# 4. FORMAL ANALYSIS USING ALLOY

## 4.1. Code

The main goal of our Alloy analysis is to validate the model, and see some samples of our eMALL system. For showing the result, we assume a world that exist one end user, one CPO, 3 cars, and more than 2 sockets.

////segments

sig Firstname{}

sig Lastname{}

sig Email{}

sig Password{}

sig NameOfCar{}

sig Amount{}

sig Company{}

sig Capacity{}

sig CompanyName{}

sig Username{}

sig Wallet{}

sig EndUser {

    firstname: one Firstname,

    lastname: one Lastname,

    email: one Email,

    password: one Password,

    own : set Car,

    receive: set suggestion,

    book : set Booking,

    pay : some Payment,

    accountBalance: one Wallet

}{

    #own > 1

}

sig Car{

    nameOfCar: one NameOfCar,

}

sig suggestion{}

sig Payment{

    amount: one Amount,

    status: one Bool,

}

sig Booking{

    date: one Date,

    socket : one Socket

}

sig session{

    haveBook: one Booking,

    havePayment: one Payment

}

sig Date{

    date: one TypeOfDate

}

sig Socket{

    type: one Type

}

sig ChargingStation{

    location: one Location,

    manageBy: some CPO,

    battery: one Battery,

    dso: some DSO,

    has: some Socket

}{

    #socket > 0

}

sig Battery{

    price: one Int,

    capacity: one Capacity

}{

    #price >0

    #capacity > 0

}

sig DSO{

    price: one Int,

    capacity: one Capacity,

    companyName: one CompanyName,

}{

    #price != 0

    #capacity != 0

}

enum Type{

    slow,

    fast,

    rapid

}

enum TypeOfDate{

    Sunday,

    Monday,

    Tuesday,

    Wednesday,

    Thursday,

    Friday,

    Saturday

}

sig CPO{

    firstname: one Firstname,

