Examples of the translation of several CDs elements

In this document, we provide examples of the translation proposal we give for different CD's elements, in particular:

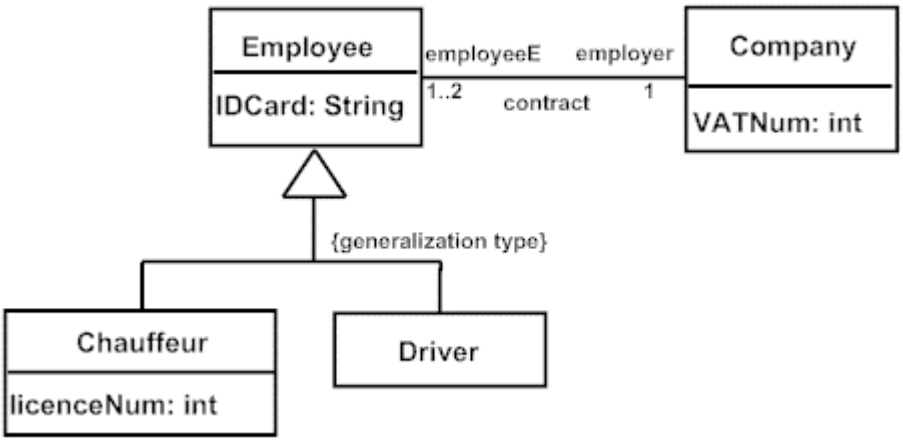
- simple inheritance
- multiple inheritance
- (strong) composition

The class diagrams used in this document and their Formula translations are available as .uml and in .4ml extension files in the *Eclipse project* `Examples` of [CD2FProject].

1. Formula representation of simple generalizations

Next, we present a same class diagram scenario which includes a simple generalization relationship. In particular, with this example we want to show the way in which we represent (simple) inheritance in Formula, taking into account the different generalization types [OMGS]:

- {**complete, disjoint**} - Indicates the generalization set is covering and its specific Classifiers have no common instances --> we identify our CD with this type of generalization as **CDCoDi.**
- **{incomplete, disjoint**} - Indicates the generalization set is not covering and its specific Classifiers have no common instances. (it is the default option) --> we identify our CD with this type of generalization as **CDInDi**
- {**complete, overlapping**} - Indicates the generalization set is covering and its specific Classifiers do share common instances.--> we identify our CD with this type of generalization as **CDCoOv.**
- {**incomplete, overlapping**} - Indicates the generalization set is not covering and its specific Classifiers do share common instances. --> we identify our CD with this type of generalization as **CDInOv.**



We divide the explanation in **two sections**: (1) The Formula code instructions corresponding to the representation of the four CDs in Formula (**Formula code instructions**), (2) the

Formula instances resulted from the application of the Formula solver to the previous Formula code instructions (**Formula generated facts**).

At the same time, since the translation of the four different versions of the class diagram shares most of the Formula code, we have divided the first section into other two subsections: the **Formula code instructions common** to the **four versions** of the CD, divided by levels, and (2) The **Formula instructions specific** to each CD. In particular, we do not include the instructions at level M2 because they are the same whatever class diagram is considered and we do not want to sound repetitive.

We have shaded in **light blue** the instructions corresponding to the elements that directly participate in the generalization relationship, while we represent in **dark blue** the parts of the code that depend on the specific generalization set type.
Later, we provide the Formula code instructions specific to each type of generalization set and which would have to be included in the corresponding “dark blue” parts.

Later in this section, we include another two CD examples with:

- An inheritance relationship with a generalization set {complete, disjoint} including three subclasses. --> we identify our CD with this type of generalization as **CDCoDiThreeSubclasses.**
- An inheritance relationship with two generalization sets {complete, disjoint} and two subclasses each one. --> we identify our CD with this type of generalization as **CDCoDiTwoGeneralizationSetsThreeAndTwoSubclasses.**

In this case, we just focus on the instructions of their Formula representation related to inheritance.

A. Formula code instructions

a. Formula code instructions common to the four CD

M1

ClassDiagram model

```
classCompany is Class("Company", false)
classEmployee is Class("Employee", false)
classDriver is Class("Driver", false)
classChauffeur is Class("Chauffeur", false)
contract is Association("contract",
classEmployee, 1, 2, classCompany, 1, 1)
VATNumCompany is IntegerProperty("VATNum", 0, 1, 1)
HasProperty(classCompany,VATNumCompany)
IDCardEmployee is StringProperty("IDCard", "", 1, 1)
HasProperty(classEmployee,IDCardEmployee)
nameChauffeur is StringProperty("name", "", 1, 1)
HasProperty(classChauffeur,nameChauffeur)
genEmployeeDriver is Generalization(classEmployee, classDriver)
genEmployeeChauffeur is Generalization(classEmployee, classChauffeur)
```

