1. Tipo Vector$< T >$

Usaremos el tipo [T] (lista) para especificar la clase vector<T>.

```
problema nuevoV () = this : [T] {
        asegura |this| == 0;
}

problema size (this: [T]) = res : ℤ {
        asegura res == |this|;
}

problema empty (this: [T]) = res : Bool {
        asegura res == (|this| == 0);
}

problema at (this: [T], i: ℤ) = res : T {
        requiere 0 ≤ i < |this|;
        asegura res == this_i;
}
```

Nota: el operador [] cumple la misma especificación, también pueden usarlo.

```
problema push_back (this: [T], e: T) {
        modifica this;
        asegura this == pre(this) ++ [e];
}

problema pop_back (this: [T]) {
        requiere |this| > 0;
        modifica this;
        asegura pre(this) == this ++ [e];
}
```

2. Tipos

```
tipo Empleado = String;
tipo Energia = ℤ;
tipo Cantidad = 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;
```

3. Combo

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

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

problema Combo (b: Bebida, h: Hamburguesa, d: Energia, this: Combo) {
        requiere energiaEnRango(d);

        modifica this;

        asegura bebida(this) == b;
        asegura sandwich(this) == h;
        asegura dificultad(this) == d;
}

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

problema sandwichC (this: Combo) = res : Hamburguesa {
        asegura result == sandwich(this);
```

}

`problema dificultadC` (this: Combo) = `res` : Energia {
    `asegura` $result == dificultad(this)$;
}

4. Pedido

```
tipo Pedido {
        observador numero (p: Pedido) : ℤ;
        observador atendio (p: Pedido) : Empleado;
        observador combos (p: Pedido) : [Combo];

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

problema Pedido (n: ℤ, e: Empleado, cs: [Combo], this: Pedido) {
        requiere n > 0;
        requiere |cs| > 0;

        modifica this;

        asegura numero(this) == n;
        asegura atendio(this) == e;
        asegura combos(this) == cs;
}

problema numeroP (this: Pedido) = res : ℤ {
        asegura result == numero(this);
}

problema atendioP (this: Pedido) = res : Empleado {
        asegura result == atendio(this);
}

problema combosP (this: Pedido) = res : [Combo] {
        asegura result == combos(this);
}

problema dificultadP (this: Pedido) = res : Energia {
        asegura result == dificultad(this);
}

problema agregarComboP (this: Pedido, c: Combo) {
        modifica this;

        asegura numero(this) == numero(pre(this));
        asegura atendio(this) == atendio(pre(this));
        asegura combos(this) == combos(pre(this)) ++ [c];
}

problema anularComboP (this: Pedido, i:ℤ) {
        requiere 0 ≤ i < |combos(this)|;
        requiere |combos(this)| > 1;

        modifica this;

        asegura numero(this) == numero(pre(this));
        asegura atendio(this) == atendio(pre(this));
        asegura combos(this) == combos(pre(this))[0..i) ++ combos(pre(this))(i..|combos(pre(this))|);
}

problema cambiarBebidaComboP (this: Pedido, b: Bebida, i:ℤ) {
        requiere 0 ≤ i < |combos(this)|;

        modifica this;

        asegura numero(this) == numero(pre(this));
        asegura atendio(this) == atendio(pre(this));
        asegura |combos(this)| == |combos(pre(this))|;
        asegura (∀j ← [0..|combos(pre(this)), j ≠ i)combos(this)_j == combos(pre(this))_j;
        asegura cambiaSoloBebida(combos(this)_i, combos(pre(this))_i, b);
}

