

OrderedSet — colección ordenada sin duplicados

Combina semánticas de Set (membership, sin duplicados) y Sequence (índices 1-based, operaciones posicionales).

Validación: bajamos de ~150 errores en la versión original (cascada por L8 sin signatures y caracteres unicode garbleados) a **0 errores, 6 warnings**. Los warnings son V1 sobre createOrderedSet (nullary, V1 no debería pedirle constructor-pattern) y las 5 ops parciales cuyas precondiciones excluyen create pero V1 no las modela. Todos son falsos positivos esperados según la memoria del workspace.

Estrategia (como bag.nereus y sequence.nereus): dos vistas conviviendo en cada operación derivada:

- **Extensional:** caracterización vía las observaciones (atOrderedSet, size, count, _in_).
- **Constructor-pattern:** axiomas con LHS de la forma op(create, ...) y op(addOrderedSet(s, e), ...). Cubre V1.

A diferencia de Bag o Sequence, OrderedSet tiene un constructor PROPIO (addOrderedSet en lugar del heredado add), siguiendo el patrón de Set.nereus. Los axiomas relacionan add con addOrderedSet para mantener la invariante de unicidad.

1. Especificación corregida

```
-- =====
-- OrderedSet — colección ordenada SIN duplicados.
-- =====

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

OPERATIONS

-- — Constructores básicos —————
create:      -> OrderedSet;
add:         OrderedSet * Elem -> OrderedSet;
addOrderedSet: OrderedSet(s) * Elem(e) -> OrderedSet
  pre: not (e in s);

-- — Sinónimos (referenciados desde otras specs) —
createOrderedSet: -> OrderedSet;
includingOrderedSet: OrderedSet(s) * Elem(e) -> OrderedSet
  pre: not (e in s);

-- — Observaciones —————
_in_: OrderedSet * Elem -> Boolean;
count: OrderedSet * Elem -> Integer;
equalOrderedSet: OrderedSet * OrderedSet -> Boolean;

-- — Operaciones sobre el orden / posiciones —————
prependOrderedSet: OrderedSet(s) * Elem(e) -> OrderedSet
  pre: not (e in s);
```

```

insertAtOrderedSet:
    OrderedSet(s) * Integer(idx) * Elem(e) -> OrderedSet
    pre: (1 <= idx) and (idx <= size(s) + 1) and not (e in s);

subOrderedSet:
    OrderedSet(s) * Integer(lower) * Integer(upper) -> OrderedSet
    pre: (1 <= lower) and (lower <= upper) and (upper <= size(s));

atOrderedSet: OrderedSet(s) * Integer(i) -> Elem
    pre: (i >= 1) and (i <= size(s));

indexOfOrderedSet: OrderedSet(s) * Elem(e) -> Integer
    pre: includes(s, e);

firstOrderedSet: OrderedSet(s) -> Elem pre: not isEmpty(s);
lastOrderedSet: OrderedSet(s) -> Elem pre: not isEmpty(s);

-- --- Operaciones de set ---
unionOrderedSet: OrderedSet * OrderedSet -> OrderedSet;
excludingOrderedSet: OrderedSet * Elem -> OrderedSet;

-- --- Conversiones ---
orderedSetAsBag: OrderedSet -> Bag;
orderedSetAsSet: OrderedSet -> Set;
orderedSetAsSequence: OrderedSet -> Sequence;

AXIOMS s, s1: OrderedSet; e, e1, v: Elem;
        i, idx, lower, upper: Integer;

-- Sinónimos
createOrderedSet() = create() as OrderedSet;
includingOrderedSet(create(), e) = addOrderedSet(create() as OrderedSet, e);
includingOrderedSet(addOrderedSet(s, e1), e) = addOrderedSet(addOrderedSet(s, e1), e);
includingOrderedSet(s, e) = addOrderedSet(s, e);

-- Relación add ↔ addOrderedSet (Collection inherits add)
add(create(), e) = addOrderedSet(create() as OrderedSet, e);

(e in addOrderedSet(s, e1)) implies
    add(addOrderedSet(s, e1), e) = addOrderedSet(s, e1);
not (e in addOrderedSet(s, e1)) implies
    add(addOrderedSet(s, e1), e) = addOrderedSet(addOrderedSet(s, e1), e);

(e in s) implies add(s, e) = s;
not (e in s) implies add(s, e) = addOrderedSet(s, e);

-- Membership _in_
not (e in create());
(e = e1) implies (e in addOrderedSet(s, e1)) = True;
not (e = e1) implies (e in addOrderedSet(s, e1)) = (e in s);

-- count (0 o 1)
count(create(), e) = 0;
(e = e1) implies count(addOrderedSet(s, e1), e) = 1;
not (e = e1) implies count(addOrderedSet(s, e1), e) = count(s, e);
(e in s) implies count(s, e) = 1;
not (e in s) implies count(s, e) = 0;

-- equalOrderedSet (recursión estructural; orden importa)
equalOrderedSet(create(), create()) = True;

