



**DEPARTAMENTO  
DE COMPUTACION**

Facultad de Ciencias Exactas y Naturales - UBA

# Trabajo Práctico 1 - Especificación

Fat Food

8 de septiembre de 2015

Algoritmos y Estructura de Datos I

## Grupo 19 Plaza Sesamo

| Integrante                        | LU     | Correo electrónico      |
|-----------------------------------|--------|-------------------------|
| Gagliardi, Hernan Gabriel         | 810/12 | hg.gagliardi@gmail.com  |
| de Monasterio, Fransisco Jesús    | 764/13 | XXX@XXX.XXX             |
| Martín Darricades, Matías Facundo | 480/13 | matiasamd@gmail.com     |
| Solari Saban, Tomás León          | 774/14 | tomassolari94@gmail.com |



**Facultad de Ciencias Exactas y Naturales**  
Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

<http://www.exactas.uba.ar>

# Índice

|                                |          |
|--------------------------------|----------|
| <b>1. Tipos</b>                | <b>3</b> |
| <b>2. Combo</b>                | <b>3</b> |
| <b>3. Pedido</b>               | <b>4</b> |
| <b>4. Local</b>                | <b>6</b> |
| <b>5. Funciones Auxiliares</b> | <b>9</b> |
| 5.1. Combo . . . . .           | 9        |
| 5.2. Pedido . . . . .          | 9        |
| 5.3. Local . . . . .           | 9        |

## 1. Tipos

```
tipo Empleado = String ;
tipo Energia =  $\mathbb{Z}$  ;
tipo Cantidad =  $\mathbb{Z}$  ;
tipo Bebida = Pesti Cola, Falsa Naranja, Se ve nada, Agua con Gags, Agua sin Gags ;
tipo Hamburguesa = Mcgyver, CukiQueFresco (Cuarto de Kilo con Queso Fresco), McPato, Big Macabra ;
```

## 2. Combo

```
tipo Combo {
    observador bebida (c: Combo) : Bebida ;
    observador sandwich (c: Combo) : Hamburguesa ;
    observador dificultad (c: Combo) : Energia ;

    invariante dificultadHasta100 : energiaEnRango(dificultad(c)) ;
}

problema nuevoC (b: Bebida, h: Hamburguesa, d: Energia) = res : Combo {
    requiere  $d \geq 0 \wedge d \leq 100$  ;
    asegura bebida(res) == b ;
    asegura sandwich(res) == h ;
    asegura dificultad(res) == d ;
}

problema bebidaC (c: Combo) = res : Bebida {
    asegura res == bebida(c) ;
}

problema sandwichC (c: Combo) = res : Hamburguesa {
    asegura res == sandwich(c) ;
}

problema dificultadC (c: Combo) = res : Energia {
    asegura res == dificultad(c) ;
}
```

