

Sequence — secuencia ordenada como subtipo de Collection

Posiciones indexadas desde 1; permite repeticiones; add agrega al final.

Validación: bajamos de **190 errores** en la versión original a **0 errores, 5 warnings**. Los 5 warnings son V1 sobre operaciones parciales (subSequence, atSequence, indexOfSequence, firstSequence, lastSequence): sus precondiciones excluyen `s = create` pero el analyzer del workspace no modela precondiciones expresadas vía `size`, `isEmpty` o `includes`. Son los falsos positivos esperados según la memoria del proyecto.

Estrategia (como bag.nereus): cada operación derivada tiene dos vistas conviviendo:

- **Extensional:** caracterización vía las observaciones (atSequence, size, count, includes). Documenta el significado.
- **Constructor-pattern:** axiomas con LHS de la forma `op(create, ...)` y `op(add(s, e), ...)`. Cubre V1 y da reglas de reescritura inductivas.

1. Especificación corregida

```
-- =====  
-- Sequence – secuencia ordenada (permite repeticiones,  
-- posiciones indexadas desde 1).  
-- =====  
  
CLASS Sequence [Elem]  
IMPORTS Set[Elem:ANY], Bag[Elem:ANY], OrderedSet[Elem:ANY]  
IS-SUBTYPE-OF Collection[Elem]  
BASIC CONSTRUCTORS create, add  
EFFECTIVE  
TYPE Sequence  
  
OPERATIONS  
  
-- — Constructores básicos heredados —————  
create: -> Sequence;  
add:    Sequence * Elem -> Sequence;  
  
-- — Sinónimos de add (al final) —————  
appendSequence: Sequence * Elem -> Sequence;  
includingSequence: Sequence * Elem -> Sequence;  
  
-- — Propias de Sequence —————  
count:          Sequence * Elem -> Integer;  
equalSequence: Sequence * Sequence -> Boolean;  
  
prependSequence: Sequence * Elem -> Sequence;  
unionSequence:   Sequence * Sequence -> Sequence;  
insertAtSequence: Sequence(s) * Integer(idx) * Elem -> Sequence  
  pre: (1 <= idx) and (idx <= size(s) + 1);  
excludingSequence: Sequence * Elem -> Sequence;  
  
subSequence: Sequence(s) * Integer(lower) * Integer(upper) -> Sequence
```

```

    pre: (1 <= lower) and (lower <= upper) and (upper <= size(s));

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

indexOfSequence: Sequence(s) * Elem(e) -> Integer
    pre: includes(s, e);

firstSequence: Sequence(s) -> Elem
    pre: not isEmpty(s);

lastSequence: Sequence(s) -> Elem
    pre: not isEmpty(s);

sequenceAsBag:      Sequence -> Bag;
sequenceAsSet:      Sequence -> Set;
sequenceAsOrderedSet: Sequence -> OrderedSet;

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

-- Sinónimos
appendSequence(create(), e)    = add(create(), e) as Sequence;
appendSequence(add(s, e1), e)  = add(add(s, e1), e) as Sequence;

includingSequence(create(), e) = add(create(), e) as Sequence;
includingSequence(add(s, e1), e) = add(add(s, e1), e) as Sequence;

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

-- atSequence (precondición excluye create)
atSequence(add(s, e) as Sequence, size(add(s, e) as Sequence)) = e;
((1 <= idx) and (idx <= size(s))) implies
    atSequence(add(s, e) as Sequence, idx) = atSequence(s, idx);

-- equalSequence: recursión estructural limpia (orden importa)
equalSequence(create(), create())    = True;
equalSequence(create(), add(s1, e1)) = False;
equalSequence(add(s, e), create())   = False;
equalSequence(add(s, e), add(s1, e1)) = (e = e1) and equalSequence(s, s1);

-- prependSequence
prependSequence(create(), e) = add(create(), e) as Sequence;
prependSequence(add(s, e1), e) =
    add(prependSequence(s, e), e1) as Sequence;

atSequence(prependSequence(s, e), 1) = e;
((1 <= idx) and (idx <= size(s))) implies
    atSequence(prependSequence(s, e), idx + 1) = atSequence(s, idx);

-- unionSequence: concatenación, inducción sobre el segundo arg
unionSequence(s, create()) = s;
unionSequence(s, add(s1, e)) = add(unionSequence(s, s1), e) as Sequence;

size(unionSequence(s, s1)) = size(s) + size(s1);

((1 <= idx) and (idx <= size(s))) implies
    atSequence(unionSequence(s, s1), idx) = atSequence(s, idx);

