



TP de Especificación

Esperando el Bondi

30 de Marzo de 2022

Algoritmos y Estructuras de Datos I

Grupo 1

Integrante	LU	Correo electrónico
Polonuer, Joaquin	1612/21	jtpolonuer@gmail.com
González, Facundo	1440/21	facundo2gonzalez2@gmail.com
Jaime, Brian David	411/18	brian.d.jaime97@gmail.com
Guberman, Diego	469/17	diego98g@hotmail.com



Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2610 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (++54 +11) 4576-3300

<http://www.exactas.uba.ar>

1. Definición de Tipos

```
type Tiempo =  $\mathbb{R}$ 
type Dist =  $\mathbb{R}$ 
type GPS =  $\mathbb{R} \times \mathbb{R}$ 
type Recorrido =  $seq\langle GPS \rangle$ 
type Viaje =  $seq\langle Tiempo \times GPS \rangle$ 
type Nombre =  $\mathbb{Z} \times \mathbb{Z}$ 
type Grilla =  $seq\langle GPS \times GPS \times Nombre \rangle$ 
type Celda =  $GPS \times GPS \times Nombre$ 
```

2. Constantes

3. Problemas

3.1. Ejercicio 1

Devolver verdadero si los puntos GPS del viaje y los tiempos están en rango.

```
proc viajeValido (in v: Viaje, out res: Bool) {
  Pre {True}
  Post {res = true  $\leftrightarrow$  esViajeValido(v)}
  pred esViajeValido (v: Viaje) {
    ( $\forall i : \mathbb{Z}$ ) (
       $0 \leq i < |v| \longrightarrow_L (esTiempoValido(v[i]_0) \wedge sonCoordenadasValidas(v[i]_1))$ 
    )
    /* no hay dos tiempos iguales en los registros */
     $\wedge \neg(\exists i, j : \mathbb{Z}) ($ 
       $0 \leq i < j < |v| \wedge_L v[i]_0 = v[j]_0$ 
    )
  }
}

pred esTiempoValido (t: Tiempo) {
   $t \geq 0$ 
}

pred sonCoordenadasValidas (c: GPS) {
   $-90,0 \leq c_0 \leq 90,0 \wedge -180,0 \leq c_1 \leq 180,0$ 
}
}
```

3.2. Ejercicio 2

Devolver verdadero si los puntos GPS del recorrido están en rango.

```
proc recorridoValido (in v: Recorrido, out res: Bool) {  
  Pre {True}  
  Post {res = true  $\leftrightarrow$  esRecorridoValido(v)}  
  pred esRecorridoValido (v: Recorrido) {  
    ( $\forall i : \mathbb{Z}$ )( $0 \leq i < |v| \longrightarrow_L$  sonCoordenadasValidas(v[i]))  
  }  
}
```

3.3. Ejercicio 3

Chequear que todos los puntos registrados en un viaje válido se encuentren dentro de un círculo de radio r kilómetros.

```
proc enTerritorio (in v: Viaje, in r: Dist, out res: Bool) {  
  Pre {esViajeValido(v)}  
  Post {res = true  $\leftrightarrow$  estaEnTerritorio(v, r)}  
  pred estaEnTerritorio (v: Viaje, r: Dist) {  
    ( $\exists c : GPS$ )(sonCoordenadasValidas(c)  $\wedge_L$  ( $\forall i : \mathbb{Z}$ )( $0 \leq i < |v| \longrightarrow_L dist(c, v[i]_1) \leq 1000 \cdot r$ ))  
    /* Multiplico r por 1000 dado que r está dado en kilómetros y la función auxiliar dist(p1, p2)  
    devuelve su resultado en metros */  
  }  
}
```