### 3. Pedido

```

tipo Pedido {
  observador numero (p: Pedido) :  $\mathbb{Z}$ ;
  observador atendio (p: Pedido) : Empleado;
  observador combos (p: Pedido) : [Combo];

  invariante numeroPositivo :  $numero(p) > 0$ ;
  invariante pideAlgo :  $|combos(p)| > 0$ ;
}

problema nuevoP (n:  $\mathbb{Z}$ , e: Empleado, cs: [Combo]) = res : Pedido {
  requiere numeroPositivo :  $n > 0$ ;
  requiere pideAlgo :  $|cs| > 0$ ;
  asegura  $n == numero(res)$ ;
  asegura  $e == atendio(res)$ ;
  asegura  $mismos(cs, combos(res))$ ;
}

problema numeroP (p: Pedido) = res :  $\mathbb{Z}$  {
  asegura  $res == numero(p)$ ;
}

problema atendioP (p: Pedido) = res : Empleado {
  asegura  $res == atendio(p)$ ;
}

problema combosP (p: Pedido) = res : [Combo] {
  asegura  $mismos(res, combos(p))$ ;
}

problema agregarComboP (p: Pedido, c: Combo) {
  modifica  $p$ ;
  asegura  $numero(p) == numero(pre(p))$ ;
  asegura  $atendio(p) == atendio(pre(p))$ ;
  asegura  $|combos(p)| == |combos(pre(p))| + 1$ ;
  asegura  $(\forall k \leftarrow [0..|combos(pre(p))|]) combosIguales(combos(p)_k, combos(pre(p))_k)$ ;
  asegura  $combosIguales(c, combos(p)_{|combos(pre(p))|})$ ;
}

problema anularComboP (p: Pedido, i:  $\mathbb{Z}$ ) {
  requiere enRango :  $0 \leq i < |combos(pre(p))|$ ;
  modifica  $p$ ;
  asegura  $numero(p) == numero(pre(p))$ ;
  asegura  $atendio(p) == atendio(pre(p))$ ;
  asegura  $|combos(p)| == |combos(pre(p))| - 1$ ;
  asegura  $(\forall k \leftarrow [0..i]) combosIguales(combos(p)_k, combos(pre(p))_k)$ ;
  asegura  $(\forall k \leftarrow [i..|combos(p)|]) combosIguales(combos(p)_k, combos(pre(p))_{k+1})$ ;
}

problema cambiarBebidaComboP (p: Pedido, b: Bebida, i:  $\mathbb{Z}$ ) {
  requiere enRango :  $0 \leq i \leq |combos(pre(p))|$ ;
  modifica  $p$ ;
  asegura  $numero(p) == numero(pre(p))$ ;
  asegura  $atendio(p) == atendio(pre(p))$ ;
  asegura  $|combos(p)| == |combos(pre(p))|$ ;
  asegura  $(\forall k \leftarrow [0..i]) combosIguales(combos(p)_k, combos(pre(p))_k)$ ;
  asegura  $(\forall k \leftarrow [i+1..|combos(pre(p))|]) combosIguales(combos(p)_k, combos(pre(p))_k)$ ;
  asegura  $sandwich(combos(p)_i) == sandwich(combos(pre(p))_i)$ ;
  asegura  $bebida(combos(p)_i) == b$ ;
}

```

```
problema elMezcladitoP (p: Pedido) {  
    requiere  $|combos(pre(p))| \leq (cantidadSandwichDistintos(combos(pre(p))) *  
        cantidadBebidaDistintos(combos(pre(p))))$  ;  
    modifica p ;  
    asegura  $numero(p) == numero(pre(p))$  ;  
    asegura  $atendio(p) == atendio(pre(p))$  ;  
    asegura  $|combos(p)| == |combos(pre(p))|$  ;  
    asegura  $(\forall i \leftarrow [0..|combos(p)|], \forall j \leftarrow [0..|combos(p)|]) i \neq j \wedge combosDistintos(combos(p)_i, combos(p)_j)$  ;  
    asegura  $mismosSandwich(combos(p), combos(pre(p)))$  ;  
    asegura  $mismosBebida(combos(p), combos(pre(p)))$  ;  
}
```