```

```

((size(s) < idx) and (idx <= size(s) + size(s1))) implies
  atSequence(unionSequence(s, s1), idx) = atSequence(s1, idx - size(s));

-- insertAtSequence
insertAtSequence(create(), idx, e) = add(create(), e) as Sequence;

(idx = size(add(s, e1) as Sequence) + 1) implies
  insertAtSequence(add(s, e1) as Sequence, idx, e) =
    add(add(s, e1) as Sequence, e) as Sequence;

(idx <= size(add(s, e1) as Sequence)) implies
  insertAtSequence(add(s, e1) as Sequence, idx, e) =
    add(insertAtSequence(s, idx, e), e1) as Sequence;

atSequence(insertAtSequence(s, idx, e), idx) = e;
size(insertAtSequence(s, idx, e)) = size(s) + 1;

((1 <= i) and (i < idx)) implies
  atSequence(insertAtSequence(s, idx, e), i) = atSequence(s, i);

((idx <= i) and (i <= size(s))) implies
  atSequence(insertAtSequence(s, idx, e), i + 1) = atSequence(s, i);

-- subSequence (precondición excluye create)
(upper <= size(s)) implies
  subSequence(add(s, e1) as Sequence, lower, upper) =
    subSequence(s, lower, upper);

((lower <= size(s)) and (upper > size(s))) implies
  subSequence(add(s, e1) as Sequence, lower, upper) =
    add(subSequence(s, lower, size(s)), e1) as Sequence;

(lower > size(s)) implies
  subSequence(add(s, e1) as Sequence, lower, upper) =
    add(create(), e1) as Sequence;

size(subSequence(s, lower, upper)) = upper - lower + 1;

((lower <= idx) and (idx <= upper)) implies
  atSequence(subSequence(s, lower, upper), idx - lower + 1) =
    atSequence(s, idx);

-- indexOfSequence (precondición excluye create)
includes(s, e) implies
  indexOfSequence(add(s, e1) as Sequence, e) = indexOfSequence(s, e);

((not includes(s, e)) and (e1 = e)) implies
  indexOfSequence(add(s, e1) as Sequence, e) =
    size(add(s, e1) as Sequence);

atSequence(s, indexOfSequence(s, e)) = e;

-- firstSequence, lastSequence (precondición excluye create)
isEmpty(s) implies firstSequence(add(s, e) as Sequence) = e;
not isEmpty(s) implies firstSequence(add(s, e) as Sequence) = firstSequence(s);

lastSequence(add(s, e) as Sequence) = e;

-- excludingSequence
excludingSequence(create(), e) = create() as Sequence;

