Universidad de Huelva

MFIS

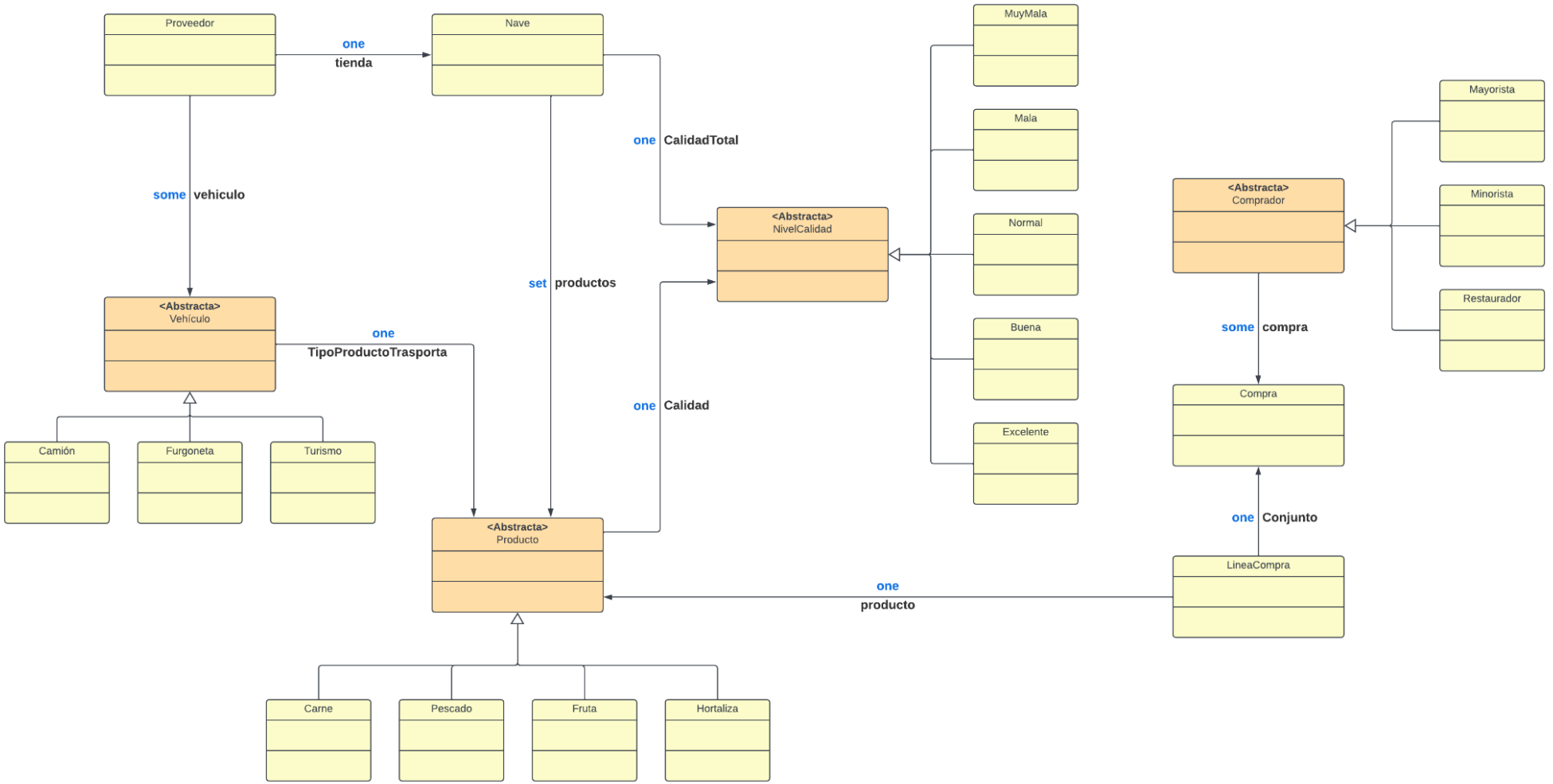
Práctica 2

Autores:

- Ismael Da Palma Fernández
- Cristian Delgado Cruz
- Juan Jimenez Serrano
- Fernando García-Palomo Albarrán

Índice

1.Diagrama de Clases



2. Cambios realizados y sus soluciones

- Todos los atributos usados en OCL no se pueden implementar ni usar en ALLOY por lo que algunas invariantes en las que usábamos atributos tendremos que rediseñarlas o bien implementar unas nuevas.

