UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación

Algoritmos y Estructura de Datos I

Segundo cuatrimestre de 2013

28 de Octubre de 2013

# Especificación del TPI - Cine 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 | x ← pre(this), x ≠ e ];
}

problema darVuelta (this : Lista⟨T⟩) {
  modifica this;
  asegura |this| == |pre(this)| ∧ (∀i ← [0..|this|)) this_i = pre_{|this|−i−1};
}

problema posicion (this : Lista⟨T⟩,e : T) = result : ℤ {
  requiere e ∈ this;
  asegura result = [ i | i ← [0..|this|), this_i = e ]_0;
}

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

problema cantidadDeApariciones (this : Lista⟨T⟩, e:T) = result : ℤ {
  asegura result = |[ e | e' ← this, e' == e ]|;
}
```

## 2. Tipos

```
tipo Actor = String;
tipo Sala = ℤ;
tipo Genero = Aventura, Comedia, Drama, Romantica, Terror;
```

## 3. Pelicula

```
tipo Pelicula {
  observador nombre (p: Pelicula) : String;
  observador generos (p: Pelicula) : [Genero];
  observador actores (p: Pelicula) : [Actor];
  observador es3D (p: Pelicula) : Bool;
  invariante sinActoresRepetidos : sinRepetidos(actores(p));
  invariante sinGenerosRepetidos : sinRepetidos(generos(p));
  invariante generosOrdenados : listaOrdenada(generos(p));
  invariante actoresOrdenados : listaOrdenada(actores(p));
}

problema nuevaP (n: Nombre, gs: [Género], as: [Actor], b: Bool) = result : Pelicula {
  requiere listaOrdenada(gs) ∧ listaOrdenada(as);
  asegura nombre(result) == n ∧ es3D(result) == b;
  asegura ((∀g ← gs)g ∈ generos(result)) ∧ ((∀g ← generos(result))g ∈ gs);
  asegura ((∀a ← as)a ∈ actores(result)) ∧ ((∀a ← actores(result))a ∈ as);
}

problema nombreP (p: Pelicula) = result : Nombre {
  asegura result == nombre(p);
}

problema generosP (p: Pelicula) = result : [Género] {
  asegura result == generos(p);
}

problema actoresP (p: Pelicula) = result : [Actor] {
  asegura result == actores(p);
}
```

```
problema es3DP (p: Pelicula) = result : Bool {
  asegura result == es3D(p);
}

problema agruparPelisPorGeneroP (ps: [Pelicula]) = result : [(Genero, [Pelicula])] {
  asegura unGrupoPorGenero : sinRepetidos(primeros(result));
  asegura sinCopiasEnGrupos : (∀d ← result)sinRepetidos(sgd(d));
  asegura estanTodas : (∀p ← ps, g ← generos(p))(∃d ← result)prm(d) == g ∧ p ∈ sgd(d);
  asegura noHayDeMas : (∀d ← result)(|sgd(d)| > 0 ∧ (∀p ← sgd(d))p ∈ ps ∧ prm(d) ∈ generos(p));
}

problema generarSagaDePeliculasP (as:[Actor], gs:[Genero], nombres:[Nombre]) = result : [Pelicula] {
  requiere nombresDistintos : sinRepetidos(nombres);
  requiere ordenadas : listaOrdenada(as) ∧ listaOrdenada(gs);
  asegura mismaLongitud : |result| == |nombres|;
  asegura unaPeliPorNombre : (∀n ← nombres)(∃p ← result)nombre(p) == n
      ∧ mismos(generos(p), sacarRepetidos(gs)) ∧ mismos(actores(p), sacarRepetidos(as));
}
```

# 4. Ticket