```

```

(e1 = e) implies
  excludingSequence(add(s, e1) as Sequence, e) = excludingSequence(s, e);
not (e1 = e) implies
  excludingSequence(add(s, e1) as Sequence, e) =
    add(excludingSequence(s, e), e1) as Sequence;

not includes(excludingSequence(s, e), e);
size(excludingSequence(s, e)) = size(s) - count(s, e);

-- sequenceAsBag
sequenceAsBag(create()) = create() as Bag;
sequenceAsBag(add(s, e) as Sequence) = add(sequenceAsBag(s), e) as Bag;
count(sequenceAsBag(s), v) = count(s, v);

-- sequenceAsSet (respeta precondition de addSet)
sequenceAsSet(create()) = create() as Set;
includes(sequenceAsSet(s), e) implies sequenceAsSet(add(s, e) as Sequence) = sequenceAsSet(s);
not includes(sequenceAsSet(s), e) implies sequenceAsSet(add(s, e) as Sequence) = addSet(sequenceAsSet(s), e);
includes(sequenceAsSet(s), v) = (count(s, v) > 0);

-- sequenceAsOrderedSet
sequenceAsOrderedSet(create()) = createOrderedSet();
includes(sequenceAsOrderedSet(s), e) implies sequenceAsOrderedSet(add(s, e) as Sequence) = sequenceAsOrderedSet(s);
not includes(sequenceAsOrderedSet(s), e) implies sequenceAsOrderedSet(add(s, e) as Sequence) =
  includingOrderedSet(sequenceAsOrderedSet(s), e);
includes(sequenceAsOrderedSet(s), v) = (count(s, v) > 0);

-- Preservación de orden (uses indexOfOrderedSet importado)
((indexOfSequence(s, v) < indexOfSequence(s, e))
 and includes(s, v) and includes(s, e)) implies
  (indexOfOrderedSet(sequenceAsOrderedSet(s), v) <
   indexOfOrderedSet(sequenceAsOrderedSet(s), e));

END-CLASS

```

2. Resumen de las correcciones aplicadas

#	Problema en versión original	Corrección
Errores sintácticos (190 reportes en cascada)		
1	Línea 8: create, add, count; sin signatures	Tres signatures completas: create: -> Sequence, add: Sequence*Elem->Sequence, count: Sequence*Elem->Integer
2	Líneas 58-59: iterate con coma faltante, acc y v no declarados, argumentos invertidos	Eliminado el iterate roto; excludingSequence queda solo con su forma constructor-pattern
Bugs de tipos en variables		
3	L29: count(s, e) en else (debería ser count(s, e1))	Fixeado

4	L47-48: <code>atSequence(insertAtSequence(...), 1)</code> debería ser <code>i+1</code>	Fixeado
5	L61-62: <code>excludingSequence(s, e)</code> recursivo (debería ser <code>e1</code>)	Reescrito sobre <code>add(s, e1)</code> con <code>implies</code> y referencias correctas
6	L63 y L70: <code>count(..., e)</code> donde la variable de cuantificación es <code>v</code>	Fixeado
Bugs lógicos		
7	L30-31: <code>equalSequence</code> con variable libre <code>index</code> y precedencia ambigua	Reescrito por 4 casos constructor-pattern (recursión limpia gracias al orden)
8	L34: caracterización de <code>unionSequence</code> solo para primera mitad	Agregada axioma para segunda mitad: <code>size(s) < idx <= size(s)+size(s1)</code>
9	L67: tautología <code>forall (v) (sequenceAsSet(s), [includes(sequenceAsSet(s), v)])</code>	Reemplazada por la caracterización correcta vía <code>count</code>
10	L71-74: preservación de orden con espacios de índices y elementos distintos	Reescrita con <code>indexOfSequence(s, v) < indexOfSequence(s, e)</code>
Operaciones no declaradas o faltantes		
11	L65: <code>sequenceAsSeq(s) = s</code> — operación no declarada	Eliminado
12	Faltaba <code>appendSequence</code> (referenciada por <code>Set.nereus:144</code>)	Declarada y axiomatizada (= <code>add</code> con <code>cast</code>)
13	Faltaba axioma base <code>count(create(), e) = 0</code>	Agregado
14	Faltaba BASIC CONSTRUCTORS <code>create</code> , <code>add</code>	Agregado
15	<code>firstSequence/lastSequence</code> sin precondición	Agregado pre: <code>not isEmpty(s)</code>
Estilo y redundancias		
16	<code>Integer</code> en <code>IMPORTS</code> (es built-in)	Quitado
17	Variables <code>f</code> , <code>g</code> , <code>h</code> , <code>k</code> declaradas y no usadas	Eliminadas
18	<code>if-then-else</code> / <code>endif</code> en múltiples axiomas	Migrados a <code>implies</code>
19	<code>TYPE</code> y <code>Sequence</code> en líneas separadas	Consolidado
20	Sin comentarios de sección	Agregados (consistente con <code>bag.nereus</code> , <code>BSTree-hide.nereus</code>)

