Algoritmos y Estructura de Datos I

Primer cuatrimestre de 2012

25 de Junio de 2012

RTPI OJOTA (Organización de Juegos Olímpicos Tp de Algoritmos 1) v1.0

1. Tipo Lista⟨T⟩

```
tipo Lista⟨T⟩ {
  observador asSeq (l : Lista⟨T⟩) : [T];
}
```

El observador asSeq devuelve una lista del lenguaje de especificación que contiene los mismos elementos, y en el mismo orden, que la Lista⟨T⟩ recibida. En adelante, haremos un abuso de notación, y omitiremos el observador asSeq en las especificaciones que involucren al tipo Lista⟨T⟩. Uds. pueden hacer lo mismo si lo desean.

```
problema Lista⟨T⟩ (this : Lista⟨T⟩) {
  modifica this;
  asegura this == [];
}

problema longitud (this : Lista⟨T⟩) = result : ℤ {
  asegura result == |this|;
}

problema iesimo (this : Lista⟨T⟩,i : ℤ) = result : T {
  requiere i ∈ [0..|this|);
  asegura result = this_i;
}

problema agregar (this : Lista⟨T⟩,e : T) {
  modifica this;
  asegura this == e : pre(this);
}

problema agregarAtras (this : Lista⟨T⟩,e : T) {
  modifica this;
  asegura this == pre(this) ++[e];
}

problema cabeza (this : Lista⟨T⟩) = result : T {
  requiere |this| > 0;
  asegura result == cabeza(this);
}

problema cola (this : Lista⟨T⟩) {
  modifica this;
  requiere |this| > 0;
  asegura this == cola(pre(this));
}

problema pertenece (this : Lista⟨T⟩,e : T) = result : Bool {
  asegura result == (e ∈ this);
}

problema concatenar (this,otraLista : Lista⟨T⟩) {
  modifica this;
  asegura this == pre(this) ++otraLista;
}

problema operator== (this,otraLista : Lista⟨T⟩) {
  asegura result == (this == otraLista);
```

}

problema sacar (this : Lista⟨T⟩,e : T) {
  modifica $this$;
  asegura $this == [\, x \mid x \leftarrow pre(this), x \neq e \,]$;
}

problema darVuelta (this : Lista⟨T⟩) {
  modifica $this$;
  asegura $|this| == |pre(this)| \wedge (\forall i \leftarrow [0..|this|))\, this_i = pre_{|this|-i-1}$;
}

problema posicion (this : Lista⟨T⟩,e : T) = result : $\mathbb{Z}$ {
  requiere $e \in this$;
  asegura $result = [\, i \mid i \leftarrow [0..|this|), this_i = e \,]_0$;
}

problema eliminarPosicion (this : Lista⟨T⟩,i : $\mathbb{Z}$) {
  modifica $this$;
  requiere $i \in [0..|pre(this)|)$;
  asegura $this = pre(this)[0..i) + +pre(this)(i..|pre(this)|)$;
}

problema cantidadDeApariciones (this : Lista⟨T⟩, e:T) = result : $\mathbb{Z}$ {
  asegura $result = |[\, e \mid e' \leftarrow this, e' == e \,]|$;
}

2. Tipos

tipo Deporte = String;
tipo Pais = String;
tipo Sexo = Femenino, Masculino;

3. Atleta

tipo Atleta {
  observador nombre (a: Atleta) : String;
  observador sexo (a: Atleta) : Sexo;
  observador añoNacimiento (a: Atleta) : $\mathbb{Z}$;
  observador nacionalidad (a: Atleta) : Pais;
  observador ciaNumber (a: Atleta) : $\mathbb{Z}$;
  observador deportes (a: Atleta) : [Deporte];
  observador capacidad (a: Atleta, d: Deporte) : $\mathbb{Z}$;
      requiere $d \in deportes(a)$;
  invariante $sinRepetidos(deportes(a))$;
  invariante $ordenada(deportes(a))$;
  invariante capacidadEnRango : $(\forall d \leftarrow deportes(a))0 \leq capacidad(a,d) \leq 100$;
}

problema Atleta (this: Atleta, nom : String, s : Sexo, a : $\mathbb{Z}$, nac : Pais, cia : $\mathbb{Z}$) {
  modifica $this$;
  asegura $nombre(this) == nom$;
  asegura $sexo(this) == s$;
  asegura $añoNacimiento(this) == a$;
  asegura $nacionalidad(this) == nac$;
  asegura $ciaNumber(this) == cia$;
  asegura $deportes(this) == []$;
}

