MARCO

24/10/16: 1h

26/10/16: 1h

27/10/16: 2h

30/10/16: 1h

05/11/16: 2h 30’

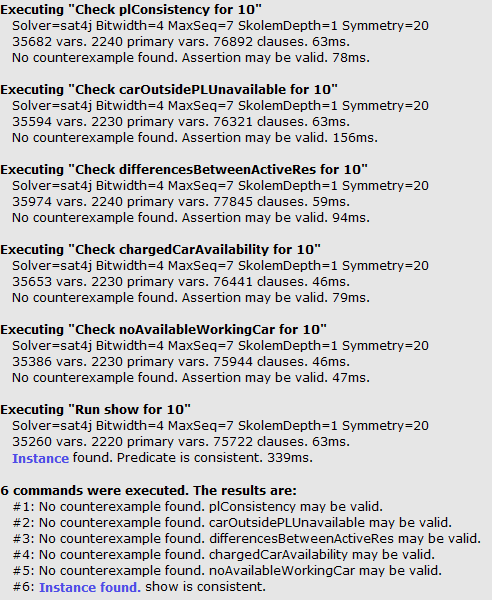
06/11/16: 1h

07/11/16: 2h

8/11/16: 8h

9/11/16: 4h

10/11/16: 30’



////////////////////////////////ENUMS///////////////////////////////////////////

enum CarStatus {

AVAILABLE,

UNAVAILABLE

}

enum CarEngine {

ON,

OFF

}

enum LotStatus {

EMPTY,

USED

}

enum ReservationStatus{

EXPIRED,

ACTIVE

}

enum BatteryStatus{

FULL,

MEDIUM,

LOW,

VERYLOW

}

////////////////////////////////PREDICATES///////////////////////////////////////////

sig ValidString{}

sig Supervisor {

manages: set Car

}{

}

sig RegisteredUser {

name: one ValidString,

creditCard: one Int,

position: one Position

}{

#creditCard>0

}

sig ReservationDB{

contains: set Reservation

}{

#ReservationDB=1

}

sig Reservation {

reservationID : Int,

bookedBy: one RegisteredUser,

relatedTo: one Car,

reservationStatus: one ReservationStatus

}{

reservationID>0

}

sig Position {

position\_x: one Int,

position\_y: one Int

}

sig Car {

carID: Int,

parkedInto: lone ParkingLot,

position: one Position,

carStatus: one CarStatus,

carEngine: one CarEngine,

battery: one BatteryStatus

}{

carID>=0

}

sig SafeArea {

contains: set ParkingLot

}

{

#contains > 0

}

sig ParkingLot {

safeArea : one SafeArea,

position : one Position,

status: one LotStatus

}

sig CHParkingLot extends ParkingLot{

}

////////////////////////////////FACTS///////////////////////////////////////////

// There's at least a Safe Area in the world

fact atLeastASafeArea{

#SafeArea>0

}

// There's at least a car in the world

fact atLeastACar{

#Car>0

}

//There's only a Supervisor

fact thereIsOnlyASupervisor{

#Supervisor=1

}

//Every car is managed by the Supervisor

fact everyCarIsManaged{

all c : Car | c in Supervisor.manages

}

//Every reservation is stored, even the Expired ones

fact theReservationDBContainsAllTheReservations{

all r: Reservation | one rDB: ReservationDB | r in rDB.contains

}

// There aren't duplicated Cars

fact noDuplicatedCar {

no car1 , car2 : Car |(car1!=car2)&&( car1.carID = car2.carID)

}

// No cloned reservations

fact noClonedReservations {

no reservation1 , reservation2 : Reservation |(reservation1!=reservation2)&&( reservation1.reservationID = reservation2.reservationID)

}

// different parkingLot must have different positions

fact PLUnity{

no pl1, pl2 : ParkingLot |(pl1!=pl2) && (pl1.position = pl2.position)

}

// different car must have different positions

fact correctCarPosition{

no c1, c2 : Car |(c1!=c2) && (c1.position = c2.position)

}

// Two car can't be parked at the same ParkingLot

fact noAbusedParkingLot {

no car1,car2 : Car |(car1!=car2)&&( car1.parkedInto=car2.parkedInto)

}

//Check for correct Pl's statuses

fact PLStatusesconsistency{

all pl : ParkingLot | (no c: Car | (c.parkedInto=pl))=>(pl.status!=USED) else (pl.status=USED)

