```
----------------Position part----------------

sig Position{
     latitude: Int,
     longitude: Int
}{
     latitude >= 0
     latitude <= 3
     longitude >= 0
     longitude <= 5
}

//the positions are uniquely determinate by the place
fact differentPosition{
     all p1, p2: Position| (p1!=p2)implies(p1.latitude !=
p2.latitude or p1.longitude != p2.longitude)
}

----------------Safe Area part----------------

sig SafeArea{
     safePositions: some Position,
}

fact differentPositionInDifferentSafeArea{
     all s1,s2: SafeArea| all p:Position| (s1 !=s2) implies
((p in s1.safePositions) and (p not in s2.safePositions))
}

//in our world there are always unsafe position
fact positionConstraint{
     all s: SafeArea| some p: Position| p not in
s.safePositions
}

----------------Special Area part----------------

sig SpecialArea{
     specialPositions: some Position,
     plugs: Int,
}{
     plugs >= 0
     plugs <= 5
}
```

```
fact existent{
#(SpecialArea)>= 1
}

fact plugsConstarint{
     all s:SpecialArea| #(s.specialPositions) = s.plugs
}

fact specialAreaConstraint{
     all p: SpecialArea.specialPositions | (p in
SafeArea.safePositions) &&
     (#(SpecialArea.specialPositions) <
#(SafeArea.safePositions))
}

fact differentPositionInDifferentSecialArea{
     all s1,s2: SpecialArea| all p: s1.specialPositions|
(s1 !=s2) implies ((p in s1.specialPositions) and (p not in
s2.specialPositions))
}

----------------Car part----------------

//there are no less than two seats
sig Car{
     code: Int,
     seats: Int,
     state: one State,
}{
     code>= 0
     seats>= 2
     seats<= 5
}

fact indentificationByCode{
     all c1, c2: Car | (c1 != c2) implies (c1.code !=
c2.code)
}
```

```
pred Car.isUsed[]{
     #(this.state.usedBy)= 1
}

pred Car.isInSafe[]{
     one s: SafeArea| this.state.statePosition in
s.safePositions
}

pred Car.isInSpecial[]{
     one s: SpecialArea| this.state.statePosition in
s.specialPositions
}

pred Car.isAvailable[]{
     !(this.isUsed) and (this.state.batteryLevel > 1) and
(this.isInSafe)
}

pred Car.isNotAvailable[]{
     this.isUsed or this.state.batteryLevel < 1 or
!(this.isInSafe)
}

pred Car.isInList[l1:List]{
     this in l1.cars
}
----------------State part----------------

sig State{
     batteryLevel: Int,
     phase: one Phase,
     passengers: Int,
     statePosition: one Position,
     usedBy: lone User
}{
     passengers >= 0
     batteryLevel >= 0
     batteryLevel <= 3
}
```

```
fact numberOfState{
     #Car = #State
}

fact passengersConstraint{
     all c:Car| ((c.state.phase=used) implies
(c.state.passengers<=c.seats)) and((c.state.phase!=used)
implies (c.state.passengers=0))
}

//this means that there are not cars in the same position
fact differentPositionInCar{
     all c1,c2 : Car | (c1!=c2) implies
(c1.state.statePosition != c2.state.statePosition)
}

----------------Phase part----------------

//these are the possible internal state of the car
enum Phase {reserved, used, charge, free,
parkUnsafeOrChargeOnSite}

fact reservedPhase{
     all c: Car| (c.state.phase = reserved) iff (c.isInSafe
and c.state.batteryLevel>1 and c.isUsed and
(c.state.phase!=used and c.state.phase!=charge and
c.state.phase!=free and
c.state.phase!=parkUnsafeOrChargeOnSite))
}

fact usedPhase{
     all c: Car| (c.state.phase = used) iff (c.isUsed and
c.state.batteryLevel != 0  and (c.state.phase!=reserved
and c.state.phase!=charge and c.state.phase!=free and
c.state.phase!=parkUnsafeOrChargeOnSite))
}

fact chargePhase{
     all c: Car| (c.state.phase = charge) iff
(!(c.isAvailable) and c.isInSpecial and
(c.state.phase!=reserved and c.state.phase!=used and
```

```
c.state.phase!=free and
c.state.phase!=parkUnsafeOrChargeOnSite))
}

fact freePhase{
     all c: Car| (c.state.phase = free) iff (c.isAvailable
and (c.state.phase!=reserved and c.state.phase!=used and
c.state.phase!=charge and
c.state.phase!=parkUnsafeOrChargeOnSite))
}

fact parkUnsafePhase{
     all c: Car| (c.state.phase =