problema nombre (this : Atleta) = result : String {
  asegura $nombre(this) == result$;
}

```
problema sexo (this : Atleta) = result : Sexo {
  asegura sexo(this) == result;
}

problema anioNacimiento (this : Atleta) = result : ℤ {
  asegura añoNacimiento(this) == result;
}

problema nacionalidad (this : Atleta) = result : Pais {
  asegura nacionalidad(this) == result;
}

problema ciaNumber (this : Atleta) = result : ℤ {
  asegura ciaNumber(this) == result;
}

problema deportes (this : Atleta) = result : [Deporte] {
  asegura deportes(this) == result;
}

problema capacidad (this : Atleta, d : Deporte) = result : ℤ {
  requiere d ∈ deportes(this);
  asegura capacidad(this, d) == result;
}

problema entrenarNuevoDeporte (this: Atleta, d: Deporte, c: ℤ) {
  requiere 0 ≤ c ≤ 100;
  modifica this;
  asegura nombre(this) == nombre(pre(this));
  asegura sexo(result) == sexo(pre(this));
  asegura añoNacimiento(this) == añoNacimiento(pre(this));
  asegura nacionalidad(this) == nacionalidad(pre(this));
  asegura ciaNumber(this) == ciaNumber(pre(this));
  asegura mismos(deportes(this), sacarRepetidos(d : deportes(pre(this))));
  asegura (∀x ← deportes(this), x ≠ d)capacidad(this, x) == capacidad(pre(this), x);
  asegura capacidad(this, d) == c;
}

problema operator== (this, a: Atleta) = result : Bool {
  asegura result == mismosDatosPersonales(this, a) ∧ mismosDeportesYCapacidades(this, a);
  aux mismosDatosPersonales (a1, a2: Atleta) : Bool = nombre(a1) == nombre(a2) ∧ sexo(a1) == sexo(a2)
      ∧ añoNacimiento(a1) == añoNacimiento(a2) ∧ nacionalidad(a1) == nacionalidad(a2)
      ∧ ciaNumber(a1) == ciaNumber(a2);
  aux mismosDeportesYCapacidades (a1, a2: Atleta) : Bool =
      deportes(a1) == deportes(a2) ∧ (∀d ← deportes(a1))capacidad(a1, d) == capacidad(a2, d);
}
```