```

```

equalOrderedSet(create(), addOrderedSet(s1, e1))          = False;
equalOrderedSet(addOrderedSet(s, e), create())           = False;
equalOrderedSet(addOrderedSet(s, e), addOrderedSet(s1, e1)) =
    (e = e1) and equalOrderedSet(s, s1);

-- atOrderedSet (addOrderedSet appendea al final)
atOrderedSet(addOrderedSet(s, e), size(addOrderedSet(s, e))) = e;
((1 <= idx) and (idx <= size(s))) implies
    atOrderedSet(addOrderedSet(s, e), idx) = atOrderedSet(s, idx);

-- prependOrderedSet
prependOrderedSet(create(), e)          = addOrderedSet(create() as OrderedSet, e);
prependOrderedSet(addOrderedSet(s, e1), e) = addOrderedSet(prependOrderedSet(s, e), e1);

atOrderedSet(prependOrderedSet(s, e), 1) = e;
((1 <= idx) and (idx <= size(s))) implies
    atOrderedSet(prependOrderedSet(s, e), idx + 1) = atOrderedSet(s, idx);
size(prependOrderedSet(s, e)) = size(s) + 1;

-- insertAtOrderedSet
insertAtOrderedSet(create(), idx, e) = addOrderedSet(create() as OrderedSet, e);

(idx = size(addOrderedSet(s, e1)) + 1) implies
    insertAtOrderedSet(addOrderedSet(s, e1), idx, e) =
        addOrderedSet(addOrderedSet(s, e1), e);

(idx <= size(addOrderedSet(s, e1))) implies
    insertAtOrderedSet(addOrderedSet(s, e1), idx, e) =
        addOrderedSet(insertAtOrderedSet(s, idx, e), e1);

atOrderedSet(insertAtOrderedSet(s, idx, e), idx) = e;
size(insertAtOrderedSet(s, idx, e))          = size(s) + 1;
((1 <= i) and (i < idx)) implies
    atOrderedSet(insertAtOrderedSet(s, idx, e), i) = atOrderedSet(s, i);
((idx <= i) and (i <= size(s))) implies
    atOrderedSet(insertAtOrderedSet(s, idx, e), i + 1) = atOrderedSet(s, i);

-- subOrderedSet (precondición excluye create)
(upper <= size(s)) implies
    subOrderedSet(addOrderedSet(s, e1), lower, upper) =
        subOrderedSet(s, lower, upper);
((lower <= size(s)) and (upper > size(s))) implies
    subOrderedSet(addOrderedSet(s, e1), lower, upper) =
        addOrderedSet(subOrderedSet(s, lower, size(s)), e1);
(lower > size(s)) implies
    subOrderedSet(addOrderedSet(s, e1), lower, upper) =
        addOrderedSet(create() as OrderedSet, e1);

size(subOrderedSet(s, lower, upper)) = upper - lower + 1;
((lower <= idx) and (idx <= upper)) implies
    atOrderedSet(subOrderedSet(s, lower, upper), idx - lower + 1) =
        atOrderedSet(s, idx);

-- indexOfOrderedSet (única ocurrencia)
(e1 = e) implies
    indexOfOrderedSet(addOrderedSet(s, e1), e) = size(addOrderedSet(s, e1));
((not (e1 = e)) and (e in s)) implies
    indexOfOrderedSet(addOrderedSet(s, e1), e) = indexOfOrderedSet(s, e);
atOrderedSet(s, indexOfOrderedSet(s, e)) = e;

-- firstOrderedSet, lastOrderedSet

