UNIVERSIDAD DE BUENOS AIRES

Departamento de Computación

Facultad de Ciencias Exactas y Naturales

Algoritmos y Estructuras de Datos I

Segundo cuatrimestre de 2012

RTPI De Hoteles y Pasajeros: Demo huespedesConPalabra

3 de Diciembre 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. Demostración huespedesConPalabra **3**
1.1. Especificación huespedesConPalabra . . . 3
1.2. Implementación huespedesConPalabra . . . 3
1.3. Demostración huespedesConPalabra . . . 3
1.3.1. Observaciones . . . 3
1.3.2. Definiciones . . . 3
1.3.3. Transición de estados . . . 4
1.3.4. Demostración de terminación y correctitud para el ciclo . . . 4
1.3.5. Demostración del problema . . . 6

2. Demostración sacarRepetidos **7**
2.1. Especificación funcion auxiliar sacarRepetidos . . . 7
2.2. Implementación funcion auxiliar sacarRepetidos . . . 7
2.3. Demostración funcion auxiliar sacarRepetidos . . . 7
2.4. Pc $\Rightarrow$ I . . . 7
2.5. (I $\wedge$ $\neg$B) $\Rightarrow$ Qc . . . 7
2.6. Funcion variante decrece . . . 8
2.7. I $\wedge$ $Fv \leq Cota \Rightarrow \neg$ B . . . 8
2.8. El cuerpo del ciclo preserva el invariante . . . 8

3. Demostración hayReserva **9**
3.1. Especificación funcion auxiliar HayReserva . . . 9
3.2. Implementación funcion auxiliar HayReserva . . . 9
3.3. Demostración funcion auxiliar HayReserva . . . 9
3.4. Pc $\Rightarrow$ I . . . 9
3.5. (I $\wedge$ $\neg$B) $\Rightarrow$ Qc . . . 9
3.6. Funcion variante decrece . . . 10
3.7. I $\wedge$ $Fv \leq Cota \Rightarrow \neg$ B . . . 10
3.8. El cuerpo del ciclo preserva el invariante . . . 10

4. Demostración noHaySalidaEnMedio **11**
4.1. Especificación funcion auxiliar noHaySalidaEnMedio . . . 11
4.2. Implementación funcion auxiliar noHaySalidaEnMedio . . . 11
4.3. Demostración funcion auxiliar noHaySalidaEnMedio . . . 11
4.4. Pc $\Rightarrow$ I . . . 11
4.5. (I $\wedge$ $\neg$B) $\Rightarrow$ Qc . . . 11
4.6. Funcion variante decrece . . . 12
4.7. I $\wedge$ $Fv \leq Cota \Rightarrow \neg$ B . . . 12
4.8. El cuerpo del ciclo preserva el invariante . . . 12

5. Demostración procesarIesimoIngreso **13**
5.1. Especificación funcion auxiliar procesarIesimoIngreso . . . 13
5.2. Implementación funcion auxiliar procesarIesimoIngreso . . . 13
5.3. Demostración funcion auxiliar procesarIesimoIngreso . . . 13
5.4. Pc $\Rightarrow$ I . . . 13
5.5. (I $\wedge$ $\neg$B) $\Rightarrow$ Qc . . . 13
5.6. Funcion variante decrece . . . 14
5.7. I $\wedge$ $Fv \leq Cota \Rightarrow \neg$ B . . . 14
5.8. El cuerpo del ciclo preserva el invariante . . . 14

1. Demostración huespedesConPalabra

1.1. Especificación huespedesConPalabra

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

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

Nota: *mismos*, *sacarRepetidos*, *noHaySalidaEnElMedio* y *salidasDe* se tomaron sin modificacíón de la sección de auxiliares de la especifiación del proyecto.

1.2. Implementación huespedesConPalabra

```
Lista<DNI> Hotel::huespedesConPalabra() const {
    Lista<DNI> result;
    int i = 0;
    //e1
    while( i < this->_ingresos.longitud() )
    {
        //e2
        result.concatenar( procesarIesimoIngreso( i ) );
        //e3
        i++;
        //e4
    }
    //e5
    result = sacarRepetidos(result);
    //e6
    return result;
}
```

1.3. Demostración huespedesConPalabra

1.3.1. Observaciones

1. El método es declarado const; es decir, this no se modifica. En particular, los valores de *ingresos(this)*, *salidas(this)* y *reservas(this)* tampoco son modificados a lo largo de los estados.
2. Por la poscondición del problema *longitud* definido sobre el tipo Lista, la expresión *this→_ingresos.longitud()* es igual a $|ingresos(this)|$.
3. Las siguientes expresiones son equivalentes para el caso particular de este problema:

   - Expresión original de la especificación que se muestra en la sección anterior
     $[prm(prm(in))|in \leftarrow ingresos(this)_{[0..i)}, co \leftarrow salidasDe(this, prm(prm(in))),$
     $noHaySalidaEnElMedio(prm(in), co, this), hayReserva(in, co, this)]$
   - Tomando el total de las salidas, y utilizando como primera condición un filtro sobre el DNI:
     $[prm(prm(in))|in \leftarrow ingresos(this)_{[0..i)}, co \leftarrow salidas(this), prm(prm(in)) == prm(co),$
     $noHaySalidaEnElMedio(prm(in), co, this), hayReserva(in, co, this)]$
   - Condensando las tres condiciones en una sola
     $[prm(prm(in))|in \leftarrow ingresos(this)_{[0..i)}, co \leftarrow salidas(this), prm(prm(in)) == prm(co)$
     $\wedge\, noHaySalidaEnElMedio(prm(in), co, this) \wedge hayReserva(in, co, this)]$