## 4. Local

```

tipo Local {
  observador stockBebidas (l: Local, b: Bebida) : Cantidad ;
    requiere  $b \in bebidasDelLocal(l)$  ;
  observador stockSandwiches (l: Local, h: Hamburguesa) : Cantidad ;
    requiere  $h \in sandwichesDelLocal(l)$  ;
  observador bebidasDelLocal (l:Local) : [Bebida] ;
  observador sandwichesDelLocal (l:Local) : [Hamburguesa] ;
  observador empleados (l: Local) : [Empleado] ;
  observador desempleados (l: Local) : [Empleado] ;
  observador energiaEmpleado (l: Local, e: Empleado) : Energia ;
    requiere  $e \in empleados(l)$  ;
  observador ventas (l: Local) : [Pedido] ;

  invariante hayBebidasYSonDistintas :  $|bebidasDelLocal(l)| > 0 \wedge distintos(bebidasDelLocal(l))$  ;
  invariante haySandwichesYSonDistintos :  $|sandwichesDelLocal(l)| > 0 \wedge distintos(sandwichesDelLocal(l))$  ;
  invariante stockBebidasPositivo :  $(\forall b \leftarrow bebidasDelLocal(l)) stockBebidas(l, b) \geq 0$  ;
  invariante stockSandwichesPositivo :  $(\forall h \leftarrow sandwichesDelLocal(l)) stockSandwiches(l, h) \geq 0$  ;
  invariante empleadosDistintos :  $distintos(empleados(l) ++ desempleados(l))$  ;
  invariante energiaHasta100 :  $(\forall e \leftarrow empleados(l)) energiaEnRango(energiaEmpleado(l, e))$  ;
  invariante empleadosQAtendieronDelLocal :  $(\forall v \leftarrow ventas(l)) atendio(v) \in empleados(l) ++ desempleados(l)$  ;
  invariante ventasCorrelativas :  $|[menorNumeroPedido(ventas(l))..mayorNumeroPedido(ventas(l))]| ==$ 
     $|ventas(l)| \wedge (\forall i \leftarrow [menorNumeroPedido(ventas(l))..mayorNumeroPedido(ventas(l))])$ 
     $j \leftarrow [0..|ventas(l)|) i == numero(ventas_j)$  ;
  invariante combosDeLocal :  $(\forall i \leftarrow sandwichLocal(ventas(l)), j \leftarrow sandwichesDelLocal(l)) i == j \wedge$ 
     $(\forall i \leftarrow bebidaLocal(ventas(l)), j \leftarrow bebidasDelLocal(l)) i == j$  ;
}

problema stockBebidasL (l: Local, b:Bebida) = res : Cantidad {
  requiere  $bebidaPerteneceLocal(b, l)$  ;
  asegura  $res == stockBebidas(l, b)$  ;
}

problema stockSandwichesL (l: Local, h:Hamburguesa) = res : Cantidad {
  requiere  $sandwichPerteneceLocal(h, l)$  ;
  asegura  $res == stockSandwiches(l, h)$  ;
}

problema bebidasDelLocalL (l: Local) = res : [Bebida] {
  asegura  $mismos(res, bebidasDelLocal(l))$  ;
}

problema sandwichesDelLocalL (l: Local) = res : [Hamburguesa] {
  asegura  $mismos(res, sandwichesDelLocal(l))$  ;
}

problema empleadosL (l: Local) = res : [Empleado] {
  asegura  $mismos(res, empleados(l))$  ;
}

problema desempleadosL (l: Local) = res : [Empleado] {
  asegura  $mismos(res, desempleados(l))$  ;
}

problema energiaEmpleadoL (l: Local, e:Empleado) = res : Energia {
  requiere  $empleadoPerteneceLocal(e, l)$  ;
  asegura  $res == energiaEmpleado(l, e)$  ;
}

problema ventasL (l: Local) = res : [Pedido] {
  asegura  $mismos(res, ventas(l))$  ;
}

