module PowEnj

//SIGNATURES

sig Position

{

latitude:Int //should be float

longitude:Int //should be float

}

abstract sig User

{

rentedCar: one Car, //field that indicates which car the user is using at the moment

payInfo: some Payment,

position: one Position

}

fact driverIsUnique

{

//can’t exist two renters who rent the same car at the same time

all r1,r2:Rent | (r1.Renter!=r2.Renter) => r1.rentedCar != r2.rentedCar and r1.startTime!=r2.startTime

}

fact applyDiscountForBattery

{

//if the battery at the end of the trip is at least at 50%, apply a discount

//not sure if it’s possible to use this notation!!

all c:Car, r:Rent | (c.battery>=50) => r.applyDiscount

//alternatively, we can suppose that the attribute “isCharge” is set to a specific value for every kind of discount appliable

all c:Car, r:Rent | (c.isCharge) => r.applyDiscount

}

fact applyDiscountForArea

{

//if the car at the end of trip is let in a service station, apply a discount

all r:Rent, s:ServiceStation | (r.finalPosition=s.position) => r.applyDiscount

}

fact applyOvertax

{

all c:Car, r:Rent, s:ServiceStation | (!(c.isCharge)) => r.applyOvertax or ((s.position – c.position)>r.maxDistance) => r.applyOvertax

}

sig Rent

{

startTime = Int, //should be Float

rentedCar: set Car,

Renter: one User,

applyDiscount: one User,

applyOvertax: one User,

carsAvailable: set Car,

maxDistance: Int //should be float

initialPos: Position,

finalPos: Position

passengers: set Passenger // don’t remember if the class (and consequently the signature) should exist or we decide to remove it; in this case, change Passenger with User}

sig Car

{

user: one User,

position:Position

battery: Int,

plate: String

}{#passengers>0}

fact plateIsUnique {

all c1,c2: Car | (c1 != c2) => c1.plate != c2.plate

}

Sig Plug{}

abstract sig safeArea {}

sig serviceStation extends safeArea

{

carsToCharge: set Car,

plugAvailable: set Plug,

position:Position

}

sig Payment

{

transactionCode: Int,

payInfo : set Payment

}

fact userIsUnique {

all u1,u2: User | u1 != u2 => u1.email != u2.email

}

fact pathDriverHasAStartAndEnd

{

all r:Rent | (r.RentedCar) => r.InitialPosition and r.finalPosition

}

//30.10.2016

fact rentIfAvailable

{

all u:user, r:rent | r.startTime ⬄ u.rentedCar in r.carsAvailable

}

fact noSafeAreaParkNoEndTrip

{

//I can terminate my trip if and only if my car’s position coincides with a parking position or a safa area position

all c:Car, r:Rent, p:Parking, s:SafeArea | r.endTime <=> (c.position = p.position) or (c.position = s.position)

}

assert allCarsWithPosition

{

all c:Car | c.position

}

fact applyDiscountForPassNumb

{

//for every trip that involves a number of passengers >=3, apply a discount for the renter

all r:Rent | (#r.passengers>= 3) => r.applyDiscount

}

assert userMustHaveAtLeastOnePaymentInfo

{

no u:User, p:Payment | u.payInfo not in p.payInfo

}