UNIVERSIDAD DE BUENOS AIRES

Departamento de Computación

Facultad de Ciencias Exactas y Naturales

Algoritmos y Estructura de Datos I

Segundo cuatrimestre de 2012

RTPE De Hoteles y Pasajeros

26 de septiembre de 2012

Grupo Los Simuladores

Integrante	LU	Correo electrónico
Almansi, Emilio Guido	674/12	ealmansi@gmail.com
Vilerino, Silvio Fernando	106/12	svilerino@gmail.com
Chapresto, Matias Nahuel	201/12	matiaschapresto@gmail.com
Erdei, Alan	170/12	alane1993@hotmail.com

Índice

1. Notas y comentarios

1.1. Notas iniciales

Los invariantes de los tipos y los auxiliares del tp realizados por nosotros estarán marcados en color *azul* para una mejor diferenciación con el enunciado.

1.2. Tipo Habitacion

Debido a que la lista de accesorios del observador correspondiente es ordenada (según el invariante accesoriosOrdenada), la igualdad básica == puede ser utilizada para elementos de tipo Habitacion, así como el operador mismos y el auxiliar sinRepetidos.

1.3. Tipo Hotel

El invariante estanAlMenosUnDia utiliza el auxiliar checkInAsociado, tomando la cabeza de una lista (obtenerCheckInsAsociados) que, en principio, podría ser vacía. Sin embargo, el invariante siSeVaEntro garantiza que esta lista nunca lo es (en particular, tiene siempre un elemento). Las listas del tipo Hotel son no ordenadas, por lo cual el operador == no admitiría como iguales a hoteles que conceptualmente sí lo son; por esta razón, se definió el auxiliar hotelesEquivalentes, y el auxiliar mismosHoteles que conlleva la semántica del operador mismos para elementos de este tipo.

1.4. Problema RegistrarHuesped

Para garantizar que luego de la modificación sobre el hotel las reservas son aquellas esperadas, se asegura: la cantidad total de reservas es conservada, aquellas reservas no relacionadas con el nuevo ingreso se mantienen en lugar, y el lugar restante corresponde a la reserva asociada al nuevo ingreso (pero ya confirmada).

1.5. Problema HuespedesConPalabra

El criterio para considerar que un huesped es de palabra es el siguiente: el huesped debe tener al menos una reserva confirmada (de lo contrario, nunca ha ingresado en el Hotel, y aún no ha sido puesto a prueba), y debe haber respetado todas sus reservas confirmadas (una reserva no confirmada puede tener fecha en el futuro, y aún no se sabe si el huesped la respetará o no).

2. Tipos

`tipo` Fecha = $\mathbb{Z}$;
`tipo` DNI = $\mathbb{Z}$;
`tipo` Dinero = $\mathbb{Z}$;
`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` : $fechaHasta(r) > fechaDesde(r)$;
}

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` : $ordenada(accesorios(h))$;
}

4.1. pretensionesDePopStar

`problema pretensionesDePopStar` (as: [Accesorio], hs: [Habitacion]) = `result` : [Habitacion] {

  `asegura` $mismos(result, [hb | hb \leftarrow hs, relativamenteSuperEquipada(as, hs, hb)])$;

  `aux` *relativamenteSuperEquipada* (as: [Accesorio], hs: [Habitacion], hb: Habitacion) : Bool =
    $(\forall ohb \leftarrow hs) cantidadAccesoriosPresentes(as, ohb) \leq cantidadAccesoriosPresentes(as, hb)$;
  `aux` *cantidadAccesoriosPresentes* (as: [Accesorio], hb: Habitacion) : $\mathbb{Z} = |[a | a \leftarrow accesorios(hb), a \in as]|$;
}

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 siEstanNoSeFueron : (∀d ← huespedes(h))|ingresosDe(h, d)| == |salidasDe(h, d)| + 1;
  invariante siSeVaEntro : (∀co ← salidas(h))existeUnicoCheckInAsociado(h, co);
  invariante estanAlMenosUnDia : (∀co ← salidas(h))
    fechaCheckIn(checkInAsociado(h, co)) ≤ fechaCheckOut(co) − 1;
  invariante noEntranDosVeces : sinRepetidos([prm(ingreso)|ingreso ← ingresos(h)]);
  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 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)));
}
```

5.1. sobreReservado

