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

# Algoritmos y Estructura de Datos I

Segundo cuatrimestre de 2013

8 de septiembre de 2013

# TPE Cine

```
tipo Actor = String;
tipo Sala = ℤ;
tipo género = Aventura, Comedia, Drama, Romántica, Terror;

tipo Pelicula {
  observador nombre (p: Pelicula) : String;
  observador géneros (p: Pelicula) : [Género];
  observador actores (p: Pelicula) : [Actor];
  observador es3D (p: Pelicula) : Bool;
  invariante sinActoresRepetidos : sinRepetidos(actores(p));
  invariante sinGénerosRepetidos : sinRepetidos(generos(p));
  invariante génerosOrdenados : generosOrd(generos(p));
  invariante actoresOrdenados : actoresOrd(actores(p));
}

problema agruparPelisPorGenero (ps:[Pelicula]) = result : [(Genero, [Pelicula])] {
  asegura generoSalidaEntrada : (∀dupla ← result)prm(dupla) ∈ obtenerGeneros(ps);
  asegura pelisSalidaEntrada : (∀dupla ← result, peli ← sgd(dupla))peli ∈ ps;
  asegura pelisSalidaSinRepetir : (∀dupla ← result)sinRepetir(sgd(dupla));
  asegura generosSalidaEnPelis : (∀dupla ← result, peli ← sgd(dupla))prm(dupla) ∈ generos(peli);
  asegura generosSalidaSinRepetir : sinRepetidos(obtenerGenerosDupla(result));
  asegura pelisEntradaEnSalida : (∀peli ← ps, dupla ← result, prm(dupla) ∈ generos(peli))peli ∈ sgd(dupla);
  asegura generosEntradaEnSalida : (∀genero ← obetenerGeneros(ps))genero ∈ obtenerGenerosDupla(result);
  aux obtenerGeneros (ps: [Pelicula]) : [Genero] = [ genero(p) | p ← ps ];
  aux obtenerGenerosDupla (x: [(Genero,[Pelicula])]) : [Genero] = [ prm(dupla) | dupla ← x ];
}

problema generarSagaDePeliculas (as:[Actor], gs:[Genero], nombres:[String]) = result : [Pelicula] {
  asegura mismosGeneros : (∀peli ← result)mismos(generos(peli), elementosSinRepetir(gs));
  asegura mismosActores : (∀peli ← result)mismos(actores(peli), elementosSinRepetir(as));
  asegura todosNombreEnNombres : mismos(listaNombrePelis(res), elementosSinRepetir(nombres));
  aux listaNombrePelis (listaPelis: [Pelicula]) : [String] = [ nombre(peli) | peli ← listaPelis ];
  aux elementosSinRepetir (s: [T]) : [T] = [ s_i | i ← [0..|s|), ¬(∃j ← [0..|s|), j ≠ i)s_i == s_j ];
}

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

problema películaMenosVista (ts : [Ticket]) = result : Bool {
  requiere ts ≠ [];
  requiere ticketsUsados(ts) ≠ [];
  asegura peliEstaEnTickets : result ∈ peliculasVistas(ticketsUsados(ts));
  asegura esLaPeliMenosVista : ¬(∃p ← peliculasVistas(ticketsUsados(ts))ticketsPorPelicula(p, ticketsUsados(ts)) <
      ticketsPorPelicula(result, ticketsUsados(ts));
  aux ticketsUsados (s: [Ticket]) : [Ticket] = [ x | x ← s, usado(x) ];
  aux peliculasVistas (s: [Ticket]) : [Pelicula] = [ pelicula(x) | x ← s ];
  aux ticketsPorPelicula (p: Pelicula, tic: [Ticket]) : ℤ = [ t | t ← tic, pelicula(t) == p ];
}

problema todosLosTicketsParaLaMismaSala (ts:[Ticket]) = result : Bool {
  asegura todosLosTicketsParaLaMismaSala : res == (∀t ← [0..|ts| − 1))sala(ts_i) == sala(ts_{i+1});
}
```

```
problema cambiarSala (ts:[Ticket], vieja: Sala, nueva: Sala) {
  modifica ts;
  asegura mismoNumeroTickets : |ts| == |pre(ts)|;
  asegura cambioDeSala : (∀i ← [0..|pre(ts)|), sala(pre(ts)_i) == vieja)sala(ts_i == nueva;
  asegura mismasPelis : (∀i ← [0..|pre(ts)|)pelicula(ts_i) == pelicula(pre(ts)_i);
  asegura ticketsSiguenUsados : (∀i ← [0..|pre(ts)|)usado(ts_i) == usado(pre(ts)_i);
  asegura mismasSalas : (∀i ← [0..|pre(ts)|, sala(pre(ts)_i ≠ vieja)sala(ts_i) == sala(pre(ts)_i);
}

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 espectadoresNoNegativos : (∀s ← salas(c))espectadores(c, s) ≥ 0;
  invariante losTicketsVendidosEstanSinUsar : (∀t ← ticketsVendidosSinUsar(c))¬usado(t);
  invariante salasConsistentes : sinRepetidos([ sala(c, peli) | peli ← peliculas(c) ]);
  invariante losTicketsVendidosSonParaPeliculasDelCine : (∀t ← ticketsVendidosSinUsar(c))
     pelicula(t) ∈ peliculas(c) && sala(t) == sala(c, pelicula(t));
}

problema cineVacio (n: String) = result : Cine {
  asegura nombreCineEntradaEsSalida : nombre(res) == n;
  asegura salasCineVacias : salas(res) == [];
}

problema agregarPelicula (c: Cine, p: Pelicula, s: Sala) {
  requiere salaEntradaEnCine : s ∈ salas(c);
  requiere nuevaSalaVacia : ¬((∃pe ← pelicula(c))sala(c, pe) == s);
  requiere nuevaPelicula : p ∉ peliculas(c);
  modifica c;
  asegura peliculasSeMantienen : mismos(peliculas(pre(c) ++ [p], peliculas(c));
  asegura salasSeMantienen : mismos(salas(c), salas(pre(c));
  asegura nuevaPeliEnCorrectaSala : sala(c, p) == s;
  asegura nombresSeMantienen : nombre(c) == nombre(pre(c));
  asegura salaParaTodasPeliculasSeMantiene : (∀p ← peliculas(pre(c)))sala(c, p) == sala(pre(c), p);
  asegura espectadoresSeMantienen : (∀s ← salas(c))espectadores(c, p) == espectadores(pre(c), p);
  asegura noSeVendieronMasTickets : mismos(ticketsVendidosSinUSar(c) == ticketsVendidosSinUSar(pre(c)));
}

problema cerrarSala (c: Cine, s: Sala) {
  requiere salaEsteEnElCine : s ∈ salas(c);
  requiere noSeCierreSalaConTicketsVendidos : (∀t ← ticketsVendidosSinUSar(c))sala(t) ≠ s;
  modifica c;
  asegura salasNuevas : mismos(salas(c), [ sal | sal ← salas(pre(c)), sal ≠ s ]);
  asegura nombreSeMantiene : nombre(c) == nombre(pre(c));
  asegura peliculasSinLasDeSalasCerradas : mismos(peliculas(c), [ peli | peli ← peliculas(pre(c)), peli ≠ peliDeSala(s, c) ]);
  asegura salaParaTodasPeliculasSeMantiene : (∀p ← peliculas(pre(c)))sala(c, p) == sala(pre(c), p);
  asegura espectadoresSeMantienen : (∀s ← salas(c))espectadores(c, p) == espectadores(pre(c), p);
  asegura noSeVendieronMasTickets : mismos(ticketsVendidosSinUsar(c), ticketsVendidosSinUsar(pre(c)));
}

problema cerrarSalas (c: Cine, e: ℤ) {
  requiere e ≥ 0;
  modifica c;
  asegura seParecen(c, pre(c), e);
}
```