4. Competencia

```
tipo Competencia {
  observador categoria (c: Competencia) : (Deporte, Sexo);
  observador participantes (c: Competencia) : [Atleta];
  observador finalizada (c: Competencia) : Bool;
  observador ranking (c: Competencia) : [Atleta];
       requiere finalizada(c);
  observador lesTocoControlAntiDoping (c: Competencia) : [Atleta];
       requiere finalizada(c);
  observador leDioPositivo (c: Competencia, a: Atleta) : Bool;
       requiere finalizada(c) ∧ a ∈ lesTocoControlAntiDoping(c);

  invariante participaUnaSolaVez : sinRepetidos(ciaNumbers(participantes(c)));
  invariante participantesPertenecenACat :
      (∀p ← participantes(c))prm(categoria(c)) ∈ deportes(p) ∧ sgd(categoria(c)) == sexo(p);
```

```
  invariante elRankingEsDeParticipantesYNoHayRepetidos :
      finalizada(c) ⇒ incluida(ranking(c), participantes(c)) ;
  invariante seControlanParticipantesYNoHayRepetidos :
      finalizada(c) ⇒ incluida(lesTocoControlAntiDoping(c), participantes(c)) ;
}

problema Competencia (this: Competencia, d: Deporte, s: Sexo, as: [Atleta])  {
  requiere sonDeEstaCategoria : (∀a ← as)d ∈ deportes(a) ∧ s == sexo(a) ;
  requiere noHayAtletasRepetidos : sinRepetidos(ciaNumbers(as)) ;
  modifica this ;
  asegura categoria(this) == (d, s) ;
  asegura mismos(participantes(this), as) ;
  asegura ¬finalizada(this) ;
}

problema categoria (this : Competencia) = result : (Deporte,Sexo)  {
  asegura categoria(this) == result ;
}

problema participantes (this : Competencia) = result : [Atleta]  {
  asegura mismos(participantes(this), result) ;
}

problema finalizada (this : Competencia) = result : Bool  {
  asegura finalizada(this) == result ;
}

problema ranking (this : Competencia) = result : [Atleta]  {
  requiere finalizada(this) ;
  asegura ranking(this) == result ;
}

problema lesTocoControlAntiDoping (this : Competencia) = result : [Atleta]  {
  requiere finalizada(this) ;
  asegura mismos(lesTocoControlAntiDoping(this), result) ;
}

problema leDioPositivo (this : Competencia, a : Atleta) = result : Bool  {
  requiere finalizada(this) ;
  requiere a ∈ lesTocoControlAntiDoping(this) ;
  asegura leDioPositivo(this, a) == result ;
}

problema finalizar (this: Competencia, posiciones: [ℤ], control: [(ℤ, Bool)])  {
  requiere ¬finalizada(this) ;
  requiere incluida(posiciones, ciaNumbers(participantes(this))) ;
  requiere incluida(primeros(control), ciaNumbers(participantes(this))) ;
  modifica this ;
  asegura seMantieneCategoria : categoria(this) == categoria(pre(this)) ;
  asegura seMantienenenParticipantes : mismos(participantes(this), participantes(pre(this))) ;
  asegura finalizo : finalizada(this) ;
  asegura rankingOrdenado : ciaNumbers(ranking(this)) == posiciones ;
  asegura quienesSeControlan : mismos(ciaNumbers(lesTocoControlAntiDoping(this)), primeros(control)) ;
  asegura resultadosDeControl : (∀x ← control)leDioPositivo(this, elAtleta(participantes(this), prm(x))) ⇔ sgd(x) ;
  aux elAtleta (as: [Atleta], x:ℤ) : Atleta = [a|a ← as, ciaNumber(a) == x]0 ;
}

problema linfordChristie (this: Competencia, ciaNum: ℤ)  {
  requiere noFinalizada : ¬finalizada(this) ;
  requiere esParticipante : ciaNum ∈ ciaNumbers(participantes(this)) ;
  modifica this ;
  asegura seMantieneCategoria : categoria(this) == categoria(pre(this)) ;
  asegura soloUnoDescalificado : mismos(participantes(this), [a|a ← participantes(pre(this)), ciaNumber(a) ≠ ciaNum]) ;
  asegura noFinalizada : ¬finalizada(this) ;
}
```

```
problema gananLosMasCapaces (this: Competencia) = result : Bool {
  requiere seConocenResultados : finalizada(this);
  asegura result == ordenada(reverso(capacidades(ranking(this), deporte(this))));
}

problema sancionarTramposos (this: Competencia) {
  requiere seConocenResultados : finalizada(this);
  modifica this;
  asegura seMantieneCategoria : categoria(this) == categoria(pre(this));
  asegura seMantienenParticipantes : mismos(participantes(this), participantes(pre(this)));
  asegura sigueFinalizada : finalizada(this);
  asegura drogadosDescalificados : ranking(this) == [a | a ← ranking(pre(this)), noLoDescubrenDopado(a, pre(this))];
  asegura seMantieneControl : mismos(lesTocoControlAntiDoping(this), lesTocoControlAntiDoping(pre(this)));
  asegura mismosResultadosControl :
    (∀a ← lesTocoControlAntiDoping(this))leDioPositivo(this, a) ⇔ leDioPositivo(pre(this), a);
  aux noLoDescubrenDopado (a: Atleta, c: Competencia) : Bool =
    a ∉ lesTocoControlAntiDoping(c) ∨ ¬leDioPositivo(c, a);
}

problema operator== (this,c: Competencia) = result : Bool {
  asegura mismosParticipantesYCategoria(this, c) ∧ finalizada(this) == finalizada(c) ∧
    (finalizada(this) ⟶ ranking(this) == ranking(c) ∧ coincidenControlesAntidoping(this, c));
  aux mismosParticipantesYCategoria (c1, c2: Competencia) : Bool = mismos(participantes(c1), participantes(c2)) ∧
    categoria(c1) == categoria(c2);
  aux coincidenControlesAntidoping (c1,c2: Competencia) : Bool =
    mismos(lesTocoControlAntidoping(c1), lesTocoControlAntidoping(c2)) ∧
    (∀a ← lesTocoControlAntidoping(c1))leDioPositivo(c1, a) == leDioPositivo(c2, a);
}

problema clasificoTarde (this: Competencia, a: Atleta) {
  requiere noFinalizada : ¬finalizada(this);
  requiere noEsParticipante : a ∉ participantes(this);
  modifica this;
  asegura seMantieneCategoria : categoria(this) == categoria(pre(this));
  asegura soloUnoDescalificado : mismos(participantes(this), a : participantes(pre(this)));
  asegura noFinalizada : ¬finalizada(this);
}
```