```
problema sobreReservado (h: Hotel, f: Fecha) = result : Bool {
  asegura result == (cantidadPlazasReservadasEnFecha(h, f) > capacidad(h));
  aux cantidadPlazasReservadasEnFecha (h: Hotel, f: Fecha) : ℤ = ∑[cantidadHuespedes(tipo(r)) | r ← reservas(h),
    reservaIncluyeFecha(r, f)];
  aux reservaIncluyeFecha (r: Reserva, f: Fecha) : Bool = ((fechaDesde(r) ≤ f) ∧ (f < fechaHasta(r)));
}
```

5.2. registrarHuesped

```
problema registrarHuesped (h: Hotel, d: DNI, f:Fecha, a: Habitacion) {
  requiere a ∈ habitaciones(h);
  requiere ¬habitacionEstaOcupada(h, d, f, a);
  requiere sinCheckOutPosterior : ¬(∃co ← salidasDe(h, d))f ≤ fechaCheckOut(co);
  requiere siVieneNoEstaba : d ∉ huespedes(h);
  requiere existeReserva : (∃r ← reservas(h))esReservaSinConfirmarAsociadaAlIngreso(r, d, f, a);
  modifica h;
  asegura mismoNombre : nombre(h) == nombre(pre(h));
  asegura mismaCadena : cadena(h) == cadena(pre(h));
  asegura mismasHabitaciones : mismos(habitaciones(h), habitaciones(pre(h)));
  asegura mismasSalidas : mismos(salidas(h), salidas(pre(h)));
  asegura mismasTarifas : mismos(tarifaHabitacionXDia(h), tarifaHabitacionXDia(pre(h)));
  asegura mismosAccesorios : mismos(precioAccesorio(h), precioAccesorio(pre(h)));
  asegura nuevoIngreso : mismos(ingresos(h), ((d, f), a) : ingresos(pre(h)));
  asegura nuevoHuesped : mismos(huespedes(h), d : huespedes(pre(h)));
  asegura igualCantidadDeReservas : |reservas(h)| == |reservas(pre(h))|;
  asegura estanLasDeAntesSalvoLaNoConfirmadaAsociadaAlIngreso : (∀r ← reservas(pre(h)),
    ¬esReservaSinConfirmarAsociadaAlIngreso(r, d, f, a))r ∈ reservas(h);
```

```
  asegura estaLaAsociadaAlIngresoYaConfirmada : (∃r ∈ reservas(h))
      esReservaConfirmadaAsociadaAlIngreso(h, d, f, a) ;

  aux habitacionEstaOcupada (h:Hotel, d: DNI, f: Fecha, a: Habitacion) : Bool =
      (∃i ← ingresos(h), sgd(i) == a)
      ((fechaCheckIn(prm(i)) ≤ f) ∧ noHizoCheckOut(h, d, prm(i), f)) ;
  aux noHizoCheckOut (h: Hotel, d: DNI, ci: CheckIn, f: Fecha) : Bool =
      ¬(∃co ← salidasDe(h, d), fechaCheckOut(co) ≤ f)esCheckInAsociado(h, co, ci) ;
  aux esReservaSinConfirmarAsociadaAlIngreso (h: Hotel, d: DNI, f:Fecha, a: Habitacion) : Bool =
      (documento(r) == d ∧ fechaDesde(r) == f ∧ tipo(r) == a ∧ ¬confirmada(r)) ;
  aux esReservaConfirmadaAsociadaAlIngreso (h: Hotel, d: DNI, f:Fecha, a: Habitacion) : Bool =
      (documento(r) == d ∧ fechaDesde(r) == f ∧ tipo(r) == a ∧ confirmada(r)) ;
}
```

5.3. desRegistrarHuesped

```
problema desRegistrarHuesped (h: Hotel, d: DNI, f:Fecha) {
  requiere estaHospedado : d ∈ huespedes(h) ;
  requiere seQuedoAlMenosUnDia(checkInAsociado(h, checkout(d, f)), checkout(d, f)) ;

  modifica h ;
  asegura mismoNombre : nombre(h) == nombre(pre(h)) ;
  asegura mismaCadena : cadena(h) == cadena(pre(h)) ;
  asegura mismasHabitaciones : mismos(habitaciones(h), habitaciones(pre(h))) ;
  asegura mismosIngresos : mismos(ingresos(h), ingresos(pre(h))) ;
  asegura mismasReservas : mismos(reservas(h), reservas(pre(h))) ;
  asegura mismasTarifas : mismos(tarifaHabitacionXDia(h), tarifaHabitacionXDia(pre(h))) ;
  asegura mismosAccesorios : mismos(precioAccesorio(h), precioAccesorio(pre(h))) ;