problema cerrarSalasDeLaCadena (cs: [Cine], e: $\mathbb{Z}$) {
  requiere $e \geq 0$;
  modifica $cs$;
  asegura mismaListaDeCines : $long(cs) == long(pre(cs))$;
  asegura cinesSeParecen : $(\forall c' \leftarrow pre(cs))(\exists c \leftarrow cs) seParecen(c, c', e)$;
}

problema pelicula (c: Cine, s: Sala) = result : Pelicula {
  requiere salaEnElCine : $s \in salas(c)$;
  requiere todasSalasConPeli : $(\exists p \leftarrow peliculas(c)) sala(c, p) == s$;
  asegura peliResultadoEstaEnElCine : $result \in peliculas(c)$;
  asegura aLaPeliResultadoLeCorrespondeSalaEntrada : $sala(c, result) == s$;
}

problema venderTicket (c: Cine, p: Pelicula) = result : Ticket {
  requiere peliculaEnElCine : $p \in peliculas(c)$;
  modifica $c$;
  asegura sumoTicketANoUsados : $mismos(ticketsVendidosSinUsar(pre(c) +\!\!+ [result], ticketsVendidosSinUsar(pre(c))$;
  asegura noCambiaNombreDelCine : $nombe(c) == nombre(pre(c))$;
  asegura salasDelCineNoCambian : $mismos(salas(pre(c)), salas(c))$;
  asegura peliculasDelCineNoCambian : $mismos(peliculas(pre(c)), peliculas(c))$;
  asegura salasDeLasPelisNoCambian : $(\forall pe \leftarrow peliculas(pre(c)) sala(pre(c), pe) == sala(c, pe)$;
  asegura espectadoresDeLasSalasNoCambian : $(\forall sa \leftarrow salas(pre(c)) espectadores(pre(c), pe) == espectadores(c, pe)$;
}

