DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

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 $\{res = \text{true} \leftrightarrow esViajeValido(v)\}$

```
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 ↔ esRecorridoValido(v)}
    pred esRecorridoValido (v: Recorrido) {
        (∀i : ℤ)(0 ≤ i < |v| ⟶_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` $\{res = \text{true} \leftrightarrow estaEnTerritorio(v, r)\}$

`pred estaEnTerritorio` (v: *Viaje*, r: *Dist*) {

$(\exists c : GPS)(sonCoordenadasValidas(c) \quad \wedge_L \quad (\forall i : \mathbb{Z})(0 \leq i < |v| \longrightarrow_L dist(c, v[i]_1) \leq 1000 \cdot r))$
/* 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*) {

$(\exists v' : Viaje)(esElViajeOrdenado(v, v') \ \wedge \ d = sumaDistanciasSucesivas(v'))$

}

`pred esElViajeOrdenado` (v,v': *Viaje*) {

$estaOrdenadoTemporalmente(v') \ \wedge \ esPermutacion(v, v')$

}

`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*/
$(\forall e : Tiempo \times GPS)(\#apariciones(v1, e) = \#apariciones(v2, e))$

}

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

`aux sumaDistanciasSucesivas` (v: *Viaje*) : *Dist* $= \frac{1}{1000} \cdot \sum_{i=0}^{|v|-2} dist(v[i]_1, v[i+1]_1)$

/* 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` $\{res = \text{true} \leftrightarrow superaVelocidad(v)\}$

`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*) {

$e1_0 < e2_0 \ \wedge \ \neg(\exists e : Tiempo \times GPS)(e \in v \ \wedge \ e1_0 < e_0 < e2_0)$

}

`aux velocidadTramo` (e1,e2 : *Tiempo* $\times$ *GPS*) : $\mathbb{R} = \frac{dist(e1_1, e2_1)}{e2_0 - e1_0} \cdot 3{,}6$
/* 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 inclusives. 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: $seq\langle Viaje\rangle$, in t_0: *Tiempo*, in t_f: *Tiempo*, out res: $\mathbb{Z}$) {

Pre $\{sonTodosViajesValidos(vs) \ \wedge \ t_0 \leq t_f \ \wedge \ esTiempoValido(t_0) \ \wedge \ esTiempoValido(t_f)\}$

Post $\{esCantidadEnRuta(vs, t_0, t_f, res)\}$

pred sonTodosViajesValidos (vs: $seq\langle Viaje\rangle$) {

$(\forall v : Viaje)(v \in vs \longrightarrow esViajeValido(v))$

}

pred esCantidadEnRuta (vs: $seq\langle Viaje\rangle$, t_0,t_f: *Tiempo*, res: $\mathbb{Z}$) {

$res = \sum_{i=0}^{|vs|-1}$ (if $estaEnRuta(v[i], t_0, t_f)$ then 1 else 0 fi)

}

pred esCantidadEnRuta (vs: $seq\langle Viaje\rangle$, t_0,t_f: *Tiempo*, res: $\mathbb{Z}$) {

$(\exists vs' : seq\langle Viaje\rangle)$ (

$(\forall v : Viaje)$ (

$(v \in vs \ \wedge \ estaEnRuta(v, t_0, t_f)) \longrightarrow_L \#aparicionesViajes(v, vs') = \#aparicionesViajes(v, vs)$

)

$\wedge \ |vs'| = res$

)

}

pred estaEnRuta (v: *Viaje*, t_0,t_f: *Tiempo*) {

$(\exists i, j : \mathbb{Z})$ (

$0 \leq i, j < |v| \ \wedge_L \ v[i]_0 \leq t_0 < t_f \leq v[j]_0$

)

$\vee \ (\exists i : \mathbb{Z})$ (

$0 \leq i < |v| \ \wedge_L \ t_0 \leq v[i]_0 \leq t_f$

)

}

pred estaEnRuta (v: *Viaje*, t_0,t_f: *Tiempo*) {

$(\exists i, j : \mathbb{Z})$ (

$0 \leq i \leq j < |v| \ \wedge_L \ (v[i]_0 \leq t_f \ \wedge \ v[j]_0 \geq t_0)$

)

}

aux #aparicionesViajes (v: *Viaje*, vs: $seq\langle Viaje\rangle$) : $\mathbb{Z} = \sum_{i=0}^{|vs|-1}$ (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) ∧ u > 0 ∧ 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*/
        (∀p : GPS) (
            p ∈ res ↔ (p ∈ r ∧ ¬estaCubierto(p, v, u))
        )
    }
    pred estaCubierto (p: GPS, v: Viaje, u: Dist) {
        (∃m : Tiempo × GPS) (
            m ∈ v ∧ dist(m_1, p) < u · 1000
        )
        /*Multiplicamos por 1000 para pasar de km a metros*/
    }
}
```