```
tipo Ticket {
  observador película (t: Ticket) : Pelicula;
  observador sala (t: Ticket) : Sala;
  observador usado (t: Ticket) : Bool;
}

problema nuevoT (p: Pelicula, s: Sala, u: Bool) = result : Ticket {
  asegura pelicula(result) == p ∧ sala(result) == s ∧ usado(result) == u;
}

problema peliculaT (this: Ticket) = result : Pelicula {
  asegura result == pelicula(this);
}

problema salaT (this: Ticket) = result : Sala {
  asegura result == sala(this);
}

problema usadoT (this: Ticket) = result : Bool {
  asegura result == usado(this);
}

problema usarT (this: Ticket) {
  modifica this;
  asegura mismoTicket : pelicula(this) == pelicula(pre(this)) ∧ sala(this) == sala(pre(this));
  asegura ahoraUsado : usado(this);
}

problema peliculaMenosVistaT (ts:[Ticket]) = result : Pelicula {
  requiere alMenosUna : |ts| > 0;
  asegura esValida : (∃t ← ts)pelicula(t) == result;
  asegura esLaMenosVista : (∀t ← ts)sumaUsado(result, ts) ≤ sumaUsado(t, ts);
}

problema todosLosTicketsParaLaMismaSalaT (ts:[Ticket]) = result : Bool {
  asegura result ↔ (∀i ← [1.. |t|))sala(t_i) == sala(t_0);
}

problema cambiarSala (ts:[Ticket], vieja: Sala, nueva: Sala) = result : [Ticket] {
  asegura mismaLongitud : |ts| == |result|;
  asegura otrasNoCambian : (∀i ← [0.. |ts|), sala(ts[i])! = vieja)ts[i] == result[i];
```

asegura viejaReemplazada : $(\forall \leftarrow [0..|ts|), sala(ts[i]) == vieja) pelicula(result[i]) == pelicula(ts[i]) \land usado(ts[i]) == usado(result[i]) \land sala(result[i]) == nueva$ ;
}

# 5. Cine

```
tipo Cine {
  observador nombre (c: Cine) : String ;
  observador películas (c: Cine) : [Peliculas] ;
  observador salas (c: Cine) : [Sala] ;
  observador sala (c: Cine, p: Pelicula) : Sala ;
       requiere p ∈ peliculas(c) ;
  observador espectadores (c: Cine, s: Sala) : ℤ ;
       requiere s ∈ salas(c) ;
  observador ticketsVendidosSinUsar (c: Cine) : [Ticket] ;
  invariante sinPeliculasRepetidas : sinRepetidos(nombresDePeliculas(c)) ;
  invariante sinSalasRepetidas : sinRepetidos(salas(c)) ;
  invariante salasDeCineSonSalas : (∀p ← peliculas(c))sala(c, p) ∈ salas(c) ;
  invariante salasSinPeliSinEspectadores : (∀s ← salas(c))¬tienePeli(c, s) ⇒ espectadores(c, s) == 0 ;
  invariante espectadoresNoNegativos : (∀s ← salas(c))espectadores(c, s) ≥ 0 ;
  invariante salasConsistentes : (∀p ← peliculas(c), q ← peliculas(c))sala(c, p) == sala(c, q) ⇒ p == q ;
  invariante losTicketsVendidosSonParaPeliculasDelCine : (∀t ← ticketsVendidosSinUsar(c))ticketOK(t, c) ;
  invariante losTicketsVendidosEstanSinUsar : (∀t ← ticketsVendidosSinUsar(c))¬usado(t) ;
}

problema nuevoC (n: Nombre) = result : Cine {
  asegura nombre(result) == n ∧ |peliculas(result)| == 0 ∧ |salas(result)| == 0 ;
}

problema nombreC (this: Cine) = result : Nombre {
  asegura result == nombre(this) ;
}

problema peliculasC (this: Cine) = result : [Peliculas] {
  asegura result == peliculas(this) ;
}

problema salasC (this: Cine) = result : [Sala] {
  asegura result == salas(this) ;
}

problema espectadoresC (this: Cine, s:Sala) = result : ℤ {
  requiere s ∈ salas(this) ;
  asegura result == espectadores(this, s) ;
}