3.4. Ejercicio 4

Dado un viaje válido, determinar el tiempo total que tardó el colectivo. Este valor debe ser calculado como el tiempo transcurrido desde el primer punto registrado y hasta el último.

```
proc tiempoTotal (in v: Viaje, out t: Tiempo) {  
  Pre {esViajeValido(v)}  
  Post {esMaximaDiferenciaTiempo(v, t)}  
  pred esMaximaDiferenciaTiempo (v: Viaje, t: Tiempo) {  
    /* t es la diferencia entre dos tiempos del v */  
    ( $\exists i, j : \mathbb{Z})(0 \leq i, j < |v| \wedge_L v[i]_0 - v[j]_0 = t) \wedge$   
    /* t es la mayor diferencia posible entre dos tiempos del viaje */  
     $\neg(\exists n, m : \mathbb{Z})(0 \leq n, m < |v| \wedge_L v[n]_0 - v[m]_0 > t)$   
  }  
}
```

3.5. Ejercicio 5

Dado un viaje válido, determinar la distancia recorrida en kilómetros aproximada utilizando toda la información registrada en el viaje, es decir, utilizando la información registrada de todos los tramos.

```

proc distanciaTotal (in v: Viaje, out d: Dist) {
  Pre {esViajeValido(v)}
  Post {distanciaViajeOrdenado(v,d)}
  pred distanciaViajeOrdenado (v: Viaje, d: Dist) {
    ( $\exists v' : \text{Viaje}$ )( $\text{esElViajeOrdenado}(v, v') \wedge d = \text{sumaDistanciasSucesivas}(v')$ )
  }
  pred esElViajeOrdenado (v,v': Viaje) {
     $\text{estaOrdenadoTemporalmente}(v') \wedge \text{esPermutacion}(v, v')$ 
  }
  pred estaOrdenadoTemporalmente (v: Viaje) {
    ( $\forall i : \mathbb{Z}$ )( $0 \leq i < |v|-1 \longrightarrow_L v[i]_0 < v[i+1]_0$ )
  }
  pred esPermutacion (v1,v2: Viaje) {
    /*Esto funciona porque no hay repetidos en los viajes*/
    ( $\forall e : \text{Tiempo} \times \text{GPS}$ )( $\#apariciones(v1, e) = \#apariciones(v2, e)$ )
  }

  aux #apariciones (v: Viaje, e: Tiempo  $\times$  GPS) :  $\mathbb{Z} = \sum_{i=0}^{|v|-1} \text{if } v[i] = e \text{ then } 1 \text{ else } 0 \text{ fi};$ 

  aux sumaDistanciasSucesivas (v: Viaje) : Dist =  $\frac{1}{1000} \cdot \sum_{i=0}^{|v|-2} \text{dist}(v[i]_1, v[i+1]_1)$ 
  /* Divido la sumatoria por 1000 dado que se pide el resultado en kilómetros y la función auxiliar
  dist(p1,p2) devuelve su resultado en metros */;
}

```

3.6. Ejercicio 6

Dado un viaje válido devolver verdadero si el colectivo superó los 80 km/h en algún momento del viaje.

```
proc excesoDeVelocidad (in v: Viaje, out res: Bool) {  
  Pre {esViajeValido(v)}  
  Post {res = true  $\leftrightarrow$  superaVelocidad(v)}  
  pred superaVelocidad (v: Viaje) {  
    ( $\exists i, j : \mathbb{Z}$ )( $0 \leq i, j < |v| \wedge_L i \neq j \wedge$  esTramo(v, v[i], v[j])  $\wedge$  velocidadTramo(v[i], v[j]) > 80)  
  }  
  pred esTramo (v: Viaje, e1, e2: Tiempo  $\times$  GPS) {  
     $e1_0 < e2_0 \wedge \neg(\exists e : \textit{Tiempo} \times \textit{GPS})(e \in v \wedge e1_0 < e_0 < e2_0)$   
  }  
  aux velocidadTramo (e1, e2 : Tiempo  $\times$  GPS) :  $\mathbb{R} = \frac{\textit{dist}(e1_1, e2_1)}{e2_0 - e1_0} \cdot 3,6$   
  /* Multiplico por 3,6 dado que se pide el resultado en kilómetros por hora y la función auxiliar  
  dist(p1, p2) devuelve su resultado en metros mientras que los tiempos están en segundos */;  
}
```

