Assignment 3
Formal Methods & Software Engineering

Haseeb Irfan-029

Abdullah Nadeem-021

Question 1 – Cinema Ticket Booking System

Basic Types and Report

```
[USER, SHOW, SEAT, BOOKING]

Report ::= ok | userExists | notRegistered | notLoggedIn
           | pastShow | seatUnavailable | bookingUnknown
```

System State

```
TicketSystem
_____________________________________________
users          : P USER
loggedIn       : P USER

shows          : P SHOW
pastShows      : P SHOW

showSeats      : SHOW ↦ P SEAT

booked         : (SHOW × SEAT) ⇸ USER
bookings       : BOOKING ⇸ (USER × SHOW × P SEAT)
_____________________________________________
dom showSeats ⊆ shows
pastShows ⊆ shows
dom booked ⊆ shows × SEAT
```

Operation: DoRegisterUser

```
DoRegisterUser
_____________________________________________
Δ TicketSystem
u?          : USER
rep!        : Report
_____________________________________________
(
  u? ∉ users ∧
  users'     = users ∪ {u?} ∧
  loggedIn'  = loggedIn ∧
  shows'     = shows ∧
  pastShows' = pastShows ∧
  showSeats' = showSeats ∧
  booked'    = booked ∧
  bookings'  = bookings ∧
  rep!       = ok
)
∨
(
  u? ∈ users ∧
  users'     = users ∧
  loggedIn'  = loggedIn ∧
  shows'     = shows ∧
  pastShows' = pastShows ∧
  showSeats' = showSeats ∧
  booked'    = booked ∧
```

```
  bookings'  = bookings ∧
  rep!       = userExists
)
```

Operation: DoLogin

```
DoLogin
_____________________________________________________
Δ TicketSystem
u?          : USER
rep!        : Report
_____________________________________________________
(
  u? ∈ users ∧

  users'     = users ∧
  loggedIn'  = loggedIn ∪ {u?} ∧
  shows'     = shows ∧
  pastShows' = pastShows ∧
  showSeats' = showSeats ∧
  booked'    = booked ∧
  bookings'  = bookings ∧
  rep!       = ok
)
∨
(
  u? ∉ users ∧

  users'     = users ∧
  loggedIn'  = loggedIn ∧
  shows'     = shows ∧
  pastShows' = pastShows ∧
  showSeats' = showSeats ∧
  booked'    = booked ∧
  bookings'  = bookings ∧
  rep!       = notRegistered
)
```

Operation: DoBook

```
DoBook
_____________________________________________________
Δ TicketSystem
user?       : USER
show?       : SHOW
seats?      : P SEAT
bid!        : BOOKING
rep!        : Report
_____________________________________________________
(
  user? ∈ users ∧
  user? ∈ loggedIn ∧
  show? ∈ shows ∧
  show? ∉ pastShows ∧
  seats? ⊆ showSeats show? ∧
  seats? ≠ ∅ ∧
```

```
  (∀ s : SEAT • s ∈ seats? ⇒ (show?, s) ∉ dom booked) ∧
  bid! ∉ dom bookings ∧

  users'     = users ∧
  loggedIn'  = loggedIn ∧
  shows'     = shows ∧
  pastShows' = pastShows ∧
  showSeats' = showSeats ∧

  booked' =
      booked ∪ { (show?, s) ↦ user? | s ∈ seats? } ∧

  bookings' =
      bookings ∪ { bid! ↦ (user?, show?, seats?) } ∧

  rep! = ok
)
∨
(
  user? ∉ users ∧
  users'     = users ∧
  loggedIn'  = loggedIn ∧
  shows'     = shows ∧
  pastShows' = pastShows ∧
  showSeats' = showSeats ∧
  booked'    = booked ∧
  bookings'  = bookings ∧
  rep!       = notRegistered
)
∨
(
  user? ∈ users ∧ user? ∉ loggedIn ∧
  users'     = users ∧
  loggedIn'  = loggedIn ∧
  shows'     = shows ∧
  pastShows' = pastShows ∧
  showSeats' = showSeats ∧
  booked'    = booked ∧
  bookings'  = bookings ∧
  rep!       = notLoggedIn
)
∨
(
  show? ∈ pastShows ∧
  users'     = users ∧
  loggedIn'  = loggedIn ∧
  shows'     = shows ∧
  pastShows' = pastShows ∧
  showSeats' = showSeats ∧
  booked'    = booked ∧
  bookings'  = bookings ∧
  rep!       = pastShow
)
∨
(
```

```
  ∃ s : SEAT • s ∈ seats? ∧ (show?, s) ∈ dom booked ∧
  users'     = users ∧
  loggedIn'  = loggedIn ∧
  shows'     = shows ∧
  pastShows' = pastShows ∧
  showSeats' = showSeats ∧
  booked'    = booked ∧
  bookings'  = bookings ∧
  rep!       = seatUnavailable
)
```