1.3.2. Definiciones

- Pc:
  $i == 0 \wedge result == []$
- I:
  $0 \leq i \leq |ingresos(this)| \wedge$
  $result == [prm(prm(in))|in \leftarrow ingresos(this)_{[0..i)}, co \leftarrow salidas(this), prm(prm(in)) == prm(co) \wedge$
  $noHaySalidaEnElMedio(prm(in), co, this) \wedge hayReserva(in, co, this)]$
- Qc:
  $result == [prm(prm(in))|in \leftarrow ingresos(this), co \leftarrow salidas(this), prm(prm(in)) == prm(co) \wedge$
  $noHaySalidaEnElMedio(prm(in), co, this) \wedge hayReserva(in, co, this)]$

- B:
$i < |ingresos(this)|$

- Fv (cota == 0):
$this \rightarrow \_ingresos.longitud() - i$

1.3.3. Transición de estados

- En e1 vale:
$i == 0 \wedge result == []$

- En e2 vale:
$B \wedge I:$
$i < |ingresos(this)| \wedge$
$0 \leq i \leq |ingresos(this)| \wedge$
$result == [prm(prm(in))|in \leftarrow ingresos(this)_{[0..i)}, co \leftarrow salidas(this), prm(prm(in)) == prm(co)$
$\wedge\, noHaySalidaEnElMedio(prm(in), co, this) \wedge hayReserva(in, co, this)]$

- En e3 vale:
$i == i@e2 \wedge$
$result == result@e2 +\!\!+ [prm(prm(in))|in \leftarrow [ingresos(this)_{i@e2}], co \leftarrow salidas(this), prm(prm(in)) == prm(co)$
$\wedge\, noHaySalidaEnElMedio(prm(in), co, this) \wedge hayReserva(in, co, this)]$[1]

- En e4 vale:
$i == i@e3 + 1 \wedge$
$result == result@e3$

- En e5 vale:
$Qc:$
$result == [prm(prm(in))|in \leftarrow ingresos(this), co \leftarrow salidas(this), prm(prm(in)) == prm(co)$
$\wedge\, noHaySalidaEnElMedio(prm(in), co, this) \wedge hayReserva(in, co, this)]$

- En e6 vale:
$result == sacarRepetidos(result@e5)$[2]

[1] Por poscondición del problema *concatenar* definido sobre el tipo Lista, y el problema *procesarIesimoIngreso*, especificado posteriormente.
[2] Por poscondición del problema *sacarRepetidos*, especificado posteriormente.

1.3.4. Demostración de terminación y correctitud para el ciclo

- Pc → I
  - como i == 0, por propiedad de enteros
  $0 \leq i$
  - además, por propiedad de listas
  $i == 0 \leq |ingresos(this)|$
  - entonces, uniendo ambas proposiciones
  $0 \leq i \leq |ingresos(this)|$✓
  - por otro lado,
  $[0..i) == [0.,0) == []$
  - y esto implica
  $ingresos(this)_{[0..i)} == ingresos(this)_{[]} == []$
  - Por definición de las listas por comprensión, si el selector se toma de una lista vacía, la lista resultante también es vacía. En particular:
  $[prm(prm(in))|in \leftarrow ingresos(this)_{[0..i)}, co \leftarrow salidas(this), prm(prm(in)) == prm(co)$
  $\wedge\, noHaySalidaEnElMedio(prm(in), co, this) \wedge hayReserva(in, co, this)]$
  $== [prm(prm(in))|in \leftarrow [], co \leftarrow salidas(this), prm(prm(in)) == prm(co)$
  $\wedge\, noHaySalidaEnElMedio(prm(in), co, this) \wedge hayReserva(in, co, this)]$
  $== []$
  - luego, como $result == []$
  $result == [prm(prm(in))|in \leftarrow ingresos(this)_{[0..i)}, co \leftarrow salidas(this), prm(prm(in)) == prm(co)$
  $\wedge noHaySalidaEnElMedio(prm(in), co, this) \wedge hayReserva(in, co, this)]$✓

- por último, la conjunción de las dos proposiciones marcadas es I

- $I \wedge \neg B \rightarrow Qc$
  - por hipótesis, vale
    $i \geq |ingresos(this)| \wedge$
    $0 \leq i \leq |ingresos(this)| \wedge$
    $result == [prm(prm(in))|in \leftarrow ingresos(this)_{[0..i)}, co \leftarrow salidas(this), prm(prm(in)) == prm(co)$
    $\wedge\, noHaySalidaEnElMedio(prm(in), co, this) \wedge hayReserva(in, co, this)]$
  - como $i \leq |ingresos(this)| \wedge i \geq |ingresos(this)|$, por propiedades de enteros
    $i == |ingresos(this)|$
  - luego, reemplazando i
    $ingresos(this)_{[0..i)} == ingresos(this)_{[0..|ingresos(this)|)} == ingresos(this)$
  - utilizando este reemplazo, se obtiene Qc
    $result == [prm(prm(in))|in \leftarrow ingresos(this), co \leftarrow salidas(this), prm(prm(in)) == prm(co)$
    $\wedge noHaySalidaEnElMedio(prm(in), co, this) \wedge hayReserva(in, co, this)]$✓

- Función variante decrece
  - expresamos Fv al principio y final del ciclo,
    $Fv@e2 == this \rightarrow \_ingresos.longitud() - i@e2$
    $Fv@e4 == this \rightarrow \_ingresos.longitud() - i@e4$
  - en la transición de estados, se observa que:
    $i@e4 == i@e3 + 1 == i@e2 + 1$
  - luego, reemplazando y por propiedad de enteros
    $Fv@e4 == this \rightarrow \_ingresos.longitud() - i@e2 - 1 < this \rightarrow \_ingresos.longitud() - i@e2 == Fv@e2$✓