problema elMezcladitoP (this: Pedido) {
```

`requiere` $|combos(this)| \leq |quitarRepetidos(bebidasDelPedido(this))| * |quitarRepetidos(sandwichesDelPedido(this))|$;

`modifica` $this$;

`asegura` $numero(this) == numero(pre(this))$;
`asegura` $atendio(this) == atendio(pre(this))$;
`asegura` $|combos(this)| == |combos(pre(this))|$;
`asegura` $mismosIngredientes(combos(this), combos(pre(this)))$;
`asegura noHayRepetidos` : $(\forall i, j \leftarrow [0..combos(this))i \neq j)$
$bebida(combos(this)_i) \neq bebida(combos(this)_j) \vee sandwich(combos(this)_i) \neq sandwich(combos(this)_j)$;
`asegura sePareceAlPedidoOriginal` : $|[1|i \leftarrow [0..|combos(this)|), combos(this)_i == combos(pre(this))_i]|$
$== |combos(this)| - |[1|i \leftarrow [0..|combos(p)|), comboRepetido(combos(pre(this)), i)]|$;
}

5. Local

```
tipo Local {
        observador stockBebidas (l: Local, b: Bebida) : Cantidad ;
            requiere b ∈ bebidasDelLocal(l) ;
        observador stockSandwiches (l: Local, h: Hamburguesa) : Cantidad ;
            requiere h ∈ 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 ∈ empleados(l) ;
        observador ventas (l: Local) : [Pedido] ;

        invariante hayBebidasySonDistintas : |bebidasDelLocal(l)| > 0 ∧ distintos(bebidasDelLocal(l)) ;
        invariante haySandwichesySonDistintos : |sandwichesDelLocal(l)| > 0 ∧ distintos(sandwichesDelLocal(l)) ;
        invariante stockBebidasPositivo : (∀b ← bebidasDelLocal(l))stockBebidas(l, b) ≥ 0 ;
        invariante stockSandwichesPositivo : (∀h ← sandwichesDelLocal(l))stockSandwiches(l, h) ≥ 0 ;
        invariante empleadosDistintos : distintos(empleados(l) + +desempleados(l)) ;
        invariante energiaHasta100 : (∀e ← empleados(l))energiaEnRango(energiaEmpleado(l, e)) ;
        invariante empleadosQAtendieronDelLocal : (∀v ← ventas(l))atendio(v) ∈ empleados(l) + +desempleados(l) ;
        invariante ventasCorrelativas : distintos(nroPedidos(l)) ∧
            (∀n ← nroPedidos(l), n ≠ maximo(nroPedidos(l)))(n + 1) ∈ nroPedidos(l) ;
        invariante combosDeLocal : (∀v ← ventas(l))combosDelPedidoSonDelLocal(l, p) ;
}

problema Local (bs: [(Bebida,Cantidad)], hs: [(Hamburguesa,Cantidad)], es : [Empleado], this: Local)  {
        requiere |bs| > 0 ∧ |hs| > 0 ;
        requiere distintos(primeros(bs)) ;
        requiere distintos(primeros(hs)) ;
        requiere todosPositivos(segundos(bs)) ;
        requiere todosPositivos(segundos(hs)) ;
        requiere distintos(es) ;

        modifica this ;

        asegura mismos(bebidasDelLocal(this), primeros(bs)) ;
        asegura mismos(sandwichesDelLocal(this), primeros(hs)) ;
        asegura mismos(empleados(this), es) ;
        asegura desempleados(this) == [] ;
        asegura (∀e ← empleados(this))energiaEmpleado(this, e) == 100 ;
        asegura (∀b ← bebidasDelLocal(this))stockBebidas(this, b) == dameSegundo(bs, b) ;
        asegura (∀h ← sandwichesDelLocal(this))stockSandwiches(this, h) == dameSegundo(hs, h) ;
        asegura ventas(this) == [] ;
}

problema stockBebidasL (this: Local, b:Bebida) = res : Cantidad  {
        requiere b ∈ bebidasDelLocal(this) ;

        asegura result == stockBebidas(this, b) ;
}

problema stockSandwichesL (this: Local, h:Hamburguesa) = res : Cantidad  {
        requiere h ∈ sandwichesDelLocal(this) ;