problema ingresarASala (c: Cine, s: Sala, t: Ticket) {
  requiere salaEntradaEnCine : $s \in salas(c)$;
  requiere salaTicketIgualEntrada : $salta(t) == s$;
  requiere ticketEstaEntreLosNoUsados : $t \in ticketsVendidosSinUsar(c)$;
  modifica $c$;
  modifica $t$;
  asegura peliculaDelTicketNoCambia : $pelicula(t) == pelicula(pre(t))$;
  asegura salaDelTicketNoCambia : $sala(t) == sala(pre(t))$;
  asegura marcarElTicketComoUsado : $usado(t)$;
  asegura sumarUnEspectador : $espectadores(c, s) == espectadores(pre(c), s) + 1$;
  asegura sacoElTicketDeSinUsar : $mismos(ticketsVendidosSinUsar(c), [\, ti \mid ti \leftarrow ticketsVendidosSinUsar(pre(c)),$
    $ti \neq t\,])$;
  asegura salasNoCambian : $mismos(salas(c), salas(pre(c)))$;
  asegura nombreDelCineNoCambia : $nombre(c) == nombre(pre(c))$;
  asegura peliculasDelCineNoCambia : $mismos(peliculas(c), peliculas(pre(c)))$;
  asegura pelisNoCambianSala : $(\forall p \leftarrow peliculas(pre(c))) sala(c, p) == sala(pre(c), p)$;
  asegura espectadoresNoCambianDeOtrasSalas : $(\forall sa \leftarrow salas(pre(c)), sa \neq s) espectadores(c, sa) == espectadores(pre(c), sa)$
}

problema pasarA3DUnaPelicula (c: Cine, nombre: String) = result : Pelicula {
  requiere ;
  modifica $c$;
  asegura ;
  asegura ;
  asegura ;
  asegura ;
  asegura ;
  asegura ;
  asegura ;
  asegura ;
  asegura ;
  asegura ;
  asegura ;
}

# 1. Auxiliares

aux `obtenerGeneros` (ps: [Pelicula]) : [Genero] $= [\, genero(p) \mid p \leftarrow ps \,]$ ;
aux `obtenerGenerosDupla` (x: [(Genero,[Pelicula])]) : [Genero] $= [\, prm(dupla) \mid dupla \leftarrow x \,]$ ;
aux `generosOrd` (gs: [Genero]) : Bool $= (\forall i \leftarrow [0..|gs|, i \neq |gs|) ord(gs_i) \leq ord(gs_{i+1})$ ;
aux `actoresOrd` (as: [Actores]) : Bool $= (\forall i \leftarrow [0..|as|, i \neq |gs|) ord(as_i) \leq ord(as_{i+1})$ ;
aux `pelisConExito` (c: Cine, e: $\mathbb{Z}$) : [Pelicula] $= [peli | sala \leftarrow salasConExito(c,e), peli \leftarrow peliculas(c), sala(c,peli) == sala]$ ;
aux `nombresDePeliculas` (c: Cine) : [String] $= [nombre(p) | p \leftarrow peliculas(c)]$ ;
aux `tienePeli` (c: Cine, s: Sala) : Bool $= (\exists p \leftarrow pelicula(c)) sala(c,p) = s$ ;
aux `seParecen` (c: Cine, c‘: Cine, e: $\mathbb{Z}$) : Bool $=$
$nombresIguales(c, c‘)\&\&$
$mismasSalas(c, c‘, e)\&\&$
$mismasPeliculas(c, c‘)\&\&$
$pelisEnMismasSalas(c, c‘)\&\&$
$mismosEspectadores(c, c‘)\&\&$
$mismosTickets(c, c‘)$ ;
aux `nombresIguales` (c: Cine, c‘: Cine) : Bool $= nombre(c) == nombre(c‘)$ ;
aux `mismasSalas` (c: Cine, c‘: Cine, e: $\mathbb{Z}$) : Bool $= mismos(salas(c), salasSinTicketsVendidosConMasDe(c‘, e))$ ;
aux `mismasPeliculas` (c: Cine, c‘: Cine, e: $\mathbb{Z}$) : Bool $= mismas(peliculas(c), pelisDeSalas(c‘, salasConMasDe(c‘, e))$ ;
aux `pelisEnMismasSalas` (c: Cine, c‘: Cine) : Bool $= (\forall p \leftarrow peliculas(c)) sala(c,p) == sala(c‘, p)$ ;
aux `mismosEspectadores` (c: Cine, c‘: Cine) : Bool $= (\forall sala \leftarrow salas(c)) espectadores(c, sala) == espectadores(c‘, sala)$ ;
aux `mismosTickets` (c: Cine, c‘: Cine) : Bool $= mismos(ticketsVendidosSinUsar(c), ticketsVendidosSinUsar(c‘))$ ;
aux `nombrePelisDeCine` (c: Cine) : [String] $= [\, nombre(p) \mid p \leftarrow peliculas(c) \,]$ ;
aux `nombrePelisDeTicketsDeCine` (c: Cine) : [String] $= [\, nombre(pelicula(t)) \mid t \leftarrow ticketsVendidosSinUsar(c) \,]$ ;
aux `películaDeNombre` (c: Cine, n: String) : Pelicula $= \mathsf{cab}[\, peli \mid peli \leftarrow peliculas(c), nombre(peli) == n \,]$ ;