problema salaC (this: Cine, n:Nombre) = result : Sala {
  requiere hayPeliConNombre : |[ p | p ← peliculas(this), nombre(p) == n ]| > 0 ;
  asegura result == sala(this, peliConNombre(this, n)) ;
}

problema ticketsVendidosC (this: Cine) = result : [Ticket] {
  asegura result == ticketsVendidosSinUsar(this) ;
}

problema abrirSalaC (this: Cine, s : Sala) {
  requiere salaNueva : ¬(s ∈ salas(this)) ;
  modifica this ;
  asegura mismoNombre : nombre(this) == nombre(pre(this)) ;
  asegura abreSala : mismos(salas(this), s : salas(pre(this))) ;
  asegura mismasPelis : mismos(peliculas(this), peliculas(pre(this))) ;
  asegura pelisEnMismaSala : (∀p ← peliculas(pre(this)))sala(this, p) == sala(pre(this), p) ;
  asegura mismosEspectadores : (∀s ← salas(pre(this)))espectadores(this, s) == espectadores(pre(this), s) ;
  asegura mismosTickets : mismos(ticketsVendidosSinUsar(this), ticketsVendidosSinUsar(pre(this))) ;
}

problema agregarPeliculaC (this:Cine, p:Pelicula, s:Sala) {
  requiere salaVacante : s ∈ salas(this) ∧ ¬(∃p2 ← peliculas(this))sala(p2) == s ;
  requiere peliNueva : ¬(∃p2 ← peliculas(this))nombre(p2) == nombre(p) ;
```

modifica $this$;
asegura mismoNombre : $nombre(this) == nombre(pre(this))$;
asegura mismasSalas : $mismos(salas(this), salas(pre(this)))$;
asegura agregaPeli : $mismos(peliculas(this), p : peliculas(pre(this)))$;
asegura otrasPelisEnMismaSala : $(\forall p2 \leftarrow peliculas(pre(this)))sala(this, p2) == sala(pre(this), p2)$;
asegura mismosEspectadores : $(\forall s \leftarrow salas(this))espectadores(this, s) == espectadores(pre(this), s)$;
asegura mismosTickets : $mismos(ticketsVendidosSinUsar(this), ticketsVendidosSinUsar(pre(this)))$;
asegura salaPedida : $sala(this, p) == s$;
}

problema cerrarSalaC (this:Cine,s:Sala) {
requiere salaValida : $s \in salas(this)$;
requiere ceroDemanda : $\neg(\exists t \leftarrow ticketsVendidosSinUsar(this))sala(t) == s$;
modifica $this$;
asegura mismoNombre : $nombre(this) == nombre(pre(this))$;
asegura sacarSala : $mismos(s : salas(this), salas(pre(this)))$;
asegura mismasPelis : $mismos(peliculas(this), [\, p \mid p \leftarrow peliculas(pre(this)), sala(pre(this), p) \neq s \,])$;
asegura pelisEnMismaSala : $(\forall p \leftarrow peliculas(this))sala(this, p) == sala(pre(this), p)$;
asegura mismosEspectadores : $(\forall s \leftarrow salas(this))espectadores(this, s) == espectadores(pre(this), s)$;
asegura mismosTickets : $mismos(ticketsVendidosSinUsar(this), ticketsVendidosSinUsar(pre(this)))$;
}

problema cerrarSalasC (this:Cine, e:$\mathbb{Z}$) {
requiere ceroDemanda : $\neg(\exists t \leftarrow ticketsVendidosSinUsar(this))espectadores(this, sala(t)) < e$;
modifica $this$;
asegura respetaCotaDeEspectadores : $igualesCerrandoLasMenosVistas(pre(this), this, e)$;
}

problema cerrarSalasDeLaCadenaC (cs:[Cine], e:$\mathbb{Z}$) {