  asegura mismos(salidas(h), checkout(d, f) : salidas(pre(h))) ;
  asegura mismos(huespedes(h), listaSinElemento(huespedes(pre(h)), d)) ;

  aux checkout (d: DNI, f: Fecha) : CheckOut = (d, f) ;
  aux seQuedoAlMenosUnDia (ci: CheckIn, co: CheckOut) : Bool = fechaCheckIn(ci) ≤ fechaCheckOut(co) − 1 ;
}
```

5.4. huespedesConPalabra

```
problema huespedesConPalabra (h: Hotel) = result : [DNI] {
  asegura mismos(result, sacarRepetidos(
      [ documento(r) | r ← reservas(h), confirmada(r), respetoSusReservas(h, documento(r)) ]
      )) ;

  aux respetoSusReservas (h: Hotel, d: DNI) : Bool = (∀r ← reservasDe(h, d), confirmada(r))
      reservaFueRespetada(h, r) ;
  aux reservaFueRespetada (h: Hotel, r: Reserva) : Bool =
      (∃ci ← ingresosDe(h, documento(r)), co ← salidasDe(h, documento(r)))
      fechaCheckIn(ci) == fechaDesde(r) ∧ fechaCheckOut(co) == fechaHasta(r) ∧
      esCheckInAsociado(h, co, ci) ;
}
```

5.5. calcularCuenta

```
problema calcularCuenta (h: Hotel, i: CheckIn, o: CheckOut, hb: Habitacion) = result : Dinero {
  requiere (i, hb) ∈ ingresos(h) ;
  requiere esCheckInAsociado(h, o, i) ;

  asegura result == (totalDias(i, o) ∗ precioHabitacion(h, hb) + importePorAccesorios(h, hb)) ;

  aux totalDias (i: CheckIn, o: CheckOut) : ℤ = fechaCheckOut(o) − fechaCheckIn(i) ;
  aux precioDeHabitacion (h: Hotel, hb: Habitacion) : ℤ = cab([ sgd(x) | x ← tarifaHabitacionXDia(h),
      tipo(hb) == prm(x) ]) ;
  aux importePorAccesorios (h: Hotel, hb: Habitacion) : ℤ =
      ∑ [ sgd(ap) | ap ← precioAccesorio(h), prm(ap) ∈ accesorios(h) ] ;
}
```

5.6. reservasSolapadas

```
problema reservasSolapadas (h: Hotel, d: DNI) = result : Bool  {
  asegura result == diasRepetidosEnReservasPorDNI(h, d) ;
  aux diasRepetidosEnReservasPorDNI (h: Hotel, d: DNI) : Bool =
      ¬sinRepetidos(fechasReservadasPorDNI(h, d)) ;
  aux fechasReservadasPorDNI (h: Hotel, d: DNI) : [Fecha] = [ f | r ← reservasDe(h, d),
      f ← [fechaDesde(r)..fechaHasta(r)], ¬confirmada(r) ] ;
}
```

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 sinHotelesRepetidos : (∀xs ← cadenasDeHoteles(m))sinRepetidos(xs) ;
  invariante sinCadenasRepetidas : sinRepetidos([ cadena(xs_0) | xs ← cadenasDeHoteles(m) ]) ;
  invariante sinProvinciasRepetidas : sinRepetidos(secretarias(m)) ;
  invariante cadenasBienFormadas : (∀lh ← cadenasDeHoteles(m))
     ((∀h ← lh)cadena(cab(lh)) == cadena(h)) ;
  invariante sinNombresRepetidosEnCadenas : (∀lh ← cadenasDeHoteles(m))
     sinRepetidos([ nombre(h) | h ← lh ]) ;
  invariante hotelesConsistentes : mismosHoteles(aplanar(cadenasDeHoteles(m)),
     aplanar([ registro(m, p) | p ← secretarias(m) ])) ;
  invariante cadenasConHoteles : (∀xs ← cadenasDeHoteles(m))|xs| > 0 ;
}
```

