Tutorial 8

1) If *shopPrice : ITEM* $\twoheadrightarrow$ *N* records the prices charged for items stocked in the shop, write expressions (using *shopPrice'* for the changed state) to show how *shopPrice* changes when the following occur:

```
__Shop______________________________________
| ...
| shopPrice : ITEM ⇸ ℕ
| ...
|___________________________________________
|
|___________________________________________
```

(a) Item *i* has its price altered to *p*.

shopPrice' = shopPrice $\oplus$ {i $\mapsto$ p}

(b) A number of items are to be given new prices and the information is given in:
priceUpdate : ITEM $\twoheadrightarrow$ $\mathbb{N}$

shopPrice' = shopPrice $\oplus$ priceUpdate

(c) Items in the set *item:* $\mathbb{P}$ *ITEM* are no longer to be stocked by the shop and their price information is to be removed from the price records.

shopPrice' = shopPrice $\lhd$ shopPrice

(d) Item *i* has its price increased by RM 2.

shopPrice' = shopPrice $\oplus$ {i $\mapsto$ shopPrice(i) + 2 }

2) Consider a scenario concerning an air traffic control of an airport.

The air traffic control keeps a record of the planes *waiting* to land and the *assignment* of planes to *gates* on the ground. There are operations to accept a plane when it *arrives* at the airport's waiting space, to *assign* a plane to a gate at the airport and to record which plane *leaves* its gate. There is a *limit* to the number of planes that can be waiting.

Given the set:

[PLANE] - the set of all possible, uniquely identified planes
[GATE] - the set of all gates at the airport

There is a limit to the number of planes that can be waiting:

| limit : ℕ

The state of the *Airport*, at any time, can be expressed in the Z state space below:

```
_Airport_______________________________________
| waiting : ℙ PLANE
| assignment : GATE ⤀ PLANE
|__________________
| #waiting ≤ limit
| waiting ∩ ran assignment = ∅
|_______________________________________________
```

(a) Design the initial state called *InitAirport* where there are no planes waiting or at any gate.

```
__InitAirport__________________________________
| Airport
|______________________________________________
| waiting = ∅
| assignment = ∅
|______________________________________________
```

(b) Design the operation schema called *Arrive* that records the arrival of a plane *p?* at the airport's waiting area. The waiting area must not be full and the plane must be neither already waiting nor assigned to a gate.

```
__Arrive_______________________________________
| Δ Airport
| p? : PLANE
|______________________________________________
| p? ∉ waiting
| p? ∉ ran assignment
| #waiting < limit
|
| waiting' = waiting ∪ {p?}
| assignment' = assignment
|______________________________________________
```

(c) Design the operation schema called *Assign* that records the assignment of a plane *p?* to a free gate *g?*. The plane must be waiting and the gate must be free.

Assign

Δ *Airport*
p? : PLANE
g? : GATE

---

p? $\in$ waiting
p? $\notin$ ran assignment ->not important
g? $\notin$ dom assignment

waiting' = waiting \ {p?}
assignment' = assignment $\cup$ {g? $\mapsto$ p?}

OR

Assign

Δ *Airport*
p? : PLANE
g? : GATE

---

p? $\in$ waiting
p? $\notin$ ran assignment ->not important
assignment (g?) = $\varnothing$

waiting' = waiting \ {p?}
assignment' = assignment $\oplus$ {g? $\mapsto$ p?}

(d) Design the operation schema called *Leave* that records the plane *p?* leaving its gate. The waiting planes are unaffected.

Leave

Δ *Airport*
p? : PLANE

---

p? $\notin$ waiting ->not important
p? $\in$ ran assignment
assignment (g?) = $\varnothing$

waiting' = waiting
assignment' = assignment $\rhd$ {p?}

OR

Leave

Δ Airport
p? : PLANE

p? ∉ waiting ->not important
p? ∈ ran assignment

waiting’ = waiting
assignment’ = assignment ⊕ {dom(assignment ▷ {p?} ↦ ∅}

OR

assignment’ = assignment ⊕ {dom(assignment~ ({p?}) ↦ ∅}