $\sum_{i=0}^{|B.blockchain|-1} |B[i][1]|;$ 

pred transaccionDeCreacion (ts : transaccion, B : $Berretacoin) {
   $(ts[1] = 0 \wedge_L |B.blockchain| = 0 \wedge_L ts[3] = 1) \vee_L (ts[3] = 1 \wedge_L (\forall i : \mathbb{Z}) ($ 
     $0 \leq i < |B.blockchain| \longrightarrow_L B.blockchain[i][1][0][2] \neq ts[2]$ 
   $) \wedge_L ts[2] \neq 0)$ 
}

pred transaccionBienDefinida (ts : transaccion, B : $Berretacoin) {
   $ts[1] \neq ts[2] \wedge_L (\forall i : \mathbb{Z}) ($ 
     $0 \leq i < 3 \longrightarrow_L ts[i] > 0$ 
   $) \wedge_L compraVenta(B.blockchain, ts[1], 2) - compraVenta(B.blockchain, ts[1], 1) \geq ts[3]$ 
}

```

1.4. Comentarios

Los siguientes comentarios facilitan la comprensión de los ejercicios. Además, tener a mano que tipos de datos contiene la blockchain resulta práctico para la lectura del TAD.

Blockchain = seq $\langle Bloque \rangle$

Bloque = tupla $\langle id_bloque : \mathbb{Z}, transacciones : seq\langle transaccion \rangle \rangle$

transaccion = tupla $\langle id_transaccion : \mathbb{Z}, id_comprador : \mathbb{Z}, id_vendedor : \mathbb{Z}, monto : \mathbb{Z} \rangle$

- **Proc agregarBloque** : Con ayuda de las **precondiciones**, definimos las condiciones necesarias para los datos de entrada. Dentro de estas precondiciones, consideramos que la lista de transacciones no debe superar las 50 transacciones (según lo especificado en el enunciado) y no debe estar vacía. Además, establecimos que la lista no puede contener ninguna transacción de creación en caso de que la longitud de la blockchain sea igual o mayor a 3000 bloques (tener en cuenta

que comienza en el bloque 0). En cambio, si la blockchain aún no alcanzó ese límite, se permite incluir una única transacción de creación, y debe ubicarse en la primera posición de la lista.

En las **postcondiciones** definimos que el `id_bloque` sea válido, es decir, que sea distinto de los demás y mayor a 0. También nos aseguramos de que la lista de transacciones mantenga su orden: definimos la metavariante `ts_0` para almacenar su orden también. Además, con ayuda de una metavariante definida en las precondiciones, verificamos que la salida del procedimiento tenga la estructura correcta.

- **Pred transaccionDeCreacion** : Del procedimiento anterior se derivan ciertos predicados utilizados. Este se ocupa de verificar que una transacción sea de creación. Por un lado, que el `id_comprador` sea 0 y por otro lado que el `id_vendedor` sea distinto para cada transacción de creación realizada previamente. (además evitamos la posibilidad que el `id_vendedor` sea 0).
- **Pred transaccionBienDefinida** : Este predicado también es derivado del procedimiento `agregarBloque`. En este caso, se aseguran las siguientes condiciones: el `id_vendedor` es distinto al `id_comprador`, todos los componentes son mayores o iguales a 0 (según lo establecido en el enunciado), y el saldo del comprador es suficiente para realizar la transacción, es decir, debe tener al menos el monto que se intenta transferir.
- **Aux saldo y compraVenta** : Este auxiliar fue muy útil para calcular el saldo de una persona. Dado que especificar todo en un solo auxiliar nos pareció demasiado complejo, decidimos dividir el problema en partes más simples. Primero, definimos `compraVenta`, que, dependiendo del índice al cual te refieras, calcula el monto total de las compras o ventas que se hayan generado por esa persona en la blockchain hasta ese momento. Con esto, definimos el auxiliar `saldo`, que utiliza `compraVenta` para calcular el estado de compra y le resta el estado de venta de la misma persona.
- **Proc maximosTenedores** : Utilizamos las **precondiciones** para verificar que la blockchain no esté vacía. Luego, dentro de las **postcondiciones**, establecimos que cada elemento del resultado pertenezca efectivamente a la blockchain; es decir, un máximo tenedor de \$Berretacoin debe aparecer, al menos una vez, en el rol de vendedor dentro de alguna transacción de la blockchain. Además, nos aseguramos de que el saldo de cada persona que aparece en la lista de `maximosTenedores` (es decir, en la salida) sea mayor o igual al saldo de cualquier otro vendedor presente en la blockchain.
- **Proc montoMedio** : Al igual que en `maximosTenedores`, usamos las **precondiciones** para verificar que la blockchain no esté vacía. Por otro lado, en las **postcondiciones** establecimos el siguiente comportamiento: Si la blockchain tiene menos de 3000 bloques, restamos la cantidad de bloques al monto total acumulado (es decir, al total de transacciones de creación, ya que cada una equivale a 1 \$Berretacoin), y luego dividimos ese valor por la cantidad total de transacciones menos la cantidad de transacciones de creación —es decir, la cantidad de bloques. En cambio, si la blockchain tiene 3000 bloques o más, aplicamos un razonamiento similar. Restamos 3000 al monto total de transacciones (ya que sabemos que existen 3000 transacciones de creación con monto igual a 1), y luego dividimos ese resultado por la cantidad total de transacciones menos 3000 (las de creación).
- **Aux cantTransacciones** : Dentro de este auxiliar decidimos hacer un contador de transacciones por bloque, dentro de toda la blockchain. Como lo establecido en el enunciado no afirma que cada bloque tiene únicamente 50 transacciones, hace falta sumar la longitud de cada lista de transacciones de cada bloque.
- **Aux sumaMontoTotal** : En este auxiliar sumamos los montos de todas las transacciones, de todos los bloques, de toda la blockchain.
- **Aux sumaMontoBloque** : En este auxiliar sumamos los montos de todas las transacciones del bloque `n`.
- **Proc cotizacionAPesos** : La **precondición** establece que la blockchain y la lista de estados de cotizaciones deben tener la misma longitud. Por otro lado, las **postcondiciones** aseguran que la lista de salida también tenga la misma longitud que la blockchain, y que cada uno de sus elementos represente el monto cotizado en pesos del bloque número `i`, siendo `i` el índice correspondiente en la lista.