requiere ceroDemanda : $\neg(\exists c \leftarrow cs, t \leftarrow ticketsVendidosSinUsar(c))espectadores(c, sala(t)) < e$;
modifica $cs$;
asegura mismaCantidad : $|cs| == |pre(cs)|$;
asegura mismosCinesRespetandoCota : $(\forall c1 \leftarrow pre(cs))(\exists c2 \leftarrow cs)igualesCerrandoLasMenosVistas(c1, c2, e)$
$\wedge\, cuentaCines(c1, pre(cs)) == cuentaCines(c2, cs)$;
}

problema peliculaC (this:Cine, s:Sala) = result : Pelicula {
requiere salaValida : $s \in salas(this) \wedge (\exists p \leftarrow peliculas(this))sala(this, p) == s$;
asegura esLaPeliDeLaSala : $result \in peliculas(this) \wedge sala(this, result) == s$;
}

problema venderTicketC (this:Cine, n:Nombre) = result : Ticket {
requiere hayPeliConNombre : $|[\, p \mid p \leftarrow peliculas(this), nombre(p) == n \,]| > 0$;
modifica $this$;
asegura mismoNombre : $nombre(this) == nombre(pre(this))$;
asegura mismasSalas : $mismos(salas(this), salas(pre(this)))$;
asegura mismasPelis : $mismos(peliculas(this), peliculas(pre(this)))$;
asegura pelisEnMismaSala : $(\forall p \leftarrow peliculas(pre(this)))$
$sala(this, peliConNombre(this, n)) == sala(pre(this), peliConNombre(pre(this), n))$;
asegura mismosEspectadores : $(\forall s \leftarrow salas(pre(this)))espectadores(this, s) == espectadores(pre(this), s)$;
asegura ticketParaLaPeli : $pelicula(result) == peliConNombre(this, n) \wedge \neg usado(result)$;
asegura ticketEnLaLista : $mismos(ticketsVendidosSinUsar(this), result : ticketsVendidosSinUsar(pre(this)))$;
}

problema ingresarASalaC (this:Cine, s:Sala, t:Ticket) = result : Ticket {
requiere salaConsistente : $s == sala(t)$;
requiere ticketValido : $t \in ticketsVendidosSinUsar(this)$;
modifica $this$;
asegura mismoNombre : $nombre(this) == nombre(pre(this))$;
asegura mismasSalas : $mismos(salas(this), salas(pre(this)))$;
asegura mismasPelis : $mismos(peliculas(this), peliculas(pre(this)))$;
asegura pelisEnMismaSala : $(\forall p \leftarrow peliculas(this))sala(this, p) == sala(pre(this), p)$;
asegura mismosEspectadores : $(\forall s2 \leftarrow salas(this), s2 \neq s)espectadores(this, s) == espectadores(pre(this), s)$;
asegura unEspectadorMas : $espectadores(this, s) == espectadores(pre(this), s) + 1$;
asegura usaElTicket : $mismos(t : ticketsVendidosSinUsar(this), ticketsVendidosSinUsar(this))$;

  asegura elTicketSeUso : $sala(result) == sala(t) \land pelicula(result) == pelicula(t) \land usado(result)$ ;
}

problema pasarA3DUnaPeliculaC (this:Cine, nombre:Nombre) = result : Pelicula {
  requiere estaLaPeli : $(\exists p \leftarrow peliculas(this))nombre(p) == nombre$ ;
  modifica $this$ ;
  asegura mismoNombre : $nombre(this) == nombre(pre(this))$ ;
  asegura mismasSalas : $mismos(salas(this), salas(pre(this)))$ ;
  asegura laPeliEs3D : $igualExcepto3D(result, peliConNombre(pre(this), nombre)) \land es3D(result)$ ;
