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	XXX@XXX.XXX
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

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` $d \geq 0 \land d \leq 100$;
`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) : ℤ ;
        observador atendio (p: Pedido) : Empleado ;
        observador combos (p: Pedido) : [Combo] ;

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

problema nuevoP (n: 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 : 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 (∀k ← [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:ℤ) {
        requiere enRango : 0 ≤ 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 (∀k ← [0..i))combosIguales(combos(p)_k, combos(pre(p))_k) ;
        asegura (∀k ← [i..|combos(p)|))combosIguales(combos(p)_k, combos(pre(p))_{k+1}) ;
}

problema cambiarBebidaComboP (p: Pedido, b: Bebida, i:ℤ) {
        requiere enRango : 0 ≤ i ≤ |combos(pre(p))| ;
        modifica p ;
        asegura numero(p) == numero(pre(p)) ;
        asegura atendio(p) == atendio(pre(p)) ;
        asegura |combos(p)| == |combos(pre(p))| ;
        asegura (∀k ← [0..i))combosIguales(combos(p)_k, combos(pre(p))_k) ;
        asegura (∀k ← [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))| ≤ (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 (∀i ← [0..|combos(p)|), ∀j ← [0..|combos(p)|))i ≠ j ∧ 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 ∈ 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 : |[menorNumeroPedido(ventas(l))..mayorNumeroPedido(ventas(l))]| ==
            |ventas(l)| ∧ (∀i ← [menorNumeroPedido(ventas(l))..mayorNumeroPedido(ventas(l))],
            j ← [0..|ventas(l)|))i == numero(ventas_j) ;
        invariante combosDeLocal : (∀i ← sandwichLocal(ventas(l)), j ← sandwichesDelLocal(l))i == j ∧
            (∀i ← bebidaLocal(ventas(l)), j ← 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 {
asegura $res == lista$;
}

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

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

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

problema elVagonetaL (l: Local) = res : Empleado {
}

problema anularPedidoL (l: Local, n: $\mathbb{Z}$) {
requiere $esPedidoLocal(\mathsf{pre}(l), n)$;
requiere empleadoDelLocal : $empleadoPerteneceLocal(atendio(pedidoLocal(\mathsf{pre}(l), n)), \mathsf{pre}(l))$;
modifica l ;
asegura $mismos(bebidasDelLocal(l), bebidasdelLocal(\mathsf{pre}(l)))$;
asegura $mismos(sandwichesDelLocal(l), sandwichesDelLocal(\mathsf{pre}(l)))$;
asegura $mismosEmpleados(empleados(l), empleados(\mathsf{pre}(l)))$;

asegura $mismosEmpleados(desempleados(l), desempleados(\text{pre}(l)))$;
asegura $|ventas(l)| == |ventas(\text{pre}(l))| - 1$;
asegura $(\forall b \leftarrow bebidaDistintos(combos(pedidoLocal(\text{pre}(l), n))))(stockBebidas(\text{pre}(l), b) + cuentaBebida(b, combos(pedidoLocal(\text{pre}(l), n)))) == stockBebidas(l, b)$;
asegura $(\forall h \leftarrow sandwichDistintos(combos(pedidoLocal(\text{pre}(l), n))))(stockSandwiches(\text{pre}(l), h) - cuentaSandwich(h, combos(pedidoLocal(\text{pre}(l), n)))) == stockSandwiches(l, b)$;
asegura $(\forall e \leftarrow empleados(\text{pre}(l)), atendio(pedidoLocal(\text{pre}(l), n)) \neq e) energiaEmplado(\text{pre}(l), e) == energiaEmpleado(l, ee)$;
asegura $(energiaEmpleado(\text{pre}(l), atendio(pedidoLocal(\text{pre}(l), n))) + energiaPedido(pedidoLocal(\text{pre}(l), n))) == energiaEmpleado(l, atendio(pedidoLocal(\text{pre}(l), n)))$;
asegura $mayorNumeroPedido(ventas(\text{pre}(l))) - 1 == mayorNumeroPedido(ventas(l))$;
asegura $mismosPedidosNoNumero(ventas(l), eliminarPedidoVentasNumero(\text{pre}(l), n))$;
asegura $(\forall i \leftarrow |ventas(\text{pre}(l))|, numero(ventas(\text{pre}(l))_i) \neq n)$ if $numero(ventas(\text{pre}(l))_i) > n$ then $numero(ventas(\text{pre}(l))_i) - 1 == numero(ventas(l)_i)$ else $numero(ventas(\text{pre}(l))_i) == numero(ventas(l)_i)$;
}

