DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 1 - Especificación

Fat Food

6 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 : ... ;
        invariante combosDeLocal : ... ;
}

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

problema stockSandwichesL (l: Local, h:Hamburguesa) = res : Cantidad {
}

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

problema sandwichesDelLocalL (l: Local) = res : [Hamburguesa] {
}

problema empleadosL (l: Local) = res : [Empleado] {
}

problema desempleadosL (l: Local) = res : [Empleado] {
}

problema energiaEmpleadoL (l: Local, e:Empleado) = res : Energia {
}

problema ventasL (l: Local) = res : [Pedido] {
}

problema unaVentaCadaUno (l:Local) = res : Bool {
}

problema venderL (l: Local, p:Pedido) {
}

problema candidatosAEmpleadosDelMesL (l: Local) = res : [Empleado] {
}

problema sancionL (l: Local, e:Empleado, n:Energia) {
```

}

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

`problema anularPedidoL` (l: Local, n: $\mathbb{Z}$) {
}

`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] $= [combos_i | i \leftarrow [0..|combos|), \forall j \leftarrow [0..|combos|), i \leq j \wedge \text{if } i < j \text{ then } sandwich(combos_i) \neq sandwich(combos_j) \text{ else True}]$;
aux `bebidaDistintos` (combos: [Combo]) : [Bebida] $= [combos_i | i \leftarrow [0..|combos|), \forall j \leftarrow [0..|combos|), i \leq j \wedge \text{if } i < j \text{ then } bebida(combos_i) \neq bebida(combos_j) \text{ else True}]$;
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