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

Algoritmos y Estructura de Datos I

Segundo cuatrimestre de 2012

29 de Octubre de 2012

Especificación del TPI - Hoteles Imperativos 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 Fecha = ℤ;
tipo DNI = ℤ;
tipo Dinero = ℤ;
tipo Cadena = String;
tipo Nombre = String;
tipo Provincia = String;
tipo CheckIn = (DNI, Fecha);
tipo CheckOut = (DNI, Fecha);
tipo TipoHabitacion = Simple, Doble, Triple, Cuadruple;
tipo Accesorio = Jacuzzi, LCD, PS3, DVD, Pelotero, Inflable;
```

3. Reserva

```
tipo Reserva {
  observador documento (r: Reserva) : DNI;
  observador fechaDesde (r: Reserva) : Fecha;
  observador fechaHasta (r: Reserva) : Fecha;
  observador tipo (r: Reserva) : TipoHabitacion;
  observador confirmada (r: Reserva) : Bool;

  invariante NoAntesDeDespues : fechaHasta(r) > fechaDesde(r);
}

problema Reserva (this: Reserva,d: DNI, fd, fh: Fecha, t: TipoHabitacion) {
  requiere fh > fd;
  modifica this;
  asegura confirmada(this) == False;
  asegura fechaDesde(this) == fd;
  asegura fechaHasta(this) == fh;
  asegura documento(this) == d;
  asegura tipo(this) == t;
}

problema documento (this: Reserva) = result : DNI {
  asegura result == documento(this);
```

}

problema fechaDesde (this: Reserva) = result : Fecha {
  asegura $result == fechaDesde(this)$;
}

problema fechaHasta (this: Reserva) = result : Fecha {
  asegura $result == fechaHasta(this)$;
}

problema tipo (this: Reserva) = result : tipoHabitacion {
  asegura $result == tipo(this)$;
}

problema confirmada (this: Reserva) = result : Bool {
  asegura $result == confirmada(this)$;
}

problema confirmar (this: Reserva) {
  modifica $this$;
  asegura $confirmada(this) == \text{True}$;
  asegura $fechaDesde(this) == fechaDesde(pre(this))$;
  asegura $fechaHasta(this) == fechaHasta(pre(this))$;
  asegura $documento(this) == documento(pre(this))$;
  asegura $tipo(this) == tipo(pre(this))$;
}

problema operator == (this,r: reserva) = result : Bool {
  asegura $result == (confirmada(this) == confirmada(r) \wedge fechaDesde(this) == fechaDesde(r) \wedge fechaHasta(this) == fechaHasta(r) \wedge documento(this) == documento(r) \wedge tipo(this) == tipo(r))$;
}

4. Habitación

tipo Habitacion {
  observador numero (h: Habitacion) : $\mathbb{Z}$;
  observador tipo (h: Habitacion) : TipoHabitacion;
  observador accesorios (h: Habitacion) : [Accesorio];

  invariante sinAccesoriosRepetidos : $sinRepetidos(accesorios(h))$;
  invariante accesoriosOrdenada : $ordenada(accesorios(h))$;
}

problema Habitacion (this: Habitacion,n: $\mathbb{Z}$, t: TipoHabitacion, as: [Accesorio]) {
  requiere sinAccesoriosRepetidos : $sinRepetidos(as)$;
  requiere accesoriosOrdenada : $ordenada(as)$;
  modifica $this$;
  asegura $numero(this) == n$;
  asegura $tipo(this) == t$;
  asegura $accesorios(this) == as$;
}

problema numero (this: habitacion) = result : $\mathbb{Z}$ {
  asegura $numero(this) == result$;
}

problema tipo (this: habitacion) = result : tipoHabitacion {
  asegura $tipo(this) == result$;
}

problema accesorios (this: habitacion) = result : [Accesorio] {
  asegura $accesorios(this) == result$;
}

