```
open util/boolean

sig Position {}

sig Plug {}

one abstract sig SafeArea{
    positions: set Position
}{    #positions >0    }

sig PowerGridStation extends SafeArea{
    plugs: set Plug
}{    #positions = #plugs    }

sig ParkingLot extends SafeArea {}

sig User {
    doReserve: lone Reservation,
    doPickUp: Bool lone -> lone Car,
    doCancel: Bool lone -> lone Reservation,
    hasRide: lone Ride,
    park: Car lone -> lone Position,
    plugIn: Car lone -> lone Plug,
    accountState: one Bool
}{
    #doPickUp =1
    Bool.doPickUp = this.(doReserve.reserve)
    (this.(doReserve.reserve).(~doPickUp) = True)
        implies doReserve.reservationStatus = COMPLETE
    Bool.doCancel = doReserve
    (doReserve.(~doCancel) = True)
        implies doReserve.reservationStatus = CANCELED
    Position.(~park) = True.doPickUp  & this.((False.(~rideStatus)).drive)
    (#plugIn =1) implies (Car.park in PowerGridStation.positions)
    Plug.(~plugIn) = (True.doPickUp & this.(False.(~rideStatus).drive))
}
sig Car {
    carStatus: one CarStatus,
    batteryLevel: one BatteryLevel,
    parkPosition: lone Position,
    chargeState: Bool,
    engineState: Bool
}{
    (carStatus = AVAILABLE)
         implies (batteryLevel = HIGH)
            and ((one parkPosition) and (parkPosition in SafeArea.positions))
            and (engineState = False)
            and ((this.(~(Reservation.reserve)) = none)
                or (no res: Reservation | this.(~(res.reserve)) != none and res.res
ervationStatus = ACTIVE)
                or (one res: Reservation | this.(~(res.reserve)) != none and res.re
servationStatus = TIMEUP)
                or (one res: Reservation | this.(~(res.reserve)) != none and res.re
servationStatus = CANCELED)
                or (one res: Reservation | this.(~(res.reserve)) != none and res.re
servationStatus = COMPLETE and
                    (one rd: Ride| rd.rideStatus = False and this.(~(Reservation.re
serve)).(rd.drive)= this)))
    (carStatus = RESERVED)
        implies (batteryLevel = HIGH)
            and ((one parkPosition) and (parkPosition in SafeArea.positions))
            and (engineState = False)
            and (one res: Reservation |this.(~(res.reserve)) != none and res.reserv
ationStatus = ACTIVE)
            and (this.(~(User.doPickUp)) = False)
    (carStatus = INUSE)
        implies (batteryLevel != EMPTY)
            and (no parkPosition)
            and (this.~(User.doPickUp) = True) and (one rd: Ride| User.(rd.drive) =
 this and rd.rideStatus = True)
            and (one res: Reservation| this.(~(res.reserve)) != none and res.reserv
ationStatus = COMPLETE)
    (carStatus = OUTOFSERVICE)
        <=> (batteryLevel = EMPTY) or ((one parkPosition) and not(parkPosition in S
afeArea.positions))
    (chargeState = True)
        implies (one parkPosition)
            and (parkPosition in PowerGridStation.positions)
    (engineState = True)
```

```
        implies (carStatus = INUSE)
    (carStatus = INUSE) <=> (parkPosition = none)
    (carStatus = INUSE)    implies this.(User.park) = none
    (this.(User.plugIn) != none) => (chargeState = True)
}

abstract sig CarStatus {}
one sig AVAILABLE extends CarStatus {}
one sig RESERVED extends CarStatus {}
one sig INUSE extends CarStatus {}
one sig OUTOFSERVICE extends CarStatus {}

abstract sig BatteryLevel {}
one sig LOW extends BatteryLevel {}
one sig HIGH extends BatteryLevel {}
