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 {
}

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

`problema sandwichC` (c: Combo) = `res` : Hamburguesa {
}

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

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 {
}

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

problema atendioP (p: Pedido) = res : Empleado {
}

problema combosP (p: Pedido) = res : [Combo] {
}

problema agregarComboP (p: Pedido, c: Combo) {
}

problema anularComboP (p: Pedido, i:ℤ) {
        modifica p ;
        requiere iEnRango : 0 < i ≤ numero(p) ;
        asegura numero(p) = numero(pre(p)) − 1 ;
        asegura combos(p) = [combos(pre(p))_j | ∀j ← [0..|combos(pre(p))|), j ≠ i] ;
}

problema cambiarBebidaComboP (p: Pedido, b: Bebida, i:ℤ) {
}

problema elMezcladitoP (p: Pedido) {
}
```

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 : (∀v ← ventas(l))(∀c ← combos(v))bebida(c) ∈ bebidasDelLocal(l) ∧ sandwich(c) ∈
            sandwichesDelLocal(l) ;
}

problema stockBebidasL (l: Local, b:Bebida) = res : Cantidad  {
}

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

problema bebidasDelLocalL (l: Local) = res : [Bebida]  {
}

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

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]  {
        asegura res = ventas(l) ;
}

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)  {
        modifica l ;
        requiere e ∈ empleados(l) ;
        requiere energiaEnRango(n) ;
```

asegura $if\ energiaEnRango(energiaEmpleado(pre(l), e) - n)\ then\ energiaEmpleado(l, e) = energiaEmpleado(pre(l), e) - n\ else\ mismos(desempleados(l), desempleados(pre(l)) + +e) \wedge mismos(empleados(l), sancionarE(pre(l), e))$;
}

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

5.3. Local

`aux sancionarE` (l: Local,e: Empleado) : [Empleados] $= [empleados(l)_i \mid \forall i \leftarrow [0..|empleados(l)|), empleados(l)_i \neq e]$;