```
abstract sig Person{
    }

sig Taxi_driver extends Person{
    incoming:lone Call  }

sig User extends Person{
    location:set Location,
    past_call:set Call,
    blank_call:set Call}

abstract sig Call{

    }

sig Delayed_call extends Call{
    caller:one User,
    start: one Address,
    destination: one Address
    }

sig Immediate_call extends Call{
    caller:one User,
    start: one Address
    }

sig Shared_call extends Call{
    start: some Address,
    destination: some Address,
    caller: some User}

sig Zone{
    address: some Address   }

sig Queue {
    zone:one Zone,
    drivers: set Taxi_driver}

sig Location{
    address: one Address}

sig Address{
    }

//FACTS

fact OneDriverPerCall{
    //ad ogni call deve corrispondere un tassista se attiva oppure essere un past call di un
    utente
    all c:Call | (lone t:Taxi_driver | t.incoming=c )
    all c:Call| (some u:User |(no t:Taxi_driver | c=t.incoming <=> (c in u.past_call)))
}

fact BlankCalls {
    //le chiamate perse non possono essere più di quelle effettuate e devono essere
    contenute in quelle passate
    all u:User|(all c: Call |(c in u.blank_call implies c in u.past_call))}
```

```
fact MaxOneActiveCall {
    //un utente può avere solo una chiamata attiva
    all u:User, i1:Immediate_call, i2:Immediate_call | ((i1!=i2) implies (i1.caller=u
    implies (i2.caller != u or i2 in u.past_call)))
    all u:User, s1:Shared_call, s2:Shared_call | ((s1!=s2) implies (u in s1.caller implies
    (u not in s2.caller  or s2 in u.past_call)))
    all u:User, s:Shared_call, i:Immediate_call | ((u in s.caller implies (u != i.caller  or
    i in u.past_call))and (u = i.caller implies (u not in s.caller  or s in u.past_call)))
    }

fact DestinationNotStart {
    //una destinazione non può essere la partenza
    all i:Delayed_call | i.start!=i.destination
}

fact NotEmptyQueue{
    //in ogni coda deve contenere almeno un tassista
    all q:Queue | (some t:Taxi_driver | t in q.drivers)
}

fact NumberAdressesShared{
    //partenze e arrivi al massimo uguali al numero di utenti
     all c:Shared_call | (#c.start <=#c.caller && #c.destination<=#c.caller)}

fact SameStartZone{
    //tutti gli utenti devono partire dalla stessa zona
    all c:Shared_call|(one z:Zone |c.start in z.address)
}

fact QueueInZone {
    //ogni zona deve avere una e una sola queue
    all z:Zone |(one q:Queue |q.zone=z)
}

fact OnlyMyCalls {
    //ogni utente ha in lista solo le sue calls
    all u:User |(all c:Immediate_call|(c in u.past_call implies c.caller=u))
    all u:User |(all c:Delayed_call|(c in u.past_call implies c.caller=u))
    all u:User |(all c:Shared_call | (c in u.past_call implies u in c.caller))
    }

fact OneQueuePerTaxi{
    //un tassista deve essere in una sola coda
    all t:Taxi_driver |(lone q:Queue |t in q.drivers)
}

fact LocationInUser{
    //una location di una chiamata deve essere nelle location dell'utente
    all l:Location|(one u:User | (l in u.location))}

fact NoOrphanAddress{
    //non ci sono indirizzi senza zona
    all a:Address| (one z:Zone |(a in z.address))}


//ASSERTIONS
```

```
assert NoOrphanCalls {
    //controlla che non ci siano chiamate senza tassista o utente
    no c: Call | ((no u: User | c in u.past_call) && (no t: Taxi_driver | t.incoming=c))
}
check NoOrphanCalls

assert NoDifferentStart {
    //controlla che non ci siano utenti che partono da zone diverse
    all c : Shared_call, a1,a2 : Address |(a1 in c.start and a2 in c.start implies (one
    z:Zone| a1 in z.address and a2 in z.address))
}
check NoDifferentStart

assert MaxBlank {
    //controlla che le chiamate perse non siano maggiori di quelle effettuate
    all u:User | (#u.blank_call<=#u.past_call)}
check MaxBlank



//PREDICATES

//un mondo con tassisti non in coda
pred DriverNotinQUeue {
some t:Taxi_driver| (all q: Queue | t not in q.drivers)
some Location
some Immediate_call
 }
run DriverNotinQUeue


//un mondo con utenti che chiamano ma non prendolo il taxi
pred UserNoRide{
all c:Call|(some u:User | c in u.blank_call)
}
run UserNoRide

//un mondo con almeno 2 utenti che chiamano un taxi condiviso
pred Sharing{
some c:Shared_call | #c.caller>1 and (no u:User | c in u.past_call)
}

run Sharing

//un mondo in cui un utente ha prenotato una chiamata da una zona preferita ad un'altra
pred Favorite {
one u:User| (some c:Delayed_call |( c.start in u.location.address and c.destination in
u.location.address and c not in u.past_call))
}

run Favorite

//un mondo con 2 zone
pred TwoZones{
#Person>2
#Zone>1
```

```
}
run TwoZones

//partenza e arrivo in zone diverse
pred DifferentZone{
one c:Delayed_call | some z1,z2:Zone | (c.start in z1.address and c.destination in
z2.address and z1!=z2)
}
run DifferentZone
```