- $I \wedge Fv \leq 0 \rightarrow \neg B$
  - por hipótesis,
    $this \rightarrow \_ingresos.longitud() - i \leq 0$
    $this \rightarrow \_ingresos.longitud() \leq i$
  - o equivalentemente (por la observación número 2.),
    $\neg(i < |ingresos(this)|)$✓

- El cuerpo del ciclo preserva el invariante
  - en e2, vale [1]
    $i < |ingresos(this)| \wedge$
    $result == [prm(prm(in))|in \leftarrow ingresos(this)_{[0..i)}, co \leftarrow salidas(this), prm(prm(in)) == prm(co)$
    $\wedge\, noHaySalidaEnElMedio(prm(in), co, this) \wedge hayReserva(in, co, this)]$
  - en e3, vale [1]
    $result == result@e2 ++ [prm(prm(in))|in \leftarrow [ingresos(this)_{i@e2}], co \leftarrow salidas(this),$
    $prm(prm(in)) == prm(co) \wedge noHaySalidaEnElMedio(prm(in), co, this) \wedge hayReserva(in, co, this)]$
  - reemplazando result@e2 y i@e2,
    $result == [prm(prm(in))|in \leftarrow ingresos(this)_{[0..i)}, co \leftarrow salidas(this), prm(prm(in)) == prm(co)$
    $\wedge\, noHaySalidaEnElMedio(prm(in), co, this) \wedge hayReserva(in, co, this)] ++$
    $.[prm(prm(in))|in \leftarrow [ingresos(this)_i], co \leftarrow salidas(this), prm(prm(in)) == prm(co)$
    $\wedge\, noHaySalidaEnElMedio(prm(in), co, this) \wedge hayReserva(in, co, this)]$
  - se puede observar que las listas concatenadas en el paso anterior difieren únicamente en el selector in:
    $in \leftarrow ingresos(this)_{[0..i)}$ ... $in \leftarrow [ingresos(this)_i]$
  - esto es equivalente a tomar una sola lista por comprensión, concatenando las listas del selector in; es decir:
    $result == [prm(prm(in))|in \leftarrow ingresos(this)_{[0..i]}, co \leftarrow salidas(this), prm(prm(in)) == prm(co)$
    $\wedge\, noHaySalidaEnElMedio(prm(in), co, this) \wedge hayReserva(in, co, this)]$
  - en e4, vale [1]
    $i == i@e3 + 1 \wedge$
    $result == result@e3$
  - reemplazando, obtengo
    $result == [prm(prm(in))|in \leftarrow ingresos(this)_{[0..i)}, co \leftarrow salidas(this), prm(prm(in)) == prm(co)$
    $\wedge noHaySalidaEnElMedio(prm(in), co, this) \wedge hayReserva(in, co, this)]$✓
  - finalmente, se observa:
    $i == i@e3 + 1 == i@e2 + 1$

- como $0 \leq i@e2 < |ingresos(this)|$, sumando 1 en cada miembro
  $1 \leq i \leq |ingresos(this)|$
- y por último,
  $0 \leq i \leq |ingresos(this)|$ ✓

[1] A partir de las transformaciones de estados, expresadas previamente.

1.3.5. Demostración del problema

- como en e1 vale
  $result == [] \wedge i == 0$
- se satisface la precondición del ciclo Pc. Luego, en e5 vale Qc;
  $result == [prm(prm(in))|in \leftarrow ingresos(this), co \leftarrow salidas(this), prm(prm(in)) == prm(co)$
  $\wedge\ noHaySalidaEnElMedio(prm(in), co, this) \wedge hayReserva(in, co, this)]$
- como en e6 vale:
  $result == sacarRepetidos(result@e5)$
- reemplazando result@e5,
  $result == sacarRepetidos([prm(prm(in))|in \leftarrow ingresos(this), co \leftarrow salidas(this), prm(prm(in)) == prm(co) \wedge noHaySalidaEnElMedio(prm(in), co, this) \wedge hayReserva(in, co, this)])$
- finalmente, notando la observación (3.) al comienzo del problema:
  $result == [prm(prm(in))|in \leftarrow ingresos(this)_{[0..i)}, co \leftarrow salidasDe(this, prm(prm(in))),$
  $noHaySalidaEnElMedio(prm(in), co, this), hayReserva(in, co, this)]$ ✓
- que es lo que buscaba demostrar.

2. Demostración sacarRepetidos

2.1. Especificación funcion auxiliar sacarRepetidos

problema sacarRepetidos (list: [T], this: Hotel) = **result** : Bool {

  asegura $result == [list_i \,|\, i \leftarrow [0..|list|), (\neg\exists j \leftarrow [0..i))list_j == list_i]$;
}