```
problema operator == (this, h: habitacion) = result : Bool  {
  asegura result == (numero(this) == numero(h) ∧ tipo(this) == tipo(h) ∧ accesorios(this) == accesorios(h)) ;
}

problema pretencionesDePopStar (as: [Accesorio], hs: [Habitacion]) = result : [Habitacion]  {
  asegura (∀r ← result)r ∈ hs ;
  asegura (∀r ← result)tieneMasAccesoriosQueTodas(r, as, hs) ;
  asegura (∀h ← hs, tieneMasAccesoriosQueTodas(h, as, hs))h ∈ result ;
  aux cantidadDeAccesoriosEnHabitacion (as: [Accesorio], h: Habitacion) : ℤ = |[ a | a ← as, a ∈ accesorios(h) ]| ;
  aux tieneMasAccesoriosQueTodas (h: Habitacion, as: [Accesorio], hs: [Habitacion]) : Bool =
      (∀x ← hs)cantidadDeAccesoriosEnHabitacion(as, x) ≤ cantidadDeAccesoriosEnHabitacion(as, h) ;
}
```

5. Hotel

```
tipo Hotel {
  observador nombre (h: Hotel) : Nombre ;
  observador cadena (h: Hotel) : Cadena ;
  observador huespedes (h: Hotel) : [DNI] ;
  observador habitaciones (h: Hotel) : [Habitacion] ;
  observador ingresos (h: Hotel) : [(CheckIn, Habitacion)] ;
  observador salidas (h: Hotel) : [CheckOut] ;
  observador reservas (h: Hotel) : [Reserva] ;
  observador tarifaHabitacionXDia (h: Hotel) : [(TipoHabitacion, Dinero)] ;
  observador precioAccesorio (h: Hotel) : [(Accesorio, Dinero)] ;
  invariante habitacionesValidas : (∀c ← ingresos(h))snd(c) ∈ habitaciones(h) ;
  invariante cuantosHay : |huespedes(h)| == |ingresos(h)| − |salidas(h)| ;
  invariante sinHuespedesRepeditos : sinRepetidos(huespedes(h)) ;
  invariante siEstanNoSeFueron : ∀d ← huespedes(h)|ingresosDe(h, d)| == |salidasDe(h, d)| + 1 ;
  invariante siSeVaEntro : (∀o ← salidas(h))existeUnIngresoSinSalida(h, o) ;
  invariante estanAlMenosUnDia : (∀o ← salidas(h))¬existeUnIngresoElMismoDia(h, dniCheckOut(o), fechaCheckOut(o)) ;
  invariante noEntranDosVeces : (∀i ← ingresos(h))existeSoloUnIngresoElMismoDia(h, dniCheckIn(i), fechaCheckIn(i)) ;
  invariante reservasValidas : (∀r ← reservas(h))existeUnaHabitacionDelTipo(h, tipo(r)) ;
  invariante sinTarifasRepetidas : sinRepetidos([prm(t)|t ← tarifaHabitacionXDia(h)]) ;
  invariante sinPreciosRepetidos : sinRepetidos([prm(p)|p ← precioAccesorio(h)]) ;
  invariante tarifasPositivas : (∀t ← tarifaHabitacionXDia(h))snd(t) > 0 ;
  invariante preciosPositivos : (∀p ← precioAccesorio(h))snd(p) > 0 ;
  invariante numerosUnicos : (∀h1 ← habitaciones(h))|[h2|h2 ← habitaciones(h), numero(h1) == numero(h2)]| == 1 ;
  invariante noValeAcaparar : (∀r1, r2 ← reservas(h), tipo(r1) == tipo(r2) ∧ documento(r1) == documento(r2) ∧
      fechaDesde(r1) == fechaDesde(r2))r1 == r2 ;
  invariante noVuelveElMismoDia : ¬(((∃ci ← ingresos(h))(∃co ← salidas(h))
      dniCheckIn(prm(ci)) == dniCheckOut(co) ∧ fechaCheckIn(prm(ci)) == fechaCheckOut(co))) ;
  invariante preciosSeCorresponden : (∀h′ ← habitaciones(h), a ← accesorios(h′))
      (∃p ← prcioAccesorio(h))a == prm(p) ;
  invariante tiposDeHabSeCorresponden : (∀h′ ← habitaciones(h))(∃p ← tarifaHabitacionXDia(h))tipo(h′) == prm(p) ;
  invariante sinRepetidos(reservas(h)) ;
}

