Software Engineering
Exercise 3

Prof. Dr. Peter Thiemann
Sergio Feo-Arenis Sergiy Bogomolov

Albert-Ludwigs-University Freiburg

June 5th, 2014

Exercise 1: The B Method

`INVARIANT` clause for the elevator: *"if the door is open then the current floor is among the pending requests."*

```
INVARIANT
    cab : floors &
    req <: floors &
    door : DSTATE
    &
    (door = open => cab : req)
```

Exercise 1: The B Method

`request(fl)` operation: adds the floor *fl* to the set (not a multiset) of pending requests. It is allowed for the given floor fl to already be a pending request.

```
request (fl) =
        PRE
            fl : floors
        THEN
            req := req \/ {fl}
        END
    ;
```

`move` operation: moves the cab to an arbitrary floor that is among the pending requests. The current floor must not be a pending request, and the door must be closed.

```
move =
      PRE
          req /= {} &
          cab /: req &
          door = closed
      THEN
          cab :: req
      END
   ;
```

`toggle` operation: Opens the door if it is closed, and closes it otherwise. The cab door must be toggled only if the cab's current floor is a pending request. If the operation closes the door, the request is removed from the set of pending requests.

```
toggle =
        PRE
            cab : req
        THEN
            IF
                door = closed
            THEN
                door := open
            ELSE
                req := req - {cab} ||
                door := closed
            END
        END
```

Exercise 1: The B Method

Consistency: for each operation `PRE` P `THEN` S `END`, show:

```
I & P => [S]I.
```

using the *weakest precondition*.

For the `Elevator` machine, we have:

```
I ≡  cab : floors & req <: floors & door : DSTATE
         & (door = open => cab : req)
```

Consistency: for each operation `PRE` P `THEN` S `END`, show:

```
I & P => [S]I.
```

using the *weakest precondition*.

For the `Elevator` machine, we have:

```
I ≡  cab : floors & req <: floors & door : DSTATE
       & (door = open => cab : req)
```

The `request(fl)` operation is consistent:

```
cab : floors & req <: floors & door : DSTATE
& (door = open => cab : req) & fl : floors =>
[req := req \/ {fl}] I
```
<=> (wp)
```
cab : floors & req <: floors & door : DSTATE
& (door = open => cab : req) & fl : floors =>
cab : floors & req \/ fl <: floors & door : DSTATE
& (door = open => cab : req \/ fl)
```
<=> (set theory, predicate logic)
```
true
```

Exercise 1: The B Method

The `move` operation is consistent:

```
cab : floors & req <: floors & door : DSTATE
& (door = open => cab : req)
& req /= {} & cab /: req & door = closed =>
[cab :: req] I
<=> (wp)
cab : floors & req <: floors & door : DSTATE
& (door = open => cab : req)
& req /= {} & cab /: req & door = closed =>
cab : req =>
cab : floors & req <: floors & door : DSTATE
& (door = open => cab : req)
<=> (predicate logic)
true
```

Exercise 1: The B Method

The `toggle` operation is consistent:

```
I & cab : req =>
[IF door = closed THEN door:= open
ELSE req := req - {cab} || door := closed]I
<=> (wp)
I & cab : req =>
(door = closed) & [door:=open]I
or (not door=closed & [req:=req-{cab} || door:=closed]I)
```

Exercise 1: The B Method

```
<=> (wp)
I & cab : req =>
(door = closed) & cab : floors & req <: floors
& open : DSTATE & (open = open => cab : req)
or (not door=closed & cab : floors & req-{cab} <: floors
& closed : DSTATE & (closed = open => cab : req-{cab}))
<=> (predicate logic)
I & cab : req =>
(door = closed) & cab : floors & req <: floors
& (true => cab : req)
or (not door=closed & cab : floors & req-{cab} <: floors
& (false => cab : req-{cab}))
<=> (set theory, predicate logic)
...
<=>
true
```

ProB Demo

Exercise 2: B Types

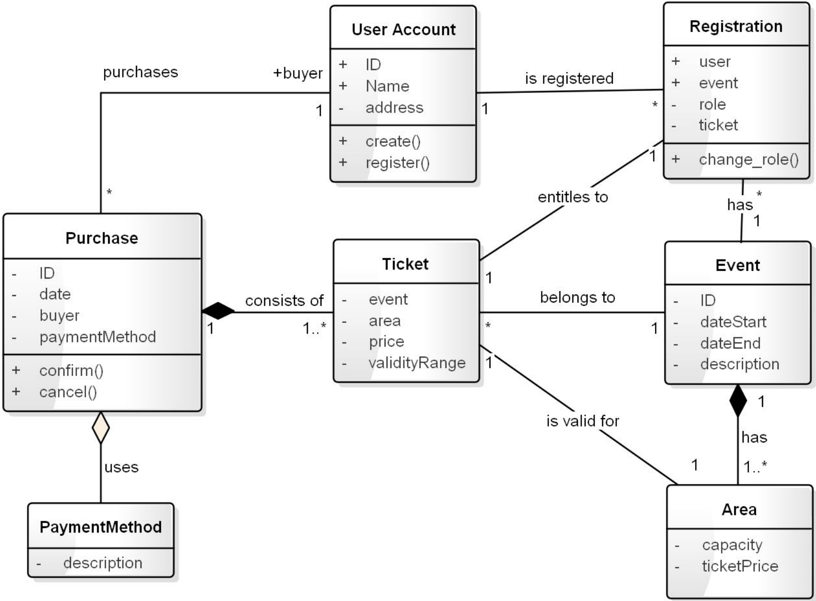
1. `NAT` : $\mathbb{P}(\mathbb{N})$
2. `POW(1)` : not well-typed, POW(S) requires S to be set
3. `{1} * {{1}, NAT}` : $\mathbb{P}(\mathbb{N}) \times \mathbb{P}(\mathbb{P}(\mathbb{N}))$
4. `1:2` : not well-typed, E : S requires S to be a set
5. `card({{{}, {1, {}}, {2}}})` :
   not well-typed, $\{E_1 \ldots\}$ requires $\forall E_i : A$
6. `{x | x = 1 or x = TRUE}` : not well-typed, implies that $x : \mathbb{N} \cup \mathbb{B}$ which is not allowed.
7. `{1}` $\cap$ `{{1}}` :
   not well-typed, $S \cap T$ requires S and T to be of the **same** set type

Exercise 3: OOA

Classes:

- Event
- Users: Organizer, Participant, etc
- Ticket
- Purchase
- Area
- Payment Method

Exercise 3: OOA



Exercise 3: OOA