```

```

}

problema unaVentaCadaUno (l:Local) = res : Bool {
  requiere hayEmpleados : |empleados(l)| > 0;
  requiere alMenosUnaVenta : ( $\forall e \leftarrow empleados(l), \exists p \leftarrow ventas(l) \text{atendio}(p) == e$ );
  asegura res == ( $\forall i \leftarrow [0..|quienAtiendeVentasEmpleado(ventas(l), l)| - |empleados(l)|]$ )
    quienAtiendeVentasEmpleado(ventas(l), l)i == quienAtiendeVentasEmpleado(ventas(l), l)i+|empleados(l)|;
}

problema venderL (l: Local, p:Pedido) {
  requiere numeroPedidoCorrecto : numero(p) == mayorNumeroPedido(ventas(pre(l))) + 1;
  requiere empleadoDelLocal : empleadoPerteneceLocal(atendio(p), pre(l));
  requiere tieneEnergiaSuficiente : energiaEmpleado(pre(l), atendio(p)) ≥ energiaPedido(p);
  requiere stockSandwichSuficiente : ( $\forall h \leftarrow sandwichDistintos(combos(p))$ )(stockSandwiches(pre(l), h)
    − cuentaSandwich(h, combos(p))) > 0;
  requiere stockBebidaSuficiente : ( $\forall b \leftarrow bebidaDistintos(combos(p))$ )(stockBebidas(pre(l), b)
    − cuentaBebida(b, combos(p))) > 0;
  modifica l;
  asegura mismos(bebidasDelLocal(l), bebidasdelLocal(pre(l)));
  asegura mismos(sandwichesDelLocal(l), sandwichesDelLocal(pre(l)));
  asegura mismosEmpleados(empleados(l) ++ desempleados(l), empleados(pre(l)) ++ desempleados(pre(l)));
  asegura ( $\forall b \leftarrow bebidaDistintos(combos(p))$ )(stockBebidas(pre(l), b)
    − cuentaBebida(b, combos(p))) == stockBebidas(l, b);
  asegura ( $\forall h \leftarrow sandwichDistintos(combos(p))$ )(stockSandwiches(pre(l), h)
    − cuentaSandwich(h, combos(p))) == stockSandwiches(l, b);
  asegura if (energiaEmpleado(pre(l), atendio(p)) − energiaPedido(p)) > 0 then
    (energiaEmpleado(pre(l), atendio(p)) − energiaPedido(p)) == energiaEmpleado(l, atendio(p)) else
    mismosEmpleados(desempleados(l), agregarDesempleadoLocal(pre(l), atendio(p))) ∧
    mismosEmpleados(empleados(l), despedirEmpleadoLocal(pre(l), atendio(p)));
  asegura |ventas(l)| == |ventas(pre(l))| + 1;
  asegura numero(p) == mayorNumeroPedido(ventas(l));
  asegura mismosPedidos(ventas(l), ventas(pre(l)) ++ p);
}

problema candidatosAEmpleadosDelMesL (l: Local) = res : [Empleado] {
  asegura mismosEmpleados(res, mejoresEmpleados(l));
}

problema sancionL (l: Local, e:Empleado, n:Energia) {
  requiere empleadoDelLocal : empleadoPerteneceLocal(e, pre(l));
  modifica l;
  asegura mismos(bebidasDelLocal(l), bebidasdelLocal(pre(l)));
  asegura mismos(sandwichesDelLocal(l), sandwichesDelLocal(pre(l)));
  asegura mismosEmpleados(empleados(l) ++ desempleados(l), empleados(pre(l)) ++ desempleados(pre(l)));
  asegura mismos(ventas(l), ventas(pre(l)));
  asegura ( $\forall b \leftarrow bebidasDelLocal(pre(l))$ ) stockBebidas(pre(l), b) == stockBebidas(l, b);
  asegura ( $\forall empleado \leftarrow empleados(pre(l)), e \neq empleado$ ) energiaEmpleado(pre(l), empleado) ==
    energiaEmpleado(l, empleado);
  asegura ( $\forall h \leftarrow sandwichesDelLocal(pre(l))$ ) stockSandwiches(pre(l), h) == stockSandwiches(l, h);
  asegura if (energiaEmpleado(pre(l), e) − n) > 0 then (energiaEmpleado(pre(l), e) − n) == energiaEmpleado(l, e)
    else mismosEmpleados(desempleados(l), agregarDesempleadoLocal(pre(l), e)) ∧
    mismosEmpleados(empleados(l), despedirEmpleadoLocal(pre(l), e));
}

problema elVagonetaL (l: Local) = res : Empleado {
  requiere hayEmpleados : |empleados(l)| > 0;
  asegura res == empleadoMayorDescanso(l);
}

problema anularPedidoL (l: Local, n: ℤ) {
  requiere esPedidoLocal(pre(l), n);
}