Operation: DoCancel

```
DoCancel
_____________________________________________________
Δ TicketSystem
user?       : USER
bid?        : BOOKING
rep!        : Report
_____________________________________________________
(
  user? ∈ users ∧
  user? ∈ loggedIn ∧
  bid? ∈ dom bookings ∧

  LET (u, sh, Ss) == bookings bid? IN
    u = user? ∧
    sh ∉ pastShows ∧

    users'     = users ∧
    loggedIn'  = loggedIn ∧
    shows'     = shows ∧
    pastShows' = pastShows ∧
    showSeats' = showSeats ∧

    booked' =
      { (sh2,s2) ↦ u2 | (sh2,s2) ↦ u2 ∈ booked ∧ ¬(sh2 = sh ∧ s2 ∈ Ss) } ∧

    bookings' = bookings ∧

    rep! = ok
)
∨
(
  (user? ∉ users ∨ user? ∉ loggedIn ∨ bid? ∉ dom bookings) ∧
  users'     = users ∧
  loggedIn'  = loggedIn ∧
  shows'     = shows ∧
  pastShows' = pastShows ∧
  showSeats' = showSeats ∧
  booked'    = booked ∧
  bookings'  = bookings ∧
  rep!       = bookingUnknown
)
```

Operation: DoResetShow

```
DoResetShow
_____________________________________________________
Δ TicketSystem
show?       : SHOW
rep!        : Report
_____________________________________________________
(
  show? ∈ shows ∧
  booked'   = { (sh,s) ↦ u | (sh,s) ↦ u ∈ booked ∧ sh ≠ show? } ∧
  bookings' = bookings ∧
  users'     = users ∧
  loggedIn'  = loggedIn ∧
  shows'     = shows ∧
  pastShows' = pastShows ∧
  showSeats' = showSeats ∧
  rep!       = ok
)
∨
(
  show? ∉ shows ∧
  users'     = users ∧
  loggedIn'  = loggedIn ∧
  shows'     = shows ∧
  pastShows' = pastShows ∧
  showSeats' = showSeats ∧
  booked'    = booked ∧
  bookings'  = bookings ∧
  rep!       = pastShow
)
```

Operation: ViewBookingHistory (admin view)

```
ViewBookingHistory
_____________________________________________________
Ξ TicketSystem
bh!        : BOOKING ⇸ (USER × SHOW × P SEAT)
_____________________________________________________
bh! = bookings
```

Question 2 – University Course Registration System

Basic Types

```
[STUDENT, COURSE]
```

System State

```
CourseReg
_____________________________________________
students      : P STUDENT
```

```
courses       : P COURSE

completed     : STUDENT ↦ P COURSE
prereq        : COURSE ↦ P COURSE
capacity      : COURSE ↦ ℕ
reg           : STUDENT ↦ P COURSE
_____________________________________________________
dom completed ⊆ students
dom reg ⊆ students
dom prereq ⊆ courses
dom capacity ⊆ courses
```

Operation: AddStudent

```
AddStudent
_____________________________________________________
Δ CourseReg
s?          : STUDENT
_____________________________________________________
(
  s? ∉ students ∧

  students'  = students ∪ {s?} ∧
  courses'   = courses ∧
  completed' = completed ∧
  prereq'    = prereq ∧
  capacity'  = capacity ∧
  reg'       = reg
)
∨
(
  s? ∈ students ∧

  students'  = students ∧
  courses'   = courses ∧
  completed' = completed ∧
  prereq'    = prereq ∧
  capacity'  = capacity ∧
  reg'       = reg
)
```