6.1. cadenasAmarretas

```
problema cadenasAmarretas (m: MinisterioDeTurismo) = result : [Cadena] {
  asegura mismos(result, [ cadena(cab(c)) | c ← cadenaDeHoteles(m), seaCadenaAmarreta(m, c) ]) ;
  aux seaCadenaAmarreta (m: MinisterioDeTurismo, c: [Hotel]) : Bool =
     ¬(∃c2 ← cadenaDeHoteles(m))|provinciasDeCadena(m, c2)| < |provinciasDeCadena(m, c)| ;
}
```

6.2. fusionAutorizada

```
problema fusionAutorizada (m: MinisterioDeTurismo, c1: Cadena, c2: Cadena) {
  requiere cadenaRegistrada(m, c1) ∧ cadenaRegistrada(m, c2) ;
  requiere sonCompatibles(m, c1, c2) ;
  requiere noCompartenProvincias(m, c1, c2) ;
  modifica m ;
  asegura noCambianLasSecretarias : mismos(secretarias(m), secretarias(pre(m))) ;
  asegura sinCambiosIndeseados : (∀p ← secretarias(m))registroModificadoCorrectamente(m, p, c1, c2) ;
  asegura dosCadenasSeConviertenEnUnaSola : |cadenasDeHoteles(m)| == |cadenasDeHoteles(pre(m))| − 1 ;
  asegura estanLasDemasCadenas(m, c1, c2) ;
  asegura estaLaCadenaFusionada(m, c1, c2) ;
  aux registroModificadoCorrectamente (m: Ministerio, p: Provincia, c1,c2: Cadena) : Bool =
     |registro(m, p)| == |registro(pre(m), p)| ∧ hotelesDeseados(m, p, c1, c2)) ;
  aux hotelesDeseados (m: Ministerio, p: Provincia, c1,c2: Cadena) : Bool = (∀h ← registro(pre(m), p))
     if cadena(h) == c2 then hotelAparaceConCadenaCambiada(h, registro(m, p), c1) else
     hotelPertenece(h, registro(m, p)) ;
  aux estanLasDemasCadenas (m: Ministerio, c1,c2: Cadena) : Bool =
     (∀lh ← cadenasDeHoteles(pre(m)), ¬cadena(cab(lh)) == c1
     ∧ ¬cadena(cab(lh)) == c2)cadenaPertenece(lh, cadenasDeHoteles(m)) ;
  aux estaLaCadenaFusionada (m: Ministerio, c1,c2: Cadena) : Bool =
     (∃lh ← cadenasDeHoteles(m))
     (|lh| == |obtenerListaDeLaCadena(pre(m), c1)| + |obtenerListaDeLaCadena(pre(m), c2)|)
     ∧ contieneC1(m, c1, lh) ∧ contieneC2Modificada(m, c1, c2, lh) ;
  aux contieneC1 (m: Ministerio, c1: Cadena, lh: [Hotel]) : Bool =
     (∀h ← obtenerListaDeLaCadena(pre(m), c1))
     hotelPertenece(h, lh) ;
  aux contieneC2Modificada (m: Ministerio, c1,c2: Cadena, lh: [Hotel]) : Bool =
     (∀h ← obtenerListaDeLaCadena(pre(m), c2))
     hotelAparaceConCadenaCambiada(h, lh, c1) ;
  aux hotelAparaceConCadenaCambiada (h: Hotel, hs: [Hotel], c: Cadena) : Bool =
     (∃oh ← hs)hotelConCadenaCambiada(h, oh, c) ;
```

```
  aux hotelConCadenaCambiada (h1: Hotel, h2: Hotel, c: nuevaCadena) : Bool =
      (nombre(h1) == nombre(h2)) ∧
      (cadena(h2) == nuevaCadena)) ∧
      (mismos(ingresos(h1), ingresos(h2))) ∧
      (mismos(huespedes(h1), huespedes(h2)) ∧
      (mismos(habitaciones(h1), habitaciones(h2))) ∧
      (mismos(salidas(h1), salidas(h2))) ∧
      (mismos(reservas(h1), reservas(h2))) ∧
      (mismos(tarifasHabitacionXDia(h1), tarifasHabitacionXDia(h2))) ∧
      (mismos(precioAccesorio(h1), precioAccesorio(h2))) ;
  aux cadenaPertenece (lh: [Hotel], llh: [[Hotel]]) : Bool = (∃olh ← llh)mismosHoteles(lh, olh) ;
}
```