        asegura result == stockSandwiches(this, h) ;
}

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

problema sandwichesDelLocalL (this: Local) = res : [Hamburguesa]  {
        asegura mismos(result, sandwichesDelLocal(this)) ;
}
```

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

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

problema energiaEmpleadoL (this: Local, e:Empleado) = res : Energia {
requiere $e \in empleados(this)$;
asegura $result == energiaEmpleado(this, e)$;
}

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

problema unaVentaCadaUnoL (this:Local) = res : Bool {
asegura $siguenSiempreElMismoOrden(empleadosDeVentasDeEmpleadosActivos(this))$;
}

problema venderL (this: Local, p:Pedido) {
requiere empleadoDelLocal : $atendio(p) \in empleados(this)$;
requiere combosSonDelLocal : $combosDelPedidoSonDelLocal(this, p)$;
requiere hayStockBebidas :
$(\forall b \leftarrow bebidasDelPedido(p))stockBebidas(this, b) \geq cuenta(b, bebidasDelPedido(p))$;
requiere hayStockSandwiches :
$(\forall s \leftarrow sandwichesDelPedido(p))stockSandwiches(this, s) \geq cuenta(s, sandwichesDelPedido(p))$;
requiere numeracionCorrelativa : $|ventas(this)| > 0 \rightarrow numero(p) == maxNroPedido(ventas(this)) + 1$;
modifica l;
asegura mismasBebidasDelLocal : $mismos(bebidasDelLocal(pre(this)), bebidasDelLocal(this))$;
asegura mismosSandwichesDelLocal : $mismos(sandwichesDelLocal(pre(this)), sandwichesDelLocal(this))$;
asegura modificaStockBebidas : $(\forall b \leftarrow bebidasDelLocal(this))stockBebidas(this, b) ==$
$stockBebidas(pre(l), b) - beta(puedeAtenderPedido(pre(this), p)) * cuenta(b, bebidasDelPedido(p))$;
asegura modificaStockSandwiches : $(\forall s \leftarrow sandwichesDelPedido(this))stockSandwiches(this, s) ==$
$stockSandwiches(pre(this), s) - beta(puedeAtenderPedido(pre(this), p)) * cuenta(s, sandwichesDelPedido(p))$;
asegura empleadoSinEnergia : $\neg puedeAtenderPedido(pre(this), p)) \rightarrow$
$mismos(sacar(empleados(pre(this)), atendio(p)), empleados(this)) \wedge$
$mismos(desempleados(pre(this)) + +[atendio(p)], desempleados(this))$;
asegura empleadoConEnergia : $puedeAtenderPedido(pre(this), p)) \rightarrow$
$mismos(empleados(pre(this)), empleados(this)) \wedge mismos(desempleados(pre(this)), desempleados(this))$;
asegura modificaEnergiaEmpleado1 :
$(\forall e \leftarrow empleados(this), e \neq atendio(p))energiaEmpleado(this, e) == energiaEmpleado(pre(this), e)$;
asegura modificaEnergiaEmpleado2 :
$puedeAtenderPedido(pre(this), p)) \rightarrow$
$energiaEmpleado(this, atendio(p)) == energiaEmpleado(pre(this), atendio(p)) - dificultad(p)$;
asegura modificaVentas : $puedeAtenderPedido(pre(this), p) \rightarrow$
$mismos(ventas(this), p : ventas(pre(this)))$;
asegura ventasIgual : $\neg puedeAtenderPedido(pre(this), p) \rightarrow$
$mismos(ventas(this), ventas(pre(this)))$;
}