Instance domain

```
//****Classes****
[Closed(type)][Unique(id -> type)]
primitive InstanceCompany ::=( id: Integer, type: Class).
className := InstanceCompany(_,y), y.name!="Company" .

[Closed(type)][Unique(id -> type)]
primitive InstanceEmployee ::=( id: Integer, type: Class).
className := InstanceEmployee(_,y), y.name!="Employee" .

[Closed(type)][Unique(id -> type)]
primitive InstanceDriver ::=( id: Integer, type: Class).
className := InstanceDriver(_,y), y.name!="Driver" .

[Closed(type)][Unique(id -> type)]
primitive InstanceChauffeur ::=( id: Integer, type: Class).
className := InstanceChauffeur(_,y), y.name!="Chauffeur" .

//****Associations and Association classes****

[Closed(type,employeeE,employeer)]
primitive Linkcontract ::=(id: Integer, type: Association, employeeE: InstanceEmployee,
employeer: InstanceCompany).
associationName:= Linkcontract(_,aso,_,_),aso.name!= "contract".

//****Generalizations****

[Closed(type, Driver, Employee)]
[Unique(Driver-> Employee)]
[Unique(Employee -> Driver)]
primitive LinkGenEmployeeDriver::=(id: Integer, type:Generalization,Employee:
InstanceEmployee, Driver: InstanceDriver).
genType:= LinkGenEmployeeDriver(_,gen,_,_),gen.sup.name!= "Employee".
genType:= LinkGenEmployeeDriver(_,gen,_,_), gen.sub.name!="Driver".
//Each instance of the child has to be associated with one and only one instance of the
parent.
error_GenOneAndOnlyOne := c is InstanceDriver, count(LinkGenEmployeeDriver(_,_,p,c))
!=1.

[Closed(type, Chauffeur, Employee)]
[Unique(Chauffeur-> Employee)]
[Unique(Employee -> Chauffeur)]
primitive LinkGenEmployeeChauffeur::=(id: Integer, type:Generalization,Employee:
InstanceEmployee, Chauffeur: InstanceChauffeur).
genType:= LinkGenEmployeeChauffeur(_,gen,_,_),gen.sup.name!= "Employee".
genType:= LinkGenEmployeeChauffeur(_,gen,_,_),gen.sub.name!="Chauffeur".
//Each instance of the child has to be associated with one and only one instance of the
parent.
error_GenOneAndOnlyOne := c is InstanceChauffeur,
count(LinkGenEmployeeChauffeur(_,_,p,c)) !=1.

//****Properties****

[Closed(owner,prop)]
[Unique(owner, prop ->value)]
primitive VATNumCompanySlot ::= (owner: Element, prop:IntegerProperty, value: Integer).
slotName :=VATNumCompanySlot(_,prop,_), prop.name!= "VATNum".
slotOwner :=VATNumCompanySlot(owner,_,_), owner.type.name!= "Company".

[Closed(owner,prop)]
[Unique(owner, prop ->value)]
```

```
primitive IDCardEmployeeSlot ::= (owner: Element, prop:StringProperty, value: String).
slotName :=IDCardEmployeeSlot(_,prop,_), prop.name!= "IDCard".
slotOwner :=IDCardEmployeeSlot(owner,_,_), owner.type.name!= "Employee".

[Closed(owner,prop)]
[Unique(owner, prop ->value)]
primitive nameChauffeurSlot ::= (owner: Element, prop:StringProperty, value: String).
slotName :=nameChauffeurSlot(_,prop,_), prop.name!= "name".
slotOwner :=nameChauffeurSlot(owner,_,_), owner.type.name!= "Chauffeur".

Instance ::=  InstanceCompany +  InstanceEmployee +  InstanceDriver +
InstanceChauffeur.
Slot ::= VATNumCompanySlot + IDCardEmployeeSlot + nameChauffeurSlot.
Element::=  Linkcontract + Instance .

//****Association queries****
//// There cannot be too few out-going/in-coming links on an instance.
error_model_tooFewLinks := a is Association, a.name="contract", i is InstanceEmployee,
count(Linkcontract(_,a,i,_)) < a.dstLower.
error_model_tooFewLinks := a is Association, a.name="contract", i is InstanceCompany,
count(Linkcontract(_,a,_,i))< a.srcLower.
error_model_tooManyLinks := a is Association, a.name="contract", i is InstanceEmployee,
count(Linkcontract(_,a,i,_)) > a.dstUpper.
error_model_tooManyLinks := a is Association, a.name="contract", i is InstanceCompany,
count(Linkcontract(_,a,_,i))>  a.srcUpper.

//****Generalization queries****

 To be substituted by the Formula instructions corresponding to the constraints given by
                        each different type of generalization

conforms:= !className & !associationName & !genType & !error_GenOneAndOnlyOne &
!slotName & !slotOwner & !error_model_tooFewLinks & !error_model_tooManyLinks &
!queryComplete & !queryDisjoint.
```

Depending on the generalization set type, the **conforms** query could include the verification of the negation of the corresponding **queryComplete** and **queryDisjoint** queries (see latter the definition of each of these queries).

M0 ClassDiagram Instance partial model

```
[Introduce(InstanceCompany,5)]

 To be substituted by the Formula instructions corresponding to the number of instances
                    of each child and parent we want Formula to generate

[Introduce(Linkcontract,5)]
[Introduce(VATNumCompanySlot,5)]
[Introduce(IDCardEmployeeSlot,5)]
[Introduce(nameChauffeurSlot,5)]
```

b. Specific Formula code instructions

o CDCoDi- {complete, disjoint}

M1- Instance domain

```
queryComplete := p is InstanceEmployee,
fail LinkGenEmployeeChauffeur(_,_,p,_), fail LinkGenEmployeeDriver(_,_,p,_).
queryDisjointG11 :=
LinkGenEmployeeChauffeur(_,_,p,_), LinkGenEmployeeDriver(_,_,p,_).
queryDisjoint := queryDisjointG11.
```

The representation in Formula of this class diagram includes the two queries, since the generalization type is both **`complete`** and **`disjoint`**.

M0- ClassDiagram Instance partial model

```
[Introduce(InstanceEmployee,10)]
[Introduce(InstanceDriver,5)]
[Introduce(InstanceChauffeur,5)]
[Introduce(LinkGenEmployeeDriver,5)]
[Introduce(LinkGenEmployeeChauffeur,5)]
```

o CDInDi - {incomplete, disjoint}

M1- Instance domain

```
queryDisjointG11 := LinkGenEmployeeChauffeur(_,_,p,_),
LinkGenEmployeeDriver(_,_,p,_).
queryDisjoint := queryDisjointG11.
```

The representation in Formula of this class diagram includes only the query **`queryDisjoint,`** since the generalization type is **`disjoint`** but **`incomplete`**.

M0- ClassDiagram Instance partial model

```
[Introduce(InstanceEmployee,15)]
[Introduce(InstanceDriver,5)]
[Introduce(InstanceChauffeur,5)]
[Introduce(LinkGenEmployeeDriver,5)]
[Introduce(LinkGenEmployeeChauffeur,5)]
```

o CDCoOv- {complete, overlapping}

M1- Instance domain

```
queryComplete := p is InstanceEmployee,
       fail LinkGenEmployeeChauffeur(_,_,p,_),
       fail LinkGenEmployeeDriver(_,_,p,_).
```

The representation in Formula of this class diagram includes only the query **querycomplete**, since the generalization type is **complete** but **overlapping.**

M0- ClassDiagram Instance partial model

```
[Introduce(InstanceEmployee,10)]
[Introduce(InstanceDriver,5)]
[Introduce(InstanceChauffeur,5)]
[Introduce(LinkGenEmployeeDriver,5)]
[Introduce(LinkGenEmployeeChauffeur,5)]
```

o CDInOv- {incomplete, overlapping}

M1- Instance domain

```
--
```

The Formula representation of this class diagram does not include the specific queries, since the generalization type is neither **complete** nor **disjoint**.