- Todos los **enumerados** se han convertido en superclases y los diferentes **estados** en subclases.

- Se han creado tres predicados, dos de ellos funcionan de forma similar a métodos y el otro es una ampliación de los facts de LineadeCompra.

- La **composición** entre Compra y LineadeCompra la sustituimos por una relación *'some'* y obligando a LineadeCompra a tener únicamente una asociación en *'Conjunto'*.

- Hemos cambiado la invariante que indicaba cuando un comprador realiza un pedido de más de 50 productos se le consideraba Mayorista a que si tiene más de 5 Compras se le considera Mayorista.

- Hemos cambiado una restricción de diseño, en la práctica anterior, todas las compras eran solo de un tipo, ahora tenemos un *fact* que controla esto para que todas las compras sean iguales, si y sólo si el comprador es Mayorista. En otro caso, este *fact* no influye para nada.

- Algunas invariantes se han podido mantener, aunque hayan cambiado su forma de expresión, por ejemplo:
    - Un mayorista solo puede comprar productos de un solo tipo.
    - Cada proveedor tiene como mínimo un camión.
    - Todos los productos de una nave deben de ser del mismo tipo.
    - ...

3. Clases

```
//Clases
sig Proveedor{tienda : one Nave, vehiculo : some Vehiculo}
//Abstracta
abstract sig Vehiculo{TipoProductoTrasporta : one Producto}
//Subclases - Enumerado
sig Camion extends Vehiculo{}
sig Furgoneta extends Vehiculo{}
sig Turismo extends Vehiculo{}
//------
//Abstracta
abstract sig NivelCalidad{}
//Subclase - Enumerado
sig Excelente extends NivelCalidad{}
sig Buena extends NivelCalidad{}
sig Normal extends NivelCalidad{}
sig Mala extends NivelCalidad{}
sig MuyMala extends NivelCalidad{}
//------
sig Nave{CalidadTotal : one NivelCalidad ,productos : set Producto}

//Abstracta
abstract sig Producto{Calidad : one NivelCalidad}
//Subclase - Enumerado
sig Pescado extends Producto{}
sig Carne extends Producto{}
sig Fruta extends Producto{}
sig Hortalizas extends Producto{}
//------
sig LineadeCompra{Conjunto : some Compra, producto : one Producto}
sig Compra{}
//Abstracta
abstract sig Comprador{compra : some Compra}
//Subclases - Enumerados
sig Mayorista extends Comprador{}
sig Minorista extends Comprador{}
sig Restaurador extends Comprador{}
```

4. Facts

Para que funcionen las clases abstractas:

```
//Restricciones de Diseño
fact{
	all v: Vehiculo | v in Camion + Furgoneta + Turismo
	all p: Producto | p in Pescado + Carne + Fruta + Hortalizas
	all n: NivelCalidad | n in MuyMala + Mala + Normal + Buena + Excelente
}
```

Nave:

```
fact{ //Nave
	all disj nl: Nave | //Todos los productos de una nave deben ser de un tipo
		#(nl.productos) > 0 implies
			nl.productos in Pescado or  nl.productos in Carne or nl.productos in Fruta or nl.productos in Hortalizas

	all disj nl,n2: Nave | //los productos de una nave no estan en otra
		nl.productos not in n2.productos

}
```

LineadeCompra:

```
fact{//Linea de Compra

	all disj cl,c2: LineadeCompra| //Todos los productos de todas las lineas de compra deben ser del mismo tipo si es de tipo mayorista el comprador
		(one mayo: Mayorista |  mayo.compra = cl.Conjunto + c2.Conjunto) implies
		((cl.producto in Pescado and c2.producto in Pescado) or
		(cl.producto in Carne and c2.producto in Carne) or (cl.producto in Fruta and c2.producto in Fruta)
		or (cl.producto in Hortalizas and c2.producto in Hortalizas))

	all disj cl,c2: LineadeCompra | //Cada linea de compra si es de una misma compra no puede tener el mismo producto
		cl.Conjunto = c2.Conjunto implies cl.producto not in c2.producto

	all cl: LineadeCompra |//Cada linea de compra esta en una unica compra (Composicion)
		#cl.Conjunto = 1

	all cl: LineadeCompra |//Cada producto de una linea de compra debe estar en una nave
		(some nl : Nave | cl.producto in nl.productos)

}
```