problema Hotel (this: Hotel,n: Nombre, c: Cadena, hs: [Habitacion], thxd: [(TipoHabitacion, Dinero)], pa: [(Accesorio,
Dinero)] ) {
  requiere sinTarifasRepetidas : sinRepetidos([ prm(t) | t ← thxd ]) ;
  requiere sinPreciosRepetidos : sinRepetidos([ prm(p) | p ← pa ]) ;
  requiere tarifasPositivas : (∀t ← thxd)snd(t) > 0 ;
  requiere preciosPositivos : (∀p ← pa)snd(t) > 0 ;
  requiere sinHabsRepetidas : sinRepetidos(hs) ;
  requiere preciosSeCorresponden : (∀h ← hs, a ← accesorios(h))(∃p ← pa)a == prm(p) ;
  requiere tiposDeHabSeCorresponden : (∀h ← hs)(∃p ← thxd)tipo(h) == prm(p) ;
  modifica this ;
  asegura nombre(this) == n ;
  asegura cadena(this) == c ;
  asegura mismos(habitaciones(this), hs) ;
  asegura |reservas(this)| == 0 ;
  asegura |huespedes(this)| == 0 ;
  asegura |ingresos(this)| == 0 ;
  asegura |salidas(this)| == 0 ;
  asegura mismos(tarifaHabitacionXDia(this), thxd) ;
  asegura mismos(precioAccesorio(this), pa) ;
}

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

problema cadena (this: Hotel) = result : Cadena {
  asegura cadena(this) == result ;
}
```

```
problema huespedes (this: Hotel) = result : [DNI]  {
  asegura mismos(huespedes(this), result);
}

problema habitaciones (this: Hotel) = result : [Habitacion]  {
  asegura mismos(habitaciones(this), result);
}

problema ingresos (this: Hotel) = result : [(checkIn, Habitacion)]  {
  asegura mismos(ingresos(this), result);
}

problema salidas (this: Hotel) = result : [CheckOut]  {
  asegura mismos(salidas(this), result);
}

problema reservas (this: Hotel) = result : [Reserva]  {
  asegura mismos(reservas(this), result);
}

problema tarifaXDia (this: Hotel, t: TipoHabitacion) = result : Dinero  {
  requiere (∃p ← tarifaHabitacionXDia(this))prm(p) == t;
  asegura result == cab([ snd(p) | p ← tarifaHabitacionXDia(this), prm(p) == t ]);
}

problema precioAccesorio (this: Hotel, a: Accesorio) = result : Dinero  {
  requiere (∃p ← precioAccesorio(this))prm(p) == a;
  asegura result == cab([ snd(p) | p ← precioAccesorio(this), prm(p) == a ]);
}

problema vender (this: Hotel, c: Cadena)  {
  modifica this;
  asegura cambioDeCadena(pre(this), this, c);
}

problema hacerReserva (this: Hotel, r: Reserva)  {
  requiere existeUnaHabitacionDelTipo(this, tipo(r));
  requiere noAcapara : (∀r' ← reservas(this), tipo(r) == tipo(r'),
      documento(r) == documento(r'), fechaDesde(r) == fechaDesde(r'))r == r';
  requiere ¬(∃r' ← reservas(this))r == r';
  modifica this;
  asegura nombre(pre(this)) == nombre(this);
  asegura cadena(pre(this)) == cadena(this);
  asegura mismos(huespedes(pre(this)), huespedes(this));
  asegura mismos(habitaciones(pre(this)), habitaciones(this));
  asegura mismos(ingresos(pre(this)), ingresos(this));
  asegura mismos(salidas(pre(this)), salidas(this));
  asegura mismos(r : reservas(pre(this)), reservas(this));
  asegura mismos(tarifaHabitacionXDia(pre(this)), tarifaHabitacionXDia(this));
  asegura mismos(precioAccesorio(pre(this)), precioAccesorio(this));
}

problema operator == (this, h: Hotel) = result : Bool  {
  asegura result ==
      (nombre(this) == nombre(h) ∧ cadena(this) == cadena(h) ∧ mismos(huespedes(this), huespedes(h)) ∧
      mismos(habitaciones(this), habitaciones(h)) ∧ mismos(ingresos(this), ingresos(h)) ∧
      mismos(salidas(this), salidas(h)) ∧ mismos(tarifaHabitacionXDia(this), tarifaHabitacionXDia(h)) ∧
      mismos(reservas(this), reservas(h)) ∧ mismos(precioAccesorio(this), precioAccesorio(h)));
}