    lastname: one Lastname,

    username: one Username,

    password: one Password,

    company: one Company

}

sig Location{

    latitude: one Int,

    longitude: one Int

}{

    latitude >= -5 and latitude =< 5

    longitude >= -5 and longitude =< 5

}

abstract sig Bool{}

one sig TRUE extends Bool{}

one sig FALSE extends Bool{}

////facts

//each email always must be associated with only one user

fact EmailAssociationEndUser{

    all e : Email | one u : EndUser | e in u.email

}

//each Username always must be associated with only one CPO

fact UsernameAssociationCPO{

    all e : Username | one u : CPO | e in u.username

}

//emails of each user are different

fact EmailOfEndUser{

    no disjoint u1,u2: EndUser | u1.email = u2.email

}

//each charging station has its own sokcets

fact SocketsInChargingStation{

    all disjoint c1,c2 : ChargingStation | c1.has != c2.has

}

//Username of each CPO is different

fact UsernameOfCPO{

    no disjoint u1,u2: CPO | u1.username = u2.username

}

// each user must have some car

fact CarsOfEndUser{

    all u : EndUser | some c : Car | c in u.own

}

//the locations of each charging station must be different

fact LocationOfChargingStation{

    all disjoint c1,c2 : ChargingStation| c1.location != c2.location

}

// latitudes and longitudes of all charging stations must be different

fact latitudesAndlongitudes{

    all disjoint l1,l2: Location |

    l1.latitude = l2.latitude => l1.longitude != l2.longitude

}

//each socket must have only one type

fact TypeOfSocket{

    all t : Type | one s : Socket | t in s.type

}

//each CPO must work only for one company

fact CompanyForCPO{

    all a : CPO | one c : Company | c in a.company

