

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

1. Tipos

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

2. 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  $d \in \text{deportes}(a)$ ;  
  
  invariante  $|\text{deportes}(a)| > 0$ ;  
  invariante  $\text{sinRepetidos}(\text{deportes}(a))$ ;  
  invariante  $\text{ordenada}(\text{deportes}(a))$ ;  
  invariante  $\text{capacidadEnRango} : (\forall d \leftarrow \text{deportes}(a)) 0 \leq \text{capacidad}(a, d) \leq 100$ ;  
}
```

```
problema especialidad (a: Atleta) = res : Deporte {  
  asegura  $\text{res} \in \text{deportes}(a)$ ;  
  asegura  $\text{capacidad}(a, \text{res}) == \text{maximoLista}(\text{listarCapacidades}(a))$ ;  
  aux listarCapacidades (a: Atleta) : [ $\mathbb{Z}$ ] =  $[\text{capacidad}(a, x) | x \leftarrow \text{deportes}(a)]$ ;  
  aux maximoLista (ls: [ $\mathbb{Z}$ ]) :  $\mathbb{Z}$  =  $[x | x \leftarrow \text{ls}, (\forall y \leftarrow \text{ls}) x \geq y][0]$ ;  
}
```

```
problema entrenarNuevoDeporte (a: Atleta, d: Deporte, c:  $\mathbb{Z}$ ) {  
  requiere  $c \geq 0 \wedge c \leq 100$ ;  
  requiere  $d \notin \text{deportes}(a)$ ;  
  modifica  $a$ ;  
  asegura  $\text{nombre}(a) == \text{nombre}(\text{pre}(a))$ ;  
  asegura  $\text{sexo}(a) == \text{sexo}(\text{pre}(a))$ ;  
  asegura  $\text{anoNacimiento}(a) == \text{anoNacimiento}(\text{pre}(a))$ ;  
  asegura  $\text{nacionalidad}(a) == \text{nacionalidad}(\text{pre}(a))$ ;  
  asegura  $\text{ciaNumber}(a) == \text{ciaNumber}(\text{pre}(a))$ ;  
  asegura  $(\forall d' \leftarrow \text{deportes}(\text{pre}(a))) \text{capacidad}(a, d') == \text{capacidad}(\text{pre}(a), d')$ ;  
  asegura  $\text{mismos}(\text{deportes}(a), d : \text{deportes}(\text{pre}(a)))$ ;  
  asegura  $\text{ordenada}(\text{deportes}(a))$ ;  
  asegura  $\text{capacidad}(a, d) == c$ ;  
}
```

3. 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)  $\wedge$   $a \in lesTocoControlAntiDoping(c)$ ;

invariante participaUnaSolaVez :  $sinRepetidos(ciaNumbers(participantes(c)))$ ;
invariante participantesPerteneceenACat :
   $(\forall p \leftarrow participantes(c)) prm(categoria(c)) \in deportes(p) \wedge sgd(categoria(c)) == sexo(p)$ ;
invariante elRankingEsDeParticipantesYNoHayRepetidos :
   $finalizada(c) \Rightarrow incluida(ranking(c), participantes(c))$ ;
invariante seControlanParticipantesYNoHayRepetidos :
   $finalizada(c) \Rightarrow incluida(lesTocoControlAntiDoping(c), participantes(c))$ ;
}

problema finalizarCompetencia (c: Competencia, posiciones: [Atleta], control: [(Atleta, Bool)]) {
  requiere  $\neg finalizada(c)$ ;
  requiere contenida(posiciones, participantes(c));
  requiere  $sinRepetidos(posiciones)$ ;
  requiere  $(\forall x \leftarrow control) prm(x) \in participantes(c)$ ;
  requiere  $sinRepetidos([prm(x)|x \leftarrow control])$ ;
  modifica c;
  asegura finalizada(c);
  asegura  $ranking(c) == posiciones$ ;
  asegura  $participantes(c) == participantes(pre(c))$ ;
  asegura  $mismos(lesTocoControlAntiDoping(c), [prm(x)|x \leftarrow control])$ ;
  asegura  $(\forall x \leftarrow control) leDioPositivo(prm(x)) == sgn(x)$ ;
  asegura  $categoria(c) == categoria(pre(c))$ ;
}

problema linfordChristie (c: Competencia, a: Atleta) {
  requiere  $\neg finalizada(c)$ ;
  requiere  $a \in participantes(c)$ ;
  modifica c;
  asegura  $mismos(participantes(pre(c)), a : participantes(c))$ ;
  asegura  $categoria(c) == categoria(pre(c))$ ;
  asegura  $\neg finalizada(c)$ ;
}

problema gananLosMasCapaces (c: Competencia) = res : Bool {
  requiere finalizada(c);
  asegura  $res == (\forall x \leftarrow [0..|ranking(c)| - 2])$ 
     $capacidad(ranking(c)[x], prm(categoria(c))) \geq capacidad(ranking(c)[x + 1], prm(categoria(c)))$ ;
}

problema sancionarTramposos (c: Competencia) {
  requiere finalizada(c);
  modifica c;
  asegura finalizada(c);
  asegura  $categoria(c) == categoria(pre(c))$ ;
  asegura  $mismos(participantes(c), participantes(pre(c)))$ ;
  asegura  $incluida(ranking(c), participantes(c))$ ;
  asegura  $incluida(lesTocoControlAntiDoping(c), participantes(c))$ ;
  asegura  $mismos(ranking(pre(c)), ranking(c) + listaDoping(c, lesTocoControlAntiDoping(c)))$ ;
  asegura  $ranking == [el|el \leftarrow ranking(pre(c)), el \notin listaDoping(c, lesTocoControlAntiDoping(c))]$ ;
  aux listaDoping (c:competencia, a:[atleta]) : [atleta] =  $[x|x \leftarrow a, leDioPositivo(c, x)]$ ;
}