problema sobreReservado (this: Hotel, f: Fecha) = result : Bool  {
  asegura result == capacidad(this) < plazasReservadas(reservasXFecha(this, f));
  aux reservasXFecha (h: Hotel, f: Fecha) : [Reserva] = [ r | r ← reservas(h), fechaDesde(r) ≤ f ≤ fechaHasta(r) ];
  aux plazasReservadas (rs: [Reserva]) : ℤ = sum([ cantidadHuespedes(tipo(r)) | r ← rs ]);
}
```

problema registrarHuesped (this: Hotel, d: DNI, f:Fecha, a: Habitacion) {
  requiere $a \in habitaciones(this)$;
  requiere $d \notin huespedes(this)$;
  requiere habitacionLibre : $(\forall i \in ingresos(this), snd(i) == a)\ tieneCheckout(this, prm(i))$;
  requiere $existeReserva(this, d, f, tipo(a), false)$;
  requiere superaUltimoCheckout :
    $|salidasDe(this, d)| > 0 \longrightarrow max([\, fechaCheckOut(co) \,|\, co \leftarrow salidasDe(this, d) \,]) < f$;

  modifica $this$;
  asegura $nombre(pre(this)) == nombre(this)$;
  asegura $cadena(pre(this)) == cadena(this)$;
  asegura $mismos(habitaciones(pre(this)), habitaciones(this))$;
  asegura $mismos(d : huespedes(pre(this)), huespedes(this))$;
  asegura $mismos(((d, f), a) : ingresos(pre(this)), ingresos(this))$;
  asegura $mismos(salidas(pre(this)), salidas(this))$;
  asegura $|reservas(pre(this))| == |reservas(this)|$;
  asegura $|reservasConfirmadas(pre(this))| == |reservasConfirmadas(this)| - 1$;
  asegura $|reservasSinConfirmar(pre(this))| == |reservasSinConfirmar(this)| + 1$;
  asegura $mismos(tarifaHabitacion(pre(this)), tarifaHabitacion(this))$;
  asegura $mismos(precioAccesorio(pre(this)), precioAccesorio(this))$;
  asegura otrasReservasIguales : $(\forall r \in reservas(pre(this)), r \neq dameReserva(pre(this), d, f, tipo(a), false)) r \in reservas(this$
  asegura $existeReserva(this, d, f, tipo(a), true)$;

  aux existeReserva (h: Hotel, d: DNI, f: Fecha, t: TipoHabitacion, c: Bool) : Bool =
    $(\exists r \leftarrow reservas(h)) documento(r) == d \wedge fechaDesde(r) == f \wedge tipo(r) == t \wedge confirmada(r) == c$;
  aux dameReserva (h: Hotel, d: DNI, f: Fecha, t: Tipo, c: Bool) : Reserva =
    $cab([r | r \leftarrow reservas(h), documento(r) == d, fechaDesde(r) == f, tipo(r) == t, confirmada(r) == c])$;
  aux tieneCheckOut (h: Hotel, ci : CheckIn) : Bool =
    $(\exists co \in salidasDe(h, dniCheckIn(ci)), fechaCheckIn(ci) < fechaCheckOut(co))$
    $\neg(\exists ci' \in ingresosDe(h, dniCheckIn(ci)), ci \neq ci') fechaCheckIn(ci) < fechaCheckIn(ci') < fechaCheckOut(co)$;
  aux reservasConfirmadas (h: Hotel) : [Reserva] $= [\, r \,|\, r \leftarrow rs, confirmada(r) \,])$;
  aux reservasSinConfirmar (h: Hotel) : [Reserva] $= [\, r \,|\, r \leftarrow rs, \neg confirmada(r) \,])$;
}

problema desRegistrarHuesped (this: Hotel, d: DNI, f:Fecha) {
  requiere $d \in huespedes(this)$;
  requiere $fechaMayorAlUltimoIngreso(f, d, this)$;

  modifica $this$;
  asegura $nombre(pre(this)) == nombre((this))$;
  asegura $cadena(pre(this)) == cadena((this))$;
  asegura $mismos(habitaciones(pre(this)), habitaciones((this)))$;
  asegura $mismos(huespedes((this)), sacar(d, huespedes(pre(this))))$;
  asegura $mismos(ingresos(pre(this)), ingresos((this)))$;
  asegura $mismos((d, f) : salidas(pre(this)), salidas((this)))$;
  asegura $mismos(reservas(pre(this)), reservas((this)))$;
  asegura $mismos(tarifaHabitacion(pre(this)), tarifaHabitacion((this)))$;
  asegura $mismos(precioAccesorio(pre(this)), precioAccesorio((this)))$;

  aux sacar (x: T, xs: [T]) : [T] $= [\, k \,|\, k \leftarrow xs, k \neq x \,]$;
  aux fechaMayorAlUltimoIngreso (f: Fecha, d: DNI, h: Hotel) : Bool $= f > max([\, fechaCheckIn(c) \,|\, c \leftarrow ingresosDe(h, d) \,])$;
}

problema huespedesConPalabra (this: Hotel) = result : [DNI] {
  asegura $mismos(result, sacarRepetidos(buscarHuespedesDePalabra(this)))$;

  aux buscarHuespedesDePalabra (h: Hotel) : [DNI] $= [\, dniCheckOut(o) \,|\, i \leftarrow ingresos(h),$
    $o \leftarrow salidasDe(h, dniCheckIn(prm(i))), noHaySalidaEnElMedio(prm(i), o, h),$
    $existeReserva(h, dniCheckIn(prm(i)), fechaCheckIn(prm(i)), fechaCheckOut(o), tipo(sgd(i)), true) \,]$;
  aux existeReserva (h: Hotel, d: DNI, fd: Fecha, fh: Fecha, t: TipoHabitacion, c: Bool) : Bool =
    $(\exists r \leftarrow reservas(h)) documento(r) == d \wedge fechaDesde(r) == fd \wedge tipo(r) == t \wedge fechaHasta(r) == fh \wedge confirmada(r) ==$
    c;
}

problema calcularCuenta (this: Hotel, i: CheckIn, o: CheckOut, hb: Habitacion) = result : Dinero {
  requiere $dniCheckIn(i) == dniCheckOut(o)$;