M0- ClassDiagram Instance partial model

```
[Introduce(InstanceEmployee,15)]
[Introduce(InstanceDriver,5)]
[Introduce(InstanceChauffeur,5)]
[Introduce(LinkGenEmployeeDriver,5)]
[Introduce(LinkGenEmployeeChauffeur,5)]
```

B. Formula generated facts

At this point we want to note that, as a result of the definition of the rules *supClass*, *inhsProp* and *inhsAsso*, Formula generates new facts in the domain space corresponding to the structure of inheritance classes, and inherited associations and properties. In particular, the facts generated in the four versions of CDs at level M1 are:

```
supClass(Class("Employee",false),Class("Chauffeur",false))
supClass(Class("Employee",false),Class("Driver",false))

inhsProp(Class("Chauffeur",false),StringProperty("IDCard","",1,1))
inhsProp(Class("Chauffeur",false),StringProperty("name","",1,1))
inhsProp(Class("Company",false),IntegerProperty("VATNum",0,1,1))
inhsProp(Class("Driver",false),StringProperty("IDCard","",1,1))
inhsProp(Class("Employee",false),StringProperty("IDCard","",1,1))

inhsAsso(Class("Chauffeur",false),
    Association("contract",Class("Chauffeur",false),1,2,Class("Company",false),1,1))
inhsAsso(Class("Driver",false),
    Association("contract",Class("Driver",false),1,2,Class("Company",false),1,1))
inhsAsso(Class("Employee",false),
    Association("contract",Class("Employee",false),1,2,Class("Company",false),1,1))
```

That is, both classes *Chauffeur* and *Driver* inherit the property *IDCard* and the association *contract*.

o CDCoDi- {complete, disjoint}

```
InstanceCompany(11,Class("Company",false))
InstanceCompany(13,Class("Company",false))
InstanceCompany(16,Class("Company",false))
InstanceEmployee(5,Class("Employee",false))
InstanceEmployee(10,Class("Employee",false))
InstanceEmployee(15,Class("Employee",false))
InstanceDriver(4,Class("Driver",false))
InstanceChauffeur(7,Class("Chauffeur",false))
InstanceChauffeur(8,Class("Chauffeur",false))

Linkcontract(9,Association("contract",Class("Employee",false),1,2,Class("Company",false),1,1),
        InstanceEmployee(10,Class("Employee",false)),
        InstanceCompany(11,Class("Company",false)))
Linkcontract(12,Association("contract",Class("Employee",false),1,2,Class("Company",false),1,1),
        InstanceEmployee(5,Class("Employee",false)),
        InstanceCompany(13,Class("Company",false)))
Linkcontract(14,Association("contract",Class("Employee",false),1,2,Class("Company",false),1,1),
        InstanceEmployee(15,Class("Employee",false)),
        InstanceCompany(16,Class("Company",false)))

LinkGenEmployeeDriver(3,Generalization(Class("Employee",false),Class("Driver",false)),
        InstanceEmployee(5,Class("Employee",false)),
        InstanceDriver(4,Class("Driver",false)))
LinkGenEmployeeChauffeur(6,Generalization(Class("Employee",false),Class("Chauffeur",false)),
        InstanceEmployee(10,Class("Employee",false)),
        InstanceChauffeur(7,Class("Chauffeur",false)))
LinkGenEmployeeChauffeur(27,Generalization(Class("Employee",false),Class("Chauffeur",false)),
        InstanceEmployee(15,Class("Employee",false)),
        InstanceChauffeur(8,Class("Chauffeur",false)))

VATNumCompanySlot(InstanceCompany(11,Class("Company",false)),
        IntegerProperty("VATNum",0,1,1),52)
VATNumCompanySlot(InstanceCompany(13,Class("Company",false)),
        IntegerProperty("VATNum",0,1,1),140666)
VATNumCompanySlot(InstanceCompany(16,Class("Company",false)),
        IntegerProperty("VATNum",0,1,1),-81159)
```

```
IDCardEmployeeSlot(InstanceEmployee(5,Class("Employee",false)),
        StringProperty("IDCard","",1,1),"~Together48General~")
IDCardEmployeeSlot(InstanceEmployee(10,Class("Employee",false)),
        StringProperty("IDCard","",1,1),"~Future1Destruction44Field~")
IDCardEmployeeSlot(InstanceEmployee(15,Class("Employee",false)),
        StringProperty("IDCard","",1,1),"~Common63Interest88Education~")
nameChauffeurSlot(InstanceChauffeur(7,Class("Chauffeur",false)),
        StringProperty("name","",1,1),"~Certain55Brain~")
nameChauffeurSlot(InstanceChauffeur(8,Class("Chauffeur",false)),
        StringProperty("name","",1,1),"~Certain55Brain~")
```

Results

Class	Parent	Associated with
Company 11	--	Employee 10
Company 13	--	Employee 5
Company 16	--	Employee 15
Employee 5	--	Company 13
Employee 10	--	Company 11
Employee 15	--	Company 16
Driver 4	Employee 5	Company 13
Chauffeur 7	Employee 10	Company 11
Chauffeur 8	Employee 15	Company 16

We note that all *employees* are either a *driver* or a *chauffeur* (complete) but not both (disjoint).

