6 Alloy Model

We will use the software Alloy Analyzer, in order to verify the consistency of our model and relative constraints.

6.1 Alloy Source Model

The complete model of the system in alloy code is reported as follows:

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.reservationStatus = ACTIVE)

or (one res: Reservation | this.(~(res.reserve)) != none and res.reservationStatus = TIMEUP)

or (one res: Reservation | this.(~(res.reserve)) != none and res.reservationStatus = CANCELED)

or (one res: Reservation | this.(~(res.reserve)) != none and res.reservationStatus = COMPLETE and

(one rd: Ride| rd.rideStatus = False and this.(~(Reservation.reserve)).(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.reservationStatus = 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.reservationStatus = COMPLETE)

(carStatus = OUTOFSERVICE)

<=> (batteryLevel = EMPTY) or ((one parkPosition) and not(parkPosition in SafeArea.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 = AVAILABLE)

(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).carStatus = RESERVED

}

check noCANCELEDreservationReserveCarhasStateRESERVED for 3

assert noTIMEUPreservationReserveCarhasStateRESERVED {

no res: Reservation| res.reservationStatus = TIMEUP and User.(res.reserve).carStatus = RESERVED

}

pred example {}

run example

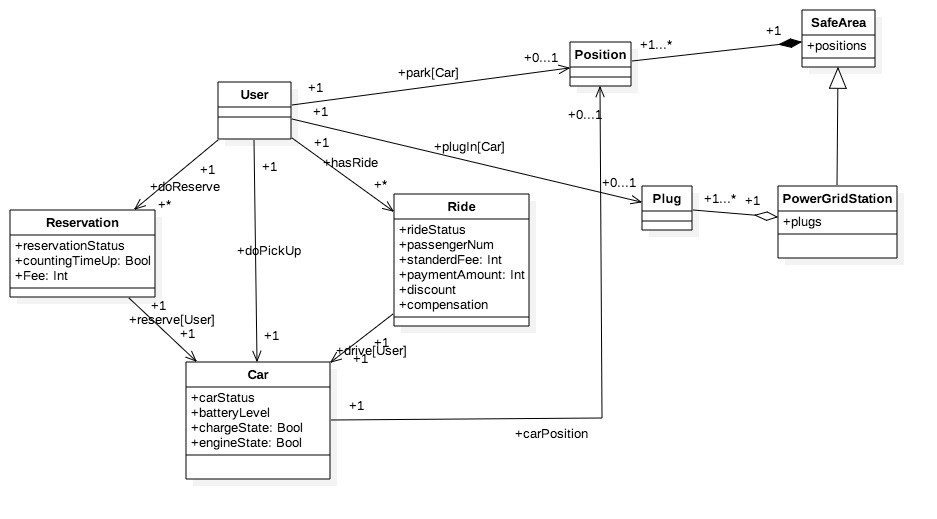
6.2 Analyzer Result

The report of Alloy Analyzer is listed as follows:



6.3 Model Description

In this subsection we will visualize some instances generated by Alloy Analyzer. The following class diagram normally represents the relevant features of our alloy model.



6.3.1 Instance I: Overall World’s Description

The first instance is an overall world’s description, which contains two users with two reservations in different status. The right one implies that User0 has reserved a car, Car1, which is parked in a power grid station. However User0 changed his idea after the reservation and cancelled the reservation, which, according to our model, makes the status of Car1 back to available again. Another user, User1, also has reserved a car, Car0. User1 picked up the car on time and made the reservation status become complete. After picking up, User1 has finished a ride with Car0. He used out the power of Car0, and finally parked it in the power grid station, as well as plugged the car into the power grid. As we can see from the model, the status of Car0 became out of service because of the empty battery level, and the charging state of Car0 stays True, thanks to the plug in action.



6.3.2 Instance II: One Time up Reservation

This particular instance describes a reservation with time up status. According to our model, a user can reserve a car for 1 hour, which means that the user is able to either pick up the car or cancel the reservation within an hour. If the user failed to pick up the car and didn’t cancel the reservation in an hour, the reservation status should become time up and our system will release the reserved car and change it status to available again.



6.3.2 Instance III: One In Use Car

This instance contains a car, which is currently using by a user. According to our model, the user should have reserved the car and have completed the reservation by picking up the car within an hour. After that the car would be used for a ride, and the car status should stay in use as long as the user doesn’t end this ride.