```

4. JJOO

```

tipo JJOO {
  observador año (j: JJOO) :  $\mathbb{Z}$ ;

```

```

observador atletas (j: JJO) : [Atleta];
observador cantDias (j: JJO) :  $\mathbb{Z}$ ;
observador cronograma (j: JJO, dia:  $\mathbb{Z}$ ) : [Competencia];
    requiere  $1 \leq dia \leq cantDias(j)$ ;
observador jornadaActual (j: JJO) :  $\mathbb{Z}$ ;

invariante atletasUnicos : sinRepetidos(ciaNumbers(atletas(j)));
invariante unaDeCadaCategoria :  $(\forall i, k \leftarrow [0..|competencias(j)|], i \neq k)$ 
     $categoria(competencias(j)_i) \neq categoria(competencias(j)_k)$ ;
invariante competidoresInscriptos :  $(\forall c \leftarrow competencias(j)) incluye(participantes(c), atletas(j))$ ;
invariante jornadaValida :  $1 \leq jornadaActual(j) \leq cantDias(j)$ ;
invariante finalizadasSiYaPasoElDia :  $lasPasadasFinalizaron(j) \wedge lasQueNoPasaronNoFinalizaron(j)$ ;
}

problema dePaseo (j: JJO) = res : [Atleta] {
    asegura mismos(res, noParticiparon(j));
    aux noParticiparon (j: JJO) : [Atleta] =  $[x | x \leftarrow atletas(j), \neg en(x, participaronConRepes(j))]$ ;
    aux participaronConRepes (j: JJO) :
        [Atleta] =  $[atl | dia \leftarrow [1..cantDias(j)], comp \leftarrow cronograma(j, dia), atl \leftarrow atletas(j), en(atl, participantes(comp))]$ ;
}

problema medallero (j: JJO) = res : [(Pais,  $\mathbb{Z}$ )] {
    asegura mismos(result, sacarSinMedallas(crearMedallero(j)));
    asegura medalleroOrdenado(result);
    aux listaNacionalidadesAtletas (j: JJO) : [Pais] =  $[nacionalidad(at) | at \leftarrow atletas(j)]$ ;
    aux sacarRepeticiones (ls : [T]) : [T] =  $[ls[x] | x \leftarrow [0..|ls| - 1], \neg en(ls[x], ls[0..x - 1])]$ ;
    aux listaPaises (j: JJO) : [Pais] = sacarRepeticiones(listaNacionalidadesAtletas(j));
    aux cuenta (x: T, a: [T]) :  $\mathbb{Z}$  =  $[y | y \leftarrow a, y == x]$ ;
    aux nacionalidadPrimerosTres (c: Competencia) : [Pais] =
         $[ifthenelse(i \leq |ranking(c)| - 1, nacionalidad(ranking(c)[i]), "Narnia") | i \leftarrow [0..2]]$ ;
    aux ganadoresMTotales (M:  $\mathbb{Z}$ , j: JJO) : [Pais] =
         $[nacionalidadPrimerosTres(c)[M] | dia \leftarrow [1..jornadaActual(j)], c \leftarrow cronograma(j, dia), finalizada(c)]$ ;