one sig EMPTY extends BatteryLevel {}

sig Reservation {
    reserve: User lone -> lone Car,
    reservationStatus: one ReservationStatus,
    countingTimeUp: Bool,
    fee: Int
}{
    // no Reservation without User
    this.(~doReserve) != none
    #reserve = 1
    Car.(~reserve) = this.(~doReserve)
    fee >=0
    (reservationStatus = TIMEUP) <=>(countingTimeUp = True)
    (reservationStatus = TIMEUP) implies (fee = 1) and (User.reserve.carStatus = AV
AILABLE)
    (reservationStatus = ACTIVE)
         implies (fee = 0)
            and (countingTimeUp = False)
            and (User.reserve.carStatus = RESERVED)
    (reservationStatus = CANCELED)
        implies (fee = 1) and (countingTimeUp = False)
            and ( True.(this.(~doReserve).doCancel) = this)
            and (User.reserve.carStatus = AVAILABLE)
    (reservationStatus = COMPLETE) implies (fee = 0)
```

```
            and (countingTimeUp = False)
            and ( True. (this.(~doReserve).doPickUp) =User.reserve)
}

abstract sig ReservationStatus {}
one sig TIMEUP extends ReservationStatus {}
one sig ACTIVE extends ReservationStatus {}
one sig CANCELED extends ReservationStatus {}
one sig COMPLETE extends ReservationStatus {}

sig Ride {
    drive: User lone -> lone Car,
    rideStatus: one Bool,
    passengerNum: one Int,
    disscount: set Discount,
    compensation: set Compensation,
    standardFee: one Int,
    paymentAmount: one Int
}{
    // no ride without user
    this.(~hasRide) != none
    #drive = 1
    Car.(~drive) = this.(~hasRide)
    (True.(this.(~hasRide).doPickUp) = User.drive)
    (rideStatus = True) implies (User.drive.carStatus = INUSE)
    (rideStatus = False) implies (User.drive.carStatus != INUSE)

}

abstract sig Discount {}
abstract sig Compensation{}

/*------------------FACTS------------------------*/
// No isolated plugs.
fact NoIsolatedPlug {
    no p: Plug | p.(~plugs) = none
}
// No two users share one reservation.
fact NoSharedReservation {
    no disj u1, u2: User| u1.doReserve = u2.doReserve
```

```
}
// No two users share one ride.
fact NoSharedRide {
    no disj u1, u2: User| u1.hasRide = u2.hasRide
}
// No two cars share one position.
fact NoSharedPosition {
    no disj c1, c2: Car| c1.parkPosition = c2.parkPosition
}
// No two reservations share one car.
fact NoSharedCar {
    no disj res1, res2: Reservation | User.(res1.reserve) = User.(res2.reserve)
}
// No two users share plug
fact NoSharedPlug {
    no disj u1, u2: User | Car.(u1.plugIn) = Car.(u2.plugIn)
}


/*------------------ASSERTS----------------------*/

assert noEMPTYbatteryCarIsINUSE {
    no c: Car | c.batteryLevel = EMPTY and c.carStatus = INUSE
}
check noEMPTYbatteryCarIsINUSE for 3
assert noLOWbatteryCarIsAVAILABLE {
    no c: Car | c.batteryLevel = LOW and c.carStatus = AVAILABLE
}
check noLOWbatteryCarIsAVAILABLE for 3
assert noCANCELEDreservationReserveCarhasStateRESERVED {
    no res: Reservation| res.reservationStatus = CANCELED and User.(res.reserve).ca
rStatus = RESERVED
}
check noCANCELEDreservationReserveCarhasStateRESERVED for 3
assert noTIMEUPreservationReserveCarhasStateRESERVED {
    no res: Reservation| res.reservationStatus = TIMEUP and User.(res.reserve).carS
tatus = RESERVED
}
check noTIMEUPreservationReserveCarhasStateRESERVED for 3
assert example{
    no c: Car| c.carStatus = INUSE}
check example for 1

pred example {}
run example
```