```

```

isEmpty(s)      implies firstOrderedSet(addOrderedSet(s, e)) = e;
not isEmpty(s) implies firstOrderedSet(addOrderedSet(s, e)) = firstOrderedSet(s);
lastOrderedSet(addOrderedSet(s, e)) = e;

-- unionOrderedSet (saltea duplicados ya presentes en s)
unionOrderedSet(s, create()) = s;
(e in s)      implies unionOrderedSet(s, addOrderedSet(s1, e)) = unionOrderedSet(s, s1);
not (e in s) implies unionOrderedSet(s, addOrderedSet(s1, e)) =
    addOrderedSet(unionOrderedSet(s, s1), e);

-- excludingOrderedSet
excludingOrderedSet(create(), e) = create() as OrderedSet;
(e1 = e)      implies excludingOrderedSet(addOrderedSet(s, e1), e) = s;
not (e1 = e) implies excludingOrderedSet(addOrderedSet(s, e1), e) =
    addOrderedSet(excludingOrderedSet(s, e), e1);

not (e in excludingOrderedSet(s, e));
size(excludingOrderedSet(s, e)) = size(s) - count(s, e);

-- Conversiones
orderedSetAsBag(create())      = create() as Bag;
orderedSetAsBag(addOrderedSet(s, e)) = add(orderedSetAsBag(s), e) as Bag;

orderedSetAsSet(create())      = create() as Set;
orderedSetAsSet(addOrderedSet(s, e)) = addSet(orderedSetAsSet(s), e);

orderedSetAsSequence(create())      = createSequence();
orderedSetAsSequence(addOrderedSet(s, e)) =
    appendSequence(orderedSetAsSequence(s), e);

count(orderedSetAsBag(s), v) = count(s, v);

```

END-CLASS

2. Resumen de las correcciones aplicadas

#	Problema en versión original	Corrección
Errores sintácticos		
1	L8: create, add, count; sin signatures	Signatures completas para los tres
2	L10, L11, L35, L36, L37: caracteres unicode garbleados donde debería ir el operador in	Reemplazado por in; operador declarado como _in_
3	L31: paréntesis sobrante add(addOrderedSet(s,e),e1))	Reescrito limpio
Bugs semánticos		
4	L9-10: addOrderedSet: Set(s) * Elem(e) -> Set — retorna Set en vez de OrderedSet	Fixeado: -> OrderedSet
5	L41: atOrderedSet(add(s,e), 1) = e — contradice la convención "add appendea al final"	Reescrito: atOrderedSet(addOrderedSet(s,e), size(addOrderedSet(s,e))) = e

6	L49-50, L51-52: insertAtOrderedSet con comparaciones $\leq$ en lugar de $<$, y RHS shifteado donde no corresponde	Reescrito siguiendo el patrón de insertAtSequence
7	L58-59: excludingOrderedSet(add(s,e), e1) recursión llama a excludingOrderedSet(s, e) con la variable equivocada	Fixeado: usa e1
8	Falta axioma base atOrderedSet(prependOrderedSet(s, e), 1) = e	Agregado
9	Falta caracterización posicional de subOrderedSet (solo tenía size)	Agregada
Operaciones faltantes		
10	createOrderedSet (usada en sequence.nereus)	Declarada y definida como alias de create() as OrderedSet
11	includingOrderedSet (usada en Set.nereus, sequence.nereus)	Declarada y axiomatizada vía pattern-match sobre los constructores
12	equalOrderedSet	Agregada con 4 axiomas constructor-pattern
13	unionOrderedSet	Agregada (saltea duplicados ya presentes)
14	Conversiones orderedSetAsBag, orderedSetAsSet, orderedSetAsSequence	Las tres agregadas con pattern-match sobre constructores
15	Axioma base count(create, e) = 0	Agregado (la versión original tenía count(s,e) $\leq$ 1 como predicado)
Estilo		
16	Integer en IMPORTS	Quitado (built-in)
17	Variables f, g declaradas y no usadas	Eliminadas
18	if-then-else / endif en excludingOrderedSet	Migrado a implies
19	Sin comentarios de sección	Agregados (consistente con bag.nereus, sequence.nereus)