6.3. sinLcdParaTodos

```
problema sinLcdParaTodos (m: MinisterioDeTurismo) = result : ℤ {

  modifica m ;
  asegura mismos(secretarias(m), secretarias(pre(m))) ;

  asegura igualCantidadDeHotelesPorRegistro : (∀p ← secretarias(m))
      |registro(m, p)| == |registro(pre(m), p)| ;
  asegura estanLosHotelesPreviosPeroSinLcds : (∀p ← secretarias(m))
      ((∀h ← registro(pre(m), p))existeHotelIgualSinLcds(h, registro(m, p))) ;

  asegura igualCantidadDeCadenas : |cadenasDeHoteles(m)| == |cadenasDeHoteles(pre(m))| ;
  asegura estanLasCadenasPreviasPeroSinLcds : (∀c ← cadenasDeHoteles(pre(m)))
      existeCadenaIgualSinLcds(c, cadenasDeHoteles(m)) ;

  asegura result == contarLcds(pre(m)) ;


  aux existeHotelIgualSinLcds (h: Hotel, hs: [Hotel]) : Bool = (∃oh ∈ hs)hotelIgualSinLcds(h, oh) ;

  aux hotelIgualSinLcds (ha, hd: Hotel) : Bool =
      (nombre(ha) == nombre(hd)) ∧
      (cadena(ha) == cadena(hd)) ∧
      (mismosIngresosSinLcds(ingresos(ha), ingresos(hd))) ∧
      (mismos(huespedes(ha), huespedes(hd)) ∧
      (mismasHabitacionesSinLcds(habitaciones(ha), habitaciones(hd))) ∧
      (mismos(salidas(ha), salidas(hd))) ∧
      (mismos(reservas(ha), reservas(hd))) ∧
      (mismos(tarifasHabitacionXDia(ha), tarifasHabitacionXDia(hd))) ∧
      (mismos(precioAccesorio(ha), precioAccesorio(hd))) ;

  aux mismosIngresosSinLcds (isAntes, isDesp: [(CheckIn,Habitacion)]) : Bool =
      |isAntes| == |isDesp| ∧
      (∀i ∈ isAntes)((∃oi ← isDesp)(prm(i) == prm(oi) ∧ habitacionIgualSinLcd(sgd(i), sgd(oi)))) ;

  aux mismasHabitacionesSinLcds (hbsAntes, hbsDesp: [Habitacion]) : Bool =
      |hbsAntes| == |hbsDesp| ∧
      (∀hb ← hbsAntes)((∃ohb ← hbsDesp)habitacionIgualSinLcd(hb, ohb)) ;

  aux habitacionIgualSinLcd (hAntes,hDesp: Habitacion) : Bool =
      numero(hAntes) == numero(hDesp) ∧
      tipo(hAntes) == tipo(hDesp) ∧
      accesorios(hDesp) == [ a | a ← accesorios(hAntes), ¬a == LCD ]
      ;

  aux existeCadenaIgualSinLcds (c: [Hotel], cs: [[Hotel]]) : Bool = (∃oc ∈ cs)cadenaIgualSinLcds(c, oc) ;

  aux cadenaIgualSinLcds (cAntes, cDesp: [Hotel]) : Bool =
      |cAntes| == |cDesp| ∧
      (∀h ← cAntes)((∃oh ← cDesp)hotelIgualSinLcds(h, oh)) ;

  aux contarLcds (m: Ministerio) : ℤ = |[ hab | h ← todosLosHoteles(m),
      hab ← habitaciones(h), LCD ∈ accesorios(hab) ]| ;

  aux todosLosHoteles (m: Ministerio) : [Hotel] = aplanar(cadenasDeHoteles(m)) ;

}
```

7. Auxiliares

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

aux *cadenaRegistrada* (m: MinisterioDeTurismo, c: Cadena) : Bool $=$
$(\exists ch \leftarrow cadenasDeHoteles(m))cadena(cab(ch)) == c$;

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

aux *contarHotel* (hs: [Hotel], h: Hotel) : $\mathbb{Z}$ $= |[\, oh \mid oh \leftarrow hs, hotelEquivalente(h, oh) \,]|$;

aux *checkInAsociado* (h: Hotel, co: CheckOut) : CheckIn $= cab(obtenerCheckInsAsociados(h, co))$;