problema agregarComboAlPedidoL (l: Local, c: Combo, n:$\mathbb{Z}$) {
}

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]) : $\mathbb{Z} = |[combos_i | i \leftarrow [0..|combos|), \forall j \leftarrow [0..|combos|), i \leq j \wedge \text{if } i < j \text{ then } sandwich(combos_i) \neq sandwich(combos_j) \text{ else True}]|$;
aux cantidadBebidaDistintos (combos: [Combo]) : $\mathbb{Z} = |[combos_i | i \leftarrow [0..|combos|), \forall j \leftarrow [0..|combos|), i \leq j \wedge \text{if } i < j \text{ then } bebida(combos_i) \neq bebida(combos_j) \text{ else True}]|$;
aux combosDistintos (cA: Combo, cB: Combo) : Bool $= bebida(cA) \neq bebida(cB) \vee sandwich(cA) \neq sandwich(cB)$;
aux sandwichDistintos (combos: [Combo]) : [Hamburguesa] $= [sandwich(combos_i) | i \leftarrow [0..|combos|), \forall j \leftarrow [0..|combos|), i \leq j \wedge \text{if } i < j \text{ then } sandwich(combos_i) \neq sandwich(combos_j) \text{ else True}]$;
aux bebidaDistintos (combos: [Combo]) : [Bebida] $= [bebida(combos_i) | i \leftarrow [0..|combos|), \forall j \leftarrow [0..|combos|), i \leq j \wedge \text{if } i < j \text{ then } bebida(combos_i) \neq bebida(combos_j) \text{ else True}]$;
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]) : $\mathbb{Z} = \text{if } /longitudventas > 1 \text{ then } |[numero(ventas_i) | i \leftarrow [0..|ventas|), \forall j \leftarrow [0..|ventas|), i \neq j \wedge numero(ventas_i) < numero(ventas_j)]| \text{ else } numero(ventas_0)$;
aux mayorNumeroPedido (ventas: [Pedido]) : $\mathbb{Z} = \text{if } /longitudventas > 1 \text{ then } |[numero(ventas_i) | i \leftarrow [0..|ventas|), \forall j \leftarrow [0..|ventas|), i \neq j \wedge numero(ventas_i) > numero(ventas_j)]| \text{ else } numero(ventas_0)$;
aux sandwichLocal (ventas: [Pedido]) : [Hamburguesa] $= [sandwichDistintos(combos(ventas_i))_j | i \leftarrow [0..|ventas|), \forall j \leftarrow [0..|sandwichDistintos(combos(ventas_i))|)]$;
aux bebidaLocal (ventas: [Pedido]) : [Bebida] $= [bebidaDistintos(combos(ventas_i))_j | i \leftarrow [0..|ventas|), \forall j \leftarrow [0..|bebidaDistintos(combos(ventas_i))|)]$;
aux quienAtiendeVentasEmpleado (ventas: [Pedido], l: Local) : [Empleado] $= [atendio(ventas_i) | i \leftarrow [0..|ventas|), empleadoPerteneceLocal(atendio(ventas_i), l)]$;
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: Hamburgesa, 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] $= [empleados(l)_i | i \leftarrow |empleados(l)|, empleados(l)_i \neq e]$;
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] $= [ventas(l)_i | i \leftarrow |ventas(l)|, atendio(ventas(l)_i) == e]$;
aux cantidadCombosPedidos (pedidos:[Pedido]) : $\mathbb{Z} = |[combos(p)_i | p \leftarrow pedidos, i \leftarrow |combos(p)|]|$;
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 $= numero(pA) == numero(pB) \wedge atendio(pA) == atendio(pB) \wedge mismosCombosDePedidos(combos(pA), combos(pB))$;

`aux mismosCombosDePedidos` (cA: [Combo], cB: [Combo]) : Bool $= |cA| == |cB| \wedge (\forall combo \leftarrow cA)\ cuentaCombosDePedidos(combo, cA) == cuentaCombosDePedidos(combo, cB)$;

`aux cuentaCombosDePedidos` (combo: Combo, combos: [Combo]) : $\mathbb{Z} = |[c|c \leftarrow combos, combosIguales(combo, c)]|$;

`aux cantidadPedidosIguales` (pedido: Pedido, pA: [Pedido]) : $\mathbb{Z} = |[p|p \leftarrow pA, pedidosIguales(p, pedido)]|$;

`aux mismosPedidos` (pA: [Pedido], pB: [Pedido]) : Bool $= |pA| == |pB| \wedge (\forall p \leftarrow pA) cantidadPedidosIguales(p, pA) == cantidadPedidosIguales(p, pB)$;

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

`aux mismosPedidosNoNumero` (pA: [Pedido], pB: [Pedido]) : Bool $= |pA| == |pB| \wedge (\forall p \leftarrow pA)\ cantidadPedidosIgualesNoNumero(p, pA) == cantidadPedidosIgualesNoNumero(p, pB)$;

`aux cantidadPedidosIgualesNoNumero` (pedido: Pedido, pA: [Pedido]) : $\mathbb{Z} = |[p|p \leftarrow pA, pedidosIgualesNoNumero(p, pedido)]|$;

`aux pedidosIgualesNoNumero` (pA: Pedido, cB: Pedido) : Bool $= atendio(pA) == atendio(pB) \wedge mismosCombosDePedidos(combos(pA), combos(pB))$;