o CDInDi - {incomplete, disjoint}

```
InstanceCompany(16,Class("Company",false))
InstanceCompany(18,Class("Company",false))
InstanceCompany(21,Class("Company",false))
InstanceCompany(24,Class("Company",false))
InstanceEmployee(6,Class("Employee",false))
InstanceEmployee(11,Class("Employee",false))
InstanceEmployee(15,Class("Employee",false))
InstanceEmployee(20,Class("Employee",false))
InstanceEmployee(23,Class("Employee",false))
InstanceDriver(5,Class("Driver",false))
InstanceDriver(8,Class("Driver",false))
InstanceChauffeur(10,Class("Chauffeur",false))
InstanceChauffeur(13,Class("Chauffeur",false))

Linkcontract(14,Association("contract",Class("Employee",false),1,2,Class("Company",false),1,1),
        InstanceEmployee(15,Class("Employee",false)),
        InstanceCompany(16,Class("Company",false)))
Linkcontract(17,Association("contract",Class("Employee",false),1,2,Class("Company",false),1,1),
        InstanceEmployee(6,Class("Employee",false)),
        InstanceCompany(18,Class("Company",false)))
Linkcontract(19,Association("contract",Class("Employee",false),1,2,Class("Company",false),1,1),
        InstanceEmployee(20,Class("Employee",false)),
        InstanceCompany(21,Class("Company",false)))
Linkcontract(22,Association("contract",Class("Employee",false),1,2,Class("Company",false),1,1),
        InstanceEmployee(23,Class("Employee",false)),
        InstanceCompany(24,Class("Company",false)))
Linkcontract(25,Association("contract",Class("Employee",false),1,2,Class("Company",false),1,1),
        InstanceEmployee(11,Class("Employee",false)),
        InstanceCompany(24,Class("Company",false)))

LinkGenEmployeeDriver(3,Generalization(Class("Employee",false),Class("Driver",false)),
        InstanceEmployee(6,Class("Employee",false)),
```

```
        InstanceDriver(5,Class("Driver",false)))
LinkGenEmployeeDriver(7,Generalization(Class("Employee",false),Class("Driver",false)),
        InstanceEmployee(15,Class("Employee",false)),
        InstanceDriver(8,Class("Driver",false)))
LinkGenEmployeeChauffeur(9,Generalization(Class("Employee",false),Class("Chauffeur",false)),
        InstanceEmployee(11,Class("Employee",false)),
        InstanceChauffeur(10,Class("Chauffeur",false)))
LinkGenEmployeeChauffeur(12,Generalization(Class("Employee",false),Class("Chauffeur",false)),
        InstanceEmployee(23,Class("Employee",false)),
        InstanceChauffeur(13,Class("Chauffeur",false)))

VATNumCompanySlot(InstanceCompany(16,Class("Company",false)),
        IntegerProperty("VATNum",0,1,1),57)
VATNumCompanySlot(InstanceCompany(18,Class("Company",false)),
        IntegerProperty("VATNum",0,1,1),58)
VATNumCompanySlot(InstanceCompany(24,Class("Company",false)),
        IntegerProperty("VATNum",0,1,1),749536)
IDCardEmployeeSlot(InstanceEmployee(11,Class("Employee",false)),
        StringProperty("IDCard","",1,1),"~Prison35Yesterday9Waiting~")
IDCardEmployeeSlot(InstanceEmployee(20,Class("Employee",false)),
        StringProperty("IDCard","",1,1),"~Middle51Distribution~")
nameChauffeurSlot(InstanceChauffeur(10,Class("Chauffeur",false)),
        StringProperty("name","",1,1),"~Value3Reward81Berry~")
nameChauffeurSlot(InstanceChauffeur(13,Class("Chauffeur",false)),
        StringProperty("name","",1,1),"~Sheep97Green~")
```

Results

Class	Parent	Associated with
Company 16	--	Employee 15
Company 18	--	Employee 6
Company 21	--	Employee 20
Company 24	--	Employee 23, 11
Employee 6	--	Company 18
Employee11	--	Company 24
Employee 15	--	Company 16
Employee 20	**--**	**Company 21**
Employee 23	--	Company 24
Driver 5	Employee 6	Company 18
Driver 8	Employee 15	Company 16
Chauffeur 10	Employee 11	Company 24
Chauffeur 13	Employee 23	Company 24

We note that *employee 20* is neither a *driver* nor a *chauffeur*.

o CDCoOv- {complete, overlapping}

```
InstanceCompany(13,Class("Company",false))
InstanceCompany(15,Class("Company",false))
InstanceEmployee(9,Class("Employee",false))
InstanceEmployee(12,Class("Employee",false))
InstanceDriver(4,Class("Driver",false))
InstanceDriver(6,Class("Driver",false))
InstanceChauffeur(8,Class("Chauffeur",false))
InstanceChauffeur(26,Class("Chauffeur",false))

Linkcontract(11,Association("contract",Class("Employee",false),1,2,Class("Company",false),1,1),
        InstanceEmployee(12,Class("Employee",false)),
        InstanceCompany(13,Class("Company",false)))
Linkcontract(14,Association("contract",Class("Employee",false),1,2,Class("Company",false),1,1),
        InstanceEmployee(9,Class("Employee",false)),
        InstanceCompany(15,Class("Company",false)))

LinkGenEmployeeDriver(3,Generalization(Class("Employee",false),Class("Driver",false)),
        InstanceEmployee(12,Class("Employee",false)),
        InstanceDriver(4,Class("Driver",false)))
LinkGenEmployeeDriver(5,Generalization(Class("Employee",false),Class("Driver",false)),
        InstanceEmployee(9,Class("Employee",false)),
        InstanceDriver(6,Class("Driver",false)))
LinkGenEmployeeChauffeur(7,Generalization(Class("Employee",false),Class("Chauffeur",false)),
        InstanceEmployee(9,Class("Employee",false)),
        InstanceChauffeur(8,Class("Chauffeur",false)))
LinkGenEmployeeChauffeur(10,Generalization(Class("Employee",false),Class("Chauffeur",false)),
        InstanceEmployee(12,Class("Employee",false)),
        InstanceChauffeur(26,Class("Chauffeur",false)))

VATNumCompanySlot(InstanceCompany(13,Class("Company",false)),
        IntegerProperty("VATNum",0,1,1),46)
VATNumCompanySlot(InstanceCompany(15,Class("Company",false)),
        IntegerProperty("VATNum",0,1,1),45)
IDCardEmployeeSlot(InstanceEmployee(9,Class("Employee",false)),
        StringProperty("IDCard","",1,1),"~Picture17Thumb53Chance~")
IDCardEmployeeSlot(InstanceEmployee(12,Class("Employee",false)),
        StringProperty("IDCard","",1,1),"~Picture17Thumb53Chance~")
nameChauffeurSlot(InstanceChauffeur(8,Class("Chauffeur",false)),
        StringProperty("name","",1,1),"~Secretary99Receipt70Waiting~")
nameChauffeurSlot(InstanceChauffeur(26,Class("Chauffeur",false)),
        StringProperty("name","",1,1),"~Black53Person4Existence~")
```

Results

Class	Parent	Associated with
Company 13	--	Employee 12
Company 15	--	Employee 9
Employee 9	--	Company 15
Employee 12	--	Company 13
Driver 4	**Employee 12**	**Company 13**
Driver 6	***Employee 9***	***Company 15***
Chauffeur 8	***Employee 9***	***Company 15***
Chauffeur 26	**Employee 12**	**Company 13**

We note that *Driver 4* and *Chauffeur 26* correspond to the same *Employee 12*. Something similar happens with *Driver 6* and *Chauffeur 8*, since they correspond to the same *Employee 9*. There are *employees* who are neither *drivers* nor *chauffeurs*.