Compra:

```
fact{//Compra
    all cl : Compra | //Cada compra pertenece a una linea de compra
        (some lc: LineadeCompra | cl in lc.Conjunto)
    all cl: Compra |//Cada compra debe tener solo un unico comprador
        (one col: Comprador | cl in col.compra)

}
```

Comprador:

```
fact{//Comprador
    all disj compl,comp2: Comprador | //Cada compra solo es de un comprador
        compl.compra not in comp2.compra
    all compl : Comprador |//Cada comprador que haga más de 5 compras será Mayorista.
        #(compl.compra) > 5 implies
        compl in Mayorista
}
```

Vehículo:

```
fact{//Vehiculo
    all vl: Vehiculo |//Cada vehiculo tiene solo un unico proveedor
        (one pl: Proveedor | vl in pl.vehiculo)
}
```

Proveedor:

```
fact{//Proveedor
    all pl: Proveedor |//Cada vehiculos de proveedor transportan un unico tipo de producto
        pl.vehiculo.TipoProductoTrasporta in Pescado or pl.vehiculo.TipoProductoTrasporta in Carne
        or pl.vehiculo.TipoProductoTrasporta in Fruta or pl.vehiculo.TipoProductoTrasporta in Hortalizas
    all pl: Proveedor | //Cada Proveedor tiene minimo un Camión
        (one cam: Camion | cam in pl.vehiculo)
}
```

5. Predicados

Método que elimina productos malos de la nave:

```
pred EliminarProductoMalo(n,n': Nave , p : Producto ){
    p in n.productos and (p.Calidad in MuyMala or p.Calidad in Mala) implies
    n'.productos = n.productos  - p
}
```

Método que añade productos nuevos a la nave desde un vehículo:

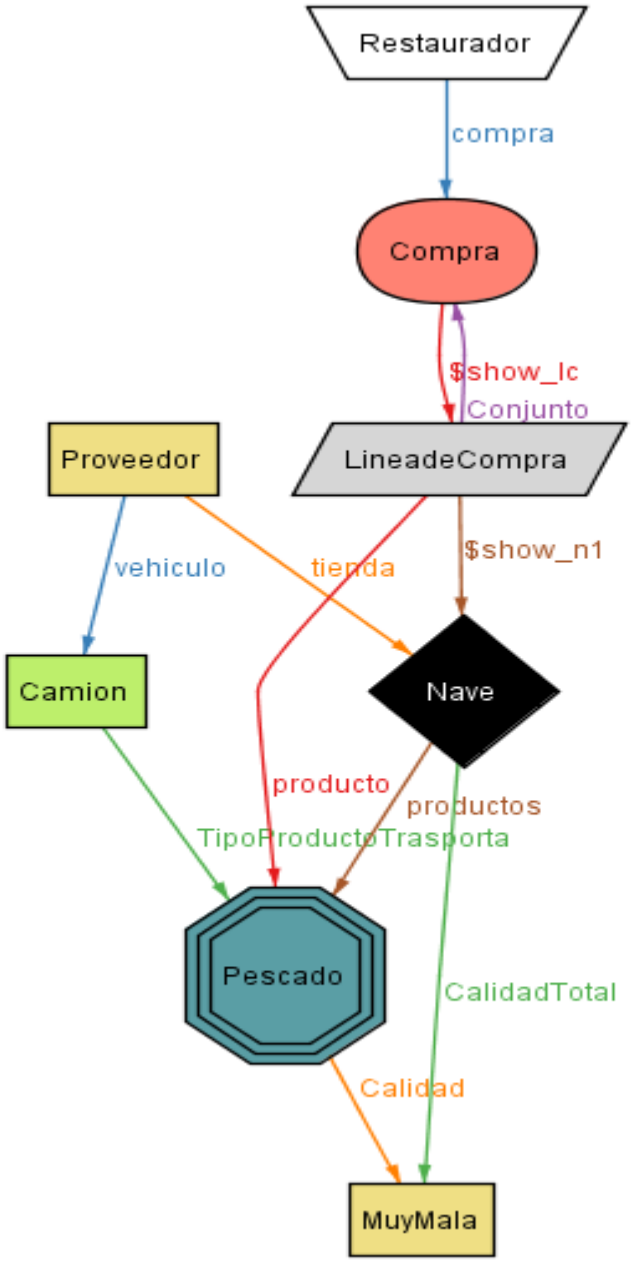
```
pred AnadirProductoNuevo(n,n': Nave, v : Vehiculo){
    v.TipoProductoTrasporta not in n.productos and (some nx : Nave | v.TipoProductoTrasporta not in nx.productos) implies
    n'.productos = n.productos + v.TipoProductoTrasporta
}
```

