Validation of OCL constraints. Single-threaded execution

Now, we show how to dynamically validate the OCL constraints in its class diagram, during the execution of such a prototype. As we assume a single threaded Maude prototype of the model, we use the `CINEMA` model previously used with simulation purposes. However now we do not import the STACK module but the `VALIDATION-STACK` because it includes some modifications on the `STACK` module which allows as to validate constraints during the simulation process.

```
mod CINEMA is
  pr CLASSES-CINEMA .
  pr VALIDATION-STACK .

  ...
endm
```

Furthermore, we need a module which defines constants `inv`, `pre` and `post` for invariants and pre- and post-conditions. We use the following module `CINEMA-CONSTRAINTS`:

```
mod CINEMA-CONSTRAINTS  is
  pr CLASSES-CINEMA .

  ops $C $T $S $T1 $T2 : -> Vid .

  op seats-available : -> OclExp .
  eq seats-available
    = (context Session inv  capacity >=  ticket -> size()) .

  op avoid-overlapping : -> OclExp .
  eq avoid-overlapping
    = context Client inv :
         ticket -> forAll(T1 | ticket -> forAll(T2 |
            (T1 = T2)
            or (T1 . session . endTime < T2 . session . startTime)
            or (T2 . session . endTime < T1 . session . startTime))))


  op avoid-overlapping : -> OclExp .
  eq avoid-overlapping
    = (context Client inv
       ticket -> forAll($T1 | ticket -> forAll($T2 |
       (($T1 = $T2)
       or ($T1 . session . endTime < $T2 . session . startTime)
       or ($T2 . session . endTime < $T1 . session . startTime))))) .

  --- ---------------- Invariants  ----------------------
```

```
  eq inv =  (seats-available and avoid-overlapping) .

  --- ---------------- Preconditions --------------------
  eq pre(buyTicket)
    = (sessions
          -> select($S | $S . startTime = startTime) -> size() = 1) .

  --- ---------------- Postconditions -------------------
  eq post(buyTicket)
    = ((result = null)
       or
      (sessions -> select($S |
             $S . startTime = startTime) . ticket -> includes(result)
      and
      (sessions -> select($S |
             $S . startTime = startTime) . ticket -> asSet()
       - sessions -> select($S |
             $S . startTime = startTime) . ticket @pre -> asSet())
       -> size() = 1)) .
endm
```

Finally, we need some configuration representing given initial states. Now we use the `init-state1` and `init-state2` configurations, already used in Appendix **??** with simulation purposes. The `init-state1` defines a correct execution and we get a correct final state:

```
Maude> rewrite in CINEMA-VALIDATION-TEST : [ init-state1 ] .
rewrite in CINEMA-VALIDATION-TEST : [init-state1] .
rewrites: 28421 in 4ms cpu (16ms real) (7105250 rewrites/second)
result Configuration+: {next-goCinema-call(6) next-ticket(6)
    < stack : Stack | state : nil >
    < cnm1 : Cinema | name : "Coronet",
                      bank : bbva,sessions : Set{s1, s2, s3} >
    < bbva : Bank | cards : (qas(111, acc1)
                             $$ qas(222, acc2) $$ qas(333, acc3)) >
    < s1 : Session | startTime : 1100,endTime : 1150,
                     capacity : 10,price : 5,ticket : Set{2, 1} >
    < s2 : Session | startTime : 1200,endTime : 1250,
                     capacity : 10,price : 8,ticket : Set{4, 3} >
    < s3 : Session | startTime : 1300,endTime : 1350,
                     capacity : 10,price : 5,ticket : Set{5} >
    < bob : Client | ticket : Set{3, 1},cinemas : Set{cnm1},
                     debitCard : 111 >
    < ada : Client | ticket : Set{4, 2},cinemas : Set{cnm1},
                     debitCard : 222 >
    < tom : Client | ticket : Set{5},cinemas : Set{cnm1},
                     debitCard : 333 >
    < acc1 : Account | bal : 87 >
    < acc2 : Account | bal : 987 >
    < acc3 : Account | bal : 9995 >
```

```
    < 1 : Ticket | seat : 0,session : s1,client : bob >
    < 2 : Ticket | seat : 0,session : s1,client : ada >
    < 3 : Ticket | seat : 0,session : s2,client : bob >
    < 4 : Ticket | seat : 0,session : s2,client : ada >
    < 5 : Ticket | seat : 0,session : s3,client : tom >}
```

However, the `init-state2` configuration defines an erroneous computation because `ada` tries to buy 2 ticket for the same sessions, thus violating the `avoid-overlapping` invariant.

```
Maude> rewrite in CINEMA-VALIDATION-TEST : [ init-state2 ] .
rewrite in CINEMA-VALIDATION-TEST : [init-state2] .
rewrites: 7175 in 4ms cpu (4ms real) (1793750 rewrites/second)
result Configuration+: {error("Invariant or postcondition violation",
    goCinema,next-goCinema-call(2) next-ticket(3)
    < cnm1 : Cinema | name : "Coronet",
                      bank : bbva,sessions : Set{s1, s2, s3} >
    < bbva : Bank | cards : (qas(111,acc1)
                             $$ qas(222, acc2) $$ qas(333, acc3)) >
    < s1 : Session | startTime : 1100,endTime : 1150,
                     capacity : 10,price : 5,ticket : Set{mt-ord} >
    < s2 : Session | startTime : 1200,endTime : 1250,
                     capacity : 10,price : 8,ticket : Set{2, 1} >
    < s3 : Session | startTime : 1300,endTime : 1350,
                     capacity : 10,price : 5,ticket : Set{mt-ord} >
    < bob : Client | ticket : Set{mt-ord},cinemas : Set{cnm1},
                     debitCard : 111 >
    < ada : Client | ticket : Set{2, 1},cinemas : Set{cnm1},
                     debitCard : 222 >
    < tom : Client | ticket : Set{mt-ord},cinemas : Set{cnm1},
                     debitCard : 333 >
    < acc1 : Account | bal : 100 >
    < acc2 : Account | bal : 984 >
    < acc3 : Account | bal : 10000 >
    < 1 : Ticket | seat : 0,session : s2,client : ada >
    < 2 : Ticket | seat : 0,session : s2,client : ada >)}
```