```
  requiere fechaCheckIn(i) < fechaCheckOut(o);
  requiere (i, hb) ∈ ingresos(this);
  requiere o ∈ salidas(this);
  requiere noHaySalidaEnElMedio(i, o, this);

  asegura result == precioXAccesorios(accesorios(hb), this) + (tarifaXHabitacion(this, tipo(hb)) * dias(i, o));

  aux precioXAccesorios (as: [Accesorio], h: Hotel) : Dinero =
      sum([ snd(x) | a ← as, x ← precioAccesorio(h), prm(x) == a ]);
  aux dias (i: CheckIn, o: CheckOut) : ℤ = fechaCheckOut(o) − fechaCheckIn(i);
  aux tarifaXHabitacion (h: Hotel, t: TipoHabitacion) : Dinero = [snd(d)|d ← tarifaHabitacionXDia(h), prm(d) ==
      t]_0;
}

problema reservasSolapadas (this: Hotel, d: DNI) = result : Bool {
  asegura result == (∃r ← reservas(this), documento(r) == d)seSolapa(this, r);
  aux seSolapa (h: Hotel, r: Reserva) : Bool = (∃r' ← reservas(h), documento(r) == documento(r'), r ≠ r')
      entreFechas(fechaDesde(r), fechaHasta(r), fechaDesde(r')) ∨
      entreFechas(fechaDesde(r), fechaHasta(r), fechaHasta(r'));
  aux entreFechas (fd, fh, f: Fecha) : Bool = fd <= f <= fh;
}
```

6. MinisterioDeTurismo

```
tipo MinisterioDeTurismo {
  observador secretarias (m: MinisterioDeTurismo) : [Provincia] ;
  observador registro (m: MinisterioDeTurismo, p: Provincia ) : [Hotel] ;
       requiere p ∈ secretarias(m) ;
  observador cadenasDeHoteles (m: MinisterioDeTurismo) : [[Hotel]] ;
  invariante sinHotelesRepetidas : (∀xs ← cadenasDeHoteles(m))sinRepetidosH(xs) ;
  invariante cadenasConHoteles : (∀xs ← cadenasDeHoteles(m))|xs| > 0 ;
  invariante sinCadenasRepetidas : sinRepetidos([ cadena(xs_0) | xs ← cadenasDeHoteles(m) ]) ;
  invariante sinProvinciasRepetidas : sinRepetidos(secretarias(m)) ;
  invariante cadenasBienFormadas : (∀xs ← cadenasDeHoteles(m))sonDeLaMismaCadena(xs) ;
  invariante sinNombresRepetidosEnCadenas : (∀xs ← cadenasDeHoteles(m))sinRepetidos(todosLosNombres(xs)) ;
  invariante hotelesConsistentes : mismosHoteles(aplanar(cadenasDeHoteles(m)), todosLosHoteles(m)) ;
}