3.7. Ejercicio 7

Dada una lista de viajes válidos, calcular la cantidad de viajes que se encontraban en ruta en cualquier momento entre t_0 y t_f inclusivos. Por ejemplo, si un viaje comenzó a las 13:30 y terminó a las 14:30 y la franja es de 14:00 a 15:00, el viaje debería estar considerado. Lo mismo ocurre si el viaje comenzó a las 14:10 y terminó a las 14:15 o si comenzó a las 13:30 y terminó a las 16:00.

```

proc flota (in vs: seq⟨Viaje⟩, in t0: Tiempo, in tf: Tiempo, out res: ℤ) {
  Pre {sonTodosViajesValidos(vs) ∧ t0 ≤ tf ∧ esTiempoValido(t0) ∧ esTiempoValido(tf) }
  Post {esCantidadEnRuta(vs, t0, tf, res)}
  pred sonTodosViajesValidos (vs: seq⟨Viaje⟩) {
    (∀v : Viaje)(v ∈ vs → esViajeValido(v))
  }
  pred esCantidadEnRuta (vs: seq⟨Viaje⟩, t0, tf: Tiempo, res: ℤ) {
    res =  $\sum_{i=0}^{|vs|-1}$  (if estaEnRuta(v[i], t0, tf) then 1 else 0 fi)
  }
  pred esCantidadEnRuta (vs: seq⟨Viaje⟩, t0, tf: Tiempo, res: ℤ) {
    (∃vs' : seq⟨Viaje⟩) (
      (∀v : Viaje) (
        (v ∈ vs ∧ estaEnRuta(v, t0, tf)) →L #aparicionesViajes(v, vs') = #aparicionesViajes(v, vs)
      )
      ∧ |vs'| = res
    )
  }
  pred estaEnRuta (v: Viaje, t0, tf: Tiempo) {
    (∃i, j : ℤ) (
      0 ≤ i, j < |v| ∧L v[i]0 ≤ t0 < tf ≤ v[j]0
    )
    ∨ (∃i : ℤ) (
      0 ≤ i < |v| ∧L t0 ≤ v[i]0 ≤ tf
    )
  }
  pred estaEnRuta (v: Viaje, t0, tf: Tiempo) {
    (∃i, j : ℤ) (
      0 ≤ i ≤ j < |v| ∧L (v[i]0 ≤ tf ∧ v[j]0 ≥ t0)
    )
  }
  aux #aparicionesViajes (v: Viaje, vs: seq⟨Viaje⟩) : ℤ =  $\sum_{i=0}^{|vs|-1}$  (if vs[i] = v then 1 else 0 fi);
}

```


3.8. Ejercicio 8

Dado un viaje v válido, un recorrido r válido y un umbral u (en kilómetros), devolver todos los puntos del recorrido que no fueron cubiertos por ningún punto del viaje. Se considera que un punto p del recorrido está cubierto si al menos un punto del viaje está a menos de u kilómetros del punto p .

```
proc recorridoCubierto (in v: Viaje, in r: Recorrido, in u: Dist, out res: seq<GPS>) {  
  Pre {esViajeValido(v)  $\wedge$  u > 0  $\wedge$  esRecorridoValido(r)}  
  Post {sonTodosLosPuntosNoCubiertos(res, v, r, u)}  
  pred sonTodosLosPuntosNoCubiertos (res: seq<GPS>, v: Viaje, r: Recorrido, u: Dist) {  
    /*Todos los puntos que están en res son puntos no cubiertos del recorrido*/  
    /*Todos los puntos no cubiertos del recorrido están en res*/  
    ( $\forall p : GPS$ ) (  
       $p \in res \leftrightarrow (p \in r \wedge \neg estaCubierto(p, v, u))$   
    )  
  }  
  pred estaCubierto (p: GPS, v: Viaje, u: Dist) {  
    ( $\exists m : Tiempo \times GPS$ ) (  
       $m \in v \wedge dist(m_1, p) < u \cdot 1000$   
    )  
    /*Multiplicamos por 1000 para pasar de km a metros*/  
  }  
}
```