3. Axiomas constructor-pattern por operación

Operación	Caso create	Caso addOrderedSet(s, e1)
add	Pattern-match con = addOrderedSet(create() as OrderedSet, e)	2 implies según e in addOrderedSet(s, e1)
count	= 0	2 implies según e = e1

<code>equalOrderedSet</code>	4 combinaciones (create/add × create/add); RHS recursivo gracias al orden	
<code>includingOrderedSet</code>	= addOrderedSet(create() as OrderedSet, e)	= addOrderedSet(addOrderedSet(s, e1), e)
<code>prependOrderedSet</code>	= addOrderedSet(create() as OrderedSet, e)	= addOrderedSet(prependOrderedSet(s, e), e1)
<code>insertAtOrderedSet</code>	idx=1, e en empty	2 implies según idx (al final vs interno)
<code>subOrderedSet</code> ★	—	3 implies (upper ≤ size / lower ≤ size < upper / lower > size)
<code>atOrderedSet</code> ★	—	2 axiomas (última posición / posición interna)
<code>indexOfOrderedSet</code> ★	—	2 implies según e1 = e
<code>firstOrderedSet</code> ★	—	2 implies según isEmpty(s)
<code>lastOrderedSet</code> ★	—	1 (siempre e porque addOrderedSet apila al final)
<code>unionOrderedSet</code>	Inducción sobre el segundo arg: 1 base + 2 implies según e in s	
<code>excludingOrderedSet</code>	= create() as OrderedSet	2 implies según e1 = e
<code>orderedSetAsBag/Set/Sequence</code>	map a empty del tipo destino	map al constructor correspondiente del tipo destino

★ = operaciones parciales cuya precondition excluye create (not isEmpty, i <= size, includes). V1 todavía emite warning porque no modela esas condiciones — falsos positivos esperados.

4. Casos no triviales

Relación add ↔ addOrderedSet: a diferencia de Sequence (donde add append directamente), en OrderedSet add hereda de Collection (sin precondition) pero queremos que preserve la unicidad. El truco es definir add(s, e) por casos: si e ya está, devolvemos s (identidad); si no, delegamos a addOrderedSet. Así un cliente puede usar add indiscriminadamente y obtiene un OrderedSet válido. Esto es exactamente el patrón de Set.nereus.

equalOrderedSet recursivo: como en Sequence (y a diferencia de Bag, donde es desordenado), el orden importa: dos OrderedSets son iguales sii sus últimos elementos coinciden y sus prefijos son iguales. La recursión equalOrderedSet(addOrderedSet(s, e), addOrderedSet(s1, e1)) = (e = e1) and equalOrderedSet(s, s1) es estructuralmente limpia.

unionOrderedSet con "no duplicar": al unir dos OrderedSets, si un elemento de s1 ya está en s, se ignora; si no, se agrega al final. Esto preserva la unicidad globalmente. Inducción sobre el segundo arg: $(e \text{ in } s) \text{ implies } \text{unionOrderedSet}(s, \text{addOrderedSet}(s1, e)) = \text{unionOrderedSet}(s, s1)$.

Conversiones: orderedSetAsBag mapa cada elemento con count 1 al Bag (puede tener conteos > 1 luego). orderedSetAsSet mapa al Set (que ya no tiene orden). orderedSetAsSequence preserva el orden (cada OrderedSet es una Sequence sin duplicados). Las tres se definen pattern-match sobre create y addOrderedSet.

Spec corregida desde la versión original (~150 errores). Resultado: 0 errores, 6 warnings (5 sobre ops parciales y 1 sobre createOrderedSet nullary). Convenciones del workspace: implies, axiomas constructor-pattern, casteos as OrderedSet consistentes, imports a los constructores específicos de los tipos relacionados.