```

```

    requiere empleadoDelLocal : empleadoPerteneceLocal(atendio(pedidoLocal(pre(l), n)), pre(l));
    modifica l;
    asegura mismos(bebidasDelLocal(l), bebidasDelLocal(pre(l)));
    asegura mismos(sandwichesDelLocal(l), sandwichesDelLocal(pre(l)));
    asegura mismosEmpleados(empleados(l), empleados(pre(l)));
    asegura mismosEmpleados(desempleados(l), desempleados(pre(l)));
    asegura |ventas(l)| == |ventas(pre(l))| - 1;
    asegura ( $\forall b \leftarrow bebidaDistintos(combos(pedidoLocal(pre(l), n)))$ )(stockBebidas(pre(l), b)
        + cuentaBebida(b, combos(pedidoLocal(pre(l), n))) == stockBebidas(l, b);
    asegura ( $\forall h \leftarrow sandwichDistintos(combos(pedidoLocal(pre(l), n)))$ )(stockSandwiches(pre(l), h)
        - cuentaSandwich(h, combos(pedidoLocal(pre(l), n))) == stockSandwiches(l, b);
    asegura ( $\forall e \leftarrow empleados(pre(l))$ , atendio(pedidoLocal(pre(l), n))  $\neq e$ )energiaEmplado(pre(l), e) ==
        energiaEmpleado(l, e);
    asegura (energiaEmpleado(pre(l), atendio(pedidoLocal(pre(l), n))) + energiaPedido(pedidoLocal(pre(l), n))) ==
        energiaEmpleado(l, atendio(pedidoLocal(pre(l), n)));
    asegura mayorNumeroPedido(ventas(pre(l)) - 1 == mayorNumeroPedido(ventas(l));
    asegura mismosPedidosNoNumero(ventas(l), eliminarPedidoVentasNumero(pre(l), n));
    asegura ( $\forall i \leftarrow [0..|ventas(pre(l))|]$ , numero(ventas(pre(l))i)  $\neq n$ ) if numero(ventas(pre(l))i) > n then
        numero(ventas(pre(l))i) - 1 == numero(ventas(l)i) else numero(ventas(pre(l))i) == numero(ventas(l)i);
}

problema agregarComboAlPedidoL (l: Local, c: Combo, n:Z) {
    requiere esPedidoLocal(pre(l), n);
    requiere empleadoDelLocal : empleadoPerteneceLocal(atendio(pedidoLocal(pre(l), n)), pre(l));
    requiere tieneEnergiaSuficiente : energiaEmpleado(pre(l), atendio(pedidoLocal(pre(l), n)))  $\geq$ 
        dificultad(c);
    requiere stockSandwichSuficiente : (stockSandwiches(pre(l), sandwich(c)) - 1) > 0;
    requiere stockBebidaSuficiente : (stockBebidas(pre(l), bebida(c)) - 1) > 0;
    modifica l;
    asegura mismos(bebidasDelLocal(l), bebidasDelLocal(pre(l)));
    asegura mismos(sandwichesDelLocal(l), sandwichesDelLocal(pre(l)));
    asegura mismosEmpleados(empleados(l), empleados(pre(l)));
    asegura mismosEmpleados(desempleados(l), desempleados(pre(l)));
    asegura ( $\forall b \leftarrow bebidasDelLocal(pre(l))$ ,  $b \neq bebida(c)$ )stockBebidas(pre(l), b) == stockBebidas(l, b);
    asegura ( $\forall e \leftarrow empleados(pre(l))$ ,  $e \neq atendio(pedidoLocal(pre(l), n))$ )energiaEmplado(pre(l), e) ==
        energiaEmpleado(l, e);
    asegura ( $\forall h \leftarrow sandwichesDelLocal(pre(l))$ ,  $h \neq sandwich(c)$ )stockSandwiches(pre(l), h)
        == stockSandwiches(l, h);
    asegura stockBebidas(pre(l), bebida(c)) == stockBebidas(l, bebida(c)) + 1;
    asegura stockSandwiches(pre(l), sandwich(c)) == stockSandwiches(l, sandwich(c)) + 1;
    asegura energiaEmplado(pre(l), atendio(pedidoLocal(pre(l), n))) ==
        energiaEmpleado(l, atendio(pedidoLocal(pre(l), n))) + dificultad(c);
    asegura |ventas(l)| == |ventas(pre(l))|;
    asegura mismosPedidos(eliminarPedidoVentasNumero(pre(l), n), eliminarPedidoVentasNumero(l, n));
    asegura numero(pedidoLocal(pre(l), n)) == numero(pedidoLocal(l, n));
    asegura atendio(pedidoLocal(pre(l), n)) == atendio(pedidoLocal(l, n));
    asegura mismosCombosDePedidos(combos(pedidoLocal(pre(l), n)) + +[c], combos(pedidoLocal(l, n)));
}