3.9. Ejercicio 9

Dados dos puntos GPS, construir una grilla de $n \times m$. Estas grillas están conformadas por celdas contiguas rectangulares. Los lados latitudinales (respectivamente longitudinales) de todas las celdas miden la misma cantidad de grados. Cada celda está caracterizada por sus puntos superior izquierdo e inferior derecho (coordenadas GPS) y un nombre, que es un par ordenado de enteros que representa la posición de la celda en la grilla. Estos pares ordenados van desde (1, 1) en el punto que se encuentre en la celda que comienza en la posición esq1 y hasta (n, m) en la posición en donde se encuentra la celda con esquina esq2. La latitud de esq1 debe ser mayor a la latitud de esq2 y la longitud de esq1 debe ser menor a la longitud de esq2.

```

proc construirGrilla (in esq1: GPS, in esq2: GPS, in n:  $\mathbb{Z}$ , in m:  $\mathbb{Z}$ , out g: Grilla) {
  Pre {sonEsquinasValidas(esq1, esq2)  $\wedge$  n > 0  $\wedge$  m > 0}
  Post {esGrillaCorrecta(esq1, esq2, n, m, g)}
  pred sonEsquinasValidas (esq1, esq2: GPS) {
    sonCoordenadasValidas(esq1)  $\wedge$  sonCoordenadasValidas(esq2)  $\wedge$  esq10 > esq20  $\wedge$  esq11 < esq21
  }
  pred esGrillaCorrecta (esq1, esq2: GPS, n, m:  $\mathbb{Z}$ , g: Grilla) {
    |g| = m · n  $\wedge$  esquinasSonCombLineales(esq1, esq2, n, m, g)
  }
  pred esquinasSonCombLineales (esq1, esq2: GPS, n, m:  $\mathbb{Z}$ , g: Grilla) {
    ( $\forall a, b: \mathbb{Z}$ ) (
      (1 ≤ a ≤ n  $\wedge$  1 ≤ b ≤ m)  $\longrightarrow_L$  ( $\exists i: \mathbb{Z}$ ) (
        0 ≤ i < |g|  $\wedge_L$ 
        /*Esquina superior izquierda*/
        esqSupIzq(g[i]) = esqSupIzqCombinacion(a, b, n, m, esq1, esq2)  $\wedge$ 
        /*Esquina inferior derecha*/
        esqInfDer(g[i]) = esqInfDerCombinacion(a, b, n, m, esq1, esq2)  $\wedge$ 
        /*Nombre*/
        nombre(g[i]) = (a, b)
      )
    )
  }
  aux esqSupIzqCombinacion (a, b, n, m:  $\mathbb{Z}$ , esq1, esq2: GPS) : GPS =
    (esq10 - (a - 1) · (tamanoCelda(esq1, esq2, n, m))0, esq11 + (b - 1) · tamanoCelda(esq1, esq2, n, m)1);
  aux esqInfDerCombinacion (a, b, n, m:  $\mathbb{Z}$ , esq1, esq2: GPS) : GPS =
    (esq10 - a · (tamanoCelda(esq1, esq2, n, m))0, esq11 + b · tamanoCelda(esq1, esq2, n, m)1);
  aux esqSupIzq (c: Celda) : GPS = c0;
  aux esqInfDer (c: Celda) : GPS = c1;
  aux nombre (c: Celda) : Nombre = c2;
  aux tamanoCelda (esq1, esq2: GPS, n, m:  $\mathbb{Z}$ ) :  $\mathbb{R} \times \mathbb{R}$  = ( $\frac{esq1_0 - esq2_0}{n}$ ,  $\frac{esq2_1 - esq1_1}{m}$ );
}