}

// in each charging station some CPOs work

fact ChargingStationCPOs{

    all c : ChargingStation | some a : CPO | a in c.manageBy

}

// each socket only for one booking   // its redundant

//fact SocketForBooking{

//    all disjoint b1,b2 : Booking | b1.socket != b2.socket

//}

// in each session, there are only one payment

fact SessionForPaymentAndBooknig{

    all s : session | one p : Payment | p in s.havePayment

}

// in each session, there are only one booking

fact SessionForPaymentAndBooknig{

    all s : session | one b : Booking | b in s.haveBook

}

// each user can have several payments

fact SeveralPaymentForEndUser{

    all u : EndUser | some p : Payment | p in u.pay

}

//each user can have some suggestion

fact SuggetionForEndUser{

    all u : EndUser | some s : suggestion | s in u.receive

}

//each charging station can have only one battery

fact BatteriesInChargingStation{

    all c : ChargingStation | one b : Battery | b in c.battery

}

//the battery of each charging stations are different

fact ChargingStationForBatteries{

    all disjoint c1,c2 : ChargingStation | c1.battery != c2.battery

}

//each cahrging station can have several DSOs

fact BatteriesInChargingStation{

    all c : ChargingStation | some d : DSO | d in c.dso

}

//each Booking must have only one date

fact DateOfBooking{

    all b : Booking | one d : Date | d in b.date

}

//each DSO must have only one companyName

fact CompanyNameOfDSO{

    all d : DSO | one n : CompanyName | n in d.companyName

}

//each DSO must have only one capacity

fact CapacityOfDSO{

    all d : DSO | one c : Capacity | c in d.capacity

}

//each car must have only one name

