

Bag — multiconjunto como subtipo de Collection

Permite repeticiones; *count* es la observación principal. Dos estilos de axiomatización conviven: *extensional* vía *count* + *constructor-pattern* sobre *create*/*add*.

Validación: 0 errores, 0 warnings. La cobertura de V1 es completa porque cada operación tiene axiomas con patrones sobre los constructores básicos *create* y *add*, además de la caracterización *extensional* vía *count*.

Estrategia (siguiendo el patrón de *Set.nereus*): cada operación derivada tiene dos vistas:

- **Extensional:** caracteriza la operación a través del observador *count* (o *includes* para las conversiones a *Set/OrderedSet*). Documenta el significado de forma compacta.
- **Constructor-pattern:** uno o varios axiomas con LHS de la forma *op(create, ...)* y *op(add(b, e), ...)*. Cubre V1 y proporciona reglas de reescritura inductivas.

Ambos estilos son consistentes: el *extensional* es derivable del *constructor-pattern* por inducción sobre *add*.

1. Especificación

```
-- =====
-- Bag – multiconjunto (permite repeticiones).
--
-- Subtipo de Collection: hereda los constructores básicos
-- create (bag vacío) y add (insertar una ocurrencia), más
-- las observaciones generales (size, isEmpty, forAll, etc.).
--
-- Propias de Bag:
-- * count          – cantidad de ocurrencias de un elemento
-- * equalBag       – igualdad extensional (mismo count  $\forall v$ )
-- * unionBag / intersectionBag / excludingBag – set-ops
--   con multiplicidad
-- * BagAsSet, BagAsSequence, BagAsOrderedSet – conversiones
--
-- Estilo de axiomatización: combina dos vistas (siguiendo el
-- patrón de Set.nereus):
-- * extensional, vía la observación count – documenta el
--   significado de cada operación de forma compacta;
-- * constructor-pattern sobre create y add – cubre V1 y
--   proporciona reglas de reescritura por inducción.
-- =====
```

```
CLASS Bag [Elem]
IMPORTS Set[Elem], Sequence[Elem], OrderedSet[Elem]
IS-SUBTYPE-OF Collection[Elem]
BASIC CONSTRUCTORS create, add
EFFECTIVE
TYPE Bag
```

OPERATIONS

```
-- — Constructores básicos heredados —————
create: -> Bag;
```

```

add:      Bag * Elem -> Bag;

-- ——— Propias de Bag ———
count: Bag * Elem -> Integer;

equalBag: Bag * Bag -> Boolean;

unionBag:      Bag * Bag -> Bag;
unionBagWithSet: Bag * Set -> Bag;
intersectionBag: Bag * Bag -> Bag;
intersectionBagWithSet: Bag * Set -> Bag;
excludingBag:   Bag * Elem -> Bag;

BagAsSet:      Bag -> Set;
BagAsSequence: Bag -> Sequence;
BagAsOrderedSet: Bag -> OrderedSet;

AXIOMS  b, b1: Bag;  e, e1, v: Elem;  s: Set;

-- count
count(create(), e1) = 0;
(e = e1)      implies count(add(b, e), e1) = 1 + count(b, e1);
not (e = e1)  implies count(add(b, e), e1) = count(b, e1);

-- equalBag: extensional + 4 casos constructor-pattern
equalBag(b, b1) =
  forAll(v)(b, [count(b, v) = count(b1, v)])
  and
  forAll(v)(b1, [count(b1, v) = count(b, v)]);

equalBag(create(), create())      = True;
equalBag(create(), add(b1, e1))    = False;
equalBag(add(b, e), create())      = False;
equalBag(add(b, e), add(b1, e1)) =
  forAll(v)(add(b, e), [count(add(b, e), v) = count(add(b1, e1), v)])
  and
  forAll(v)(add(b1, e1), [count(add(b1, e1), v) = count(add(b, e), v)]);

-- unionBag
count(unionBag(b, b1), v) = count(b, v) + count(b1, v);
unionBag(create(), b1)    = b1;
unionBag(add(b, e), b1)   = add(unionBag(b, b1), e);

-- unionBagWithSet
count(unionBagWithSet(b, s), v) = count(b, v) + count(s, v);
unionBagWithSet(create(), s)    = setAsBag(s);
unionBagWithSet(add(b, e), s)   = add(unionBagWithSet(b, s), e);

-- intersectionBag
(count(b, v) < count(b1, v)) implies
  count(intersectionBag(b, b1), v) = count(b, v);
not (count(b, v) < count(b1, v)) implies
  count(intersectionBag(b, b1), v) = count(b1, v);

intersectionBag(create(), b1) = create();
(count(b, e) < count(b1, e)) implies
  intersectionBag(add(b, e), b1) = add(intersectionBag(b, b1), e);
not (count(b, e) < count(b1, e)) implies