Expande los Facts de LineadeCompra:

```
pred LineadeCompraProductoBueno(){
    all ln: LineadeCompra |
        ln.producto.Calidad in Buena or ln.producto.Calidad in Normal or ln.producto.Calidad in Excelente
}
```

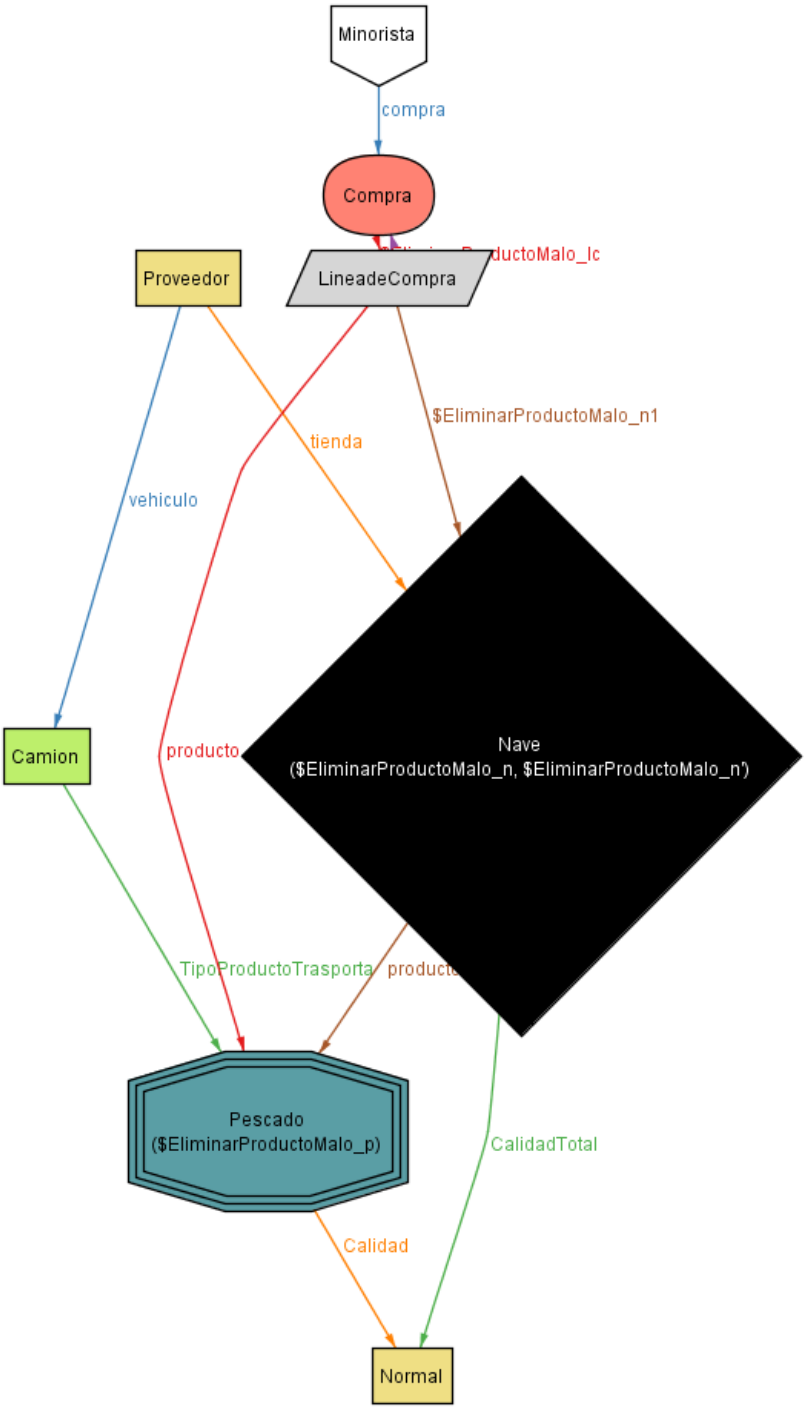
6. Pruebas de verificación

Si utilizamos el run del predicado show, pidiendo exactamente una instancia de cada tipo nos sale el siguiente esquema:



EliminarProductoMalo

Si utilizamos el run del predicado **EliminarProductoMalo**, pidiendo exactamente una instancia de cada tipo nos sale el siguiente esquema:

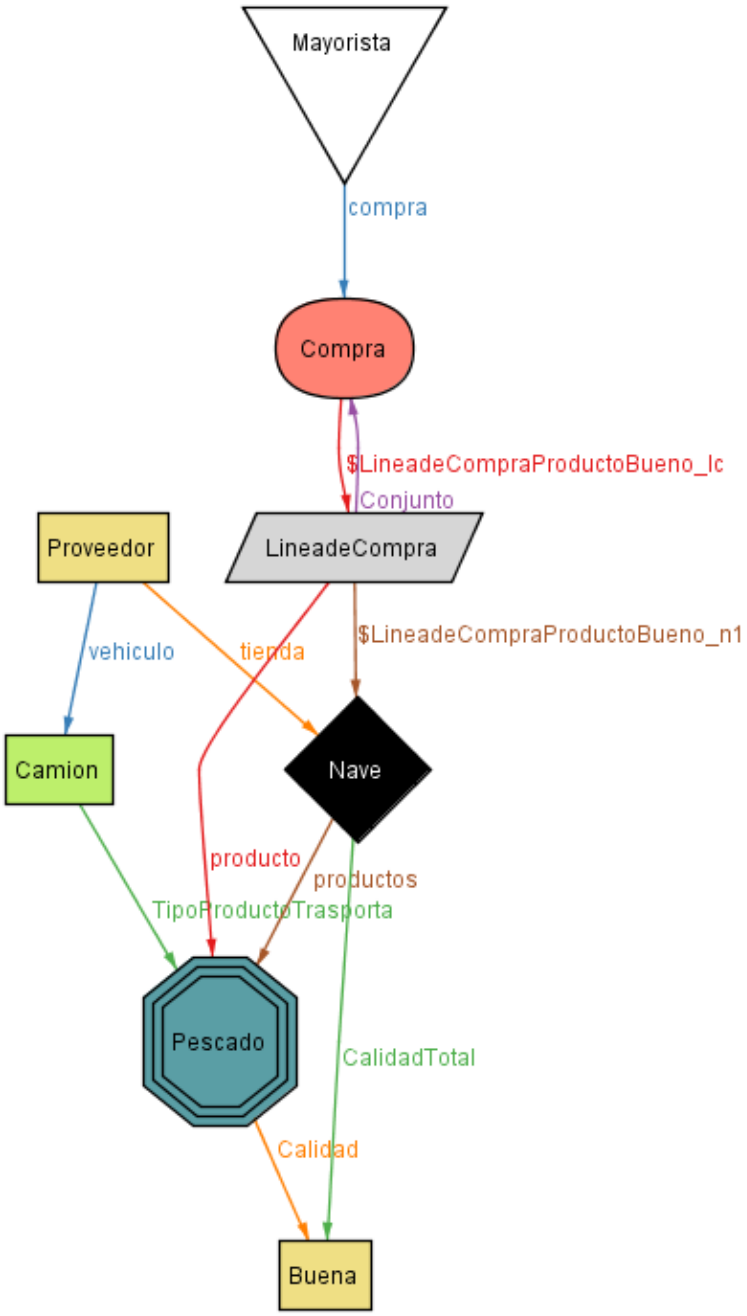


AnadirProductoNuevo

Si utilizamos el run del predicado **AnadirProductoNuevo**, pidiendo exactamente una instancia de cada tipo nos sale el siguiente esquema:

