EAST WEST UNIVERSITY
Department of Computer Science & Engineering

LTSA Lab Assignment

Name **: Abrar Khatib Lajim**

#ID **: 2022-3-60-043**

Course **: CSE430**

Section **: 01**

Instructor **: Dr. Shamim H. Ripon**

Submission **: 23/10/2025**

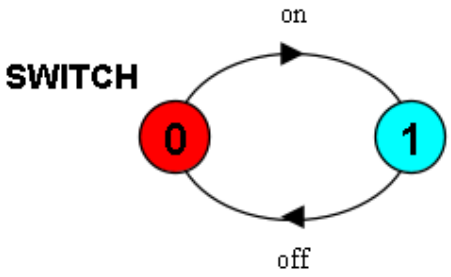
---

1. Toggle Switch (SWITCH)

Model a switch that toggles between `ON` and `OFF`.

```
SWITCH = OFF,
OFF = (on -> ON),
ON  = (off -> OFF).
```

Diagram: Two states connected by `on` and `off` transitions.



Analysis: Verify it alternates correctly and there's no deadlock.

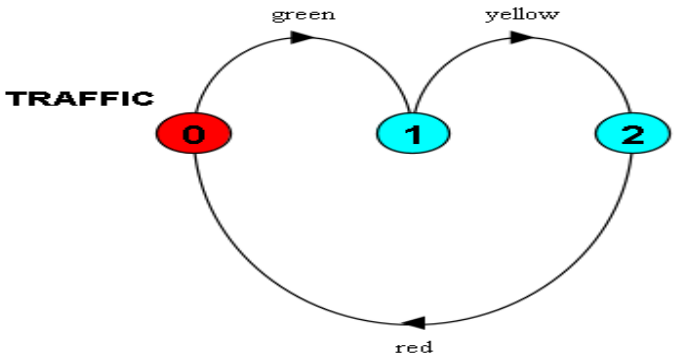
```
Composition:
DEFAULT = SWITCH
State Space:
 2 = 2 ** 1
Progress Check...
-- States: 2 Transitions: 2 Memory used: 70072K
No progress violations detected.
```

2. Traffic Light Controller

Cycle through Red → Green → Yellow → Red.

```
TRAFFIC = RED,
RED     = (green -> GREEN),
GREEN   = (yellow -> YELLOW),
YELLOW  = (red -> RED).
```

Diagram: 3 states in a cycle.



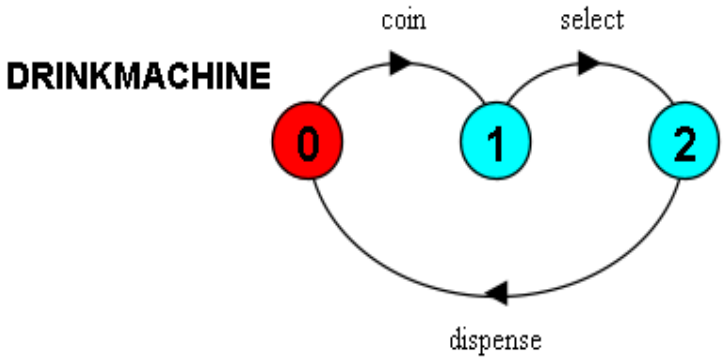
Observation: It must never skip a color or stop cycling.

3. Drinks Machine (with Cancel Option)

Add a cancel button to an existing drink machine model.

Base model (simplified):