```

3.10. Ejercicio 10

Dado un recorrido, devolver la secuencia ordenada de regiones visitadas por el colectivo.

```

proc regiones (in r: Recorrido, in g: Grilla, out res: seq⟨Nombre⟩) {
  Pre {esRecorridoValido(r) ∧ esGrillaDelRecorrido(g,r)}
  Post {esSecuenciaDelRecorrido(res,r,g)}
  pred esSecuenciaDelRecorrido (res: seq⟨Nombre⟩, r: Recorrido, g: Grilla) {
    |res|=|r|
    ∧ (∀i: ℤ) (
      0 ≤ i < |res| →L (∃c: Celda) (
        c ∈ g ∧ (nombre(c) = res[i] ∧ estaEnCelda(r[i],c))
      )
    )
  }
  pred estaEnCelda (p: GPS, c: Celda) {
    (esqInfDer(c)0 ≤ p0 ≤ esqSupIzq(c)0) ∧ (esqSupIzq(c)1 ≤ p1 ≤ esqInfDer(c)1)
  }
  pred esGrillaDelRecorrido (g: Grilla, r: Recorrido) {
    (∀i: ℤ) (
      0 ≤ i < |r| →L (∃c: Celda) (
        c ∈ g ∧ estaEnCelda(r[i],c)
      )
    )
    ∧ (∃esq1, esq2: GPS) (
      (∃n, m: ℤ) (
        esGrillaCorrecta(esq1, esq2, n, m, g)
      )
    )
  }
}

```

3.11. Ejercicio 11

Dado un viaje válido y una grilla, determinar cuántos saltos hay en el viaje.

```

proc cantidadDeSaltos (in g: Grilla, in v: Viaje, out res: seq⟨ℤ⟩) {
  Pre {esViajeValido(v) ∧ esGrillaDelViaje(g, v)}
  Post {esCantidadDeSaltos(g, v, res)}
  pred esCantidadDeSaltos (g: Grilla, v: Viaje, res: ℤ) {
    (∃v' : Viaje) (
      esElViajeOrdenado(v', v)
      ∧ (∃R : seq⟨Nombre⟩) (
        esSecuenciaDelViaje(R, v', g) ∧ cantidadDeSaltos(R) = res
      )
    )
  }
}

aux cantidadDeSaltos (R: seq⟨Nombre⟩) : ℤ =  $\sum_{i=0}^{|R|-2}$  (if esCeldaContigua(R[i], R[i + 1]) then 0 else 1 fi);

pred esCeldaContigua (n1, n2: Nombre) {
  |n10 - n20| ≤ 1 ∧ |n11 - n21| ≤ 1
}

pred esSecuenciaDelViaje (R: seq⟨Nombre⟩, v: Viaje, g: Grilla) {
  |R| = |v|
  /*Esto funciona porque el viaje está ordenado*/
  ∧ (∀i : ℤ) (
    0 ≤ i < |R| →L (∃c : Celda) (
      c ∈ g ∧ (nombre(c) = R[i] ∧ estaEnCelda(v1[i], c))
    )
  )
}

pred esGrillaDelViaje (g: Grilla, v: Viaje) {
  (∀i : ℤ) (
    0 ≤ i < |v| →L (∃c : Celda) (
      c ∈ g ∧ estaEnCelda(v[i]1, c)
    )
  )
  ∧ (∃esq1, esq2 : GPS) (
    (∃n, m : ℤ) (
      esGrillaCorrecta(esq1, esq2, n, m, g)
    )
  )
}