o CDInOv- {incomplete, overlapping}

```
InstanceCompany(13,Class("Company",false))
InstanceCompany(16,Class("Company",false))
InstanceEmployee(5,Class("Employee",false))
InstanceEmployee(9,Class("Employee",false))
InstanceEmployee(15,Class("Employee",false))
InstanceDriver(4,Class("Driver",false))
InstanceDriver(7,Class("Driver",false))
InstanceDriver(8,Class("Driver",false))
InstanceChauffeur(11,Class("Chauffeur",false))
InstanceChauffeur(30,Class("Chauffeur",false))

Linkcontract(12,Association("contract",Class("Employee",false),1,2,Class("Company",false),1,1),
        InstanceEmployee(9,Class("Employee",false)),
        InstanceCompany(13,Class("Company",false)))
Linkcontract(14,Association("contract",Class("Employee",false),1,2,Class("Company",false),1,1),
        InstanceEmployee(15,Class("Employee",false)),
        InstanceCompany(16,Class("Company",false)))
Linkcontract(17,Association("contract",Class("Employee",false),1,2,Class("Company",false),1,1),
        InstanceEmployee(5,Class("Employee",false)),
        InstanceCompany(16,Class("Company",false)))

LinkGenEmployeeDriver(3,Generalization(Class("Employee",false),Class("Driver",false)),
        InstanceEmployee(5,Class("Employee",false)),
        InstanceDriver(4,Class("Driver",false)))
LinkGenEmployeeDriver(6,Generalization(Class("Employee",false),Class("Driver",false)),
        InstanceEmployee(15,Class("Employee",false)),
        InstanceDriver(7,Class("Driver",false)))
LinkGenEmployeeDriver(28,Generalization(Class("Employee",false),Class("Driver",false)),
        InstanceEmployee(9,Class("Employee",false)),
        InstanceDriver(8,Class("Driver",false)))
LinkGenEmployeeChauffeur(10,Generalization(Class("Employee",false),Class("Chauffeur",false)),
        InstanceEmployee(5,Class("Employee",false)),
        InstanceChauffeur(11,Class("Chauffeur",false)))
LinkGenEmployeeChauffeur(29,Generalization(Class("Employee",false),Class("Chauffeur ",false)),
        InstanceEmployee(9,Class("Employee",false)),
        InstanceChauffeur(30,Class("Chauffeur",false)))

VATNumCompanySlot(InstanceCompany(13,Class("Company",false)),
        IntegerProperty("VATNum",0,1,1),49)
VATNumCompanySlot(InstanceCompany(16,Class("Company",false)),
        IntegerProperty("VATNum",0,1,1),50)
IDCardEmployeeSlot(InstanceEmployee(9,Class("Employee",false)),
        StringProperty("IDCard","",1,1),"~Tongue43Breath~")
IDCardEmployeeSlot(InstanceEmployee(15,Class("Employee",false)),
        StringProperty("IDCard","",1,1),"~Woman49Copper67Hollow~")
nameChauffeurSlot(InstanceChauffeur(11,Class("Chauffeur",false)),
        StringProperty("name","",1,1),"~Scissors52Harmony75Boiling~")
nameChauffeurSlot(InstanceChauffeur(30,Class("Chauffeur",false)),
        StringProperty("name","",1,1),"~Scissors52Harmony75Boiling~")
```

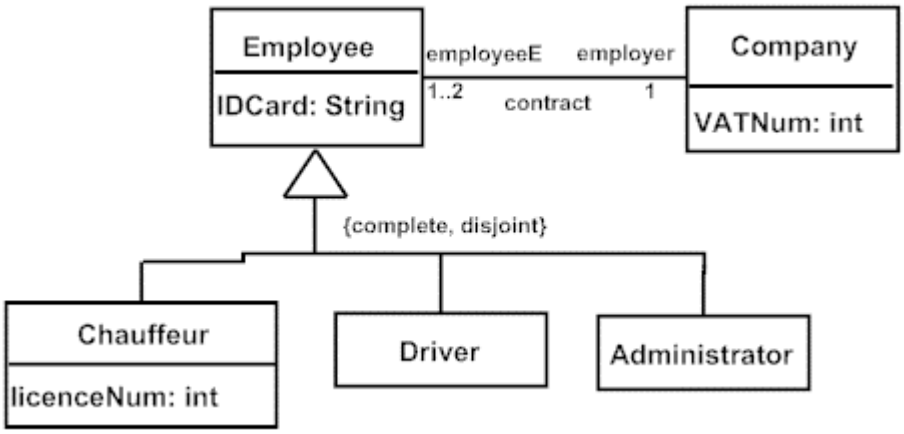
Results

Class	Parent	Associated with
Company 13	--	Employee 9
Company 16	--	Employee 15, 5
Employee 5	--	Company 16
Employee 9	--	Company 13
Employee 15	--	Company 16
Driver 4	**Employee 5**	Company 16
Driver 7	Employee 15	Company 16
Driver 8	***Employee 9***	Company 13
Chauffeur 11	**Employee 5**	Company 16
Chauffeur 30	***Employee 9***	Company 13

We note that *Driver 4* and *Chauffeur 11* correspond to the same *Employee 5*. Something similar happens with *Driver 8* and *Chauffeur 30*, since they correspond to the same *Employee 9*. In contrast *Employee 15* is only a driver (*Driver 7*).

CDCoDiThreeSubclasses

The CD corresponds to:



In this case, inheritance instructions of the M1- Instance domain are:

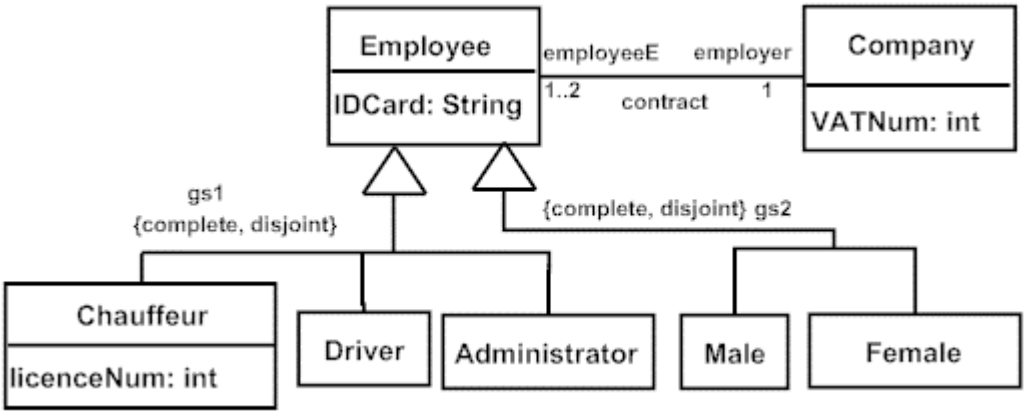
M1- Instance domain

```
//****Generalization queries****
queryComplete := p is InstanceEmployee,
   fail LinkGenEmployeeChauffeur(_,_,p,_),
   fail LinkGenEmployeeDriver(_,_,p,_),
   fail LinkGenEmployeeAdministrator(_,_,p,_).
queryDisjointg11:= LinkGenEmployeeChauffeur(_,_,p,_), LinkGenEmployeeDriver(_,_,p,_).
queryDisjointg12:= LinkGenEmployeeChauffeur(_,_,p,_), LinkGenEmployeeAdministrator(_,_,p,_).
queryDisjointg13:= LinkGenEmployeeDriver(_,_,p,_), LinkGenEmployeeAdministrator(_,_,p,_).
queryDisjoint:= ( queryDisjointg11 |  queryDisjointg12 |  queryDisjointg13).
```

The representation in Formula of this class diagram includes the two queries, since the generalization set type is both **complete** and **disjoint**.