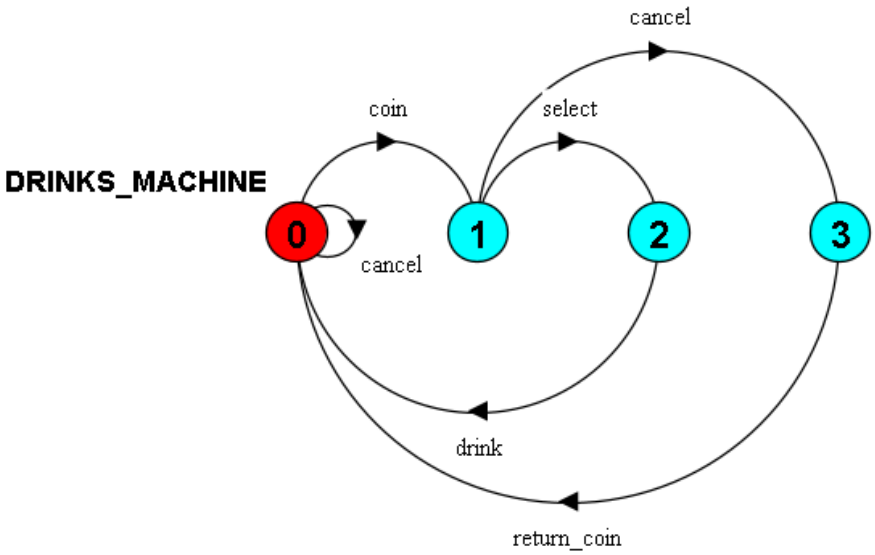
```
DRINKMACHINE = (coin -> select -> dispense -> DRINKMACHINE).
```



Extended version:

```
DRINKS_MACHINE = IDLE, IDLE = (coin -> READY | cancel ->
IDLE), READY = (select -> DISPENSE | cancel -> REFUND),
DISPENSE = (drink -> IDLE), REFUND = (return_coin -> IDLE).
```

Diagram: Add a `cancel/refund` branch before `dispense`.



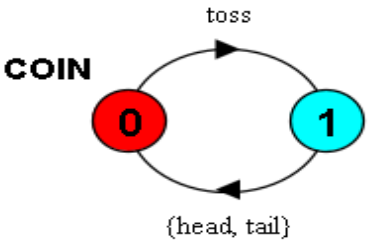
Observation: Ensure `cancel` skips `dispense`.

4. Coin Toss Machine (Non-deterministic choice)

Simulate a coin toss producing HEAD or TAIL non-deterministically.

```
COIN = (toss -> (head -> COIN | tail -> COIN)).
```

Diagram: From initial state, toss can lead to two outcomes.



Observation: Each toss should be independent and unpredictable.

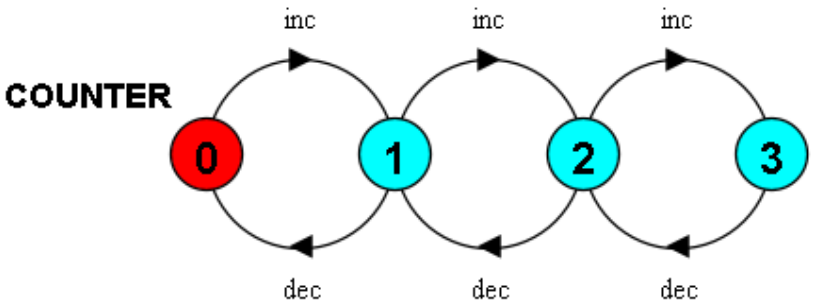
5. Counter (with Guards)

Restrict increment/decrement operations to avoid overflow/underflow.

Example: Counter ranges 0–3

```
const MIN = 0 const MAX = 3 COUNTER = COUNT[0],
COUNT[i:MIN..MAX] = (when (i < MAX) inc -> COUNT[i+1]
|when (i > MIN) dec -> COUNT[i-1]).
```

Diagram: Four states (0–3) with transitions for `inc` and `dec`.



Observation: No transition occurs beyond range (guarded).

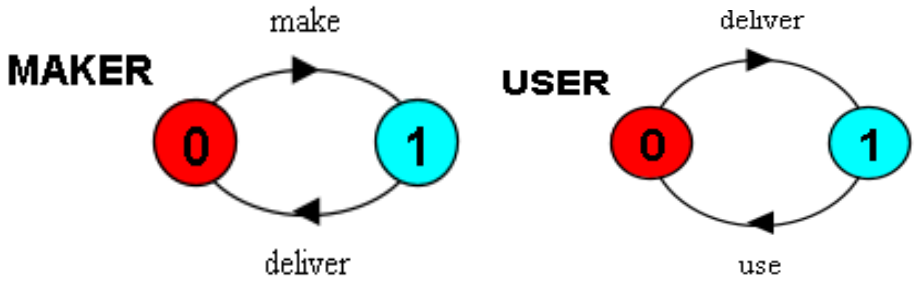
6. Maker–User Synchronization

Shared actions ensure coordination.

```
MAKER = (make -> deliver -> MAKER).
USER  = (deliver -> use -> USER).

||SYSTEM = (MAKER || USER).
```

Diagram: Shared action `deliver` synchronizes both.