    aux crearMedallero (j: JJO) : [(Pais,  $\mathbb{Z}$ )] =
         $[(p, [cuenta(p, ganadoresMTotales(0, j)), cuenta(p, ganadoresMTotales(1, j)),$ 
         $cuenta(p, ganadoresMTotales(2, j))]) | p \leftarrow listaPaises(j)]$ ;
    aux sacarSinMedallas (lista : [(Pais,  $\mathbb{Z}$ )] : [(Pais,  $\mathbb{Z}$ )] =  $[x | x \leftarrow lista, sgn(x) \neq [0, 0, 0]]$ ;
    aux medalleroOrdenado (lista : [(Pais,  $\mathbb{Z}$ )] : Bool =
         $(\forall x \leftarrow [0..|lista| - 2])((sgd(lista[x])[0] \geq sgd(lista[x + 1])[0])$ 
         $\vee ((sgd(lista[x])[0] == sgd(lista[x + 1])[0]) \wedge (sgd(lista[x])[1] \geq sgd(lista[x + 1])[1]))$ 
         $\vee ((sgd(lista[x])[0] == sgd(lista[x + 1])[0]) \wedge (sgd(lista[x])[1] == sgd(lista[x + 1])[1]))$ 
         $\wedge (sgd(lista[x])[2] \geq sgd(lista[x + 1])[2]))$ ;
}

problema boicotPorDisciplina (j: JJO, cat: (Deporte, Sexo), p: Pais) = res :  $\mathbb{Z}$  {
    requiere existe( $c \leftarrow todasLasCompetencias(j)$ )  $categoria(c) == cat$ ;
    requiere finalizada(competenciaFiltrada(j, cat)) == false;
    // asumimos que se saca a los participantes antes de finalizar la competencia
    modifica j;
    asegura finalizada(competenciaFiltrada(j, cat)) == finalizada(competenciaFiltrada(pre(j), cat));
    asegura ano(j) == ano(pre(j));
    asegura mismos(atletas(j), atletas(pre(j)));
    asegura cantDias(j) == cantDias(pre(j));
    asegura jornadaActual(j) == jornadaActual(pre(j));
    asegura  $(\forall dia \leftarrow [1..cantDias(j)])$ 
         $[cronograma(j, dia)] == [cronograma(pre(j), dia)]$ ;
    asegura  $(\forall dia \leftarrow [1..cantDias(j)])(\forall (x \leftarrow [1..(long(cronograma(j, dia)) - 1)]))$ 
         $((cronograma(j, dia)[x] == cronograma(pre(j), dia)[x])$ 
         $\vee (categoria(cronograma(j, dia)[x]) == (categoria(cronograma(pre(j), dia)[x]))$ 
         $\wedge (categoria(cronograma(j, dia)[x]) == cat))$ ;
    asegura  $(\forall comp \leftarrow todasLasCompetencias(j), categoria(comp) == cat)$ 
         $(\forall (atl \leftarrow participantes(comp)) nacionalidad(atl) \neq p$ ;
    asegura mismos(participantes(competenciaFiltrada(j, cat)),
         $[x | x \leftarrow participantes(competenciaFiltrada(pre(j), cat)), nacionalidad(x) \neq p]$ );
}