2.2. Implementación funcion auxiliar sacarRepetidos

```
Lista<DNI> Hotel::sacarRepetidos(const Lista<DNI>& dnis) const {
    Lista<DNI> result;
    int i = 0;
    int longitud = dnis.longitud();
    //Pc: i == 0 ∧ longitud == |dnis| ∧ result == []
    //B: i < longitud
    while( i < longitud ){
        //Fv: longitud − i
        //Cota: 0
        //I: 0≤i≤longitud ∧ longitud == |dnis| ∧ result == [dnis_k| k←[0..i), (¬∃j←[0..k)) dnis_j == dnis_k]
        //Estado e1
        //Vale B ∧ I
        if( !result.pertenece(dnis.iesimo(i)) ){
            result.agregarAtras(dnis.iesimo(i));
        }
        //Qif:((dnis_i ∈ result) ∧ (result == result@e1) )∨( (dnis_i ∉ result) ∧ (result == result@e1 ++ [dnis_i]))
        //estado e2
        //vale Qif ∧ i == i@e1
        i++;
        //estado e3
        //vale i == i@e2 + 1 ∧ result == result@e2
    }
    //Qc: result == [dnis_i| i←[0..longitud), (¬∃j←[0..i)) dnis_j == dnis_i]
    return result;
    //vale result == [dnis_i| i←[0..longitud), (¬∃j←[0..i)) dnis_j == dnis_i]
}
```

2.3. Demostración funcion auxiliar sacarRepetidos

Las implicaciones mas importantes estan en AZUL
Referirse a los estados mas arriba cuando sea necesario

2.4. Pc ⇒ I

- [1] longitud == $|dnis|$
- [2] i == 0 ⇒ i ≤ longitud
- [2] i == 0 ⇒ 0 ≤ i
- 0 ≤ i ≤ longitud
- [3] i == 0 ⇒ result == [] == [$dnis_k$ | k←[0..0), (¬∃j←[0..k)) $dnis_j$ == $dnis_k$]
- 0 ≤ i ≤ longitud ∧ longitud == $|dnis|$ ∧ result == [$dnis_k$ | k←[0..i), (¬∃j←[0..k)) $dnis_j$ == $dnis_k$] ✓

[1] Por poscondición del problema *longitud* definido para el tipo Lista.
[2] Porque longitud ≥ 0 (por ser *longitud* == $|dnis|$) y por propiedades de los numeros naturales
[3] Propiedades de listas

2.5. (I ∧ ¬B) ⇒ Qc

- [1] (I ∧ ¬B) ⇒ i == longitud
- [2] result == [$dnis_k$| k←[0..longitud), (¬∃j←[0..k)) $dnis_j$ == $dnis_k$] ✓

[1] Propiedades de los numeros naturales
[2] Se infiere de I

2.6. Funcion variante decrece

- Fv: longitud - i
- $i@e3 == i@e2 + 1 == i@e1 + 1$
- [1] $i@e3 > i@e1$
- [1] $-i@e3 < -i@e1$
- [2] $longitud - i@e3 < longitud - i@e1$ ✓

[1] Propiedades de los numeros naturales
[2] Sumando longitud a ambos lados

2.7. I $\wedge$ $Fv \leq Cota$ $\Rightarrow$ $\neg$ B

- Fv: longitud - i
- Cota: 0
- Fv $\leq$ Cota $\Rightarrow$ $longitud - i \leq 0$
- longitud $\leq$ i
- B: i $<$ longitud $\Rightarrow$ $\neg$ B: longitud $\leq$ i
- $\neg$ B $\Leftrightarrow$ Fv $\leq$ Cota ✓

2.8. El cuerpo del ciclo preserva el invariante

- [1] [2] 0 $\leq$ i@e1 $\Rightarrow$ 0 $\leq$ i@e1 + 1
- [3] 0 $\leq$ i@e3
- [1] [2] i@e1 $<$ longitud $\Rightarrow$ i@e1 + 1 $<$ longitud + 1
- [3] i@e3 $\leq$ longitud
- 0 $\leq$ i@e3 $\leq$ longitud ✓
- Si vale (($dnis_{i@e1}$ $\in$ result) $\wedge$ (result == result@e1)) entonces
- [4] result == [$dnis_k$| k←[0..i@e1 + 1), (¬∃j←[0..k)) $dnis_j$ == $dnis_k$]
- [3] result == [$dnis_k$| k←[0..i@e3), (¬∃j←[0..k)) $dnis_j$ == $dnis_k$] ✓
- Si vale (($dnis_{i@e1}$ $\notin$ result) $\wedge$ (result == result@e1 ++ [$dnis_i@e1$])) entonces
- result@e1 == [$dnis_k$| k←[0..i@e1), (¬∃j←[0..k)) $dnis_j$ == $dnis_k$]
- result@e1 ++ [$dnis_i@e1$] == [$dnis_k$— k←[0..i@e1), (¬∃j←[0..k)) $dnis_j$ == $dnis_k$] ++ [$dnis_i@e1$]
- [5] result@e1 ++ [$dnis_i@e1$] == [$dnis_k$ | k←[0..i@e1], (¬∃j←[0..k)) $dnis_j$ == $dnis_k$]
- [6] [7] result@e3 == [$dnis_k$| k←[0..i@e1 + 1), (¬∃j←[0..k)) $dnis_j$ == $dnis_k$]
- [3] result@e3 == [$dnis_k$| k←[0..i@e3), (¬∃j←[0..k)) $dnis_j$ == $dnis_k$] ✓

[1] Propiedades de los numeros naturales
[2] Vale I $\wedge$ B en e1
[3] i@e3 == i@e1 + 1
[4] Por propiedad de listas como la condicion de la lista no se cumple, podemos utilizar i@e1 + 1
[5] Por propiedad de listas podemos incluir al elemento concatenado como parte de la lista por comprension
[6] Por propiedad de intervalos
[7] result@e3 == result@e2 == result@e1 ++ [$dnis_i@e1$]