problema candidatosAEmpleadosDelMesL (this: Local) = res : [Empleado] {
asegura $mismos(result, candidatosAEmpleadosDelMes(this))$;
}

problema sancionL (this: Local, e:Empleado, n:Energia) {
requiere $e \in empleados(this)$;
requiere $energiaEnRango(n)$;
modifica $this$;
asegura mismasBebidasDelLocal : $mismos(bebidasDelLocal(this), bebidasDelLocal(pre(this)))$;
asegura mismosSandwichesDelLocal : $mismos(sandwichesDelLocal(this), sandwichesDelLocal(pre(this)))$;
asegura modificaStockBebidas : $(\forall b \leftarrow bebidasDelLocal(this))stockBebidas(l, b) == stockBebidas(pre(this), b)$;

asegura modificaStockSandwiches :
$(\forall s \leftarrow sandwichesDelPedido(this))stockSandwiches(l, s) == stockSandwiches(pre(this), s)$;
asegura mismasVentas : $mismos(ventas(this), ventas(pre(this)))$;
asegura mismosEmpleados : $energiaEmpleado(pre(this), e) - n \geq 0 \rightarrow$
$mismos(empleados(this), empleados(pre(this))) \wedge mismos(desempleados(this), desempleados(pre(this)))$;
asegura despedirEmpleado : $energiaEmpleado(pre(this), e) - n < 0 \rightarrow$
$mismos(empleados(this), sacar(empleados(pre(this)), e)) \wedge mismos(desempleados(this), e : desempleados(pre(this)))$;
asegura mismaEnergiaEmpleadosAnteriores :
$(\forall e1 \leftarrow empleados(this), e1 \neq e)energiaEmpleado(this, e) == energiaEmpleado(pre(this), e)$;
asegura modificaEnergiaEmpleado :
$energiaEmpleado(pre(this), e) - n \geq 0 \rightarrow energiaEmpleado(this, e) == energiaEmpleado(pre(this), e) - n$;
}

problema elVagonetaL (this: Local) = res : Empleado {
requiere $|empleados(this)| > 0$;
asegura $result \in empleados(this) \wedge (\forall e \leftarrow empleados(this))$
$descansoMasLargo(this, e) \leq descansoMasLargo(this, result)$;
}

problema anularPedidoL (this: Local, n: $\mathbb{Z}$) {
requiere $empleadoQueAtendio(this, n) \in empleados(this)$;
requiere $(\exists p \leftarrow ventas(this))numero(p) == n$;
requiere $energiaEnRango(energiaEmpleado(this, empleadoQueAtendio(this, n))$
$+ dificultad(pedidosPorNro(this, n)))$;
modifica $this$;
asegura $mismos(bebidasDelLocal(this), bebidasDelLocal(pre(this)))$;
asegura $mismos(sandwichesDelLocal(this), sandwichesDelLocal(pre(this)))$;
asegura $(\forall b \leftarrow bebidasDelLocal(this))$
$stockBebidas(this, b) == stockBebidas(pre(this), b) + cuenta(b, bebidasDelPedido(pedidoPorNro(pre(this), n))$;
asegura $(\forall s \leftarrow sandwichesDelLocal(this))stockSandwiches(this, s) ==$
$stockSandwiches(pre(this), s) + cuenta(s, sandwichesDelPedido(pedidoPorNro(pre(this), n))$;
asegura $mismos(empleados(pre(this)), empleados(this))$;
asegura $mismos(desempleados(pre(this)), desempleados(this))$;
asegura $(\forall e \leftarrow empleados(this), e \neq empleadoQueAtendio(pre(this), n))$
$energiaEmpleado(this, e) == energiaEmpleado(pre(this), e)$;
asegura $energiaEmpleado(this, empleadoQueAtendio(pre(this), n)) ==$