parkUnsafeOrChargeOnSite) iff (!(c.isAvailable)and
/*(c.isInSafe or !(c.isInSafe)) and */
(c.state.phase!=reserved and c.state.phase!=used and
c.state.phase!=charge and c.state.phase!=free))
}


----------------User part----------------

sig User {
     licenseID: Int, //should be an alphanumeric value in
the real world
}{
     licenseID > 0
}

//this means that there are not equal users
fact differentIDUser{
     all u1, u2: User | (u1 != u2) implies (u1.licenseID !=
u2.licenseID)
}

fact differentUserOnCar{
     all c1,c2: Car| (c1 != c2) implies (c1.state.usedBy
!= c2.state.usedBy)
}
```

```
----------------Employee part----------------

sig Employee {
     employeeID: Int,
} {
     employeeID > 0
}

fact uniqueEmployee{
     all e1,e2: Employee| (e1 != e2) implies (e1.employeeID
!= e2. employeeID)
}

----------------PairOfWorkers part----------------

sig PairOfWorkers {
     employees: some Employee
} {
     #employees = 2
}

fact uniquePairOfWorkersForEmployee {
     all e :Employee| lone p : PairOfWorkers| ((e in
p.employees))
}

fact mapPairOfWorkersOnLists{
     #PairOfWorkers = #List
}

fact differentPairPerList{
     all l1,l2: List| (l1!=l2) implies (l1.pair!=l2.pair)
}
```

```
----------------List part----------------

sig List{
     code: Int,
     cars: some Car,
     pair: one PairOfWorkers,
}{
     code > 0
#cars > 0
}

fact uniqueList{
     all l1,l2: List| (l1!=l2) implies (l1.code!=l2.code)
}

fact phaseOfCarInTheList{
     all c: List.cars|
c.state.phase=parkUnsafeOrChargeOnSite
}

fact differentList{
     all l1,l2: List| all c: l1.cars| (l1!=l2) implies
!(c.isInList[l2])
}

fact listConstraints{
     all c:Car| (c.state.phase=parkUnsafeOrChargeOnSite)
implies (c in List.cars)
}

----------------Assertions part----------------

assert NoAllSafeAreaInSpacialeArea{
     no s:SafeArea | all p:s.safePositions| p in
SpecialArea.specialPositions
}

assert NoCarUsedAndInFreePhase {
     all c: Car |all u: User | (c.state.usedBy = u) implies
!(c.state.phase = free)
}
```

```
assert NoCarUsedWithZeroBattery {
     no c: Car | (c.state.phase = used) &&
c.state.batteryLevel = 0

}

assert NoCarWithMoreThanOnePhase {
     all c: Car | (c.state.phase = reserved ) iff
(!(c.state.phase = used) && !(c.state.phase = charge) &&
!(c.state.phase = free) && !(c.state.phase =
parkUnsafeOrChargeOnSite))
     all c: Car | (c.state.phase = used) iff
(!(c.state.phase = reserved) &&  !(c.state.phase = charge)
&&!(c.state.phase = free) && !(c.state.phase =
parkUnsafeOrChargeOnSite))
     all c: Car | (c.state.phase = charge) iff
(!(c.state.phase = reserved) && !(c.state.phase = used) &&
!(c.state.phase = free) && !(c.state.phase =
parkUnsafeOrChargeOnSite))
     all c: Car | (c.state.phase = free) iff
(!(c.state.phase = reserved) && !(c.state.phase = used) &&
!(c.state.phase = charge) && !(c.state.phase =
parkUnsafeOrChargeOnSite))
     all c: Car | (c.state.phase =
parkUnsafeOrChargeOnSite) iff (!(c.state.phase = reserved)
&& !(c.state.phase = used) && !(c.state.phase = charge) &&
!(c.state.phase = free))
}

assert NoEveryPhaseInTheList {
     all c:Car | all l: List | (c in l.cars) iff
(!(c.state.phase = reserved) && !(c.state.phase = used) &&
!(c.state.phase = charge) && !(c.state.phase = free))
}

assert NoPairOfWorkersWithSameEmployee {
     all p: PairOfWorkers | all e1,e2: p.employees | e1!=e2
}
```

```
----------------Execution part----------------

pred show(){
/*
some c:Car| c.state.phase=reserved
some c:Car| c.state.phase=used
some c:Car| c.state.phase=charge
some c:Car| c.state.phase=free
some c:Car| c.state.phase=parkUnsafeOrChargeOnSite
*/
}

run show for 5 but exactly 5 Car// but  exactly 8 Position,
exactly 1 SafeArea, exactly 2 SpecialArea, exactly 7 Car,
exactly 7 State, exactly 6 User, exactly 5 Employee, exactly
2 PairOfWorkers, exactly 2 List
check NoEveryPhaseInTheList
check NoEveryPhaseInTheList
check NoCarWithMoreThanOnePhase
check NoAllSafeAreaInSpacialeArea
check NoCarUsedWithZeroBattery
check NoCarUsedAndInFreePhase
```