```

3.12. Ejercicio 12

Se cuenta con un viaje válido de más de 5 puntos, y la lista errores que indica cada momento para el cual el valor registrado por el GPS fue erróneo y que debe ser corregido automáticamente. Para la corrección, se buscan los dos puntos más cercanos temporalmente (y correctos), que permiten calcular la velocidad media del vehículo en ese tramo del viaje. Luego, esos dos puntos definen una recta, sobre la cual se va a definir el punto GPS corregido, de acuerdo a la distancia recorrida, usando para ello la velocidad media.

```

proc corregirViaje (inout v: Viaje, in errores: seq(Tiempo)) {
  /*Pedimos como precondition que el primero y el ultimo tiempo sean correctos*/
  /*De no ser así, no podríamos corregirlos*/
  Pre {
    |v| > 5 ∧ esViajeValido(v) ∧ sonTiemposValidos(errores)
    ∧ 10 · |errores| ≤ |v|
    ∧ primeroYUltimoSinErrores(v, errores)
    ∧ v = v0
  }
  Post {esViajeCorregido(v, v0, errores)}

  pred primeroYUltimoSinErrores (v: Viaje, e: seq(Tiempo)) {
    v[0]0 ∉ e ∧ v[|v|-1]0 ∉ e
  }

  pred sonTiemposValidos (e: seq(Tiempo)) {
    (∀i : ℤ) (
      0 ≤ i < |e| →L esTiempoValido(e[i])
    )
  }

  pred esViajeCorregido (v, v0: Viaje, e: seq(Tiempo)) {
    |v| = |v0| ∧L
    (∀i : ℤ) (
      0 ≤ i < |v0| →L ((v0[i]0 ∉ e → v[i] = v0[i]) ∨ (v0[i]0 ∈ e → esElPuntoCorregido(v0, v, i, e)))
    )
  }

  pred esElPuntoCorregido (v, v0: Viaje, i: ℤ, e: seq(Tiempo)) {
    /*Cada coordenada es la velocidad por el tiempo transcurrido desde el tiempo más
    cercano sin errores, sumado a la posición inicial en esa coordenada*/
    (∃k, j : ℤ) (
      0 ≤ k < j < |v|
      ∧L esMenorTiempoCorrectoMasCercano(v0, i, k, e)
      ∧ esMayorTiempoCorrectoMasCercano(v0, i, j, e)
      ∧ (v[i]1)0 = vectorVelocidad(v0[k], v0[j])0 · (v0[i]0 - v0[k]0) + (v0[k]1)0
      ∧ (v[i]1)1 = vectorVelocidad(v0[k], v0[j])1 · (v0[i]0 - v0[k]0) + (v0[k]1)1
      ∧ v[i]0 = v0[i]0
    )
  }
}

aux vectorVelocidad (m1, m2: Tiempo × GPS) : ℝ × ℝ = (  $\frac{(m2_1)_0 - (m1_1)_0}{m2_0 - m1_0}$ ,  $\frac{(m2_1)_1 - (m1_1)_1}{m2_0 - m1_0}$  );

pred esMenorTiempoCorrectoMasCercano (v: Viaje, i, k: ℤ, e: seq(Tiempo)) {
  v[k]0 < v[i]0 ∧ v[k]0 ∉ e
  ∧ ¬(∃h : ℤ) (
    0 ≤ h < |v| ∧L v[k]0 < v[h]0 < v[i]0 ∧ v[h]0 ∉ e
  )
}

```

```

pred esMayorTiempoCorrectoMasCercano (v: Viaje, i,j:  $\mathbb{Z}$ , e: seq(Tiempo)) {
    v[i]0 < v[j]0  $\wedge$  v[j]0  $\notin$  e
     $\wedge$   $\neg(\exists h : \mathbb{Z})$  (
        0  $\leq$  h < |v|  $\wedge_L$  v[i]0 < v[h]0 < v[j]0  $\wedge$  v[h]0  $\notin$  e
    )
}