$energiaEmpleado(pre(this), empleadoQueAtendio(pre(this), n)) + dificultad(pedidosPorNro(pre(this), n))$;
asegura $coincidenPedidos(pedidosOrdenados(this), [p | p \leftarrow pedidosOrdenados(pre(this)), numero(p) \neq n], numero(p))$;
} .

problema agregarComboAlPedidoL (this: Local, c: Combo, n:$\mathbb{Z}$) {
requiere $(\exists p \leftarrow ventas(this))numero(p) == n$;
requiere $atendio(pedidoPorNro(this, n)) \in empleados(this)$;
requiere $energiaEmpleado(this, atendio(pedidoPorNro(this, n)) \geq dificultad(c)$;
requiere $bebida(c) \in bebidasDelLocal(this) \wedge stockBebidas(this, bebida(c)) > 0$;
requiere $sandwich(c) \in sandwichesDelLocal(this) \wedge stockSandwiches(this, sandwich(c)) > 0$;
modifica $this$;
asegura mismasBebidasDelLocal : $mismos(bebidasDelLocal(pre(this)), bebidasDelLocal(this))$;
asegura mismosSandwichesDelLocal : $mismos(sandwichesDelLocal(pre(this)), sandwichesDelLocal(this))$;
asegura modificaStockBebidas : $(\forall b \leftarrow bebidasDelLocal(this), b \neq bebida(c))$
$stockBebidas(l, b) == stockBebidas(pre(this), b)$
$\wedge stockBebidas(this, bebida(c)) == stockBebidas(pre(this), bebida(c)) - 1$;
asegura modificaStockSandwiches :
$(\forall s \leftarrow sandwichesDelPedido(this), s \neq sandwich(c))stockSandwiches(this, s) == stockSandwiches(pre(this), s)$
$\wedge stockSandwiches(this, sandwich(c)) == stockSandwiches(pre(this), sandwich(c)) - 1$;
asegura mismosEmpleados : $mismos(empleados(this), empleados(pre(this)))$;
asegura mismosDesempleados : $mismos(desempleados(this), desempleados(pre(this)))$;
asegura mantienenEnergia : $(\forall e \leftarrow empleados(this), e \neq atendio(pedidoPorNro(pre(l), n)))energiaEmpleado(this, e) ==$
$energiaEmpleado(pre(this), e)$;
asegura elQueAtendioPierdeEnergia : $energiaEmpleado(this, atendio(pedidoPorNro(pre(this), n))) ==$
$energiaEmpleado(pre(this), atendio(pedidoPorNro(pre(this), n))) - dificultad(c)$;
asegura mismaCantidadPedidos : $|ventas(this)| == |ventas(pre(this))|$;
asegura losOtrosPedidosSeMantienen : $(\forall v \leftarrow ventas(pre(this)), numero(v) \neq n)(\exists w \leftarrow ventas(this))v == w$;

asegura seAgregaElComboAlPedido : $(\exists v \leftarrow ventas(this))numero(v) == n \wedge$
$atendio(v) == atendio(pedidoPorNro(pre(this), n)) \wedge combos(v) == combos(pedidoPorNro(pre(this), n)) +$
$+[c]$;
}

6. Funciones Auxiliares

aux dameSegundo (l:[<T1,T2>],elem:<T1>) : [T2] $= [sgd(x) | x \leftarrow l, prm(x) == elem]_0$;
aux distintos (ls:[T]) : Bool $= (\forall i, j \leftarrow [0..|ls|), i \neq j) ls_i \neq ls_j$;
aux energiaEnRango (e: Energia) : Bool $= 0 \leq e \leq 100$;
aux estaOrdenada (l:[T]) : [T] $= (\forall i \leftarrow [0..|l|-1)) l_i \leq l_{i+1}$;
aux incluido (es1,es2:[T]) : Bool $= (\forall e1 \leftarrow es1) e1 \in es2$;
aux incluidoConRepeticiones (c1:[T], c2:[T]) : Bool $=$
$(\forall x \leftarrow c1)(cuenta(x, c1) \leq cuenta(x, c2))$;
aux maximo (ls:[T]) : T $= [x | x \leftarrow ls, (\forall y \leftarrow ls), x \geq y)]_0$;