5. JJOO

```
tipo JJOO {
  observador año (j: JJOO) : ℤ;
  observador atletas (j: JJOO) : [Atleta];
  observador cantDias (j: JJOO) : ℤ;
  observador cronograma (j: JJOO, dia: ℤ) : [Competencia];
      requiere 1 ≤ dia ≤ cantDias(j);
  observador jornadaActual (j: JJOO) : ℤ;
  invariante atletasUnicos : sinRepetidos(ciaNumbers(atletas(j)));
  invariante unaDeCadaCategoria : (∀i, k ← [0..|competencias(j)|), i ≠ k)
    categoria(competencias(j)_i) ≠ categoria(competencias(j)_k);
  invariante competidoresInscriptos : (∀c ← competencias(j))incluida(participantes(c), atletas(j));
  invariante jornadaValida : 1 ≤ jornadaActual(j) ≤ cantDias(j);
  invariante finalizadasSiiYaPasoElDia : lasPasadasFinalizaron(j) ∧ lasQueNoPasaronNoFinalizaron(j);
}

problema JJOO (this: JJOO, año: ℤ, as: [Atleta], cron: [[Competencia]]) {
  requiere sinRepetidos(ciaNumbers(as));
  requiere (∀cs ← concat(cron))(¬(∃i, j ← [0..|concat(cron)|), i ≠ j)categoria(cs_i) == categoria(cs_j)));
  requiere (∀cs ← concat(cron))incluida(participantes(cs), as);
  requiere |cron| ≥ 1;
  requiere (∀c ← concat(cron))¬finalizada(c);
```

```
  modifica this;
  asegura año == año(this);
  asegura mismos(atletas(this), as);
  asegura |cron| == cantDias(this);
  asegura (∀j ← [0..|cron|))mismos(cron_j, cronograma(this, j + 1));
  asegura jornadaActual(this) == 1;
}

problema año (this: JJOO) = result : ℤ {
  asegura año(this) == result;
}

problema atletas (this: JJOO) = result : [Atleta] {
  asegura mismos(atletas(this), result);
}

problema cantDias (this: JJOO) = result : ℤ {
  asegura cantDias(this) == result;
}

problema jornadaActual (this: JJOO) = result : ℤ {
  asegura jornadaActual(this) == result;
}

problema cronograma (this: JJOO, d: ℤ) = result : [Competencia] {
  requiere 1 ≤ d ≤ cantDias(this);
  asegura cronograma(this, d) == result;
}

problema competencias (this: JJOO) = result : [Competencia] {
  asegura result == competencias(j);
}

problema competenciasFinalizadasConOroEnPodio (this: JJOO) = result : [Competencia] {
  asegura result == [c|c ← competencias(j), finalizada(c) ∧ |ranking(c)| > 0];
}

problema dePaseo (this: JJOO) = result : [Atleta] {
  asegura noParticipanEnNinguna : mismos(result, fueronAPasear(this));

  aux fueronAPasear (j: JJOO) : [Atleta] = [ a | a ← atletas(j), ¬(∃c ← competencias(j))a ∈ participantes(c) ];
}

problema medallero (this: JJOO) = result : [(Pais, [ℤ])] {
  asegura paisesConMedallas : mismos(primeros(result), paisesQueGanaron(this));
  asegura cantidadMedallasCorrecta : (∀m ← result) |sgd(m)| == 3 ∧
      sgd(m)_0 == |filtrarPorPais(medallistasOro(this), prm(m))| ∧
      sgd(m)_1 == |filtrarPorPais(medallistasPlata(this), prm(m))| ∧
      sgd(m)_2 == |filtrarPorPais(medallistasBronce(this), prm(m))|;
  asegura bienOrdenada : (∀i ← (0.. |result|))masMedallas(sgd(result_{i−1}), sgd(result_i));

  aux paisesQueGanaron (j: JJOO) : [Pais] = sacarRepetidos(nacionalidades(medallistasOro(j) ++
      medallistasPlata(j) ++medallistasBronce(j)));
  aux masMedallas (x, y: [ℤ]) : Bool = x_0 > y_0 ∨ (x_0 == y_0 ∧ x_1 > y_1) ∨ (x_0 == y_0 ∧ x_1 == y_1 ∧ x_2 ≥ y_2);
}

problema boicotPorDisciplina (this: JJOO, cat: (Deporte, Sexo), p: Pais) = result : ℤ {
  requiere esCategoriaValida : (∃c ← competencias(this))categoria(c) == cat;
  modifica this;
  asegura soloCambiaCronograma : año(this) == año(pre(this)) ∧ cantDias(this) == cantDias(pre(this))
      ∧ jornadaActual(this) == jornadaActual(pre(this)) ∧ mismos(atletas(this), atletas(pre(this)));
  asegura mismaCantDeCompetencias : (∀d ← [1..cantDias(this)]) |cronograma(this, d)| == |cronograma(pre(this), d)|;
  asegura lasOtrasCompetenciasNoCambian : (∀d ← [1..cantDias(this)])(∀c ← cronograma(pre(this), d), categoria(c) ≠
      cat)laCompetenciaSeMantiene(this, d, c);
  asegura boicotAEsaCat : (∃c ← cronograma(this, elDiaDeEsaCat(pre(this), cat)))
      igualSalvoBoicot(c, competenciaDeCat(pre(this), cat), p);
  asegura result == |filtrarPorPais(participantes(competenciaDeCat(pre(this), cat)), p)|;
```