problema MinisterioDeTurismo (this: MinisterioDeTurismo,ps: [Provincia]) {
  requiere sinRepetidos(ps) ;
  modifica this ;
  asegura mismos(secretarias(this), ps) ;
  asegura |cadenasDeHoteles(this)| == 0 ;
  asegura (∀p ← ps)|registro(this, p)| == 0 ;
}

problema secretarias (this: MinisterioDeTurismo) = result : [Provincia] {
  asegura mismos(secretarias(this), result) ;
}

problema registro (this: MinisterioDeTurismo, p: Provincia) = result : [Hotel] {
  requiere p ∈ secretarias(this) ;
  asegura mismosHoteles(registro(this, p), result) ;
}

problema cadenasDeHoteles (this: MinisterioDeTurismo) = result : [[Hotel]] {
  asegura |cadenasDeHoteles(this)| == |result| ;
  asegura (∀hs ← result)(∃hs' ← cadenasDeHoteles(this))mismosHoteles(hs, hs') ;
  asegura (∀hs ← cadenasDeHoteles(this))(∃hs' ← result)mismosHoteles(hs, hs') ;
}

problema agregarHotel (this: MinisterioDeTurismo, h: Hotel, p: Provincia) {
  requiere p ∈ secretarias(this) ;
  requiere ¬(∃h' ← todosLosHoteles(this))hotelesIguales(h, h') ;
  requiere sinNombresRepetidosEnCadenas : sinRepetidos(todosLosNombres(h : hotelesXCadena(cadena(h), this))) ;
  modifica this ;
  asegura mismos(secretarias(pre(this)), secretarias(this)) ;
  asegura registrosIgualesSalvoP : (∀p' ← secretarias(pre(this)), p' ≠ p)
     hotelesIguales(registro(pre(this), p'), registro(this, p')) ;
  asegura PtieneAH : mismosHoteles(registro(this, p), h : registro(pre(this), p)) ;
  aux hotelesXCadena (c: Cadena, m: MinisterioDeTurismo) : [Hotel] =
     [ h | hs ← cadenasDeHoteles(m), h ← hs, cadena(h) == c ]) ;
}

problema operator == (this, m: MinisterioDeTurismo) = result : Bool {
  asegura result ==
     (mismos(provincias(this), provincias(m)) ∧ (∀p ← provincias(m))mismos(registro(m, p), registro(this, p))) ;
}

problema cadenasAmarretas (this: MinisterioDeTurismo) = result : [Cadena] {
  asegura mismos(result, cadenasQueEnMenosProvinciasEstan(this)) ;
  aux cadenasQueEnMenosProvinciasEstan (m: MinisterioDeTurismo) : [Cadena] =
     [ c | c ← todasLasCadenas(m), estaEnCantidadMinima(c, m) ] ;
  aux estaEnCantidadMinima (c: Cadena, m: MinisterioDeTurismo) : Bool =
     (∀c2 ← todasLasCadenas(m))cantidadDeProvinciasConPresencia(c, m) ≤ cantidadDeProvinciasConPresencia(c2, m) ;
```