aux minimo (ls:[T]) : T $= [x | x \leftarrow ls, (\forall y \leftarrow ls), x \leq y)]_0$;
aux primeros (l:[<T1,T2>]) : [T1] $= [prm(x) | x \leftarrow l]$;
aux quitarRepetidos (l:[T]) : [T] $= [x_i | i \leftarrow [0..|l|), (\forall j \leftarrow (i..|l|)) x_i \neq x_j]$;
aux sacar (es:[T],e:T) : [T] $= [x | x \leftarrow es, x \neq e]$;
aux segundos (l:[<T1,T2>]) : [T2] $= [sgd(x) | x \leftarrow l]$;
aux sinRepetidos (xs: [T]) : Bool $= (\forall i, j \leftarrow [0..|xs|), i \neq j) xs_i \neq xs_j$;
aux todosPositivos (l:[Cantidad]) : Bool $= (\forall x \leftarrow l) x \geq 0$;

6.1. Combo

aux bebidas (cs: [Combo]) : [Bebida] $= [bebida(c) | c \leftarrow cs]$;
aux cambiaSoloBebida (cn, cv :Combo, b: Bebida) : Bool $=$
$sandwich(cn) == sandwich(cv) \land dificultad(cn) == dificultad(cv) \land bebida(cn) == b$;
aux dificultades (cs: [Combo]) : [Dificultad] $= [dificultad(c) | c \leftarrow cs]$;
aux mismosIngredientes (cns, cvs :[Combo]) : Bool $=$
$incluido(bebidas(cns), bebidas(cvs)) \land incluido(sandwiches(cns), sandwiches(cvs))$;
aux sandwiches (cs: [Combo]) : [Hamburguesa] $= [sandwich(c) | c \leftarrow cs]$;

6.2. Pedido

aux bebidasDelPedido (p:Pedido) : [Bebida] $= bebidas(combos(p))$;
aux comboRepetido (cs: [Combo], i:$\mathbb{Z}$) : Bool $= (\exists j \leftarrow [0..i)) sandwich(cs_i) == sandwich(cs_j) \land bebida(cs_i) == bebida(cs_j)$;
aux dificultad (p: Pedido) : $\mathbb{Z} = \sum dificultades(combos(p))$;
aux maxNroPedido (ps: [Pedido]) : $\mathbb{Z} = maximo([numero(p) | p \leftarrow ps])$;
aux minNroPedido (ps: [Pedido]) : $\mathbb{Z} = minimo([numero(p) | p \leftarrow ps])$;
aux sandwichesDelPedido (p:Pedido) : [Hamburguesa] $= [sandwich(c) | c \leftarrow combos(p)]$;

6.3. Local

aux bebidasAgotadas (l: Local, bs: [Bebida]) : [Bebida] $= [b | b \leftarrow bebidasDelLocal(l), stockBebidas(l, b) \leq cuenta(b, bs)]$;
aux candidatosAEmpleadosDelMes (l: Local) : [Empleado] $=$
$empleadosConMasCombos(l, (empleadosConMasVentas(l, empleados(l))))$;
aux cantHamburguesasVendidasDelEmpleado (l:Local, e:Empleado, h:Hamburguesa) : $\mathbb{Z} =$
$cuenta(h, [sandwich(c) | c \leftarrow combosVendidosPorElEmpleado(l, e)])$;
aux coincidenPedidos (p1:[Pedido], p2:[Pedido], n:$\mathbb{Z}$) : Bool $= |p1| == |p2|$
$\land ((\forall i \leftarrow [0..n)) atendio(p1_i) == atendio(p2_i) \land combos(p1_i) == combos(p2_i) \land numero(p1_i) == numero(p2_i))$
$\land ((\forall i \leftarrow [n..|p1|)) atendio(p1_i) == atendio(p2_i) \land combos(p1_i) == combos(p2_i) \land numero(p1_i) == numero(p2_i) - 1)$;
aux combosDelPedidoSonDelLocal (l:local,p:Pedido) : Bool $= (\forall c \leftarrow combos(p))$
$sandwich(c) \in sandwichesDelLocal(l) \land bebida(c) \in bebidasDelLocal(l)$;
aux combosVendidosPorElEmpleado (l:Local, e:Empleado) : [Combo] $=$
$concat[combos(p) | p \leftarrow ventas(l), atendio(p) == e]$;
