

BSTree — Constructors BTree_Empty + BSTree_Insert (with HIDE)

Nereus formal specification — versión corregida (casteo del axioma base)

Corrección aplicada: en el caso base de BSTree_Insert, los subárboles vacíos del lado derecho ahora llevan as BSTree explícito (resaltado en amarillo). Sin ese casteo, la reescritura iterativa del tipo BSTree_Insert(BSTree_Insert(BTree_Empty() as BSTree, 10), 8) no puede aplicar el axioma recursivo, porque b1, b2 están declarados BSTree y no se ligan a un BTree_Empty() "pelado". El resto de la spec preserva el estilo original con HIDE de BTree_Create.

```
-- =====
-- BSTree – Especificación con constructores básicos
-- BTree_Empty (heredado) y BSTree_Insert (propio).
--
-- Encapsulamiento: BTree_Create se OCULTA con La cláusula HIDE
-- para que La API pública de BSTree exponga solo Los dos
-- constructores básicos (BTree_Empty y BSTree_Insert) y Las
-- operaciones derivadas. Esto garantiza que el cliente solo
-- pueda construir BSTs válidos: La única forma de añadir un
-- elemento es mediante BSTree_Insert, cuya precondition
-- (not belongs(e, t)) preserva La invariante BST.
--
-- BTree_Create permanece visible internamente, donde Los
-- axiomas La usan para expresar La estructura de Los términos
-- (subárboles izquierdo y derecho de un nodo). Esta es La
-- separación clásica entre INTERFAZ pública (Los dos
-- constructores) e IMPLEMENTACIÓN interna (La forma canónica
-- con BTree_Create), expresada formalmente.
-- =====
```

```
CLASS BSTree [Element: WithOrder]
IS-SUBTYPE-OF BTree[Element]
BASIC CONSTRUCTORS BTree_Empty, BSTree_Insert
EFFECTIVE
```

OPERATIONS

```
-- Constructor propio: insertar un elemento (sin duplicados).
BSTree_Insert: BSTree(t) * Element(e) -> BSTree
    pre: not belongs(e, t);

-- Operaciones derivadas.
BSTree_Delete: BSTree(t) * Element(e) -> BSTree
    pre: belongs(e, t);

BSTree_Min: BSTree(t) -> Element
    pre: not BTree_IsEmpty(t);
```

AXIOMS b, b1, b2: **BSTree**; i, j: **Element**;

```
-- -----  
-- BSTree_Insert: reescritura a términos con BTree_Create  
-- (forma canónica interna; BTree_Create estará oculto al cliente)  
-- -----  
  
-- Insertar en un árbol vacío produce un singleton.  
-- Los subárboles vacíos llevan `as BSTree` para que la  
-- reescritura recursiva pueda ligarlos a variables b1, b2: BSTree.  
BSTree_Insert(BTree_Empty() as BSTree, i) =  
    BTree_Create(BTree_Empty() as BSTree, BTree_Empty() as BSTree, i) as BSTree;  
  
-- Insertar i < raíz: i va al subárbol izquierdo.  
(i < j) implies  
BSTree_Insert(BTree_Create(b1, b2, j) as BSTree, i) =  
    BTree_Create(BSTree_Insert(b1 as BSTree, i), b2, j) as BSTree;  
  
-- Insertar i > raíz: i va al subárbol derecho.  
(i > j) implies  
BSTree_Insert(BTree_Create(b1, b2, j) as BSTree, i) =  
    BTree_Create(b1, BSTree_Insert(b2 as BSTree, i), j) as BSTree;  
  
-- El caso i = j está excluido por la precondition not belongs(i, t).  
  
-- -----  
-- BSTree_Min: mínimo del árbol (elemento más a la izquierda)  
-- -----  
  
BTree_IsEmpty(b1) implies  
BSTree_Min(BTree_Create(b1, b2, i) as BSTree) = i;  
  
not BTree_IsEmpty(b1) implies  
BSTree_Min(BTree_Create(b1, b2, i) as BSTree) =  
    BSTree_Min(b1 as BSTree);  
  
-- -----  
-- BSTree_Delete: seis casos según la posición de j respecto a la raíz  
-- -----  
  
-- Caso A: j < raíz, recurrir en el subárbol izquierdo.  
(j < i) implies  
BSTree_Delete(BTree_Create(b1, b2, i) as BSTree, j) =  
    BTree_Create(BSTree_Delete(b1 as BSTree, j), b2, i) as BSTree;  
  
-- Caso B: j > raíz, recurrir en el subárbol derecho.  
(j > i) implies  
BSTree_Delete(BTree_Create(b1, b2, i) as BSTree, j) =  
    BTree_Create(b1, BSTree_Delete(b2 as BSTree, j), i) as BSTree;
```

```

-- Caso C: j = raíz y ambos subárboles vacíos -> resultado vacío.
(j = i and BTree_IsEmpty(b1) and BTree_IsEmpty(b2)) implies
BTree_Delete(BTree_Create(b1, b2, i) as BSTree, j) =
    BTree_Empty() as BSTree;

-- Caso D: j = raíz y solo el subárbol derecho no es vacío -> derecho.
(j = i and BTree_IsEmpty(b1) and not BTree_IsEmpty(b2)) implies
BTree_Delete(BTree_Create(b1, b2, i) as BSTree, j) =
    b2 as BSTree;

-- Caso E: j = raíz y solo el subárbol izquierdo no es vacío -> izquierdo.
(j = i and not BTree_IsEmpty(b1) and BTree_IsEmpty(b2)) implies
BTree_Delete(BTree_Create(b1, b2, i) as BSTree, j) =
    b1 as BSTree;

-- Caso F: j = raíz y ambos subárboles no vacíos -> reemplazar la raíz
-- por el mínimo del subárbol derecho, y borrar ese mínimo del derecho.
(j = i and not BTree_IsEmpty(b1) and not BTree_IsEmpty(b2)) implies
BTree_Delete(BTree_Create(b1, b2, i) as BSTree, j) =
    BTree_Create(
        b1,
        BTree_Delete(b2 as BSTree, BTree_Min(b2 as BSTree)),
        BTree_Min(b2 as BSTree)
    ) as BSTree;

-- Encapsulamiento: BTree_Create se oculta de la API pública de BSTree.
-- Solo BTree_Empty y BSTree_Insert quedan disponibles para el cliente
-- como constructores. La invariante BST queda garantizada por construcción.
HIDE BTree_Create

```

END-CLASS

Generado desde BTree-hide.nereus. Casteo del axioma base corregido para permitir reescritura iterativa coherente con las variables tipadas b1, b2: BSTree.