3. Demostración hayReserva

3.1. Especificación funcion auxiliar HayReserva

problema hayReserva (list: [T], this: Hotel) = **result** : Bool {
  asegura $result = (\exists j \leftarrow [0..|reservas(this)|)) esReservaValida(snd(ingreso), reservas(this)_j, fst(ingreso), checkout)$;
  aux esReservaValida (Habitacion hab, Reserva reserva, CheckIn checkIn, CheckOut checkOut) : Bool $= ((documento(reserva) == fst(checkIn) \wedge (fechaDesde(reserva) == snd(checkIn)) \wedge (fechaHasta(reserva) == snd(checkOut)) \wedge (tipo(reserva) == tipo(hab)) \wedge (confirmada(reserva)));$;
}

3.2. Implementación funcion auxiliar HayReserva

```
bool Hotel::hayReserva(const pair<CheckIn, Habitacion>& ingreso, const CheckOut& checkOut) const {
    bool hayUnaReserva = false;
    int i = 0;
    int longitud = this->_reservas.longitud();
    //Pc: (i == 0) ∧ (hayUnaReserva==false) ∧ (longuitud == |reservas(this)|)
    //B: i < longitud
    //Fv: longitud − i
    //cota: 0
    while(i < longitud){
        //I: (0≤i≤longuitud) ∧ (longuitud == |reservas(this)|) ∧ hayUnaReserva = (∃ j ← [0..i))
        //esReservaValida(snd(ingreso), reservas(this)_j , fst(ingreso), checkout)
        //estado e1
        //Vale I ∧ B
        hayUnaReserva = hayUnaReserva ||
        (esReservaValida(ingreso.second, (this->_reservas.iesimo(i)), ingreso.first, checkOut));
        //estado e2
        //Vale i==i@e1 ∧ hayUnaReserva = hayUnaReserva@e1 ∨
        //esReservaValida(snd(ingreso), reservas(this)_i , fst(ingreso), checkout)
        i++;
        //estado e3
        //Vale i==i@e2 + 1 ∧ hayUnaReserva == hayUnaReserva@e2
    }
    //Qc: hayUnaReserva = (∃ j ← [0..longitud))
    //esReservaValida(snd(ingreso), reservas(this)_j , fst(ingreso), checkout)
    return hayUnaReserva;
    //Estado final res == (∃ j ← [0..|reservas(this)|))
    //esReservaValida(snd(ingreso), reservas(this)_j , fst(ingreso), checkout)
    //(porque longitud == |reservas|)
}
```

3.3. Demostración funcion auxiliar HayReserva

Las implicaciones mas importantes estan en AZUL
Referirse a los estados mas arriba cuando sea necesario

3.4. Pc ⇒ I

- [1] $(longuitud == |reservas(this)|)$ ✓
- [2] longitud >= 0 ∧ i == 0 ⇒ i ≤ longitud
- [2] i == 0 ⇒ 0 ≤ i
- 0 ≤ i ≤ longitud ✓
- hayUnaReserva = (∃ j ← [0..i)) esReservaValida(snd(ingreso), $reservas(this)_j$, fst(ingreso), checkout)
- [3] hayUnaReserva == false == (∃ j ← [0..0)) esReservaValida(snd(ingreso), $reservas(this)_j$, fst(ingreso), checkout) ✓

[1] Por poscondición del problema *longitud* definido para el tipo Lista.
[2] Porque longitud ≥ 0 (por ser $longitud == |reservas(this)|$) y por propiedades de los números naturales
[3] i==0, luego existencial de vacio es false

3.5. (I ∧ ¬B) ⇒ Qc

- [1] (I ∧ ¬B) ⇒ i == longitud
- [2] hayUnaReserva == (∃ j ← [0..longitud)) esReservaValida(snd(ingreso), $reservas(this)_j$, fst(ingreso), checkout) ✓

[1] Propiedades de los numeros naturales
[2] Se infiere de I

3.6. Funcion variante decrece

- Fv: longitud - i
- $i@e3 == i@e2 + 1 == i@e1 + 1$
- [1] $i@e3 > i@e1$
- [1] $-i@e3 < -i@e1$
- [2] $longitud - i@e3 < longitud - i@e1$ ✓

[1] Propiedades de los numeros naturales
[2] Sumando longitud a ambos lados

3.7. I $\wedge$ $Fv \leq Cota$ $\Rightarrow$ $\neg$ B

- Fv: longitud - i
- Cota: 0
- Fv $\leq$ Cota $\Rightarrow$ $longitud - i \leq 0$
- longitud $\leq$ i
- B: i < longitud $\Rightarrow$ $\neg$ B: longitud $\leq$ i
- $\neg$ B $\Leftrightarrow$ Fv $\leq$ Cota ✓

3.8. El cuerpo del ciclo preserva el invariante

- [1] [2] 0 $\leq$ i@e1 $\Rightarrow$ 0 $\leq$ i@e1 + 1
- [3] 0 $\leq$ i@e3
- [1] [2] i@e1 < longitud $\Rightarrow$ i@e1 + 1 < longitud + 1
- [3] i@e3 $\leq$ longitud
- 0 $\leq$ i@e3 $\leq$ longitud ✓
- hayUnaReserva == hayUnaReserva@e2 == hayUnaReserva@e1 $\vee$ esReservaValida(snd(ingreso), $reservas(this)_i$, fst(ingreso), checkout)
- hayUnaReserva == ($\exists$ j $\leftarrow$ [0..i@e1)) esReservaValida(snd(ingreso), $reservas(this)_j$, fst(ingreso), checkout) $\vee$ esReservaValida(snd(ingreso), $reservas(this)_i$, fst(ingreso), checkout)
- [4] hayUnaReserva == ($\exists$ j $\leftarrow$ [0..i@e1 + 1)) esReservaValida(snd(ingreso), $reservas(this)_j$, fst(ingreso), checkout)
- [3] hayUnaReserva == ($\exists$ j $\leftarrow$ [0..i@e3)) esReservaValida(snd(ingreso), $reservas(this)_j$, fst(ingreso), checkout) ✓