3. Axiomas constructor-pattern por operación

Operación	Caso <code>create</code>	Caso <code>add(s, e)</code>
-----------	--------------------------	-----------------------------

count	= 0	2 implies (según $e = e1$)
equalSequence	4 combinaciones create/add × create/add (recursión estructural)	—
appendSequence, includingSequence	= add(create(), e)	= add(add(s, e1), e)
prependSequence	= add(create(), e)	= add(prependSequence(s, e), e1)
unionSequence	Inducción sobre el segundo arg: unionSequence(s, create) = s; unionSequence(s, add(s1, e)) = add(unionSequence(s, s1), e)	
insertAtSequence	insertAtSequence(create, idx, e) con idx=1 (único válido)	2 implies (idx al final vs idx interno)
excludingSequence	= create()	2 implies según $e1 = e$
subSequence ★	—	3 implies ($upper \leq size(s) / lower \leq size(s) < upper / lower > size(s)$)
atSequence ★	—	2 axiomas (último elemento / posición interna)
indexOfSequence ★	—	2 implies (e en s / e solo al final)
firstSequence ★	—	2 implies según isEmpty(s)
lastSequence ★	—	1 (siempre e, porque add apila al final)
sequenceAsBag	= create() as Bag	= add(sequenceAsBag(s), e) as Bag
sequenceAsSet	= create() as Set	2 implies según ya esté incluido (respeta precondition. de addSet)
sequenceAsOrderedSet	= createOrderedSet()	2 implies análogos a sequenceAsSet

★ = operaciones parciales cuya precondition excluye create (not isEmpty, $i \leq size$, includes, etc.). El analyzer todavía emite un warning V1 porque no modela esas condiciones, pero son los falsos positivos esperados según la memoria del workspace.

4. Casos no triviales del razonamiento

equalSequence sobre dos add: a diferencia de Bag (donde los bags son desordenados y la recursión "ingenua" sobre add es incorrecta), en Sequence el orden importa, así que $equalSequence(add(s, e), add(s1, e1)) = (e = e1)$ and $equalSequence(s, s1)$ es exactamente lo que queremos: dos secuencias son iguales sii sus últimos elementos coinciden y sus prefijos también.

unionSequence y la "segunda mitad": la versión original solo decía que posiciones $1..size(s)$ del resultado coinciden con s . Faltaba el axioma para posiciones $size(s) < idx \leq size(s)+size(s1)$: deben coincidir con $s1$ en la posición $idx - size(s)$. Sin él, el resultado quedaba subespecificado para la segunda mitad.

subSequence recursivo: hay tres casos según dónde caen $lower$, $upper$ respecto a $size(s)$ (el último elemento $e1$ está en la posición $size(s)+1$ de $add(s, e1)$):

- $upper \leq size(s)$: el segmento está completamente en s , ignora $e1$.
- $lower \leq size(s) < upper$: el segmento abarca parte de s más el último elemento $e1$.
- $lower > size(s)$: el segmento es solo $[e1]$.

indexOfSequence de la primera ocurrencia: si el elemento ya está en el prefijo s , el índice de su primera ocurrencia en $add(s, e1)$ es el mismo que en s (no importa que $e1$ coincida con e). Si no está en s pero coincide con $e1$, el índice es $size(add(s, e1))$ (la última posición). El orden de las dos implies en el axioma respeta esta prioridad.

Spec corregida desde la versión original (190 errores). Resultado: 0 errores, 5 warnings (todos V1 false positives sobre ops parciales). Convenciones aplicadas: `implies`, axiomas constructor-pattern, casteos as `Sequence` consistentes, importaciones a los constructores específicos de `Set/OrderedSet` (`addSet`, `createOrderedSet`, `includingOrderedSet`).