

Traversable + PostBTree

Recorrido post-orden de un árbol binario como instancia de Traversable

Resumen: PostBTree es un subtipo de BTree que además implementa la interfaz abstracta Traversable (first, rest, IsEmpty). El recorrido elegido es post-orden: para BTree_Create(L, R, x) visita primero los elementos de L, después los de R, y por último x. La operación heredada IsEmpty se renombra a PostBTree_IsEmpty para distinguirla del BTree_IsEmpty ya presente en la superclase.

Cambios respecto a la versión anterior:

- Se movió AXIOMS fuera de OPERATIONS (estaba mal indentado).
- Se agregaron precondiciones a first y rest: not PostBTree_IsEmpty(t).
- Se agregaron los dos axiomas de PostBTree_IsEmpty sobre los constructores básicos heredados (BTree_Empty, BTree_Create).
- Las condicionales if-then-else / endif se reescribieron como tres implicaciones mutuamente excluyentes por operación.
- Las llamadas recursivas a first y rest llevan as PostBTree explícito para ser consistentes con la convención de casteo del workspace.

1. Especificación

```
CLASS Traversable [elem]
DEFERRED
TYPE Traversable
OPERATIONS
  IsEmpty: Traversable -> Boolean;
  first: Traversable(t) -> elem
    pre: not IsEmpty(t);

  rest: Traversable(t) -> Traversable
    pre: not IsEmpty(t);

END-CLASS


CLASS PostBTree [Elem]
IS-SUBTYPE-OF BTree[Elem], Traversable[Elem]
  rename IsEmpty as PostBTree_IsEmpty
EFFECTIVE
TYPE PostBTree
OPERATIONS

  -- Renombrado heredado de Traversable.
  PostBTree_IsEmpty: PostBTree -> Boolean;

  -- Recorrido post-orden: primer elemento y resto.
  first: PostBTree(t) -> Elem
```

```

pre: not PostBTree_IsEmpty(t);

rest: PostBTree(t) -> PostBTree
pre: not PostBTree_IsEmpty(t);

AXIOMS t1, t2: PostBTree; x: Elem;

-- -----
-- PostBTree_IsEmpty (renombrado de Traversable.IsEmpty)
-- -----
PostBTree_IsEmpty(BTree_Empty() as PostBTree) = True;
PostBTree_IsEmpty(BTree_Create(t1, t2, x) as PostBTree) = False;

-- -----
-- first: primer elemento del recorrido post-orden.
-- post(BTree_Create(L, R, x)) = post(L) ++ post(R) ++ [x]
-- por lo tanto first baja primero por L (si no es vacío),
-- después por R (si L es vacío y R no), y solo si ambos
-- subárboles son vacíos devuelve la raíz x (nodo hoja).
-- -----

-- Hoja: ambos subárboles vacíos -> el primer elemento es la raíz.
(BTree_IsEmpty(t1) and BTree_IsEmpty(t2)) implies
  first(BTree_Create(t1, t2, x) as PostBTree) = x;

-- Izquierdo vacío, derecho no -> bajar a la derecha.
(BTree_IsEmpty(t1) and not BTree_IsEmpty(t2)) implies
  first(BTree_Create(t1, t2, x) as PostBTree) = first(t2 as PostBTree);

-- Izquierdo no vacío -> bajar a la izquierda.
not BTree_IsEmpty(t1) implies
  first(BTree_Create(t1, t2, x) as PostBTree) = first(t1 as PostBTree);

-- -----
-- rest: árbol resultante de eliminar el first.
-- -----

-- Hoja: el first era la raíz -> queda el árbol vacío.
(BTree_IsEmpty(t1) and BTree_IsEmpty(t2)) implies
  rest(BTree_Create(t1, t2, x) as PostBTree) = BTree_Empty() as PostBTree;

-- Izquierdo vacío, derecho no -> el rest se aplica al subárbol derecho.
(BTree_IsEmpty(t1) and not BTree_IsEmpty(t2)) implies
  rest(BTree_Create(t1, t2, x) as PostBTree) =
    BTree_Create(t1, rest(t2 as PostBTree), x) as PostBTree;

-- Izquierdo no vacío -> el rest se aplica al subárbol izquierdo.
not BTree_IsEmpty(t1) implies
  rest(BTree_Create(t1, t2, x) as PostBTree) =
    BTree_Create(rest(t1 as PostBTree), t2, x) as PostBTree;

```

2. Estructura de los casos

Cada una de las dos operaciones `first` y `rest` se define por tres implicaciones mutuamente excluyentes sobre el estado de los subárboles. La precondition `not PostBTree_IsEmpty(t)` garantiza que el argumento es siempre de la forma `BTree_Create(t1, t2, x)` as `PostBTree`.

Condición sobre los subárboles	<code>first(BTree_Create(t1, t2, x))</code>	<code>rest(BTree_Create(t1, t2, x))</code>
<code>IsEmpty(t1)</code> and <code>IsEmpty(t2)</code> (nodo hoja)	<code>x</code>	<code>BTree_Empty()</code> as <code>PostBTree</code>
<code>IsEmpty(t1)</code> and not <code>IsEmpty(t2)</code> (solo subárbol derecho)	<code>first(t2 as PostBTree)</code>	<code>BTree_Create(t1, rest(t2 as PostBTree), x)</code> as <code>PostBTree</code>
not <code>IsEmpty(t1)</code> (subárbol izquierdo)	<code>first(t1 as PostBTree)</code>	<code>BTree_Create(rest(t1 as PostBTree), t2, x)</code> as <code>PostBTree</code>

Exhaustividad: las tres condiciones cubren todo el dominio bajo la precondition. Como not `PostBTree_IsEmpty(t)` excluye `BTree_Empty`, el argumento siempre es `BTree_Create(t1, t2, x)` as `PostBTree`. Las tres alternativas para `(IsEmpty(t1), IsEmpty(t2))` son: (T,T), (T,F) y cualquiera con not `IsEmpty(t1)`. No hay solapamiento.

Generado desde `Traversable_PostBTree.nereus`. Spec validada con el analyzer del workspace: 0 errores. El único warning (`BTree` no declarado en este archivo) se resuelve al indexar el workspace completo en VS Code.