aux descansoMasLargo (l:Local, e:Empleado) : $\mathbb{Z} = maximo(descansos(l, e))$;
aux descansos (l:Local, e:Empleado) : [$\mathbb{Z}$] $=$ if $noTrabajoNunca(l, e)$ then $[|ventas(l)|]$ else $[primerDescanso(l, e)]$
$++ descansosIntermedios(l, e) ++ [ultimoDescanso(l, e)]$;
aux descansosIntermedios (l:Local, e:Empleado) : [$\mathbb{Z}$] $=$
$[numero(pedidosOrdenadosDeEmpleado(l, e))_{i+1} - numero(pedidosOrdenadosDeEmpleado(l, e))_i |$
$i \leftarrow [0..|pedidosOrdenadosDeEmpleado(l, e)| - 1)]$;
aux empleadoQueAtendio (l: Local, n: $\mathbb{Z}$) : Empleado $= atendio(pedidoPorNro(l, n))$;
aux empleadosConMasCombos (l:Local, es:[Empleado]) : [Empleado] $=$
$[e1 | e1 \leftarrow es, (\forall e2 \leftarrow es) | combosVendidosPorElEmpleado(l, e1)| \geq |combosVendidosPorElEmpleado(l, e2)|]$;
aux empleadosConMasVentas (l:Local,es:[Empleado]) : [Empleado] $=$
$[e1 | e1 \leftarrow es, (\forall e2 \leftarrow es) | ventasDeEmpleado(l, e1)| \geq |ventasDeEmpleado(l, e2)|]$;

aux `empleadosDeVentasDeEmpleadosActivos` (l:Local) : [Empleado] = $[atendio(p)|p \leftarrow pedidosOrdenados(l), atendio(p) \in empleados(l)]$;

aux `empleadosVentas` (l:Local) : [Empleado] = $[atendio(p)|p \leftarrow pedidosOrdenados(l)]$;

aux `esLaHamburguesaMasVendidaDelEmpleado` (l:Local, e:Empleado, h:Hamburguesa) : Bool = $(\forall h1 \leftarrow sandwichesDelLocal(l))cantHamburguesasVendidasDelEmpleado(l, e, h) \geq cantHamburguesasVendidasDelEmpleado(l, e, h1)$;

aux `noTrabajoNunca` (l:Local, e:Empleado) : Bool = $|ventasDeEmpleado(l, e)| == 0$;

aux `nrosPedidos` (l:Local) : $[\mathbb{Z}]$ = $[numero(v)|v \leftarrow ventas(l)]$;

aux `pedidoPorNro` (l:Local, n:$\mathbb{Z}$) : Pedido = $[p|p \leftarrow ventas(l), numero(p) == n]_0$;

aux `pedidosOrdenados` (l: Local) : [Pedido] = $[pedidoPorNro(l, n)|n \leftarrow [minNroPedidos(ventas(l))..maxNroPedidos(ventas(l))]]$;

aux `pedidosOrdenadosDeEmpleado` (l:Local, e:Empleado) : [Pedido] = $[p|p \leftarrow pedidosOrdenados(l), atendio(p) == e]$;

aux `primerDescanso` (l:Local, e:Empleado) : $\mathbb{Z}$ = $numero(cab(pedidosOrdenadosDeEmpleado(l, e))) - minimo(nrosPedidos(l))$;

aux `puedeAtenderPedido` (l: Local, p: Pedido) : Bool = $energiaEmpleado(l, atendio(p)) - dificultad(p) \geq 0$;

aux `sandwichesAgotados` (l: Local, hs: [Hamburguesa]) : [Hamburguesa] = $[h|h \leftarrow sandwichesDelLocal(l), stockSandwiches(l, h) \leq cuenta(h, hs)]$;

aux `siguenSiempreElMismoOrden` (es:[Empleado]) : Bool = $(\forall i, j \leftarrow [0..|es| - 1), i < j \wedge es_i == es_j)es_{i+1} == es_{j+1}$;

aux `ultimoDescanso` (l:Local, e:Empleado) : $\mathbb{Z}$ = $maximo(nrosPedidos(l)) - numero(ultimo(pedidosOrdenadosDeEmpleado(l, e)))$;

aux `ventasDeEmpleado` (l:Local, e:Empleado) : [Pedido] = $[p|p \leftarrow ventas(l), atendio(p) == e]$;

aux `ventasDeEmpleadosActuales` (l:Local) : [Pedido] = $[p|p \leftarrow ventas(l), atendio(p) \in empleados(l)]$;