aux elDiaDeEsaCat (j: JJOO, cat: (Deporte, Sexo)) : $\mathbb{Z}$ = $[\, d \,|\, d \leftarrow [1..cantDias(j)], (\exists c \leftarrow cronograma(j,d))categoria(c) == cat\,]_0$;
aux competenciaDeCat (j: JJOO, cat: (Deporte, Sexo)) : Competencia = $[\, c \,|\, c \leftarrow competencias(j), categoria(c) == cat\,]_0$;
aux igualSalvoBoicot (c, prec: Competencia, p: Pais) : Bool = $categoria(c) == categoria(prec) \land mismos(participantes(c), sacarLosDePais(participantes(prec), p) \land finalizada(c) \Leftrightarrow finalizada(prec) \land finalizada(c) \Rightarrow (ranking(c) == sacarLosDePais(ranking(prec), p) \land mismos(lesTocoControlAntiDoping(c), sacarLosDePais(lesTocoControlAntiDoping(prec), p)) \land (\forall a \leftarrow lesTocoControlAntiDoping(c))leDioPositivo(c, a) \Leftrightarrow leDioPositivo(prec, a))$;
aux sacarLosDePais (as: [Atleta], p: Pais) : [Atleta] = $[\, a \,|\, a \leftarrow as, nacionalidad(a) \neq p\,]$;
}

problema losMasFracasados (this: JJOO, p: Pais) = result : [Atleta] {
asegura $mismos(result, noGanaronMedallas(this, losMasParticipantes(this, atletasDelPais(this, p))))$;
aux atletasDelPais (j: JJOO, p: Pais) : [Atleta] = $[\, a \,|\, a \leftarrow atletas(j), nacionalidad(a) == p\,]$;
aux losMasParticipantes (j: JJOO, as: [Atleta]) : [Atleta] = $[\, a \,|\, a \leftarrow as, (\forall x \leftarrow as)cantCompetencias(j, a) \geq cantCompetencias(j, x)\,]$;
aux cantCompetencias (j: JJOO, a: Atleta) : $\mathbb{Z}$ = $|[\, c \,|\, c \leftarrow competencias(j), a \in participantes(c)\,]|$;
aux noGanaronMedallas (j: JJOO, as: [Atleta]) : [Atleta] = $[\, a \,|\, a \leftarrow as, cantMedallas(j, a) == 0\,]$;
aux cantMedallas (j: JJOO, a: Atleta) : $\mathbb{Z}$ = $|[\, c \,|\, c \leftarrow competencias(j), estaEnElPodio(c, a)\,]|$;
aux estaEnElPodio (c: Competencia, a: Atleta) : Bool = $finalizada(c) \land (salioPrimero(c, a) \lor salioSegundo(c, a) \lor salioTercero(c, a))$;
aux salioPrimero (c: Competencia, a: Atleta) : Bool = $|ranking(c)| \geq 1 \land ranking(c)_0 == a$;
aux salioSegundo (c: Competencia, a: Atleta) : Bool = $|ranking(c)| \geq 2 \land ranking(c)_1 == a$;
aux salioTercero (c: Competencia, a: Atleta) : Bool = $|ranking(c)| \geq 3 \land ranking(c)_2 == a$;
}