CDCoDiTwoGeneralizationSetsThreeAndTwoSubclasses

The CD corresponds to:



In this case, inheritance instructions of the M1- Instance domain are:

M1- Instance domain

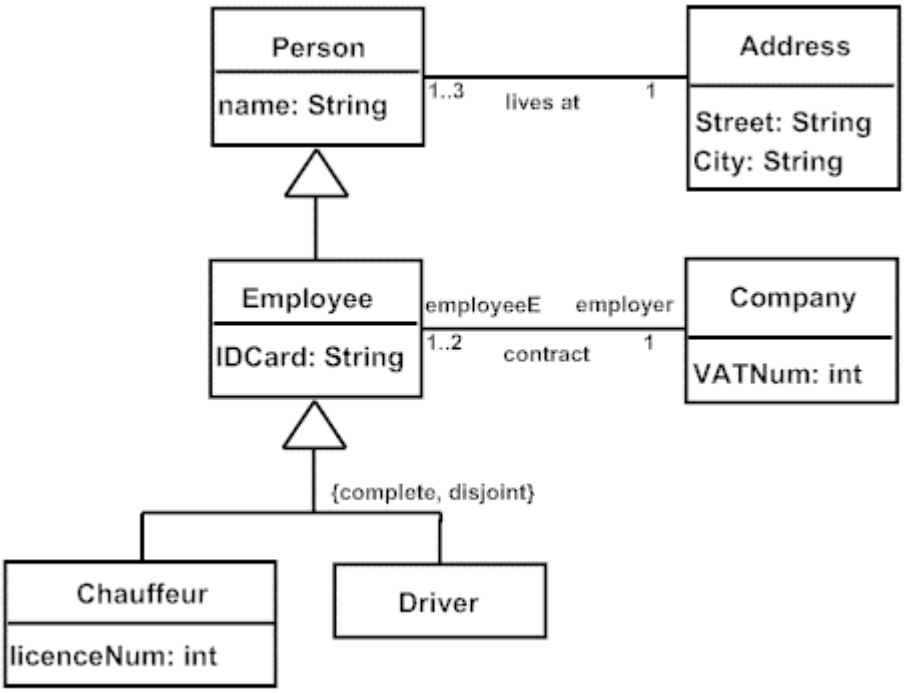
```
//****Generalization queries****
queryComplete := p is InstanceEmployee,
  fail LinkGenEmployeeChauffeur(_,_,p,_),
  fail LinkGenEmployeeDriver(_,_,p,_),
  fail LinkGenEmployeeAdministrator(_,_,p,_).
queryDisjointg11:= LinkGenEmployeeChauffeur(_,_,p,_), LinkGenEmployeeDriver(_,_,p,_).
queryDisjointg12:= LinkGenEmployeeChauffeur(_,_,p,_), LinkGenEmployeeAdministrator(_,_,p,_).
queryDisjointg13:= LinkGenEmployeeDriver(_,_,p,_), LinkGenEmployeeAdministrator(_,_,p,_).
queryDisjoint:= ( queryDisjointg11 |  queryDisjointg12 |  queryDisjointg13).

queryComplete := p is InstanceEmployee,
  fail LinkGenEmployeeFemale(_,_,p,_),
  fail LinkGenEmployeeMale(_,_,p,_).
queryDisjointg21:= LinkGenEmployeeFemale(_,_,p,_), LinkGenEmployeeMale(_,_,p,_).
queryDisjoint:=  queryDisjointg21.
```

The representation in Formula of this class diagram includes the two **queryComplete** queries **and** two **queryDisjoint** queries, since the generalization set types are both **complete** and **disjoint**.

2. Formula representation of multiple generalizations

Next, we present an example of a class diagram which includes multiple generalization relationships. In particular, with this example we want to show the way in which we represent multiple inheritance in Formula. For this example, we consider the default type of generalizations, that is, {incomplete, disjoint}, in the generalization between *Person* and *Employee*, and the {complete, disjoint} type in the generalization between *Chauffeur* and *Driver- Employee*. This CD is identified as **CDMultGen**.



Again, we divide the explanation in **two sections**: (1) The Formula code instructions corresponding to the representation of the CD in Formula (**Formula code instructions**), (2) the Formula instances resulted from the application of the Formula solver to the previous Formula code instructions (**Formula generated facts**).

Additionally, we have shaded in **light blue** the instructions corresponding to the class diagram elements that are associated with inheritance.

A.Formula code instructions

M1

ClassDiagram model

```
classPerson is Class("Person", false)
classCompany is Class("Company", false)
classEmployee is Class("Employee", false)
classChauffeur is Class("Chauffeur", false)
classDriver is Class("Driver", false)
```

```
classAddress is Class("Address", false)
livesAt is Association("livesAt", classAddress, 1, 1, classPerson, 1, 3)
contract is Association("contract", classCompany, 1, 1, classEmployee, 1, 2)
genPersonEmployee is Generalization(classPerson, classEmployee)
genEmployeeChauffeur is Generalization(classEmployee, classChauffeur)
genEmployeeDriver is Generalization(classEmployee, classDriver)
```