```

```

    asegura res == [|x|comp ← todasLasCompetencias(pre(j)),
    x ← participantes(comp), (categoria(comp) == cat ∧ nacionalidad(x) == p)]];
    aux todasLasCompetencias (j:JJO) : [competencia] = [x|y ← [1..cantDIas(j)], x ← cronogramas(j,y)];
    aux competenciaFiltrada (j:JJO, c:(deporte,sexo) : competencia =
        cab([x|y ← todasLasCompetencias(j), c == categoria(x)]);
}

problema losMasFracasados (j: JJO, p: Pais) = res : [Atleta] {
    asegura noGanaronMedallas : (∀x ← [noGanoMedallas(atl, comp)
    |atl ← result, comp ← todasLasCompetencias(j), finalizada(comp)]);
    asegura todosDelPais : (∀x ← result)nacionalidad(x) == p;
    asegura (∀x ← result, y ← atletas(j), nacionalidad(y) == p)
        competenciasEnLasQueParticipo(j, x) >= competenciasEnLasQueParticipo(j, y);
    aux competenciasEnLasQueParticipo (j: JJO, a:Atleta) : ℤ =
        long([z|z ← todasLasCompetencias(j), en(a, participantes(z))]);
    aux noGanoMedallas (atl: Atleta, comp: Competencia) : Bool =
        ciaNumber(ranking(comp)[0]) ≠ ciaNumber(atl) ∧ ciaNumber(ranking(comp)[1]) ≠ ciaNumber(atl)
        ∧ ciaNumber(ranking(comp)[2]) ≠ ciaNumber(atl);
}

problema liuSong (j: JJO, a: Atleta, p: País) {
    requiere a ∈ atletas(j);
    modifica j;
    asegura ano(j) == ano(pre(j));
    asegura (∀i ← [0..|atletas(pre(j))| - 1], atletas(pre(j))i! = a)atletas(pre(j))i == atletas(j)i;
    asegura (∀x ← atletas(j), ciaNumber(x) == ciaNumber(a))
        (nombre(x) == nombre(a) ∧ sexo(x) == sexo(a) ∧ añoNacimiento(x) == añoNacimiento(a)
        ∧ nacionalidad(x) == p ∧ deportes(x) == deportes(a) ∧ capacidad(x) == capacidad(a));
    asegura cantDias(j) == cantDias(pre(j));
    asegura jornadaActual(j) == jornadaActual(pre(j));
    asegura (∀d ← [1..cantDias(j)]|cronograma(d, j)| == |cronograma(d, pre(j))|
        ∧ (∀i ← [0..|cronograma(d, j)| - 1])
        categoria(cronograma(d, j)[i]) == categoria(cronograma(d, pre(j))[i])
        ∧ finalizada(cronograma(d, j)[i]) == finalizada(cronograma(d, pre(j))[i]);
    asegura todosIgualesMenosLiuSong(partComp(j), partComp(pre(j)));
    asegura todoIgualMenosNacionalidad(partComp(j));
    asegura todosIgualesMenosLiuSong(rankComp(j), rankComp(pre(j)));
    asegura todoIgualMenosNacionalidad(rankComp(j));
    asegura todosIgualesMenosLiuSong(doppComp(j), doppComp(pre(j)));
    asegura todoIgualMenosNacionalidad(doppComp(j));
    asegura ∀(dia ← [1..jornadaActual(j)], comp ← [0..|cronograma(j, dia) - 1|], finalizada(cronograma(j, dia)))
        leDioPositivo(cronograma(j, dia)[comp], atl) == leDioPositivo(cronograma(pre(j), dia)[comp], atl);
    aux rankComp (j: JJO) : [Atleta] =
        [atl|dia ← [1..cantDias(j)], comp ← cronograma(j, dia), atl ← ranking(comp), finalizada(comp)];
    aux doppComp (j: JJO) : [Atleta] =
        [atl|dia ← [1..cantDias(j)], comp ← cronograma(j, dia),
        atl ← lesTocoControlAntiDopping(comp), finalizada(comp)];
    aux partComp (j: JJO) : [Atleta] =
        [atl|dia ← [1..cantDias(j)], comp ← cronograma(j, dia), atl ← participantes(comp)];
    aux todosIgualesMenosLiuSong (ls : [Atleta], lsOld : [Atleta]) : Bool =
        ∀(x ← [0..|ls| - 1])(ls[x] == lsOld[x] ∨ ciaNumber(ls) == ciaNumber(a));
    aux todoIgualMenosNacionalidad (ls: [Atleta]) : Bool =
        ∀(x ← ls, ciaNumber(x) == ciaNumber(a))(nombre(x) == nombre(a) ∧ sexo(x) == sexo(a)
        ∧ anoNacimiento(x) == anoNacimiento(a) ∧ nacionalidad(x) == p ∧ deportes(x) == deportes(a)
        ∧ capacidad(x) == capacidad(a));
}

problema stevenBradbury (j: JJO) = res : Atleta {
    requiere algunOro :
        long([comp|comp ← todasLasCompetencias(j), finalizada(comp) ∧ |ranking(comp)| > 0]) > 0;
    asegura ganoOro :
        existe(f ← [ranking(comp)[0]|comp ← todasLasCompetencias(j),
        finalizada(comp) ∧ |ranking(comp)| > 0])ciaNumber(result) == ciaNumber(f);
}