problema liuSong (this: JJOO, a: Atleta, p: País) {
requiere estaLiu : $a \in atletas(this)$;
modifica $this$;
asegura loDemasIgual : $año(this) == año(pre(this)) \land cantDias(this) == cantDias(pre(this)) \land jornadaActual(this) == jornadaActual(pre(this))$;
asegura mismaCantidadAtletas : $|atletas(this)| == |atletas(pre(this))|$;
asegura atletasIguales : $(\forall at1 \leftarrow atletas(pre(this)), \neg(at1 == a))at1 \in atletas(this)$;
asegura cambioLiu : $(\forall at1 \leftarrow atletas(pre(this)), at1 == a)(\exists at2 \leftarrow atletas(this))igualSalvoPais(at1, at2, p)$;
asegura mismaCantDeCompetencias : $(\forall d \leftarrow [1..cantDias(this)])\, |cronograma(pre(this), d)| == |cronograma(this, d)|$;
asegura lasOtrasCompetenciasNoCambian : $(\forall d \leftarrow [1..cantDias(this)])(\forall c \leftarrow cronograma(pre(this), d), a \notin participantes(c))\, laCompetenciaSeMantiene(this, d, c)$;
asegura cambianLasDeLiu : $(\forall d \leftarrow [1..cantDias(this)])(\forall c \leftarrow cronograma(pre(this), d), a \in participantes(c))\, (\exists c2 \leftarrow cronograma(this, d))igualSalvoLiu(c, c2, a, p)$;
aux igualSalvoPais (at1: Atleta, at2: Atleta, p: Pais) : Bool = $nombre(at1) == nombre(at2) \land sexo(at1) == sexo(at2) \land añoNacimiento(at1) == añoNacimiento(at2) \land ciaNumber(at1) == ciaNumber(at2) \land deportes(at1) == deportes(at2) \land (\forall d \leftarrow deportes(at1))capacidad(at1, d) == capacidad(at2, d) \land nacionalidad(at2) == p$;
aux igualSalvoLiu (c1: Competencia, c2: Competencia, a: Atleta, p: Pais) : Bool = $categoria(c1) == categoria(c2) \land participantesYLiu(c1, c2, a, p) \land finalizada(c1) \Leftrightarrow finalizada(c2) \land finalizada(c1) \Rightarrow (rankingYLiu(c1, c2, a, p) \land mismosControladosYLiu(c1, c2, a, p))$;
aux participantesYLiu (c1: Competencia, c2: Competencia, a: Atleta, p: Pais) : Bool = $|participantes(c1)| == |participantes(c2)| \land (\forall at1 \leftarrow participantes(c1), at1! = a)at1 \in participantes(c2) \land (\forall at1 \leftarrow participantes(c1), at1 == a)(\exists at2 \leftarrow participantes(c2))igualSalvoPais(at1, at2, p)$;
aux rankingYLiu (c1: Competencia, c2: Competencia, a: Atleta, p: Pais) : Bool = $|ranking(c1)| == |ranking(c2)| \land (\forall i \leftarrow [0..|ranking(c1)|), ranking(c1)_i! = a)ranking(c2)_i == ranking(c1)_i \land (\forall i \leftarrow [0..|ranking(c1)|), ranking(c1)_i == a)igualSalvoPais(ranking(c1)_i, ranking(c2)_i, p)$;
aux mismosControladosYLiu (c1: Competencia, c2: Competencia, a: Atleta, p: Pais) : Bool = $|lesTocoControlAntiDoping(c1)| == |lesTocoControlAntiDoping(c2)| \land (\forall at1 \leftarrow lesTocoControlAntiDoping(c1), at1! = a)at1 \in lesTocoControlAntiDoping(c2) \land leDioPositivo(c1, at1) == leDioPositivo(c2, at1) \land (\forall at1 \leftarrow lesTocoControlAntiDoping(c1), at1 == a)(\exists at2 \leftarrow lesTocoControlAntiDoping(c2))\, igualSalvoPais(at1, at2, p) \land leDioPositivo(c1, at1) == leDioPositivo(c2, at2)$;
}

