/\* General objects \*/

**sig** Time {}

**sig** Position {}

**abstract** **sig** Boolean {}

**one** **sig** True **extends** Boolean {}

**one** **sig** False **extends** Boolean {}

/\* Battery status \*/

**abstract** **sig** Battery {}

**one** **sig** Low **extends** Battery {}

**one** **sig** High **extends** Battery {}

/\* Possible states in which a car can be \*/

**abstract sig** CarState {}

**one sig** Available **extends** CarState {}

**one sig** Reserved **extends** CarState {}

**one sig** InUse **extends** CarState {}

**one sig** NotAvailable **extends** CarState {}

/\* Event that may occur during a ride \*/

**abstract** **sig** Event {

timestamp: **one** Time

}

/\* Types of event that can occur during a ride \*/

**one sig** Unlock **extends** Event {}

**one sig** StartDriving **extends** Event {}

**one sig** StopDriving **extends** Event {}

/\* Piece of information retrieved from a car during the ride \*/

**sig** Sample {

location: **one** Position,

timestamp: **one** Time

}

/\* Safe area definition \*/

**sig** SafeArea{

location: **one** Position

}

/\* Types of payment \*/

**sig** Bill {}

**abstract sig** PaymentInformation {}

**sig** CreditCard **extends** PaymentInformation {}

**sig** BankAccount **extends** PaymentInformation {}

/\* User definitions \*/

**abstract** **sig** User {}

sig UnregisteredUser extends User{}

sig RegisteredUser extends User {

reservations: set Reservation,

rides: set Ride,

bills: set Bill,

paymentType : one PaymentInformation

} {

/\* If the user has never reserved any car, no bills should be charged \*/

#reservations = 0 implies #bills = 0

}

/\* Car definition \*/

sig Car {

currentPosition: one Position,

state: one CarState,

locked: one Boolean,

batteryLevel : one Battery

} {

/\* If the battery level is Low, then the car should not be available \*/

batteryLevel = Low implies state = NotAvailable

/\* If car is available then it must be locked \*/

state = Available implies locked = True

/\* If car is not available then it must be locked \*/

state = NotAvailable implies locked = True

}

/\* Reservations \*/

sig Reservation {

car: one Car,

timestamp: one Time

}

/\* Rides \*/

sig Ride {

car: one Car,

from: lone Sample,

to: lone Sample,

parks: lone SafeArea,

event: set Event,

reservation: one Reservation,

intermediatePosition: set Sample,

initTimestamp: lone Time,

finalTimestamp: lone Time

} {

/\* From position is set only if there is a start driving event \*/

(#from > 0) <=> StartDriving **in** event

/\* To position is set only if there is a stop driving event \*/

(#to > 0) <=> StopDriving **in** event

/\* Ride has samples only if a start driving event is present \*/

(#intermediatePosition > 0) **implies** StartDriving **in** event

/\* User can't park before Stop Driving \*/

#to > 0 <=> #parks > 0

}

/\* Stop driving event is present only if there is at least one start driving event \*/

**fact** StopDrivingOnlyIfStartDriving {

**all** ride : Ride | StopDriving **in** ride.event **implies** StartDriving **in** ride.event

}

/\* Start driving event is present only if there is at least one unlock event \*/

**fact** StartDrivingOnlyIfUnlock {

**all** ride : Ride | StartDriving **in** ride.event **implies** Unlock **in** ride.event

}

/\* Only one car can be found in a given position \*/

**fact** OneCarInEachLocation {

**all** **disj** c1, c2 : Car | c1.currentPosition != c2.currentPosition

}

/\* User must reserve before ride a car \*/

**fact** ReservationBeforeRide {

**all** ride: Ride, ru : RegisteredUser | ride **in** ru.rides **implies** ride.reservation **in** ru.reservations

}

/\* Every Safe Area is located in a different position \*/

**fact** DifferentPositionForEachSafeArea {

**all** **disj** sa1, sa2 : SafeArea | sa1.location != sa2.location

}

/\* You can ride a car only once per reservation \*/

**fact** OneRidePerReservation {

**all** **disj** r1, r2 : Ride | r1.reservation != r2.reservation

}

/\* Every sample must be associated to a ride \*/

**fact** SampleMustBelongToRide {

**all** s : Sample | some r : Ride | s **in** (r.from + r.to + r.intermediatePosition)

}

/\* Every bill must be associated only to one user \*/

**fact** OnlyOneUserPerBill {

**all** r1, r2 : RegisteredUser | r1 != r2 **implies** **not** (**some** b : Bill | b **in** r1.bills **and** b **in** r2.bills)

**all** b : Bill | (**some** regUser : RegisteredUser | b **in** regUser.bills)

}

/\* Car can be unlock only if there is at least one unlock event \*/

fact CarIsUnlockedIfThereIsAtLeastOneUnlockEvent {

all c : Car | c.locked = False implies (some ride : Ride | (ride.car = c and (Unlock in ride.event)))

}

/\* User can stop driving only once \*/

fact AtMostOneStopDrivingPerRide {

all ride : Ride | (lone ev : Event | (ev in ride.event and ev = StopDriving))

}

/\* Every event must belong to one ride \*/

fact EventMustBelongToRide {

all ev : Event | (some ride : Ride | ev in ride.event)

}

/\* Every reservation can be made only by one user \*/

fact OnlyOneUserPerReservation {

all res : Reservation | (one usr : RegisteredUser | res in usr.reservations)

}

/\* The payment information must belong only to one user \*/

fact OnlyOneUserPerPayment {

all pi : PaymentInformation | one usr : RegisteredUser | pi in usr.paymentType

}

fact ParksPositionequalToPosition {

all ride : Ride | #ride.parks > 0 implies ride.parks.location = ride.to.location

}

pred show {}

pred reserveCar[user, user' : RegisteredUser, c, c' : Car] {

c.state = Available

implies (

(one r : Reservation | not r in user.reservations

and r in user'.reservations

and r.car = c'

and user.reservations = (user'.reservations - r))

and

c'.state = Reserved

)

}

pred unlockCar[ride, ride' : Ride] {

Unlock not in ride.event

Unlock in ride'.event

ride.car = ride'.car

one ru : RegisteredUser | ride.reservation in ru.reservations and ride'.reservation in ru.reservations and ride in ru.rides and ride' in ru.rides

}

pred startDriving[ride, ride' : Ride] {

StartDriving not in ride.event

StartDriving in ride'.event

ride.car = ride'.car

one ru : RegisteredUser | ride.reservation in ru.reservations and ride'.reservation in ru.reservations and ride in ru.rides and ride' in ru.rides

#ride.from = 0

#ride'.from = 1

#ride.to = 0

}

pred stopDriving[ride, ride' : Ride] {

one ev : Event |

(

not ev in ride.event

and ev in ride'.event

and ev = StopDriving

)

ride.car = ride'.car

one ru : RegisteredUser | ride.reservation in ru.reservations and ride'.reservation in ru.reservations and ride in ru.rides and ride' in ru.rides

ride.from = ride'.from

#ride.to = 0

#ride'.to = 1

}

//run show for 6 but exactly 2 Ride

run unlockCar for 6 but exactly 2 Ride, 1 RegisteredUser, 2 Reservation, 2 Sample, 1 Car