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

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

aux *esCheckInAsociado* (h: Hotel, co: CheckOut, ci: CheckIn) : Bool $= ((dniCheckIn(ci) == dniCheckOut(co)) \wedge$
$(fechaCheckIn(ci) < fechaCheckOut(co)) \wedge noExisteCheckOutEnMedio(h, ci, co))$;

aux existeUnaHabitacionDelTipo (h: Hotel, t: TipoHabitacion) : Bool $= (\exists x \leftarrow habitaciones(h))tipo(x) == t$;

aux *existeUnicoCheckInAsociado* (h: Hotel, co: CheckOut) : Bool $= |obtenerCheckInsAsociados(h, co)| == 1$;

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

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

aux *hotelEquivalente* (h1: Hotel, h2: Hotel) : Bool $=$
$(nombre(h1) == nombre(h2)) \wedge$
$(cadena(h1) == cadena(h2)) \wedge$
$(mismos(ingresos(h1), ingresos(h2))) \wedge$
$(mismos(huespedes(h1), huespedes(h2)) \wedge$
$(mismos(habitaciones(h1), habitaciones(h2))) \wedge$
$(mismos(salidas(h1), salidas(h2))) \wedge$
$(mismos(reservas(h1), reservas(h2))) \wedge$
$(mismos(tarifasHabitacionXDia(h1), tarifasHabitacionXDia(h2))) \wedge$
$(mismos(precioAccesorio(h1), precioAccesorio(h2)))$;

aux *hotelPertenece* (h: Hotel,hs: [Hotel]) : Bool $= (\exists oh \leftarrow hs)hotelEquivalente(h, oh)$;

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

aux *listaSinElemento* (lista: [T], elem: T) : [T] $= [\, item \mid item \leftarrow lista, \neg item == elem \,]$;

aux *mismosHoteles* (a: [Hotel], b: [Hotel]) : Bool $= (|a| == |b| \wedge (\forall h \leftarrow a)contarHotel(h, a) == contarHotel(h, b))$;

aux *noCompartenProvincias* (m: Ministerio, c1: Cadena, c2: Cadena) : Bool $=$
$sinRepetidos(provinciasDeCadena(m, obtenerListaDeLaCadena(m, c1))$
$++ provinciasDeCadena(m, obtenerListaDeLaCadena(m, c2)))$;

aux *noExisteCheckOutEnMedio* (h:Hotel, ci: CheckOut, co: CheckOut) : Bool $= \neg(\exists oco \leftarrow salidasDe(h, dniCheckIn(ci)))$
$((fechaCheckIn(ci) < fechaCheckOut(oco)) \wedge (fechaCheckOut(oco) < fechaCheckOut(co)))$;

aux *obtenerCheckInsAsociados* (h: Hotel, co: CheckOut) : [CheckIn] $= [\, prm(i) \mid i \leftarrow ingresos(h),$
$esCheckInAsociado(h, co, prm(i)) \,]$;

aux *obtenerListaDeLaCadena* (m: Ministerio, c: Cadena) : [Hotel] $= cab([\, lh \mid lh \leftarrow cadenasDeHoteles(m),$
$cadena(cab(lh)) == c \,])$;

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

aux *provinciaDeHotel* (m: Ministerio, h: Hotel) : Provincia $= cab([\, p \mid p \leftarrow secretarias(m),$
$oh \leftarrow registro(m, p), hotelEquivalente(oh, h) \,])$;

aux *provinciasDeCadena* (m: MinisterioDeTurisimo, c: [Hotel]) : [Provincia] $=$
$sacarRepetidos([\, provinciaDeHotel(m, h) \mid h \leftarrow c \,])$;

aux *reservasDe* (h: Hotel, dniHuesped: DNI) : [Reserva] $= [\, res \mid res \leftarrow reservas(h), documento(res) == dniHuesped \,]$;

aux sacarRepetidos (l: [T]) : [T] $= [\, l_i \mid i \leftarrow [0..\, |l|), l_i \notin l_{[i+1..|l|-1)} \,]$;

aux salidasDe (h: Hotel, d: DNI) : [CheckOut] $= [x | x \leftarrow salidas(h), dniCheckOut(x) == d]$;

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

```
aux sonCompatibles (m: MinisterioDeTurismo, c1: Cadena, c2: Cadena)) : Bool =
  sinRepetidos([ nombre(h) | h ← unionHotelesDeCadenas(c1, c2) ]) ;

aux unionHotelesDeCadenas (c1: Cadena, c2: Cadena) : [Hotel] =
  [ h | ch ← cadenasDeHoteles(m), h ← ch, cadena(h) == c1 ∨ cadena(h) == c2 ] ;
```