[1] Propiedades de los numeros naturales
[2] Vale I $\wedge$ B en e1
[3] i@e3 == i@e1 + 1
[4] Por propiedad de existencial e intervalos podemos incluirlo

4. Demostración noHaySalidaEnMedio

4.1. Especificación funcion auxiliar noHaySalidaEnMedio

problema noHaySalidaEnMedio (CheckIn ci, CheckOut co, this: Hotel) = **result** : Bool {

  requiere $prm(ci) == prm(co)$;
  asegura $result == \neg(\exists oco \leftarrow salidas(this)), prm(oco) == prm(ci)) fechaCheckIn(ci) < fechaCheckOut(oco) < fechaCheckOut(co))$;
}

4.2. Implementación funcion auxiliar noHaySalidaEnMedio

```
bool Hotel::noHaySalidaEnElMedio(const CheckIn& ci, const CheckOut& co) const {
    bool result = false;
    int i = 0;
    int longitud = this->_salidas.longitud();
    //Pc: result == false ∧ i == 0 ∧ longitud == |salidas(this)|
    //Fv: longitud - i
    //Cota: 0
    while( i < longitud )
    {//E1: vale B && I: result == (( ∃ co ← salidas(this)[0..i)), prm(co) == prm(ci))
    //fechaCheckIn(ci) < fechaCheckOut(co) < fechaCheckOut(o)) ∧ 0≤i≤n ∧ longitud == |salidas(this)|
        CheckOut oco = this->_salidas.iesimo(i);
        //E2: vale i == i@E1 ∧ result == result@E1 ∧ oco == salidas_i
        result = result || (ci.first == oco.first && ci.second < oco.second && oco.second < co.second);
        //E3: vale i == i@E2 ∧ result == result@E2 ∨
        //(prm(oco) == prm(ci) ∧ sgd(ci) < sgd(oco) ∧ sgd(oco) < sgd(co)) ∧ oco == oco@E2
        i++;
        //E4: vale i == i@E3 + 1 ∧ result == result@E3 ∧ oco == oco@E3
    }
    //Qc: (∃ co \selec salidas(this), prm(co) == prm(ci)) fechaCheckIn(ci) < fechaCheckOut(co) < fechaCheckOut(o)
    return !result;
    //vale result = ¬ (∃ co \selec salidas(this), prm(co) == prm(ci)) fechaCheckIn(ci) < fechaCheckOut(co) < fechaCheckOut(o)
}
```

4.3. Demostración funcion auxiliar noHaySalidaEnMedio

Las implicaciones mas importantes estan en AZUL
Referirse a los estados mas arriba cuando sea necesario

4.4. Pc ⇒ I

- [1] $(longitud == |salidas(this)|)$ ✓
- [2] i == 0 ⇒ i ≤ longitud
- [2] i == 0 ⇒ 0 ≤ i
- 0 ≤ i ≤ longitud ✓
- result == [] == ((∃ co ← $salidas(this)_{[0.,0)}$), prm(co) == prm(ci))$fechaCheckIn(ci) < fechaCheckOut(co) < fechaCheckOut(o)$) ∧ 0≤i≤n ∧ longitud == |salidas(this)| ✓

[1] Por poscondición del problema *longitud* definido para el tipo Lista.
[2] Porque longitud ≥ 0 (por ser $longitud == |salidas(this)|$) y por propiedades de los números naturales
[3] Existencial de vacio es false pues como i == 0, $salidas(this)_{[0..i)} == [\,]$

4.5. (I ∧ ¬B) ⇒ Qc

- [1] (I ∧ ¬B) ⇒ i == longitud
- [2] result == ((∃ co ← $salidas(this)_{[0..longitud)}$), prm(co) == prm(ci)) $fechaCheckIn(ci) < fechaCheckOut(co) < fechaCheckOut(o)$) ∧ 0≤i≤n ∧ longitud == |salidas(this)|
- [2] result == ((∃ co ← $salidas(this)$), prm(co) == prm(ci)) $fechaCheckIn(ci) < fechaCheckOut(co) < fechaCheckOut(o)$) ∧ 0≤i≤n ∧ longitud == |salidas(this)| ✓

[1] Propiedades de los numeros naturales
[2] Se infiere de I

4.6. Funcion variante decrece

- Fv: longitud - i
- $i@e4 == i@e3 + 1 == i@e2 + 1 == i@e1 + 1$
- [1] $i@e4 > i@e1$
- [1] $-i@e4 < -i@e1$
- [2] $longitud - i@e4 < longitud - i@e1$ ✓

[1] Propiedades de los numeros naturales
[2] Sumando longitud a ambos lados

4.7. I $\wedge$ $Fv \leq Cota$ $\Rightarrow$ ¬ B

- Fv: longitud - i
- Cota: 0
- Fv ≤ Cota ⇒ $longitud - i \leq 0$
- longitud ≤ i
- B: i < longitud ⇒ ¬ B: longitud ≤ i
- ¬ B ⇔ Fv ≤ Cota ✓