```



## 5. Funciones Auxiliares

```
aux distintos (ls:[T]) : Bool = ( $\forall i, j \leftarrow [0..|ls|], i \neq j$ )  $ls_i \neq ls_j$ ;
aux energiaEnRango (e: Energia) : Bool =  $0 \leq e \leq 100$ ;
```

### 5.1. Combo

### 5.2. Pedido

```
aux combosIguales (cA: Combo, cB: Combo) : Bool = bebida(cA) == bebida(cB)  $\wedge$  sandwich(cA) == sandwich(cB);
aux cantidadSandwichDistintos (combos: [Combo]) :  $\mathbb{Z}$  =  $|[combos_i | i \leftarrow [0..|combos|], \forall j \leftarrow [0..|combos|], i \leq j \wedge$ 
if  $i < j$  then sandwich( $combos_i$ )  $\neq$  sandwich( $combos_j$ ) else True]|;
aux cantidadBebidaDistintos (combos: [Combo]) :  $\mathbb{Z}$  =  $|[combos_i | i \leftarrow [0..|combos|], \forall j \leftarrow [0..|combos|], i \leq j \wedge$ 
if  $i < j$  then bebida( $combos_i$ )  $\neq$  bebida( $combos_j$ ) else True]|;
aux combosDistintos (cA: Combo, cB: Combo) : Bool = bebida(cA)  $\neq$  bebida(cB)  $\vee$  sandwich(cA)  $\neq$  sandwich(cB);
aux sandwichDistintos (combos: [Combo]) : [Hamburguesa] = [sandwich( $combos_i$ ) |  $i \leftarrow [0..|combos|]$ ,
 $\forall j \leftarrow [0..|combos|], i \leq j \wedge$  if  $i < j$  then sandwich( $combos_i$ )  $\neq$  sandwich( $combos_j$ ) else True];
aux bebidaDistintos (combos: [Combo]) : [Bebida] = [bebida( $combos_i$ ) |  $i \leftarrow [0..|combos|]$ ,  $\forall j \leftarrow [0..|combos|], i \leq j \wedge$ 
if  $i < j$  then bebida( $combos_i$ )  $\neq$  bebida( $combos_j$ ) else True];
aux mismosSandwich (combosA: [Combo], combosB: [Combo]) : Bool = |sandwichDistintos( $combosA$ )| ==
|sandwichDistintos( $combosB$ )|  $\wedge$  ( $\forall sandwichA \leftarrow$  sandwichDistintos( $combosA$ ), sandwichB  $\leftarrow$ 
sandwichDistintos( $combosB$ )) sandwichA == sandwichB;
aux mismosBebida (combosA: [Combo], combosB: [Combo]) : Bool = |bebidaDistintos( $combosA$ )| ==
|bebidaDistintos( $combosB$ )|  $\wedge$  ( $\forall bebidaA \leftarrow$  bebidaDistintos( $combosA$ ), bebidaB  $\leftarrow$  bebidaDistintos( $combosB$ ))
bebidaA == bebidaB;
```