fact NameOfCar{

    all c : Car | one n : NameOfCar | n in c.nameOfCar

}

//each user must have different payment

fact PaymentOFEndUser{

     all disjoint u1,u2: EndUser | u1.pay != u2.pay

}

//each session has different booking

fact SessionForBooking{

    all disjoint s1,s2: session | s1.haveBook != s2.haveBook

}

//each session has different payment

fact SessionForBooking{

    all disjoint s1,s2: session | s1.havePayment != s2.havePayment

}

//cars of each user must be different

fact CarsOfEndUsers{

    all disjoint u1,u2 : EndUser | u1.own != u2.own

}

//each wallet always must be associated with only one user

fact WalletOnlyForEndUser{

   all w : Wallet | one u : EndUser | w in u.accountBalance

}

//wallets of each end user must be different

fact WalletsOfEndUsers{

    no disjoint u1,u2: EndUser | u1.accountBalance = u2.accountBalance

}

//predicates

pred AddSuggestion[u1,u2: EndUser, s: suggestion]{

    u2.receive = u1.receive + s

}

pred CreateBooking[u1,u2: EndUser, b: Booking]{

    u2.book = u1.book + b

}

pred AddPayment[u1,u2: EndUser, p: Payment]{

    u2.pay = u1.pay + p

}

pred world1{

    #EndUser =1

    #CPO=1

    #Car=3

    #Socket >2

}

run world1 for 5

## 4.2. Results

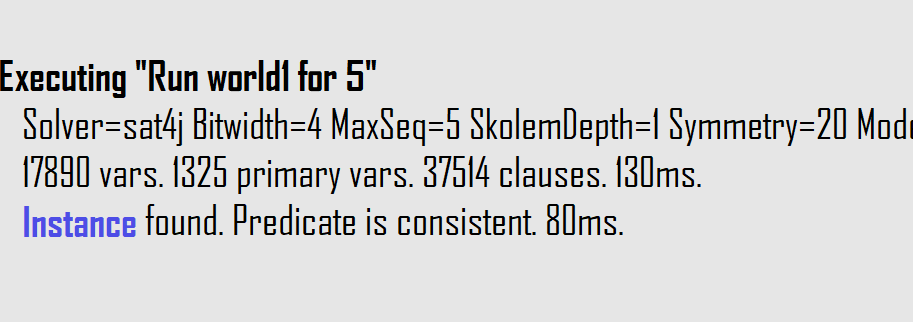
****

Figure 20 result for world 1

## 4.3. Generated Instances

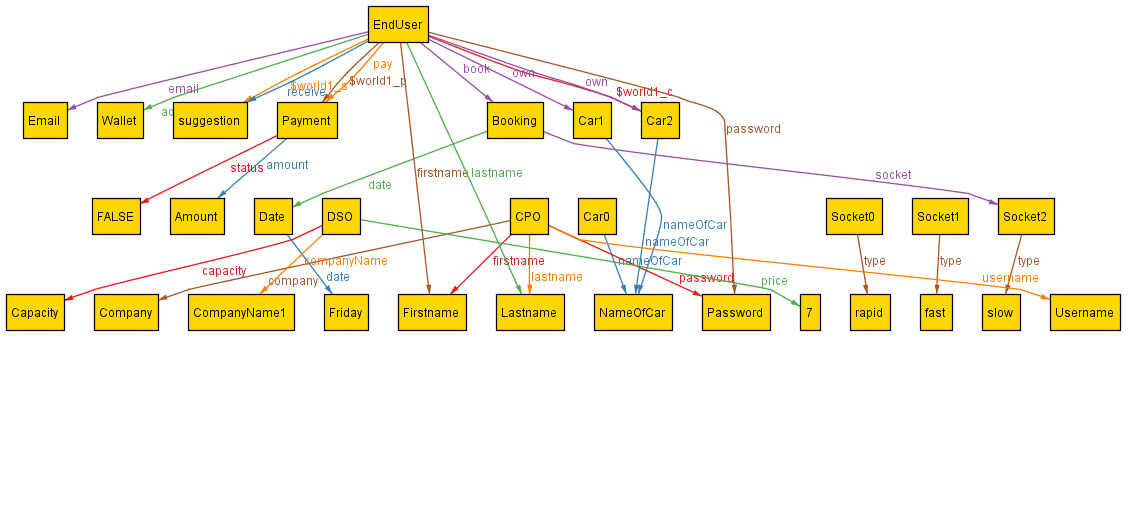


Figure 21 generated instance 1 for world 1

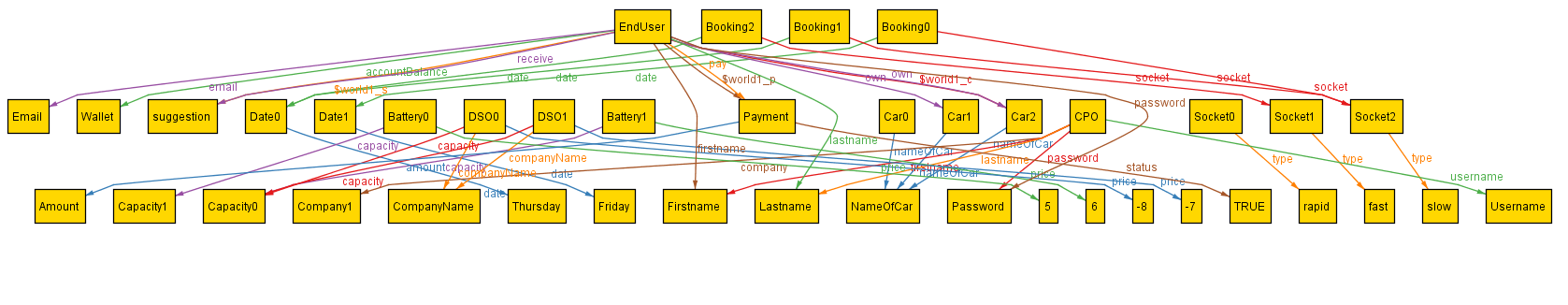


Figure 22 generated instance 2 for world 1

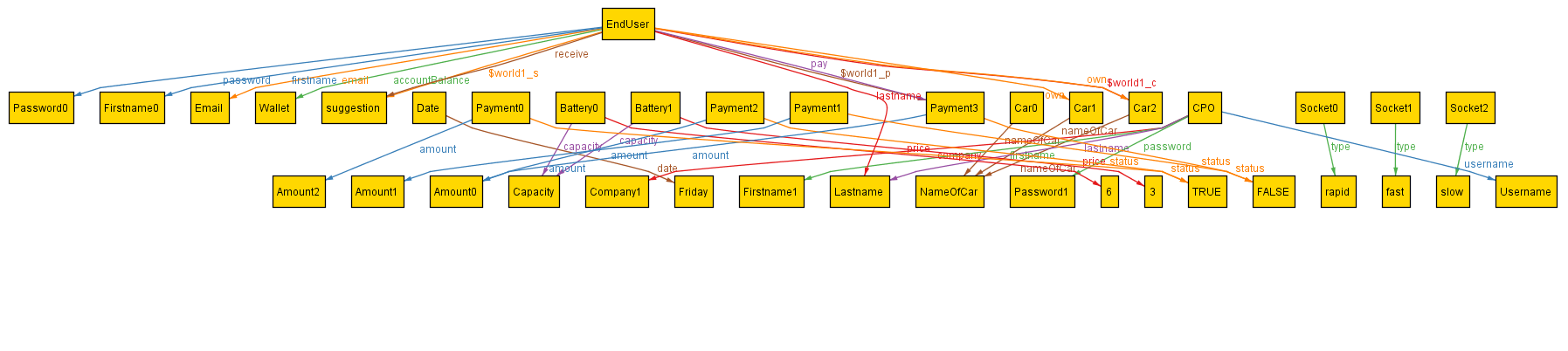


Figure 23 generated instance 3 for world 1

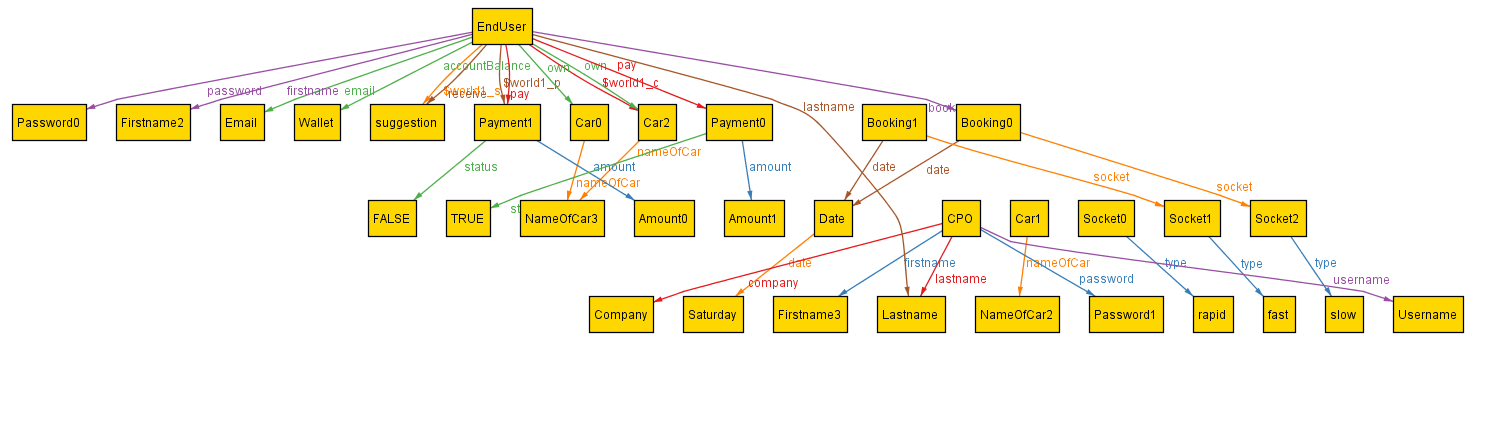


Figure 24 generated instance 4 for world 1