Observation: `USER` can't use until `MAKER` delivers.

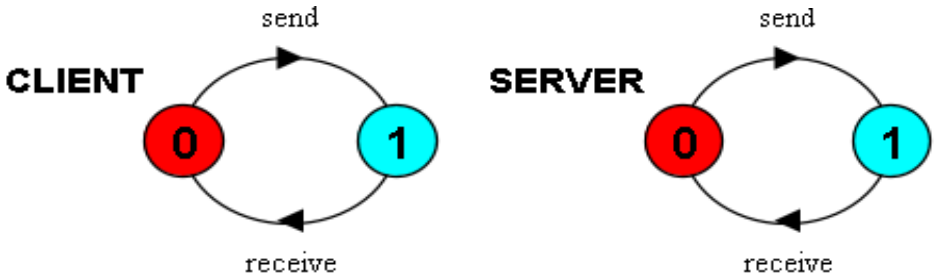
7. Client–Server Model with Relabelling

Use relabelling to synchronize `request` and `reply`.

```
CLIENT = (send -> receive -> CLIENT).
SERVER = (request -> reply -> SERVER).

||SYSTEM = (CLIENT || SERVER)
          / { send/request, receive/reply }.
```

Diagram: Synchronized `send/request` and `receive/reply`.



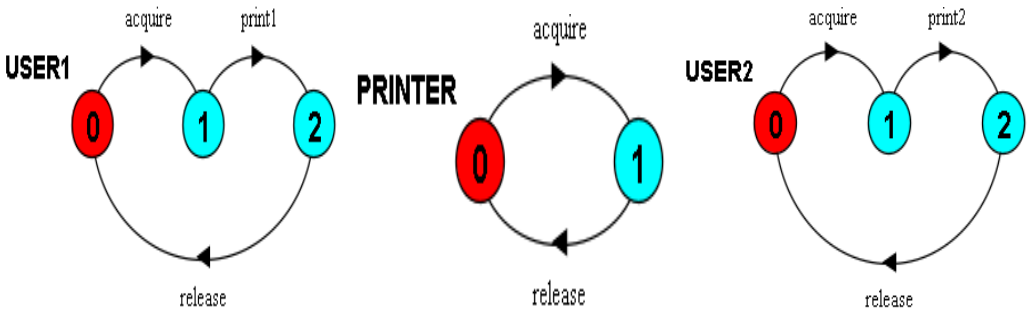
Observation: Each client request matches one server reply.

8. Shared Printer

Ensure only one user prints at a time.

```
USER1 = (acquire -> print1 -> release -> USER1). USER2
= (acquire -> print2 -> release -> USER2). PRINTER =
(acquire -> release -> PRINTER).
```

Diagram: Only one `use` at a time.



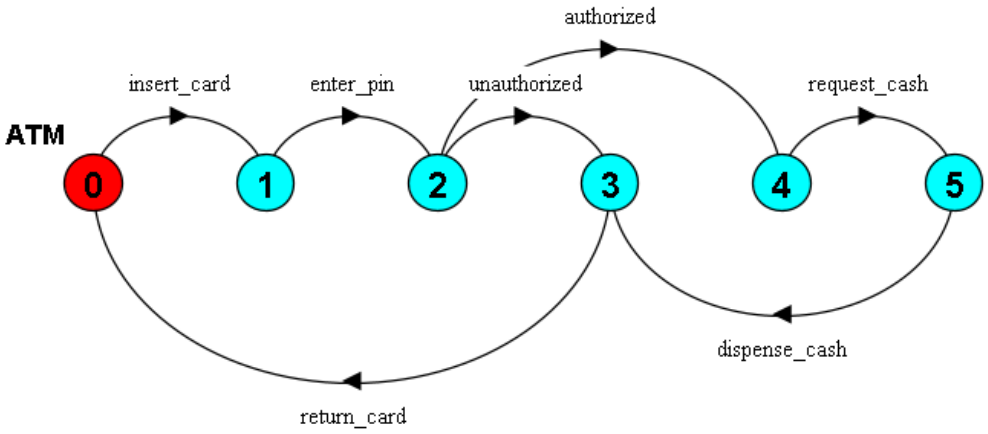
Analysis: Verify mutual exclusion (no two users using printer simultaneously).