```
  aux cantidadDeProvinciasConPresencia (c: Cadena, m: MinisterioDeTurismo) : ℤ =
      |[p | p ← secretarias(m), (∃h ← registro(m, p))cadena(h) == c]| ;
}

problema fusionAutorizada (this: MinisterioDeTurismo, c1: Cadena, c2: Cadena) {
  requiere cadenasRegistradas : existeCadena(c1, this) ∧ existeCadena(c2, this) ;
  requiere fusionPosible :
      sinRepetidos(todosLosNombres(hotelesXCadena(c1, this)) + +todosLosNombres(hotelesXCadena(c2, this))) ;

  modifica this ;

  asegura mismasProvincias : mismos(secretarias(this), secretarias(pre(this))) ;
  asegura noSePerdieronRegistros : (∀p ← secretarias(pre(this)))|registro(pre(this), p)| == |registro(this, p)| ;
  asegura noSePerdieronHoteles : |todosLosHoteles(pre(this))| == |todosLosHoteles(this)| ;
  asegura losOtrosHotelesNoCambiaron :
      mismosHoteles([ h | h ← todosLosHoteles(pre(this)), cadena(h) ≠ c1 ∧ cadena(h) ≠ c2 ],
      [ h | h ← todosLosHoteles(this), cadena(h) ≠ c1 ∧ cadena(h) ≠ c2 ]) ;
  asegura noPerdimosCadenas : |todasLasCadenas(pre(this))| == |todasLasCadenas(this)| + 1 ;
  asegura otrasCadenas :
      (∀c ← todasLasCadenas(pre(this)), c ≠ c1∧c ≠ c2)mismosHoteles(hotelesXCadena(c, pre(this)), hotelesXCadena(c, this)
  asegura chauC2 : |hotelesXCadena(c2, this)| == 0 ;
  asegura holaC1yC2 : |hotelesXCadena(c1, this)| == |hotelesXCadena(c1, pre(this))++hotelesXCadena(c2, pre(this))| ;
  asegura cambioDeCadena :
      (∀h ← todosLosHoteles(pre(this)), cadena(h) == c2)∃h2 ← todosLosHoteles(this)cambioDeCadena(h, h2, c1) ;
  asegura noCambioDeCadena : (∀h ← todosLosHoteles(pre(this)), cadena(h) == c1)
      (∃h2 ← todosLosHoteles(this))hotelesIguales(h2, h) ;
  asegura hotelesQueExistenMantienenProvincia : (∀s ← secretarias(pre(this)))
      (∀h ← registro(pre(this), s), cadena(h) ≠ c2)
      (∃h′ ← registro(this, s))hotelesIguales(h, h′) ;
  asegura hotelesQueCambiaronMantienenProvincia : (∀s ← secretarias(pre(this)))
      (∀h ← registro(pre(this), s), cadena(h) == c2)
      (∃h′ ← registro(this, s))cambioDeCadena(h, h′, c1) ;

  aux existeCadena (c: Cadena, m: MinisterioDeTurismo) : Bool =
      (∃hs ← cadenasDeHoteles(m))(∃h ← hs)cadena(h) == c ;
  aux hotelesXCadena (c: Cadena, m: MinisterioDeTurismo) : [Hotel] =
      [ h | hs ← cadenaDeHoteles(m), h ← hs, cadena(h) == c ] ;
  aux provinciasXCadena (c: Cadena, m: MinisterioDeTurismo) : [Provincia] =
      sacarRepetidos([ p | p ← secretarias(m), (∃h ← registro(m, p))cadena(h) == c ]) ;
}
```

7. Auxiliares

```
aux dniCheckIn (c: CheckIn) : DNI = prm(c) ;

aux dniCheckOut (c: CheckOut) : DNI = prm(c) ;

aux fechaCheckIn (c: CheckIn) : Fecha = snd(c) ;

aux fechaCheckOut (c: CheckOut) : Fecha = snd(c) ;

aux ingresosDe (h: Hotel, d: DNI) : [CheckIn] = [x|x ← ingresos(h), dniCheckIn(prm(x)) == d] ;

aux salidasDe (h: Hotel, d: DNI) : [CheckOut] = [x|x ← salidas(h), dniCheckOut(prm(x)) == d] ;

aux noHaySalidaEnElMedio (i: CheckIn, o: CheckOut, h: Hotel) : Bool = ¬((∃co ← salidasDe(h, dniCheckOut(o)))
  fechaCheckIn(i) < fechaCheckOut(co) < fechaCheckOut(o)) ;

aux existeUnIngresoSinSalida (h: Hotel, o1: CheckOut) : Bool =
  (∃i ← ingresosDe(h, dniCheckOut(o1)), fechaCheckIn(i) < fechaCheckOut(o1))
  ¬(∃o2 ← salidasDe(h, dniCheckOut(o1)), fechaCheckIn(i) < fechaCheckOut(o2) < fechaCheckOut(o1)) ;