```

```

asegura esElPeor :
  existe(dep ← deportesEnLosQueGano(j, result))
  paraTodo(f ← [ranking(comp)[0]|comp ← todasLasCompetencias(j),
    finalizada(comp) ∧ |ranking(comp)| > 0]) capacidad(result, dep) ≤ capacidad(f, prm(categoria(comp)));
aux deportesEnLosQueGanoOro (j: JJOO, a: Atleta) : [Deportes] =
  [prm(categoria(comp))|comp ← todasLasCompetencias(j), finalizada(comp)
    ∧ |ranking(comp)| > 0 ∧ ciaNumber(ranking(comp)[0]) == ciaNumber(a)];
}

problema uyOrdenadoAsíHayUnPatrón (j: JJOO) = res : Bool {
  asegura res == ifthenelse|mejoresPaises(j)| == |sinRepes(mejoresPaises(j))|, true,
    ifthenelse(contar(cab(mejoresPaises(j))) == 1 ∧ |mejoresPaises(j)| > |sinRepes(mejoresPaises(j))|, false,
      (∀i ← [0..|mejoresPaises(j)| - 1], k ← [0..|mejoresPaises(j)| - 1], i mod
        |patron(mejoresPaises(j))| == k mod |patron(mejoresPaises(j))|) s[i] == s[k];
  aux mejoresPaises (j: JJOO) : [Pais] = [cab(ordenar(masOrosDelDia(j, d))
    |d ← [1..jornadaActual(j) - 1], cronograma(j, d) ≠ [ ], masOrosDelDia(j, d) ≠ [ ]];
  aux patron ((ls: [T])) : [T] =
    ifthenelse(|ls| == 1, ls, sub(ls, 0, min([i | i ← [1..|ls| - 1], ls[i] == ls[0]]));
  aux masOrosDelDia (j: JJOO, d: dia) : [Pais] =
    filtrarTuplaPorPrmMax(j, uniqSgdLista(tuplasMedallOroComaPais(j, d), d);
  aux tuplasMedallaOroComaPais (j: JJOO, d: Dia) : [(Z, Pais)] =
    [(contar(pais, PaisesConOroTalDia(j, d), p)|p ← paisesConOroTalDia(j, d)];
  aux paisesConOroTalDia (j: JJOO, d: Dia) : [Pais] =
    [pais(cab(ranking(c))|c ← cronograma(j, d), finalizada(c) ∧ |ranking(c)| > 0)];
  aux uniqSgdLista (ls: [(Z, Z)]) : [(Z, Z)] =
    [ls(i)|i ← [0..|ls| - 1], j ← [0..|ls| - 1], i ≠ j, sgd(ls[i]) ≠ sgd(ls[j])];
  aux filtrarTuplaPorPrmMax (j: JJOO, ls: [(Z, Pais)], d: Dia) : [Pais] =
    [sgd(tupla)|tupla ← tuplasMedallaOroComaPais(j, d), prm(tupla) ≥
      maxListaTuplas(tuplasMedallaOroComaPais(j, d))];
  aux maxListaTuplas (ls: [(ent, Pais)]) : Z = max([prm(tupla)|tupla ← ls]);
}

problema sequiaOlimpica (j: JJOO) = res : [País] {
  asegura (∀p ← result) sequiaPais(j, p) == maximaSequia(j);
  asegura (∀p ← listaPaisesConSequia(j), sequiaPais(j, p) == maximaSequia(j)) p ∈ result;
  aux sequiaPais (j: JJOO, p: pais) : Z =
    cerosSeguidos([ganoEnElDia(j, dia, p)|dia ← [1..cantDias(j)]]);
  aux ganoEnElDia (j: JJOO, dia: Z, pais: Pais) : Bool =
    long([1 | c ← cronograma(j, dia), en(pais, paisesQueGanaronCompetencia(c))]) > 0;
  aux listaPaisesConSequia (j: JJOO) : [(Pais, Int)] =
    [(p, sequiaPais(j, p)) | p ← listaPaises(j)];
  aux maximaSequia (j: JJOO) : Z =
    maximoLista([sequiaPais(j, p)|p ← listaPaises(j)]);
  aux paisesQueGanaronCompetencia (c: Competencia) : [paises] =
    [pais(ranking(c)[a]) | a ← [0..|ranking(c)| - 1], a ≤ 2];
  aux generoTodosLosParesCeroComaUno (ls: [Bool]) : [(Z, Z)] =
    [(c, c1) | c ← [0..|ls| - 1], c1 ← [c..|ls| - 1], ls[c] == False ∧ ls[c1] == True];
  aux uniqPrmLista (ls: [(Z, Z)]) : [(Z, Z)] =
    [ls(i)|i ← [0..|ls| - 1], j ← [0..|ls| - 1], i ≠ j, prm(ls[i]) ≠ prm(ls[j])];
  aux uniqSgdLista (ls: [(Z, Z)]) : [(Z, Z)] =
    [ls(i)|i ← [0..|ls| - 1], j ← [0..|ls| - 1], i ≠ j, sgd(ls[i]) ≠ sgd(ls[j])];
  aux diferenciaPrmSgd (ls: [(Z, Z)]) : [Z] = [sgd(el) - prm(el) | el ← ls];
  aux cerosSeguidos (ls: [Bool]) : Z =
    max(diferenciaPrmSgd(uniqSgdLista(uniqPrmLista(generoTodosLosParesCeroComaUno(ls))))) ;
}

problema transcurrirDia (j: JJOO) {
  modifica j;
  asegura cantDias(j) == cantDias(pre(j));
  asegura jornadaActual(j) == jornadaActual(pre(j)) + 1;
  asegura año(j) == año(pre(j));
  asegura mismos(atletas(j), atletas(pre(j)));
  asegura (∀dia ← [1..jornadaActual(j) - 1])
    mismos(cronograma(j, dia), cronograma(pre(j), dia));
}