9. ATM Safety Property

Verify **cash never dispensed without authorization**.

```
USER1 = (acquire -> print1 -> release -> USER1). USER2
= (acquire -> print2 -> release -> USER2). PRINTER =
(acquire -> release -> PRINTER).
```

Check:
LTS → **Analyze → Safety → Check property**
Result: Must confirm "No cash before authorization."



10. Progress Property (Card Always Returned)

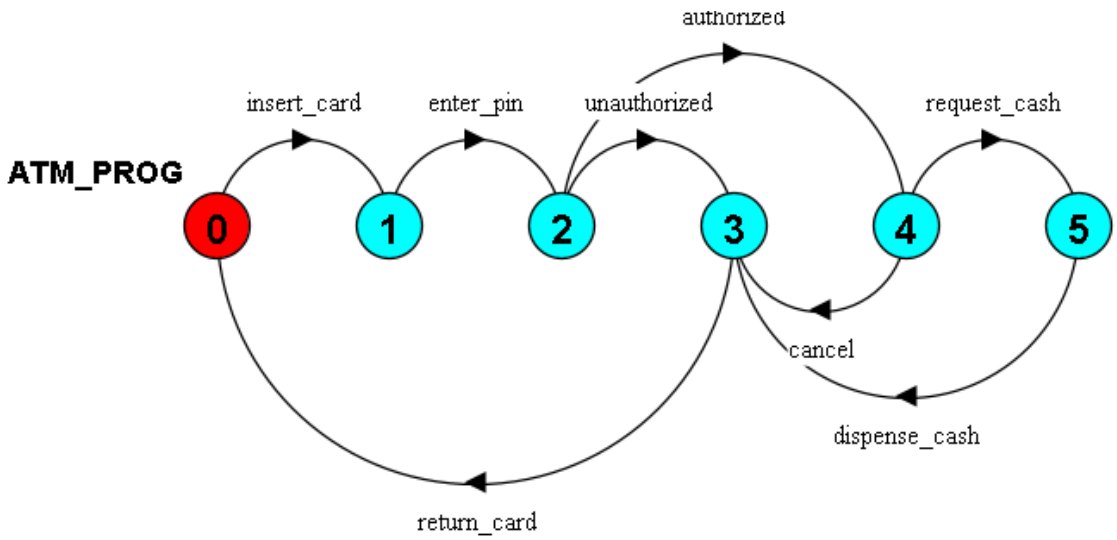
Ensure **card is always returned** after transaction.

Progress Property:

```
progress CARD_RETURNED = { ejectCard }.
```

```
ATM_PROG = IDLE, IDLE = (insert_card -> AUTH), AUTH =
(enter_pin -> VERIFY), VERIFY = (authorized ->
TRANSACTION | unauthorized -> RETURN_CARD), TRANSACTION
= (request_cash -> DISPENSE | cancel -> RETURN_CARD),
DISPENSE = (dispense_cash -> RETURN_CARD), RETURN_CARD
= (return_card -> IDLE).
```

Diagram: Every path from insert_card leads to return_card. No infinite loops before card return. LTSA will verify no progress violation exists.



Analysis:

Progress property ensures return_card action eventually occurs.

Every path from insert_card leads to return_card.

No infinite loops before card return.

LTSA will verify no progress violation exists.

Source Code: [Software-Quality-Assurance/LTSA.fsp at main · AbrarBb/Software-Quality-Assurance]