### 5.3. Local

```
aux menorNumeroPedido (ventas: [Pedido]) :  $\mathbb{Z}$  = if /longitudventas > 1 then |[numero( $ventas_i$ ) |  $i \leftarrow [0..|ventas|]$ ,
 $\forall j \leftarrow [0..|ventas|], i \neq j \wedge$  numero( $ventas_i$ ) < numero( $ventas_j$ )]| else numero( $ventas_0$ );
aux mayorNumeroPedido (ventas: [Pedido]) :  $\mathbb{Z}$  = if /longitudventas > 1 then |[numero( $ventas_i$ ) |  $i \leftarrow [0..|ventas|]$ ,
 $\forall j \leftarrow [0..|ventas|], i \neq j \wedge$  numero( $ventas_i$ ) > numero( $ventas_j$ )]| else numero( $ventas_0$ );
aux sandwichLocal (ventas: [Pedido]) : [Hamburguesa] = [sandwichDistintos( $combos(ventas_i)$ ) |  $i \leftarrow [0..|ventas|]$ ,
 $\forall j \leftarrow [0..|sandwichDistintos( $combos(ventas_i)$ )|]|];
aux bebidaLocal (ventas: [Pedido]) : [Bebida] = [bebidaDistintos( $combos(ventas_i)$ ) |  $i \leftarrow [0..|ventas|]$ ,
 $\forall j \leftarrow [0..|bebidaDistintos( $combos(ventas_i)$ )|]|];
aux quienAtiendeVentasEmpleado (ventas: [Pedido], l: Local) : [Empleado] = [atendio( $ventas_i$ ) |  $i \leftarrow [0..|ventas|]$ ,
empleadosPerteneceLocal(atendio( $ventas_i$ ), l)];
aux empleadoPerteneceLocal (e: Empleado, l: Local) : Bool = ( $\exists i \leftarrow$  empleados(l))  $e == i$ ;
aux bebidaPerteneceLocal (b: Bebida, l: Local) : Bool = ( $\exists i \leftarrow$  bebidasDelLocal(l))  $e == i$ ;
aux sandwichPerteneceLocal (h: Hamburguesa, l: Local) : Bool = ( $\exists i \leftarrow$  sandwichesDelLocal(l))  $e == i$ ;
aux energiaPedido (p: Pedido) :  $\mathbb{Z}$  =  $\sum [dificultad( $combos(p)_i$ ) |  $i \leftarrow [0..|combos(p)|]$ ];
aux cuentaSandwich (h: Hamburguesa, combos: [Combo]) :  $\mathbb{Z}$  = |[sandwich(c) |  $c \leftarrow combos$ , sandwich(c) == h]|;
aux cuentaBebida (b: Bebida, combos: [Combo]) :  $\mathbb{Z}$  = |[bebida(c) |  $c \leftarrow combos$ , bebida(c) == b]|;
aux despedirEmpleadoLocal (l: Local, e: Empleado) : [Empleado] = [empleados(l)_i |  $i \leftarrow [0..|empleados(l)|]$ ,
empleados(l)_i  $\neq e$ ];
aux agregarDesempleadoLocal (l: Local, e: Empleado) : [Empleado] = [desempleados(l)] ++ [e];
aux mismosEmpleados (empleadosA: [Empleado], empleadosB: [Empleado]) : Bool = |empleadosA| ==
|empleadosB|  $\wedge$  ( $\forall empleadoA \leftarrow$  empleadosA, empleadoB  $\leftarrow$  empleadosB) empleadoA == empleadoB;
aux ventasEmpleadoPedidos (l: Local, e: Empleado) : [Pedido] = [ventas(l)_i |  $i \leftarrow [0..|ventas(l)|]$ ,
atendio( $ventas(l)_i$ ) == e];
aux cantidadCombosPedidos (pedidos: [Pedido]) :  $\mathbb{Z}$  = |[combos( $p_i$ ) |  $p \leftarrow pedidos, i \leftarrow [0..|combos(p)|]$ ]|;
aux mejoresEmpleados (l: Local) : [Empleado] = [e |  $e \leftarrow$  empleados(l),  $\neg((\exists m \leftarrow$  empleados(l))
(mejorEmpleado(m, e, l)))]];
aux mejorEmpleado (eA: Empleado, eB: Empleado, l: Local) : Bool = |ventasEmpleadoPedidos(l, eA)| >