```

```

asegura |cronograma(j, jornadaActual(j))| == |cronograma(pre(j), jornadaActual(j))|;
asegura (∀i ← [0, |cronograma(j, jornadaActual(j))| - 1])
  categoria(cronograma(j, jornadaActual(j))[i]) == categoria(cronograma(pre(j), jornadaActual(j))[i]);
asegura (∀i ← [0, |cronograma(j, jornadaActual(j))| - 1])
  mismos(participantes(cronograma(j, jornadaActual(j))[i]),
    participantes(cronograma(pre(j), jornadaActual(j))[i]));
asegura (∀c ← cronograma(j, jornadaActual(j)))
  finalizada(c) == True ∧ incluida(ranking(c), participantes(c))
  ∧ (∀x ← [0..|ranking(c)| - 2]) capacidadEnOrden(x, c);
asegura asegura (∀c ← cronograma(j, jornadaActual(j)))
  (|lesTocoControlAntiDoping(c)| == 1 ∧ lesTocoControlAntiDoping(c)[0] ∈ atletas(c)
  ∨ (|lesTocoControlAntiDoping(c)| == 0 ∧ |atletas(c)| == 0));
asegura (∀c ← cronograma(j, jornadaActual(j)))
  |listaDoping(c, atletas(c))| * 20 ≤
  Σ(|long(lesTocoControlAntiDoping(comp))| comp ← cronograma(j, jornadaActual(j)));
aux capacidadEnOrden (x: ℤ, c: Competencia) : Bool =
  capacidad(ranking(c)[x], prm(categoria(c))) ≥ capacidad(ranking(c)[x + 1], prm(categoria(c)));
aux listaDoping (c: Competencia, a: [Atleta]) : [Atleta] = [x | x ← a, leDioPositivo(c, x)];
}

```

5. Auxiliares

```

aux ciaNumbers (as: [Atleta]) : [ℤ] = [ciaNumber(a) | a ← as];
aux competencias (j: JJOO) : [Competencia] = [c | d ← [1..cantDias(j)], c ← cronograma(j, d)];
aux incluida (l1, l2: [T]) : Bool = (∀x ← l1) cuenta(x, l1) ≤ cuenta(x, l2);
aux lasPasadasFinalizaron (j: JJOO) : Bool = (∀d ← [1..jornadaActual(j)]) (∀c ← cronograma(j, d)) finalizada(c);
aux lasQueNoPasaronNoFinalizaron (j: JJOO) : Bool =
  (∀d ← (jornadaActual(j)..cantDias(j))) (∀c ← cronograma(j, d)) ¬finalizada(c);
aux ordenada (l: [T]) : Bool = (∀i ← [0..|l| - 1]) li ≤ li+1;
aux sinRepetidos (l: [T]) : Bool = (∀i, j ← [0..|l|], i ≠ j) li ≠ lj;

```