aux existeUnIngresoElMismoDia (h: Hotel, d: DNI, f: Fecha) : Bool =
  (∃i ← ingresosDe(h, d))fechaCheckIn(i) == f ;

aux existeSoloUnIngresoElMismoDia (h: Hotel, d: DNI, f: Fecha) : Bool =
  |[i|i ← ingresosDe(h, d), fechaCheckIn(i) == f]| == 1 ;

aux existeUnaHabitacionDelTipo (h: Hotel, t: TipoHabitacion) : Bool = (∃x ← habitaciones(h))tipo(x) == t ;
```

aux capacidad (h: Hotel) : $\mathbb{Z} = \sum [\, cantidadHuespedes(tipo(h)) \mid h \leftarrow habitaciones(h) \,]$;

aux cantidadHuespedes (t: TipoHabitacion) : $\mathbb{Z} =$
if $t == Simple$ then 1 else (if $t == Doble$ then 2 else (if $t == Triple$ then 3 else 4)) ;

aux sonDeLaMismaCadena (hs: [Hotel]) : Bool $= (\forall h1, h2 \leftarrow hs) cadena(h1) == cadena(h2)$;

aux todosLosHoteles (m: MinisterioDeTurismo) : [Hotel] $= [\, h \mid p \leftarrow secretarias(m), h \leftarrow registro(m, p) \,]$;

aux todosLosNombres (hs: [Hotel]) : [String] $= [\, nombre(h) \mid h \leftarrow hs \,]$;

aux todasLasCadenas (m: MinisterioDeTurismo) : [Cadena] $= sacarRepetidos([\, cadena(h) \mid h \leftarrow todosLosHoteles(m) \,])$;

aux aplanar (xss: [[T]]) : [T] $= [\, x \mid xs \leftarrow xss, x \leftarrow xs \,]$;

aux ordenada (l:[T]) : Bool $= (\forall i \leftarrow [0..|l| - 1)) l_i \leq l_{i+1}$;

aux sinRepetidos (l: [T]) : Bool $= (\forall i, j \leftarrow [0..|l|), i \neq j) l_i \neq l_j$;

aux sinRepetidosH (l: [Hotel]) : Bool $= (\forall i, j \leftarrow [0..|l|), i \neq j) \neg hotelesIguales(l_i, l_j)$;

aux sacarRepetidos (l: [T]) : Bool $= (\forall i \leftarrow [0..|l|)) l_i \notin l_[i + 1..longitud(l) - 1)$;

aux hotelesIguales (h, h' : Hotel) : Bool $= \{$
$nombre(h) == nombre(h') \wedge$
$cadena(h) == cadena(h') \wedge$
$mismos(huespedes(h), huespedes(h')) \wedge$
$mismos(habitaciones(h), habitaciones(h')) \wedge$
$mismos(ingresos(h), ingresos(h')) \wedge$
$mismos(salidas(h), salidas(h')) \wedge$
$mismos(reservas(h), reservas(h')) \wedge$
$mismos(tarifaHabitacionXDia(h), tarifaHabitacionXDia(h')) \wedge$
$mismos(precioAccesorio(h), precioAccesorio(h'))$
$\}$;

aux cambioDeCadena (h1: Hotel, h2:Hotel, c: Cadena) : Bool $= \{nombre(h1) == nombre(h2) \wedge$
$cadena(h2) == c \wedge$
$mismos(huespedes(h1), huespedes(h2)) \wedge$
$mismos(habitaciones(h1), habitaciones(h2)) \wedge$
$mismos(ingresos(h1), ingresos(h2)) \wedge$
$mismos(salidas(h1), salidas(h2)) \wedge$
$mismos(reservas(h1), reservas(h2)) \wedge$
$mismos(tarifaHabitacionXDia(h1), tarifaHabitacionXDia(h2)) \wedge$
$mismos(precioAccesorio(h1), precioAccesorio(h2))$
$\}$;

aux mismosHoteles (hs, hs': [Hotel]) : Bool $= |hs| == |hs'| \wedge$
$(\forall h \leftarrow hs)(\exists h' \leftarrow hs') hotelesIguales(h, h') \wedge$
$(\forall h \leftarrow hs')(\exists h' \leftarrow hs) hotelesIguales(h, h')$;