3.9. Ejercicio 9

Dados dos puntos GPS, construir una grilla de n × 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*) {

$sonCoordenadasValidas(esq1) \ \wedge \ sonCoordenadasValidas(esq2) \ \wedge \ esq1_0 > esq2_0 \ \wedge \ esq1_1 < esq2_1$

}

`pred esGrillaCorrecta` (esq1,esq2: *GPS*, n,m: $\mathbb{Z}$, g: *Grilla*) {

$|g| = m \cdot n \ \wedge \ esquinasSonCombLineales(esq1, esq2, n, m, g)$

}

`pred esquinasSonCombLineales` (esq1,esq2: *GPS*, n,m: $\mathbb{Z}$, g: *Grilla*) {

$(\forall a, b : \mathbb{Z})$ (

$(1 \leq a \leq n \ \wedge \ 1 \leq b \leq m) \longrightarrow_L (\exists i : \mathbb{Z})$ (

$0 \leq i < |g| \ \wedge_L$
/*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* =
$(esq1_0 - (a-1) \cdot (tamanoCelda(esq1, esq2, n, m))_0, \ esq1_1 + (b-1) \cdot tamanoCelda(esq1, esq2, n, m)_1)$;

`aux esqInfDerCombinacion` (a,b,n,m: $\mathbb{Z}$, esq1,esq2: *GPS*) : *GPS* =
$(esq1_0 - a \cdot (tamanoCelda(esq1, esq2, n, m))_0, \ esq1_1 + b \cdot tamanoCelda(esq1, esq2, n, m)_1)$;

`aux esqSupIzq` (c: *Celda*) : *GPS* $= c_0$;

`aux esqInfDer` (c: *Celda*) : *GPS* $= c_1$;

`aux nombre` (c: *Celda*) : *Nombre* $= c_2$;

`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 = Σ_{i=0}^{|R|-2} (if esCeldaContigua(R[i], R[i+1]) then 0 else 1 fi);
    pred esCeldaContigua (n1,n2: Nombre) {
        |n1_0 − n2_0| ≤ 1 ∧ |n1_1 − n2_1| ≤ 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(v_1[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: $seq\langle Tiempo\rangle$) {

/*Pedimos como precondición que el primero y el ultimo tiempo sean correctos*/
/*De no ser así, no podríamos corregirlos*/
`Pre` {
$|v| > 5 \ \wedge \ esViajeValido(v) \ \wedge \ sonTiemposValidos(errores)$
$\wedge \ 10 \cdot |errores| \leq |v|$
$\wedge \ primeroYUltimoSinErrores(v, errores)$
$\wedge \ v = v_0$
}

`Post` {$esViajeCorregido(v, v_0, errores)$}

`pred primeroYUltimoSinErrores` (v: *Viaje*, e: $seq\langle Tiempo\rangle$) {

$v[0]_0 \notin e \ \wedge \ v[|v|-1]_0 \notin e$

}

`pred sonTiemposValidos` (e: $seq\langle Tiempo\rangle$) {

$(\forall i : \mathbb{Z})$ (

$0 \leq i < |e| \longrightarrow_L esTiempoValido(e[i])$

)

}

`pred esViajeCorregido` (v,v_0: *Viaje*, e: $seq\langle Tiempo\rangle$) {

$|v| = |v_0| \ \wedge_L$
$(\forall i : \mathbb{Z})$ (

$0 \leq i < |v_0| \longrightarrow_L ((v_0[i]_0 \notin e \longrightarrow v[i] = v_0[i]) \ \vee \ (v_0[i]_0 \in e \longrightarrow esElPuntoCorregido(v_0, v, i, e)))$

)

}

`pred esElPuntoCorregido` (v,v_0: *Viaje*, i: $\mathbb{Z}$, e: $seq\langle Tiempo\rangle$) {

/*Cada coordenada es la velocidad por el tiempo transcurrido desde el tiempo más
cercano sin errores, sumado a la posición inicial en esa coordenada*/
$(\exists k, j : \mathbb{Z})$ (

$0 \leq k < j < |v|$
$\wedge_L \ \ esMenorTiempoCorrectoMasCercano(v_0, i, k, e)$
$\wedge \ esMayorTiempoCorrectoMasCercano(v_0, i, j, e)$
$\wedge \ (v[i]_1)_0 = vectorVelocidad(v_0[k], v_0[j])_0 \cdot (v_0[i]_0 - v_0[k]_0) + (v_0[k]_1)_0$
$\wedge \ (v[i]_1)_1 = vectorVelocidad(v_0[k], v_0[j])_1 \cdot (v_0[i]_0 - v_0[k]_0) + (v_0[k]_1)_1$
$\wedge \ v[i]_0 = v_0[i]_0$

)

}

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

`pred esMenorTiempoCorrectoMasCercano` (v: *Viaje*, i,k: $\mathbb{Z}$, e: $seq\langle Tiempo\rangle$) {