problema stevenBradbury (this: JJOO) = result : Atleta {

```
  requiere alguienGanoMedalla : (∃d ← [1..jornadaActual(this)])(∃c ← cronograma(this, d), finalizada(c)) |ranking(c)| >
      0 ;
  asegura ganoMedallaDeOro : result ∈ primeros(ganadoresPorCategoria(this)) ;
  asegura elMenosCapaz : (∀a ← primeros(ganadoresPorCategoria(this)))
      peorDesempeño(result, this) ≤ peorDesempeño(a, this) ;

  aux ganadoresPorCategoria (j: JJOO) : [(Atleta, (Deporte, Sexo))] = [ (ranking(c)_0, categoria(c)) |
      d ← [1..jornadaActual(j)], c ← cronograma(j, d), finalizada(c) ∧ |ranking(c)| > 0 ] ;
  aux peorDesempeño (a: Atleta, j: JJOO) : ℤ =
      minimo([ capacidad(a, prm(sgd(g))) | g ← ganadoresPorCategoria(j), prm(g) == a ]) ;
}

problema uyOrdenadoAsíHayUnPatrón (this: JJOO) = result : Bool {
  asegura siguenSiempreElMismoOrden(losMejoresPaises(this)) == result ;

  aux losMejoresPaises (j: JJOO) : [Pais] = [ mejorEseDia(j, i) | i ← [1..jornadaActual(j)], alguienGanoOro(j, i)) ] ;
  aux mejorEseDia (j: JJOO, d: ℤ) : Pais = [ p | p ← paises(j),
      ¬(∃p2 ← paises(j))(cantOro(j, p2, d) > cantOro(j, p, d) ∨ (cantOro(j, p2, d) == cantOro(j, p, d) ∧ p2 < p)) ]_0 ;
  aux cantOro (j: JJOO, p: Pais, d: ℤ) : ℤ = |[ 1 | c ← cronograma(j, d), finalizada(c)
      ∧ ranking(c) ≥ 1 ∧ nacionalidad(ranking(c)_0) == p ]| ;
  aux alguienGanoOro (j: JJOO, d: ℤ) : Bool = (∃c ← cronograma(j, d))finalizada(c) ∧ ranking(c) ≥ 1 ;
  aux siguenSiempreElMismoOrden (ps:[Pais]) : Bool = (∀i, j ← [0.. |ps| − 1), i < j ∧ ps_i == ps_j)ps_{i+1} == ps_{j+1} ;
}

problema sequíaOlímpica (this: JJOO) = result : [País] {
  asegura mismos(result, secosOlimpicos(this)) ;

  aux secosOlimpicos (j: JJOO) : [País] = [ p | p ← paises(j), masDiasSinMedallas(j, p) == maxDiasSinMedallas(j) ] ;
  aux masDiasSinMedallas (j: JJOO, p: País) : ℤ = maxDif(0 : [ i | i ← [1..jornadaActual(j)),
      GanoMedallaEseDia(j, p, i) ] + +[jornadaActual(j)]) ;
  aux maxDif (ls:[ℤ]) : ℤ = max([ ls_i − ls_{i−1} | i ← [1.. |ls|) ]) ;
  aux GanoMedallaEseDia (j: JJOO, p: Pais, i: ℤ) : Bool = (∃c ← cronograma(j, i))
      (|ranking(c)| ≥ 1 ∧ nacionalidad(ranking(c)_0) == p)
      ∨ (|ranking(c)| ≥ 2 ∧ nacionalidad(ranking(c)_1) == p)
      ∨ (|ranking(c)| ≥ 3 ∧ nacionalidad(ranking(c)_2) == p) ;
  aux maxDiasSinMedallas (j: JJOO) : ℤ = max([ masDiasSinMedallas(j, p) | p ← paises(j) ]) ;
}

problema transcurrirDia (this: JJOO) {
  requiere losJuegosNoTerminaron : jornadaActual(this) < cantDias(this) ;
  modifica this ;
  asegura seMantieneAño : año(this) == año(pre(this)) ;
  asegura seMantienenAtletas : mismos(atletas(this), atletas(pre(this))) ;
  asegura seMantienenDias : cantDias(this) == cantDias(pre(this)) ;
  asegura avanzaDia : jornadaActual(this) == jornadaActual(pre(this)) + 1 ;
  asegura mismaCantDeCompetencias : (∀d ← [1..cantDias(this)]) |cronograma(this, d)| == |cronograma(pre(this), d)| ;
  asegura cronogramaDeOtrosDiasNoCambia : (∀d ← [1..cantDias(this)], d ≠ jornadaActual(pre(this)))
      (∀c ← cronograma(pre(this), d))laCompetenciaSeMantiene(this, d, c) ;
  asegura lasFinalizadasSeMantinen : (∀c ← cronograma(pre(this), jornadaActual(pre(this))), finalizada(c))
      laCompetenciaSeMantiene(this, jornadaActual(pre(this)), c) ;
  asegura finalizanCompetencias : (∀c ← cronograma(pre(this), jornadaActual(pre(this))), ¬finalizada(c))
      finaliza(this, c, jornadaActual(pre(this))) ;

  aux finaliza (j: JJOO, c: Competencia, dia: ℤ) : Bool = (∃x ← cronograma(j, dia))categoria(x) == categoria(c) ∧
      mismos(participantes(x), participantes(c)) ∧ finalizada(x) ∧ mismos(ranking(x), participantes(x)) ∧
      ordenada(reverso(capacidades(ranking(x), deporte(x)))) ∧
      |ranking(x)| ≥ 1 ⇒ |lesTocoControlAntiDoping(x)| == 1 ;
}

problema operator== (this, j: JJOO) = result : Bool {
  asegura result == (año(this) == año(j) ∧ cantDias(this) == cantDias(j)
      ∧ jornadaActual(this) == jornadaActual(j) ∧ mismos(atletas(this), atletas(j)) ∧ mismoCronograma(this, j)) ;

  aux mismoCronograma (j1, j2: JJOO) : Bool = (∀d ← [1..cantDias(j1)])mismos(cronograma(j1, d), cronograma(j2, d)) ;
}

problema deportesNoOlimpicos (this: JJOO) = result : [Deporte] {
```

asegura $mismos(result, [\, d \mid d \leftarrow deportesQuePractican(this), noHayCompetencia(this, d)\,])$;

aux deportesQuePractican (j: JJOO) : [Deporte] $= sacarRepetidos(concat([\, deportes(a) \mid a \leftarrow atletas(j)\,]))$;
aux noHayCompetencia (j: JJOO, d: Deporte) : Bool $= \neg(\exists c \leftarrow competencias(j))deporte(c) == d$;
}