Instance domain

```
//****Classes****

[Closed(type)][Unique(id -> type)]
primitive InstancePerson ::=( id: Integer, type: Class).
className := InstancePerson(_,y), y.name!="Person" .

[Closed(type)][Unique(id -> type)]
primitive InstanceCompany ::=( id: Integer, type: Class).
className := InstanceCompany(_,y), y.name!="Company" .

[Closed(type)][Unique(id -> type)]
primitive InstanceEmployee ::=( id: Integer, type: Class).
className := InstanceEmployee(_,y), y.name!="Employee" .

[Closed(type)][Unique(id -> type)]
primitive InstanceChauffeur ::=( id: Integer, type: Class).
className := InstanceChauffeur(_,y), y.name!="Chauffeur" .

[Closed(type)][Unique(id -> type)]
primitive InstanceDriver ::=( id: Integer, type: Class).
className := InstanceDriver(_,y), y.name!="Driver" .

[Closed(type)][Unique(id -> type)]
primitive InstanceAddress ::=( id: Integer, type: Class).
className := InstanceAddress(_,y), y.name!="Address" .

//****Associations and Association classes****

[Closed(type,address,person)]
primitive LinklivesAt ::=(id: Integer, type: Association, address: InstanceAddress,
person: InstancePerson).
associationName:= LinklivesAt(_,aso,_,_),aso.name!= "livesAt".

[Closed(type,employer,employee)]
primitive Linkcontract ::=(id: Integer, type: Association, employer: InstanceCompany,
employee: InstanceEmployee).
associationName:= Linkcontract(_,aso,_,_),aso.name!= "contract".

//****Generalizations****

[Closed(type, Employee, Person)]
[Unique(Employee-> Person)]
[Unique(Person -> Employee)]
primitive LinkGenPersonEmployee::=(id: Integer, type:Generalization, Person:
InstancePerson, Employee: InstanceEmployee).
genType:= LinkGenPersonEmployee(_,gen,_,_),gen.sup.name!= "Person".
genType:= LinkGenPersonEmployee(_,gen,_,_), gen.sub.name!="Employee".
//Each instance of the child has to be associated with one and only one instance of the
parent.
error_GenOneAndOnlyOne := c is InstanceEmployee, count(LinkGenPersonEmployee(_,_,c,p))!=1.

[Closed(type, Chauffeur, Employee)]
```

```
[Unique(Chauffeur-> Employee)]
[Unique(Employee -> Chauffeur)]
primitive LinkGenEmployeeChauffeur::=(id: Integer, type:Generalization, Employee:
InstanceEmployee, Chauffeur: InstanceChauffeur).
genType:= LinkGenEmployeeChauffeur(_,gen,_,_),gen.sup.name!= "Employee".
genType:= LinkGenEmployeeChauffeur(_,gen,_,_), gen.sub.name!="Chauffeur".
//Each instance of the child has to be associated with one and only one instance of the
parent.
error_GenOneAndOnlyOne := c is InstanceChauffeur,
count(LinkGenEmployeeChauffeur(_,_,c,p))!=1.

[Closed(type, Driver, Employee)]
[Unique(Driver-> Employee)]
[Unique(Employee -> Driver)]
primitive LinkGenEmployeeDriver::=(id: Integer, type:Generalization, Employee:
InstanceEmployee, Driver: InstanceDriver).
genType:= LinkGenEmployeeDriver(_,gen,_,_),gen.sup.name!= "Employee".
genType:= LinkGenEmployeeDriver(_,gen,_,_), gen.sub.name!="Driver".
//Each instance of the child has to be associated with one and only one instance of the
parent.
error_GenOneAndOnlyOne := c is InstanceDriver, count(LinkGenEmployeeDriver(_,_,c,p))!=1.


//****Properties****

Instance ::=  InstancePerson +  InstanceCompany +  InstanceEmployee +  InstanceChauffeur +
InstanceDriver +  InstanceAddress.
Link ::=  LinklivesAt +  Linkcontract.
Element::=  Instance  + Link .


//****Association queries****
//// There cannot be too few out-going/in-coming links on an instance.
error_model_tooFewLinks := a is Association, a.name="livesAt", i is InstanceAddress,
count(LinklivesAt(_,a,i,_)) < a.dstLower.
error_model_tooFewLinks := a is Association, a.name="livesAt", i is InstancePerson,
count(LinklivesAt(_,a,_,i))< a.srcLower.
error_model_tooManyLinks := a is Association, a.name="livesAt", i is InstanceAddress,
count(LinklivesAt(_,a,i,_)) > a.dstUpper.
error_model_tooManyLinks := a is Association, a.name="livesAt", i is InstancePerson,
count(LinklivesAt(_,a,_,i))>  a.srcUpper.
error_model_tooFewLinks := a is Association, a.name="contract", i is InstanceCompany,
count(Linkcontract(_,a,i,_)) < a.dstLower.
error_model_tooFewLinks := a is Association, a.name="contract", i is InstanceEmployee,
count(Linkcontract(_,a,_,i))< a.srcLower.
error_model_tooManyLinks := a is Association, a.name="contract", i is InstanceCompany,
count(Linkcontract(_,a,i,_)) > a.dstUpper.
error_model_tooManyLinks := a is Association, a.name="contract", i is InstanceEmployee,
count(Linkcontract(_,a,_,i))>  a.srcUpper.

   //****Generalization queries****
queryComplete := p is InstanceEmployee,
fail LinkGenEmployeeChauffeur(_,_,p,_), fail LinkGenEmployeeDriver(_,_,p,_).
queryDisjointgs21:= LinkGenEmployeeChauffeur(_,_,p,_), LinkGenEmployeeDriver(_,_,p,_).
queryDisjoint:=  queryDisjointgs21.
conforms:= !className & !associationName & !genType & !error_GenOneAndOnlyOne &
!error_model_tooFewLinks & !error_model_tooManyLinks & !queryComplete & !queryDisjoint.
```

M0 ClassDiagram Instance partial model

```
[Introduce(InstancePerson,15)]
[Introduce(InstanceCompany,10)]
[Introduce(InstanceEmployee,10)]
[Introduce(InstanceChauffeur,5)]
[Introduce(InstanceDriver,5)]
[Introduce(InstanceAddress,5)]
[Introduce(LinklivesAt,5)]
[Introduce(Linkcontract,5)]
[Introduce(LinkGenPersonEmployee,10)]
[Introduce(LinkGenEmployeeChauffeur,5)]
[Introduce(LinkGenEmployeeDriver,5)]
```

B. Formula generated facts

The instances generated at level M0 are:

M0

```
InstancePerson(5,Class("Person",false))
InstancePerson(15,Class("Person",false))
InstancePerson(18,Class("Person",false))
InstanceCompany(31,Class("Company",false))
InstanceCompany(34,Class("Company",false))
InstanceEmployee(8,Class("Employee",false))
InstanceEmployee(12,Class("Employee",false))
InstanceEmployee(47,Class("Employee",false))
InstanceChauffeur(7,Class("Chauffeur",false))
InstanceChauffeur(9,Class("Chauffeur",false))
InstanceDriver(11,Class("Driver",false))
InstanceAddress(14,Class("Address",false))
InstanceAddress(17,Class("Address",false))

LinklivesAt(13,Association("livesAt",Class("Address",false),1,1,Class("Person",false),1,3),
        InstanceAddress(14,Class("Address",false)),
        InstancePerson(15,Class("Person",false)))
LinklivesAt(16,Association("livesAt",Class("Address",false),1,1,Class("Person",false),1,3),
        InstanceAddress(17,Class("Address",false)),
        InstancePerson(18,Class("Person",false)))
LinklivesAt(19,Association("livesAt",Class("Address",false),1,1,Class("Person",false),1,3),
        InstanceAddress(17,Class("Address",false)),
        InstancePerson(5,Class("Person",false)))
Linkcontract(30,Association("contract",Class("Company",false),1,1,Class("Employee",false),1,2),
        InstanceCompany(31,Class("Company",false)),
        InstanceEmployee(47,Class("Employee",false)))
Linkcontract(32,Association("contract",Class("Company",false),1,1,Class("Employee",false),1,2),
        InstanceCompany(34,Class("Company",false)),
        InstanceEmployee(12,Class("Employee",false)))
Linkcontract(33,Association("contract",Class("Company",false),1,1,Class("Employee",false),1,2),
        InstanceCompany(34,Class("Company",false)),
        InstanceEmployee(8,Class("Employee",false)))
LinkGenPersonEmployee(4,Generalization(Class("Person",false),Class("Employee",false)),
        InstancePerson(5,Class("Person",false)),
        InstanceEmployee(8,Class("Employee",false)))
LinkGenPersonEmployee(45,Generalization(Class("Person",false),Class("Employee",false)),
        InstancePerson(15,Class("Person",false)),
        InstanceEmployee(12,Class("Employee",false)))
LinkGenPersonEmployee(46,Generalization(Class("Person",false),Class("Employee",false)),
        InstancePerson(18,Class("Person",false)),
        InstanceEmployee(47,Class("Employee",false)))
LinkGenEmployeeChauffeur(6,Generalization(Class("Employee",false),Class("Chauffeur",false)),
        InstanceEmployee(8,Class("Employee",false)),
        InstanceChauffeur(7,Class("Chauffeur",false)))
```

```
LinkGenEmployeeChauffeur(48,Generalization(Class("Employee",false),Class("Chauffeur",false)),
        InstanceEmployee(47,Class("Employee",false)),
        InstanceChauffeur(9,Class("Chauffeur",false)))
LinkGenEmployeeDriver(10,Generalization(Class("Employee",false),Class("Driver",false)),
        InstanceEmployee(12,Class("Employee",false)),
        InstanceDriver(11,Class("Driver",false)))
```

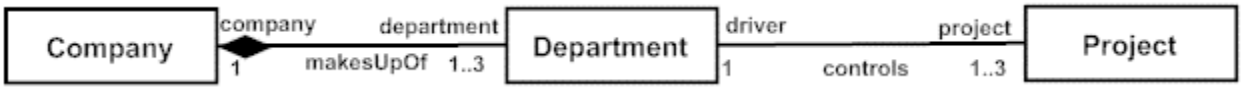
Results

Class	Parent	Associated with
Company 31	--	Employee 47 (contract)
Company 34	--	Employee 12, 8 (contract)
Address 14	--	Person 15 (livesAt)
Address 17	--	Person 18, 5 (livesAt)
Person 5	--	Address 17 (livesAt)
Person 15	--	Address 14 (livesAt)
Person 18	--	Address 17 (livesAt)
Employee 8	Person 5	Company 34 (contract), Address 17 (livesAt)
Employee 12	Person 15	Company 34 (contract), Address 14 (livesAt)
Employee 47	Person 18	Company 31 (contract), Address 17 (livesAt)
Chauffeur 7	Employee 8, Person 5	Company 34 (contract), Address 17 (livesAt)
Chauffeur 9	Employee 47, Person 18	Company 31 (contract), Address 17 (livesAt)
Driver 11	Employee 12, Person 15	Company 34 (contract), Address 14 (livesAt)

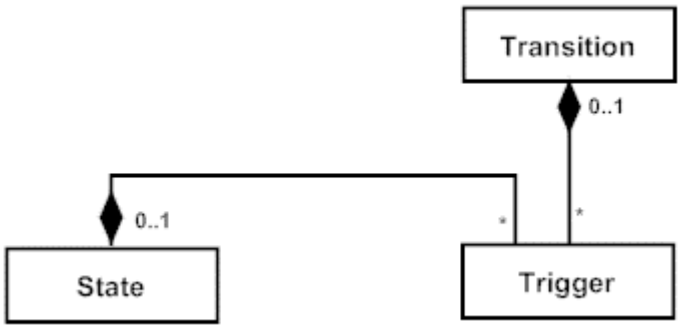
3. Formula representation of (strong) composition

Next, we present two class diagrams examples with different types of composition. Firstly, we tackle strong composition with the usual example of "*Companies* are composed of *Departments*" (the corresponding CD is identified as **CD1Ca** in the Eclipse project `Examples` provided also in [CD2FProject]). Secondly, we have considered an extract of the metamodel of UML Statemachines v. 2.4.1 [OMGS] which represents the fact that a *Trigger* can be owned by *Transition* or *State* (the corresponding CD is identified as **CDSupersComp** in the Eclipse project `Examples` provided also in [CD2FProject])

CD1Ca



CDSupersComp



In this case, we just focus on the specific Formula constrains that are generated al level M1 to ensure the semantics of strong composition in **CD1Ca** and composition in **CDSupersComp**.

Formula code instructions

CD1Ca

```
queryComposition:= p is InstanceCompany, count(LinkmakesUpOf(_,_,p,_))!=1.
conforms:= !className & !associationName & !error_model_tooFewLinks &
!error_model_tooManyLinks & !queryComposition.
```

In this case, a *part* is owned by an only *whole*.

CDSupersComp

```
queryComposition:= p is InstanceTrigger, count(LinktrigStat(_,_,p,_)) +
count(LinktrigTran(_,_,p,_))!=1.
conforms:= !className & !associationName & !error_model_tooFewLinks &
!error_model_tooManyLinks & !queryComposition.
```

In this other case, a *part* can be owned by more than one *whole*.

References

[CD2FProject] CD2Formula. Examples of CDs as an Eclipse project. Website: http://www.unirioja.es/cu/beperev/plugin/Examples.zip. Last visited on May 2015.

[OMGS] OMG UML 2.4.1 Superstructure Specification. Document formal/2011-08-06, August, 2012. Available at: http://www.omg.org/. Last visited on May 2015.