}

// A parked car must have the same position of a parkingLot

fact carAndLotInTheSameZone{

all c : Car |#c.parkedInto>0 => ( c.position=c.parkedInto.position)

}

fact carNotParkedHasNoSamePositionOfPL {

all c : Car, pl: ParkingLot |#c.parkedInto=0 => ( c.position != pl.position)

}

//containment relation must be bidirectional

fact bidirectional1{

all s: SafeArea, pl: s.contains | pl.safeArea = s

}

fact bidirectional2{

all pl: ParkingLot, s: pl.safeArea | pl in s.contains

}

// A user can't have two "Active" reservations

fact{

no r1, r2: Reservation |( (r1!=r2)&&(r1.reservationStatus=ACTIVE)&& (r2.reservationStatus=ACTIVE))&&(r1.bookedBy=r2.bookedBy)

}

//A car should be related for just one "ACTIVE" reservation at a time

fact{

no r1, r2: Reservation |( (r1!=r2)&&(r1.reservationStatus=ACTIVE)&& (r2.reservationStatus=ACTIVE))&&(r1.relatedTo=r2.relatedTo)

}

//A car should be "Available" only if it's parked in a safe area

fact CarAvailableOnlyIfInSafeArea{

no c: Car | c.parkedInto = none && c.carStatus=AVAILABLE

}

//a car whose engine is on, must be connected to an active reservation

fact WorkingCarMeansActiveReservation{

all c : Car| some r : Reservation | c.carEngine=ON => r.relatedTo=c && r.reservationStatus=ACTIVE

}

//If a car is being used, should be "Unavailable"

fact carStatusConsistency{

all r : Reservation |(r.reservationStatus=ACTIVE)=>(r.relatedTo.carStatus=UNAVAILABLE)

}

// all car with low battery and not already in use, are not availabe for booking

fact batteryConsistency{

all c : Car | (c.battery=VERYLOW)=>(c.carStatus=UNAVAILABLE)

}

// all car parked in a safe area, with not empty battery and not booked, should be available

fact batteryConsistency2{

all c : Car | c.parkedInto!=none && c.battery!=VERYLOW && ((Reservation:>relatedTo.c)&(Reservation:>reservationStatus.ACTIVE))=none=>(c.carStatus=AVAILABLE)

}

////////////////////////////////ASSERTIONS///////////////////////////////////////////

pred show{}

// A parking lot whose position coincides with the position of a car, should not be marked as empty

assert plConsistency{

no c: Car, pl: ParkingLot | (c.position=pl.position && pl.status=EMPTY)

}

check plConsistency for 10

//A car whose position does not coincide with any parking lot's position should be UNAVAILABLE

assert carOutsidePLUnavailable{

no c: Car | c.position&ParkingLot.position = none && c.carStatus= AVAILABLE

}

check carOutsidePLUnavailable for 10

// active reservations should refers to different cars and different users

assert differencesBetweenActiveRes{

no r1, r2: Reservation| (r1!=r2)&&(r1.reservationStatus=ACTIVE)&&(r2.reservationStatus=ACTIVE)&&((r1.bookedBy=r2.bookedBy)||(r1.relatedTo=r2.relatedTo))

}

check differencesBetweenActiveRes for 10

// all car with not empty battery, not booked and parked, should be available

assert chargedCarAvailability{

no c: Car| c.parkedInto!=none && c.battery!=VERYLOW && Reservation:>relatedTo.c&Reservation:>reservationStatus.ACTIVE=none && c.carStatus=UNAVAILABLE

}

check chargedCarAvailability for 10

//a car whose engine is working should not be available

assert noAvailableWorkingCar{

no c: Car| c.carEngine = ON && c.carStatus=AVAILABLE

}

check noAvailableWorkingCar for 10

///////////////////////////////////////////////////////////////////////////////////

/\*

assert noCarAvailable{

no r1, r2, r3: Car|(r1!=r2)&&(r2!=r3)&&(r1!=r3)&&(r1.carStatus=AVAILABLE)&&(r2.carStatus=AVAILABLE)&&(r3.carStatus=AVAILABLE)

}

check noCarAvailable for 5

assert noReservationActive{

no r1, r2, r3: Reservation |(r1!=r2)&&(r2!=r3)&&(r1!=r3)&&(r1.reservationStatus=ACTIVE)&&(r2.reservationStatus=ACTIVE)&&(r3.reservationStatus=ACTIVE)

}

check noReservationActive for 5

\*/

run show for 5