```

3.13. Ejercicio 13

Dada una lista de viajes válidos, calcular el histograma de velocidades máximas registradas entre todos los viajes.

```

proc histograma (in xs: seq⟨Viaje⟩, in bins: ℤ, out cuentas: seq⟨ℤ⟩, out limites: seq⟨ℝ⟩) {
  Pre {sonViajesValidos(xs) ∧ bins > 0}
  Post {sonLimitesCorrectos(limites, xs, bins) ∧L sonCuentasCorrectas(xs, bins, limites, cuentas)}
  pred sonCuentasCorrectas (xs: seq⟨Viaje⟩, bins: ℤ, limites: seq⟨ℝ⟩, cuentas: seq⟨ℤ⟩) {
    |cuentas| = bins ∧L
    (∃ vels : seq⟨ℝ⟩) (
      sonVelocidadesMaximas(vels, xs)
      ∧ (∀ i : ℤ) (
        0 ≤ i < |cuentas| - 1 →L cuentas[i] = cantEnIntervalo(vels, limites[i], limites[i + 1])
      )
      ∧ cuentas[|cuentas| - 1] = cantEnIntervaloCerrado(vels, limites[|cuentas| - 1], limites[|cuentas|])
    )
  }
  pred sonLimitesCorrectos (limites: seq⟨ℝ⟩, xs: seq⟨Viaje⟩, bins: ℤ) {
    |limites| = bins + 1 ∧ estaOrdenado(limites)
    ∧ (∃ vels : seq⟨ℝ⟩) (
      sonVelocidadesMaximas(vels, xs)
      ∧ esMinimo(vels, limites[0]) ∧ esMaximo(vels, limites[|limites| - 1])
      ∧ (∀ i : ℤ) (
        0 ≤ i < |limites| →L limites[i] = limites[0] + i ·  $\frac{\text{limites}[|limites| - 1] - \text{limites}[0]}{\text{bins}}$ 
      )
    )
  }
  pred sonVelocidadesMaximas (vels: seq⟨ℝ⟩, xs: seq⟨Viaje⟩) {
    (∀ i : ℤ) (
      0 ≤ i < |xs| →L esVelocidadMaxima(vels[i], xs[i])
    )
  }
  pred esVelocidadMaxima (vel: ℝ, v: Viaje) {
    (∃ i, j : ℤ) (
      (0 ≤ i, j < |v| ∧L esTramo(v, v[i], v[j])) ∧ velocidadTramo(v[i], v[j]) = vel
    )
    ∧ (∀ i, j : ℤ) (
      (0 ≤ i, j < |v| ∧L esTramo(v, v[i], v[j])) →L velocidadTramo(v[i], v[j]) ≤ vel
    )
  }
  aux cantEnIntervalo (vels: seq⟨ℝ⟩, lim1, lim2: ℝ) : ℝ =  $\sum_{i=0}^{|vels|-1}$  (if lim1 ≤ vels[i] < lim2 then 1 else 0 fi) ;
  aux cantEnIntervaloCerrado (vels: seq⟨ℝ⟩, lim1, lim2: ℝ) : ℝ =  $\sum_{i=0}^{|vels|-1}$  (if lim1 ≤ vels[i] ≤ lim2 then 1 else 0 fi) ;

```

```

pred esMinimo (vels: seq( $\mathbb{R}$ ), min:  $\mathbb{R}$ ) {
  min  $\in$  vels  $\wedge$  ( $\forall i : \mathbb{Z}$ ) (
     $0 \leq i < |vels| \longrightarrow_L min \leq vels[i]$ 
  )
}

pred esMaximo (vels: seq( $\mathbb{R}$ ), max:  $\mathbb{R}$ ) {
  max  $\in$  vels  $\wedge$  ( $\forall i : \mathbb{Z}$ ) (
     $0 \leq i < |vels| \longrightarrow_L vels[i] \leq max$ 
  )
}

```