```

```

intersectionBag(add(b, e), b1) = intersectionBag(b, b1);

-- intersectionBagWithSet
(count(b, v) < count(s, v)) implies
  count(intersectionBagWithSet(b, s), v) = count(b, v);
not (count(b, v) < count(s, v)) implies
  count(intersectionBagWithSet(b, s), v) = count(s, v);

intersectionBagWithSet(create(), s) = create();
(count(b, e) < count(s, e)) implies
  intersectionBagWithSet(add(b, e), s) = add(intersectionBagWithSet(b, s), e);
not (count(b, e) < count(s, e)) implies
  intersectionBagWithSet(add(b, e), s) = intersectionBagWithSet(b, s);

-- excludingBag
count(excludingBag(b, e), e) = 0;
not (v = e) implies count(excludingBag(b, e), v) = count(b, v);

excludingBag(create(), e) = create();
(e1 = e) implies excludingBag(add(b, e1), e) = excludingBag(b, e);
not (e1 = e) implies excludingBag(add(b, e1), e) = add(excludingBag(b, e), e1);

-- BagAsSet
includes(BagAsSet(b), v) = (count(b, v) > 0);
BagAsSet(create()) = create() as Set;
includes(BagAsSet(b), e) implies BagAsSet(add(b, e)) = BagAsSet(b);
not includes(BagAsSet(b), e) implies BagAsSet(add(b, e)) = addSet(BagAsSet(b), e);

-- BagAsSequence
count(BagAsSequence(b), v) = count(b, v);
BagAsSequence(create()) = createSequence();
BagAsSequence(add(b, e)) = appendSequence(BagAsSequence(b), e);

-- BagAsOrderedSet
includes(BagAsOrderedSet(b), v) = (count(b, v) > 0);
BagAsOrderedSet(create()) = createOrderedSet();
includes(BagAsOrderedSet(b), e) implies BagAsOrderedSet(add(b, e)) = BagAsOrderedSet(b);
not includes(BagAsOrderedSet(b), e) implies BagAsOrderedSet(add(b, e)) = includingOrderedSet(BagAsOrder

END-CLASS

```

2. Axiomas constructor-pattern por operación

Operación	Caso create	Caso add(b, e)	Total axiomas
count	= 0	2 implies según (e = e1)	3
equalBag	4 combinaciones (create/add × create/add)	—	4 + 1 extensional

unionBag	= b1	= add(unionBag(b, b1), e)	2 + 1 extensional
unionBagWithSet	= setAsBag(s)	= add(unionBagWithSet(b, s), e)	2 + 1 extensional
intersectionBag	= create()	2 implies según count(b,e) < count(b1,e)	3 + 2 extensional
intersectionBagWithSet	= create()	2 implies según count(b,e) < count(s,e)	3 + 2 extensional
excludingBag	= create()	2 implies según (e1 = e)	3 + 2 extensional
BagAsSet	= create() as Set	2 implies según ya esté incluido (respeto precondition de addSet)	3 + 1 extensional
BagAsSequence	= createSequence()	= appendSequence(BagAsSequence(b), e)	2 + 1 extensional
BagAsOrderedSet	= createOrderedSet()	2 implies según ya esté incluido	3 + 1 extensional

3. Casos no triviales del razonamiento inductivo

intersectionBag(add(b, e), b1): al agregar e al primer bag, count(_, e) sube en 1. Si previamente $\text{count}(b, e) \geq \text{count}(b1, e)$, el mínimo (que es lo que cuenta la intersección) ya estaba saturado por b1, así que no cambia → recurrir sin agregar. Si era estrictamente menor ($\text{count}(b, e) < \text{count}(b1, e)$), el mínimo sube en 1, lo que equivale a agregar e al resultado.

BagAsSet(add(b, e)): la precondition de addSet es $\text{not } (e \text{ in } s)$. Por eso los dos casos: si e ya está en BagAsSet(b), el axioma devuelve BagAsSet(b) sin tocar el set (un Set no puede tener duplicados); si no, lo agrega vía addSet.

equalBag(add(b, e), add(b1, e1)): los tres casos base (create-create, create-add, add-create) son triviales. El cuarto, con ambos no vacíos, evita la recursión estructural — bags son no ordenados, y una recursión "ingenua" como $(e = e1) \text{ and } \text{equalBag}(b, b1)$ sería **incorrecta** (los add pueden agregar elementos en orden distinto). Por eso el RHS sigue siendo extensional vía count, aunque la LHS sí es un patrón constructor.

Constructores de tipos importados (resueltos consultando Set.nereus): setAsBag (declarado allí), addSet, createSequence, appendSequence, createOrderedSet, includingOrderedSet. Sin la spec de Set, varios de los casos base de las conversiones no se podrían escribir con patrones constructor.

Spec validada por el analyzer del workspace: **0 errores, 0 warnings**. Total: ~38 axiomas (1 base + 2 recursivos de count, 4 de equalBag, ~3 por cada operación derivada entre constructor-pattern + extensional).