  asegura cambiaPeliA3D : $mismos(peliConNombre(pre(this), nombre) : peliculas(this), result : peliculas(pre(this)))$ ;
  asegura pelisEnMismaSala : $(\forall p \leftarrow peliculas(pre(this)), nombre(p) \neq nombre)sala(this, p) == sala(pre(this), p)$ ;
  asegura esaPeliEnMismaSala : $sala(this, result) == sala(pre(this), peliConNombre(pre(this), nombre))$ ;
  asegura mismosEspectadores : $(\forall s \leftarrow salas(pre(this)))espectadores(this, s) == espectadores(pre(this), s)$ ;
  asegura mismosTickets : $(\forall s \leftarrow salas(pre(this)))$
    $contarSala(s, ticketsVendidosSinUsar(this)) == contarSala(s, ticketsVendidosSinUsar(this))$ ;
}

# 6. Auxiliares

aux sinRepetidos (l: [T]) : Bool $= (\forall i, j \leftarrow [0..|l|), i \neq j)l_i \neq l_j$ ;
aux listaOrdenada (l: [T]) : Bool $= (\forall i \leftarrow [0..|l|-1))l_i \leq l_{i+1}$ ;
aux ticketOK (t: Ticket, c:Cine) : Bool $= (\exists p \leftarrow peliculas(c), s \leftarrow salas(c))p == pelicula(t) \land s == sala(t) \land sala(c, p) == s$ ;
aux primeros (ls : [($T_1$, $T_2$) ]) : [$T_1$] $= [\, prm(x) \mid x \leftarrow ls \,]$ ;
aux sacarRepetidos (l: [T]) : [T] $= [\, l_i \mid i \leftarrow [0..|l|), \neg(l_i \in l[0..i)) \,]$ ;
aux tienePeli (c: Cine, s:Sala) : Bool $= (\exists p \leftarrow pelicula(c))sala(c, p) == s$ ;
aux sumaUsado (p : Pelicula, ts : [Ticket]) : $\mathbb{Z}$ $= |[\, 1 \mid t \leftarrow ts, pelicula(t) == p \land usado(t) \,]|$ ;
aux cuentaCines (c:Cine,cs:[Cine]) : $\mathbb{Z}$ $= |[\, 1 \mid c2 \leftarrow cs, cinesIguales(c, c2) \,]|$ ;
aux cinesIguales (c1,c2:Cine) : Bool $= nombre(c1) == nombre(c2) \land mismos(salas(c1), salas(c2))$
  $\land\, mismos(peliculas(c1), peliculas(c2)) \land (\forall p \leftarrow peliculas(c1))sala(c1, p) == sala(c2, p)$
  $\land\, (\forall s \leftarrow salas(c1))espectadores(c1, s) == espectadores(c2, s)$
  $\land\, mismos(ticketsVendidosSinUsar(c1), ticketsVendidosSinUsar(c2))$ ;
aux igualesCerrandoLasMenosVistas (c1,c2:Cine,e:$\mathbb{Z}$) : Bool $= nombre(c1) == nombre(c2)$
  $\land\, mismos(salas(c2), [\, s \mid s \leftarrow salas(c1), espectadores(c1, s) \geq e \,]$
  $\land\, mismos(peliculas(c2), [\, p \mid p \leftarrow peliculas(c1), espectadores(c1, sala(c1, p)) \geq e \,])$
  $\land\, (\forall p \leftarrow peliculas(c2))sala(c2, p) == sala(c1, p)$
  $\land\, (\forall s \leftarrow salas(c2))espectadores(c2, s) == espectadores(c1, s)$
  $\land\, mismos(ticketsVendidosSinUsar(c1), ticketsVendidosSinUsar(c2))$ ;
aux peliConNombre (c:Cine,n:Nombre) : Pelicula $= [\, p \mid p \leftarrow peliculas(c), nombre(p) == n \,]_0$ ;
aux igualExcepto3D (p1,p2 : Pelicula) : Bool $= nombre(p1) == nombre(p2) \land mismos(generos(p1), generos(p2))$
  $\land\, mismos(actores(p1)actores(p2))$ ;
aux contarSala (s:Sala, ts:[Ticket]) : $\mathbb{Z}$ $= |[\, 1 \mid t \leftarrow ts, sala(t) == s \,]|$ ;