Operation: AddCourse

```
AddCourse
_____________________________________________________
Δ CourseReg
c?          : COURSE
prereq?     : P COURSE
cap?        : ℕ
_____________________________________________________
(
  c? ∉ courses ∧
  prereq? ⊆ courses ∧

  students'  = students ∧
  courses'   = courses ∪ {c?} ∧
```

```
  completed' = completed ∧
  reg'       = reg ∧

  prereq'    = prereq ∪ { c? ↦ prereq? } ∧
  capacity'  = capacity ∪ { c? ↦ cap? }
)
∨
(
  c? ∈ courses ∧

  students'  = students ∧
  courses'   = courses ∧
  completed' = completed ∧
  prereq'    = prereq ∧
  capacity'  = capacity ∧
  reg'       = reg
)
```

Operation: RegisterCourse

```
RegisterCourse
_____________________________________________________
Δ CourseReg
s?          : STUDENT
c?          : COURSE
_____________________________________________________
(
  s? ∈ students ∧
  c? ∈ courses ∧
  s? ∈ dom reg ∧
  c? ∉ reg s? ∧

  c? ∈ dom prereq ∧
  c? ∈ dom capacity ∧
  s? ∈ dom completed ∧
  prereq c? ⊆ completed s? ∧

  # { st : STUDENT | st ∈ dom reg ∧ c? ∈ reg st } < capacity c? ∧

  students'  = students ∧
  courses'   = courses ∧
  completed' = completed ∧
  prereq'    = prereq ∧
  capacity'  = capacity ∧

  reg'       = reg ⊕{ s? ↦ (reg s? ∪ {c?}) }
)
∨
(
  ¬(
    s? ∈ students ∧
    c? ∈ courses ∧
    s? ∈ dom reg ∧
    c? ∉ reg s? ∧
    c? ∈ dom prereq ∧
    c? ∈ dom capacity ∧
```

```
    s? ∈ dom completed ∧
    prereq c? ⊆ completed s? ∧
    # { st : STUDENT | st ∈ dom reg ∧ c? ∈ reg st } < capacity c?
   ) ∧

  students'  = students ∧
  courses'   = courses ∧
  completed' = completed ∧
  prereq'    = prereq ∧
  capacity'  = capacity ∧
  reg'       = reg
)
```

Operation: DropCourse

```
DropCourse
_____________________________________________________
Δ CourseReg
s?          : STUDENT
c?          : COURSE
_____________________________________________________
(
  s? ∈ dom reg ∧
  c? ∈ reg s? ∧

  let Cs == reg s? \ {c?} in
    students'  = students ∧
    courses'   = courses ∧
    completed' = completed ∧
    prereq'    = prereq ∧
    capacity'  = capacity ∧

    reg' =
      (if Cs = ∅
       then { st ↦ R | st ↦ R ∈ reg ∧ st ≠ s? }
       else reg ⊕ { s? ↦ Cs }
      )
)
∨
(
  ¬(s? ∈ dom reg ∧ c? ∈ reg s?) ∧

  students'  = students ∧
  courses'   = courses ∧
  completed' = completed ∧
  prereq'    = prereq ∧
  capacity'  = capacity ∧
  reg'       = reg
)
```

Question 3 – Reasoning About Loops

Part 1 – Sum = 1 + 2 + … + n

Program:

```
sum := 0;
i   := 1;

while i ≤ n do
    sum := sum + i;
    i   := i + 1
od
```

Loop invariant:

I: sum = $\Sigma_{k=1}^{i-1} k$ and $1 \le i \le n + 1$.

Sketch: initially sum = 0 and i = 1, so the invariant holds.

Each iteration adds i to sum and then increases i, preserving sum = $\Sigma_{k=1}^{i-1} k$.

When the loop exits, i = n + 1 and therefore sum = $\Sigma_{k=1}^{n} k$.

Part 2 – Relationship Between i and j

Example program:

```
i := 0;
j := 0;

while i < n do
    i := i + 1;
    j := j + 2*i - 1
od
```

Tracing a few steps gives:

i = 1,2,3,4 → j = 1,4,9,16 = $1^2, 2^2, 3^2, 4^2$.

So when the loop terminates, i = n and $j = n^2$.

Loop invariant:

I: $j = i^2$ and $0 \le i \le n$.

Sketch: initially i = 0 and j = 0, so $j = i^2$. Each iteration increases i to i+1 and adds 2*i − 1 (with the new i) to j, which turns i^2 into $(i+1)^2$. When the loop exits, i = n and $j = n^2$.