4.8. El cuerpo del ciclo preserva el invariante

- [1] [2] 0 ≤ i@e1 ⇒ 0 ≤ i@e1 + 1
- [3] 0 ≤ i@e4
- [1] [2] i@e1 < longitud ⇒ i@e1 + 1 < longitud + 1
- [3] i@e4 ≤ longitud
- 0 ≤ i@e4 ≤ longitud ✓
- [2] result@e1 == ((∃ co ← $salidas(h)_{[0..i@E1)}$), prm(co) == prm(ci)) fechaCheckIn(ci)<fechaCheckOut(co)<fechaCheckOut(o)) ∧ 0≤i≤n ∧ longitud == |salidas(this)|
- [5] [6] [7] result == ((∃ co ← $salidas(this)_{[0..i@E1)}$), prm(co) == prm(ci)) fechaCheckIn(ci) < fechaCheckOut(co) < fechaCheckOut(o)) ∨ (prm(oco) == prm(ci) ∧ sgd(ci) < sgd(oco) ∧ sgd(oco) < sgd(co))
- [8] result == ((∃ co ← $salidas(this)_{[0..i@E1+1)}$), prm(co) == prm(ci)) fechaCheckIn(ci)<fechaCheckOut(co)<fechaCheckOut(o)) ∧ 0≤i≤n ∧ longitud == |salidas(this)|
- [3]result == ((∃ co ← $salidas(this)_{[0..i@E4)}$), prm(co) == prm(ci)) fechaCheckIn(ci)<fechaCheckOut(co)<fechaCheckOut(o)) ∧ 0≤i≤n ∧ longitud == |salidas(this)| ✓

[1] Propiedades de los numeros naturales
[2] Vale I ∧ B en e1
[3] i@e4 == i@e1 + 1
[4] Por propiedad de existencial e intervalos podemos incluirlo
[5] En E2 vale: i == i@E1 ∧ result == result@E1 ∧ oco == $salidas_i$
[6] En E3: vale result == result@E2 ∨ (prm(oco) == prm(ci) ∧ sgd(ci) < sgd(oco) ∧ sgd(oco) < sgd(co))
[7] (Reemplazando result@E2==result@E1)En E3 vale: result == ((∃ co ← $salidas(this)_{[0..i@E1)}$), prm(co) == prm(ci)) fechaCheckIn(ci) < fechaCheckOut(co) < fechaCheckOut(o)) ∨ (prm(oco) == prm(ci) ∧ sgd(ci) < sgd(oco) ∧ sgd(oco) < sgd(co))
[8] Siendo oco $salidas_i$, entonces tenemos el existencial hasta i@E1 abierto, y tenemos un or con la condicion de $salidas(this)_{i@E1}$, podemos meterlo dentro del existencial
[9] Propiedades de intervalos

5. Demostración procesarIesimoIngreso

5.1. Especificación funcion auxiliar procesarIesimoIngreso

problema procesarIesimoIngreso (i: $\mathbb{Z}$, this: Hotel) = **result** : [Dni] {

requiere $i < |ingresos(this)|$;
asegura $result == [\,prm(prm(in))\,|\,in \leftarrow [ingresos(this)_i], co \leftarrow salidas(this),$
$prm(prm(in)) == prm(co) \land noHaySalida(prm(in), co, this) \land hayReserva(in, co, this)]\,]$;

}

5.2. Implementación funcion auxiliar procesarIesimoIngreso

```
Lista<DNI> Hotel::procesarIesimoIngreso( int i ) const {
    pair<CheckIn,Habitacion> in = this->_ingresos.iesimo(i);
    CheckIn ci = in.first;
    Lista<DNI> result;
    int longitud = this->_salidas.longitud();
    int j = 0;
    //Fv: longitud - j
    //Cota: 0
    //Pc: vale ci == prm(ingresos_i) ∧ result == [] ∧ j == 0 ∧ longitud == |salidas(this)|
    //∧ in == ingresos_i
    while( j < longitud )
    {//E1: vale I ∧ B
        //I:vale result == [prm(prm(in)) | in ← [ingresos(this)_i], co ← salidas(this)[0..j), prm(prm(in))) == prm(co)
        //∧ noHaySalidaEnElmedio(prm(in),co,this) ∧ hayReserva(in,co,this)]
        // ∧ 0≤j≤longitud ∧ longitud == |salidas(this)| ∧ ci == prm(ingresos_i) ∧ in == ingresos_i
        CheckOut co = this->_salidas.iesimo(j);
        //E2: vale j == j@E1 ∧ co == salidas(this)_j ∧ result == result@E1
        if( ci.first == co.first && noHaySalidaEnElMedio(ci,co) && hayReserva(in,co) )
            result.agregarAtras( ci.first );
         //Qif: vale (( prm(ci) == prm(co) ∧ noHaySalidaEnElMedio(ci,co) ∧ hayReserva(in,co) ∧ result == result@E2 ++ [
         //          (( prm(ci) != prm(co) ∨ !noHaySalidaEnElMedio(ci,co) ∨ !hayReserva(in,co)) ∧ result == result@E2)
         //   ∧ j == j@E2 ∧ co == co@E2 ∧ ci == ci@E2

         j++;
         //E3: vale  j == j@Qif + 1 ∧ co == co@Qif ∧ ci == ci@Qif ∧ result == result@Qif

    }
    //Qc:result== [prm(prm(in)) | in ← [ingresos(this)_i], co ← salidas(this),
    //prm(prm(in))) == prm(co) ∧ noHaySalidaEnElmedio(prm(in),co,this) ∧ hayReserva(in,co,this)]
    return result;
    //Q: vale res == [prm(prm(in)) | in ← [ingresos(this)_i], co ← salidas(this),
    //prm(prm(in))) == prm(co) ∧ noHaySalidaEnElmedio(prm(in),co,this) ∧ hayReserva(in,co,this)]
}
```

5.3. Demostración funcion auxiliar procesarIesimoIngreso

Las implicaciones mas importantes estan en AZUL
Referirse a los estados mas arriba cuando sea necesario