LineadeCompraProductoBueno

Si utilizamos el run del predicado **LineadeCompraProductoBueno**, pidiendo exactamente una instancia de cada tipo nos sale el siguiente esquema, podemos comprobar que solo saldrían conjuntos de instancias con calidad Normal Buena o Excelente:



Sobre los facts tenemos algunos ejemplos:

Si creamos dos compras y una única línea de compra, claramente no se podrá crear, porque cada línea de compra solo puede estar en una compra

```
Executing "Run show for exactly 1 Proveedor, exactly 1 Vehiculo, exactly 1 NivelCalidad, exactly 1 Nave, exactly 1 Producto, exactly 1 LineadeCompra, exactly 2 Compra, exactly 1 Comprador"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
   139 vars. 29 primary vars. 166 clauses. 8ms.
   No instance found. Predicate may be inconsistent. 0ms.
```

Si creamos dos compradores y una única compra, ocurre lo mismo

```
Executing "Run show for exactly 1 Proveedor, exactly 1 Vehiculo, exactly 1 NivelCalidad, exactly 1 Nave, exactly 1 Producto, exactly 1 LineadeCompra, exactly 1 Compra, exactly 2 Comprador"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
   192 vars. 34 primary vars. 248 clauses. 4ms.
   No instance found. Predicate may be inconsistent. 0ms.
```

Si creamos dos naves y un único producto, también falla porque cada producto está en una única nave.

```
Executing "Run show for exactly 1 Proveedor, exactly 1 Vehiculo, exactly 1 NivelCalidad, exactly 2 Nave, exactly 1 Producto, exactly 1 LineadeCompra, exactly 1 Compra, exactly 1 Comprador"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
   152 vars. 30 primary vars. 193 clauses. 0ms.
   No instance found. Predicate may be inconsistent. 0ms.
```

Utilizando assert tenemos algunos ejemplos:

```
check VehiculosDobles{//Un vehiculo pertenece a más de 1 persona
      all disj v1 : Vehiculo, p1,p2 : Proveedor|
      v1 in p1.vehiculo and v1 in p2.vehiculo }
```

```
Executing "Check VehiculosDobles"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
   2266 vars. 165 primary vars. 3659 clauses. 8ms.
   Counterexample found. Assertion is invalid. 8ms.
```

```
check MayoristaCompraDosCosasDiferentes{//Un Mayorista tiene una compra de un tipo de producto y otra de otro tipo.
        all disj c1,c2: LineadeCompra|
        (one mayo: Mayorista |  mayo.compra = c1.Conjunto + c2.Conjunto) implies
        ((c1.producto in Pescado and c2.producto not in Pescado) or (c1.producto in Carne and c2.producto not in Carne)
        or (c1.producto in Hortalizas and c2.producto not in Hortalizas) or (c1.producto in Fruta and c2.producto not in Fruta))
}
```

```
Executing "Check MayoristaCompraDosCosasDiferentes"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
   2344 vars. 162 primary vars. 3880 clauses. 4ms.
   Counterexample found. Assertion is invalid. 8ms.
```

```
check{//Existe un producto que no esta en una nave ni en un coche
      all disj v: Vehiculo, n: Nave |
            (one  p: Producto | p not in v.TipoProductoTrasporta and p not in n.productos)
}
```

```
Executing "Check check$1"
   Solver=sat4j Bitwidth=0 MaxSeq=0 SkolemDepth=1 Symmetry=20
   2248 vars. 162 primary vars. 3683 clauses. 8ms.
   Counterexample found. Assertion is invalid. 8ms.
```

7. Conclusión

Tal y como la práctica anterior, nuestro trabajo está basado en una interpretación del sistema ya definida, creada desde nuestra propia perspectiva sobre nuestro diseño anterior, sobre esta hemos actualizado varias invariantes a algunos facts, pero otros no han podido ser correspondidos en alloy, concluyendo así que en, en cuanto al diseño y control de asociaciones el uso de alloy es mucho más óptimo, análogamente si queremos una mayor exactitud al controlar entre otras cosas los atributos y métodos de esas clases, USE será siempre la herramienta más operativa.