$v[k]_0 < v[i]_0 \ \wedge \ v[k]_0 \notin e$
$\wedge \ \neg(\exists h : \mathbb{Z})$ (

$0 \leq h < |v| \ \wedge_L \ v[k]_0 < v[h]_0 < v[i]_0 \ \wedge \ v[h]_0 \notin e$

)

}

```
    pred esMayorTiempoCorrectoMasCercano (v: Viaje, i,j: ℤ, e: seq⟨Tiempo⟩) {
        v[i]_0 < v[j]_0  ∧  v[j]_0 ∉ e
        ∧  ¬(∃h : ℤ) (
            0 ≤ h < |v|  ∧_L  v[i]_0 < v[h]_0 < v[j]_0  ∧  v[h]_0 ∉ 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: $seq\langle Viaje\rangle$, in bins: $\mathbb{Z}$, out cuentas: $seq\langle\mathbb{Z}\rangle$, out limites: $seq\langle\mathbb{R}\rangle$) {

Pre $\{sonViajesValidos(xs) \ \wedge \ bins > 0\}$

Post $\{sonLimitesCorrectos(limites, xs, bins) \ \wedge_L \ sonCuentasCorrectas(xs, bins, limites, cuentas)\}$

pred sonCuentasCorrectas (xs: $seq\langle Viaje\rangle$, bins: $\mathbb{Z}$, limites: $seq\langle\mathbb{R}\rangle$, cuentas: $seq\langle\mathbb{Z}\rangle$) {

$|cuentas| = bins \ \wedge_L$
$(\exists vels : seq\langle\mathbb{R}\rangle)$ (

$sonVelocidadesMaximas(vels, xs)$
$\wedge \ (\forall i : \mathbb{Z})$ (

$0 \leq i < |cuentas|-1 \longrightarrow_L cuentas[i] = cantEnIntervalo(vels, limites[i], limites[i+1])$

)
$\wedge \ cuentas[|cuentas|-1] = cantEnIntervaloCerrado(vels, limites[|cuentas|-1], limites[|cuentas|])$

)

}

pred sonLimitesCorrectos (limites: $seq\langle\mathbb{R}\rangle$, xs: $seq\langle Viaje\rangle$, bins: $\mathbb{Z}$) {

$|limites| = bins + 1 \ \wedge \ estaOrdenado(limites)$
$\wedge \ (\exists vels : seq\langle\mathbb{R}\rangle)$ (

$sonVelocidadesMaximas(vels, xs)$
$\wedge \ esMinimo(vels, limites[0]) \ \wedge \ esMaximo(vels, limites[|limites|-1])$
$\wedge \ (\forall i : \mathbb{Z})$ (

$0 \leq i < |limites| \longrightarrow_L limites[i] = limites[0] + i \cdot \frac{limites[|limites|-1] - limites[0]}{bins}$

)

)

}

pred sonVelocidadesMaximas (vels: $seq\langle\mathbb{R}\rangle$, xs: $seq\langle Viaje\rangle$) {

$(\forall i : \mathbb{Z})$ (

$0 \leq i < |xs| \longrightarrow_L esVelocidadMaxima(vels[i], xs[i])$

)

}

pred esVelocidadMaxima (vel: $\mathbb{R}$, v: $Viaje$) {

$(\exists i, j : \mathbb{Z})$ (

$(0 \leq i, j < |v| \ \wedge_L \ esTramo(v, v[i], v[j])) \ \wedge \ velocidadTramo(v[i], v[j]) = vel$

)
$\wedge \ (\forall i, j : \mathbb{Z})$ (

$(0 \leq i, j < |v| \ \wedge_L \ esTramo(v, v[i], v[j])) \longrightarrow_L velocidadTramo(v[i], v[j]) \leq vel$

)

}

aux cantEnIntervalo (vels: $seq\langle\mathbb{R}\rangle$, lim1,lim2: $\mathbb{R}$) : $\mathbb{R}$ $= \sum_{i=0}^{|vels|-1}$ (if $lim1 \leq vels[i] < lim2$ then 1 else 0 fi) ;

aux cantEnIntervaloCerrado (vels: $seq\langle\mathbb{R}\rangle$, lim1,lim2: $\mathbb{R}$) : $\mathbb{R}$ $= \sum_{i=0}^{|vels|-1}$ (if $lim1 \leq vels[i] \leq lim2$ then 1 else 0 fi) ;

```
    pred esMinimo (vels: seq⟨ℝ⟩, min: ℝ) {
        min ∈ vels  ∧  (∀i : ℤ) (
            0 ≤ i < |vels| ⟶_L min ≤ vels[i]
        )
    }
    pred esMaximo (vels: seq⟨ℝ⟩, max: ℝ) {
        max ∈ vels  ∧  (∀i : ℤ) (
            0 ≤ i < |vels| ⟶_L vels[i] ≤ max
        )
    }
}
```