6. Auxiliares

aux ciaNumbers (as: [Atleta]) : $[\mathbb{Z}] = [\, ciaNumber(a) \mid a \leftarrow as\,]$;
aux competencias (j: JJOO) : [Competencia] $= [\, c \mid d \leftarrow [1..cantDias(j)], c \leftarrow cronograma(j, d)\,]$;
aux incluida (l_1, l_2:[T]) : Bool $= (\forall x \leftarrow l_1)cuenta(x, l_1) \leq cuenta(x, l_2)$;
aux lasPasadasFinalizaron (j: JJOO) : Bool $= (\forall d \leftarrow [1..jornadaActual(j)))(\forall c \leftarrow cronograma(j, d))finalizada(c)$;
aux lasQueNoPasaronNoFinalizaron (j: JJOO) : Bool $=$
$(\forall d \leftarrow (jornadaActual(j)..cantDias(j)])(\forall c \leftarrow cronograma(j, d))\neg finalizada(c)$;
aux ordenada (l:[T]) : Bool $= (\forall i \leftarrow [0..\,|l| - 1))l_i \leq l_{i+1}$;
aux sinRepetidos (l: [T]) : Bool $= (\forall i, j \leftarrow [0..\,|l|), i \neq j)l_i \neq l_j$;
aux capacidades (as: [Atleta], d: Deporte) : $[\mathbb{Z}] = [\, capacidad(a, d) \mid a \leftarrow as\,]$;
aux deporte (c: Competencia) : Deporte $= prm(categoria(c))$;
aux filtrarPorPais (as: [Atleta], p: Pais) : [Atleta] $= [\, a \mid a \leftarrow as, nacionalidad(a) == p\,]$;
aux laCompetenciaSeMantiene (j: JJOO, d: $\mathbb{Z}$, c: Competencia) : Bool $=$
$(\exists x \leftarrow cronograma(j, d))categoria(x) == categoria(c) \wedge mismos(participantes(x), participantes(c))$
$\wedge\, finalizada(x) \Leftrightarrow finalizada(c) \wedge finalizada(x) \Rightarrow (ranking(x) == ranking(c) \wedge mismosControlados(x, c))$;
aux medallistasOro (j: JJOO) : [Atleta] $= [\, ranking(c)_0 \mid d \leftarrow [1..jornadaActual(j)], c \leftarrow cronograma(j, d),$
$finalizada(c) \wedge |ranking(c)| \geq 1\,]$;
aux medallistasPlata (j: JJOO) : [Atleta] $= [\, ranking(c)_1 \mid d \leftarrow [1..jornadaActual(j)], c \leftarrow cronograma(j, d),$
$finalizada(c) \wedge |ranking(c)| \geq 2\,]$;
aux medallistasBronce (j: JJOO) : [Atleta] $= [\, ranking(c)_2 \mid d \leftarrow [1..jornadaActual(j)], c \leftarrow cronograma(j, d),$
$finalizada(c) \wedge |ranking(c)| \geq 3\,]$;
aux minimo (l: [$\mathbb{Z}$]) : $\mathbb{Z} = [\, x \mid x \leftarrow l, (\forall y \leftarrow l)x \leq y\,]_0$;
aux mismosControlados (c_1, c_2: Competencia) : Bool $=$
$mismos(lesTocoControlAntiDoping(c_1), lesTocoControlAntiDoping(c_2)) \wedge$
$(\forall p \leftarrow lesTocoControlAntiDoping(c_1))leDioPositivo(c_1, p) \Leftrightarrow leDioPositivo(c_2, p)$;
aux nacionalidades (as: [Atleta]) : [Pais] $= [\, nacionalidad(a) \mid a \leftarrow as\,]$;
aux paises (j: JJOO) : [País] $= sacarRepetidos(nacionalidades(atletas(j)))$;
aux primeros (l: [(T,S)]) : [T] $= [\, prm(x) \mid x \leftarrow l\,]$;
aux reverso (l: [T]) : [T] $= [\, l_{|x|-i-1} \mid i \leftarrow [0..\,|l|)\,]$;
aux sacarRepetidos (l: [T]) : [T] $= [\, l_i \mid i \leftarrow [0..\,|l|), l_i \notin l_{[0..i)}\,]$;