|ventasEmpleadoPedidos(l, eB)|  $\vee$  (|ventasEmpleadoPedidos(l, eA)| == |ventasEmpleadoPedidos(l, eB)|  $\wedge$ 
cantidadCombosPedidos(ventasEmpleadoPedidos(l, eA)) >
cantidadCombosPedidos(ventasEmpleadoPedidos(l, eB)));
aux pedidoLocal (l: Local, n:  $\mathbb{Z}$ ) : Pedido = [p |  $p \leftarrow$  ventas(l), numero(p) == n]0;
aux esPedidoLocal (l: Local, n:  $\mathbb{Z}$ ) : Bool = ( $\exists p \leftarrow$  ventas(l), numero(p) == n);$$$ 
```

```

    aux pedidosIguales (pA: Pedido, cB: Pedido) : Bool = numero(pA) == numero(pB) ∧ atendio(pA) == atendio(pB)
    ∧ mismosCombosDePedidos(combos(pA), combos(pB));
    aux mismosCombosDePedidos (cA: [Combo], cB: [Combo]) : Bool = |cA| == |cB| ∧ (∀ combo ← cA)
    cuentaCombosDePedidos(combo, cA) == cuentaCombosDePedidos(combo, cB);
    aux cuentaCombosDePedidos (combo: Combo, combos: [Combo]) : ℤ = |[c | c ← combos, combosIguales(combo, c)]|;
    aux cantidadPedidosIguales (pedido: Pedido, pA: [Pedido]) : ℤ = |[p | p ← pA, pedidosIguales(p, pedido)]|;
    aux mismosPedidos (pA: [Pedido], pB: [Pedido]) : Bool = |pA| == |pB| ∧ (∀ p ← pA) cantidadPedidosIguales(p, pA)
    == cantidadPedidosIguales(p, pB);
    aux eliminarPedidoVentasNumero (l: Local, n: ℤ) : [Pedido] = [p | p ← ventas(l), numero(p) ≠ n];
    aux mismosPedidosNoNumero (pA: [Pedido], pB: [Pedido]) : Bool = |pA| == |pB| ∧ (∀ p ← pA)
    cantidadPedidosIgualesNoNumero(p, pA) == cantidadPedidosIgualesNoNumero(p, pB);
    aux cantidadPedidosIgualesNoNumero (pedido: Pedido, pA: [Pedido]) : ℤ = |[p | p ← pA,
    pedidosIgualesNoNumero(p, pedido)]|;
    aux pedidosIgualesNoNumero (pA: Pedido, cB: Pedido) : Bool = atendio(pA) == atendio(pB) ∧
    mismosCombosDePedidos(combos(pA), combos(pB));
    aux listaIndicesVentasEmpleados (e: Empleado, l: Local) : [ℤ] = [i | e ←
    [0..|quienAtiendeVentasEmpleado(ventas(l), l)|], quienAtiendeVentasEmpleado(ventas(l), l)i == e];
    aux listaDistanciaEntreIndices (e: Empleado, l: Local) : [ℤ] = if |listaIndicesVentasEmpleados(e, l) == 0|
    then |[0..|quienAtiendeVentasEmpleado(ventas(l), l)| - 1]| else [if i == -1
    then |[0..listaIndicesVentasEmpleados(e, l)0]| else if i == |listaIndicesVentasEmpleados(e, l)| - 1 then
    |[listaIndicesVentasEmpleados(e, l)i..|quienAtiendeVentasEmpleado(ventas(l), l)| - 1]|
    else [listaIndicesVentasEmpleados(e, l)i..listaIndicesVentasEmpleados(e, l)i+1]]
    |i ← [-1..|listaIndicesVentasEmpleados(e, l)|];
    aux mayorDistanciaEntreIndices (e: Empleado, l: Local) : ℤ = [listaDistanciaEntreIndices(e, l)i |
    ∀ i ← [0..|listaDistanciaEntreIndices(e, l)|], ∃ j ← [0..|listaDistanciaEntreIndices(e, l)|],
    listaDistanciaEntreIndices(e, l)i ≥ listaDistanciaEntreIndices(e, l)j]0;
    aux empleadoMayorDescanso (l: Local) : Empleado = [empleados(l)i | ∀ i ← (0..|empleados(l)|),
    ∃ j ← (0..|empleados(l)|), i ≠ j, mayorDistanciaEntreIndices(empleados(l)i, l) >
    mayorDistanciaEntreIndices(empleados(l)j, l)]0;

```