5.4. Pc ⇒ I

- [1] ci == prm($ingresos(this)_i$) y longitud == $|salidas(this)|$ ✓
- [2] j == 0 ⇒ j ≤ longitud
- [2] j == 0 ⇒ 0 ≤ j
- 0 ≤ j ≤ longitud ✓
- [3] [4] result == [] == [prm(prm(in)) | in ← [ingresos(i)], o ← $salidas(this)_{[0..,0)}$, prm(prm(in)) == prm(co) ∧ noHaySalidaEnElmedio(prm(in),co) ∧ hayReserva(in,co)] ✓

[1] Por poscondición del problema *longitud* definido para el tipo Lista.
[2] Porque longitud ≥ 0 (por ser *longitud* == $|salidas(this)|$) y por propiedades de los números naturales
[3] Propiedades de listas
[4] j == 0

5.5. (I ∧ ¬B) ⇒ Qc

- [1] (I ∧ ¬B) ⇒ j == longitud
- [2] result == [prm(prm(in)) | in ← [$ingresos(this)_i$], o ← salidas(this), prm(prm(in)) == prm(co) ∧ noHaySalidaEnElmedio(prm(in),co) ∧ hayReserva(in,co)]] ✓

[1] Propiedades de los numeros naturales
[2] Se infiere de I y propiedades de intervalos

5.6. Funcion variante decrece

- Fv: longitud - j
- $j@e3 == j@Qif + 1 == j@e2 + 1 == j@e1 + 1$
- [1] $j@e3 > j@e1$
- [1] $-j@e3 < -j@e1$
- [2] $longitud - j@e3 < longitud - j@e1$ ✓

[1] Propiedades de los numeros naturales
[2] Sumando longitud a ambos lados

5.7. I $\wedge$ $Fv \leq Cota$ $\Rightarrow$ $\neg$ B

- Fv: longitud - j
- Cota: 0
- Fv $\leq$ Cota $\Rightarrow$ $longitud - j \leq 0$
- longitud $\leq$ j
- B: j < longitud $\Rightarrow$ $\neg$ B: longitud $\leq$ j
- $\neg$ B $\Leftrightarrow$ Fv $\leq$ Cota ✓

5.8. El cuerpo del ciclo preserva el invariante

- [1] [2] 0 $\leq$ j@e1 $\Rightarrow$ 0 $\leq$ j@e1 + 1
- [3] 0 $\leq$ j@e3
- [1] [2] j@e1 < longitud $\Rightarrow$ j@e1 + 1 < longitud + 1
- [3] j@e3 $\leq$ longitud
- 0 $\leq$ j@e3 $\leq$ longitud ✓
- si vale ((prm(ci) == prm(co) $\wedge$ noHaySalidaEnElMedio(ci,co) $\wedge$ hayReserva(in,co) $\wedge$ result == result@E2 ++ [ci]):
- result@E2 == result@E1 == [prm(prm(in)) | in $\leftarrow$ [ingresos(i)], o $\leftarrow$ $salidas(this)_{[0..j)}$, prm(prm(in)) == prm(co) $\wedge$ noHaySalidaElMedio(prm(in),co) $\wedge$ hayReserva(in,co)]
- result @Qif == [prm(prm(in)| in $\leftarrow$ [ingresos(i)], o $\leftarrow$ $salidas(this)_{[0..j)}$, prm(prm(in)) == prm(co) $\wedge$ noHaySalidaElMedio(prm(in),co) $\wedge$ hayReserva(in,co)] ++ [ci]
- [4] result == [prm(prm(in)) | in $\leftarrow$ [ingresos(i)], o $\leftarrow$ $salidas(this)_{[0..j+1)}$, prm(prm(in)) == prm(co) $\wedge$ noHaySalidaElMedio(prm(in),co) $\wedge$ hayReserva(in,co)]
- [3] result == [prm(prm(in)) | in $\leftarrow$ [ingresos(i)], o $\leftarrow$ $salidas(this)_{[0..j@e3)}$, prm(prm(in)) == prm(co) $\wedge$ noHaySalidaEnElmEdio(prm(in),co) $\wedge$ hayReserva(in,co)] ✓
- si vale (prm(ci) != prm(co) $\vee$!noHaySalidaEnElMedio(ci,co) $\vee$!hayReserva(in,co)) $\wedge$ result == result@E2)
- result@E2 == result@E1 == [prm(prm(in)) | in $\leftarrow$ [ingresos(i)], co $\leftarrow$ $salidas(this)_{[0..j)}$, prm(prm(in)) == prm(co) $\wedge$ noHaySalidaElMedio(prm(in),co) $\wedge$ hayReserva(in,co)]
- [5] las listas son iguales con estos intervalos [0..j] == [0..j@E3)
- result == [prm(prm(in)) | in $\leftarrow$ [ingresos(i)], co $\leftarrow$ $salidas(this)_{[0..j@e3)}$, prm(prm(in)) == prm(co) $\wedge$ noHaySalidaElMedio(prm(in),co) $\wedge$ hayReserva(in,co)] ✓

[1] Propiedades de los numeros naturales
[2] Vale I $\wedge$ B en e1
[3] j@e3 == j@e1 + 1
[4] Podemos agregar la concatenacion al resultado porque sabemos que vale la guarda del if que es la condicion de la lista por comprension
[5] Sabemos que el elemento j-ésimo de salidas no cumple la condicion de la lista, podemos cerrar el intervalo [0..j] ya que la lista hasta j-1 y hasta j son identicas (por prop de listas por comprension)