East West University

Lab Report 01

Topic: LTSA - Labelled Transition System Analyser

Course Title: Software Testing and Quality Assurance

Course Code: CSE 430

Section No: 01

Submitted By

Md. Saiful Islam

2022-3-60-045

Submitted To

Dr. Shamim H Ripon

Professor

Department of Computer Science and Engineering

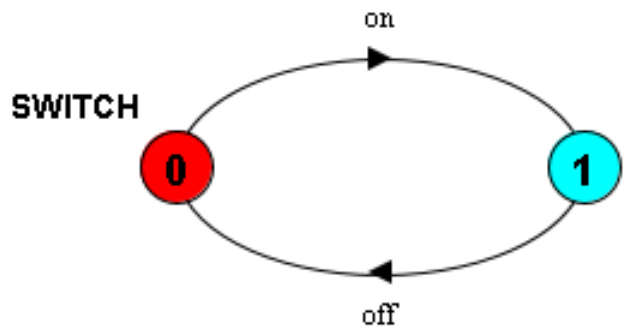
East West University

Lab Task 1

Model a toggle switch (SWITCH) that alternates between on and off states.

```
SWITCH = ( on -> off -> SWITCH ).
```

Output:



Analysis & Observation: The SWITCH model shows a repeating ON/OFF behavior. It starts with the on action, then performs off, and returns to the start, forming a continuous cycle.

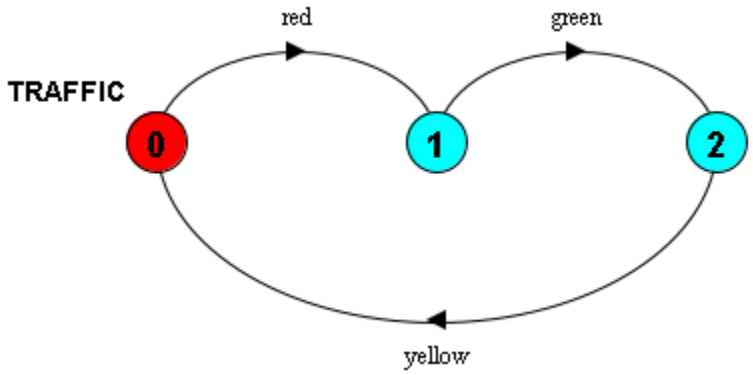
- Two states connected in a loop.
- on moves to the ON state; off returns to the start.

Lab Task 2

Develop a simple Traffic Light Controller with states Red → Green → Yellow → Red.

```
TRAFFIC = ( red -> green -> yellow -> TRAFFIC ).
```

Output:



Analysis & Observation: The TRAFFIC model cycles through three actions — red, green, and yellow — before repeating. Each action represents a traffic light change, forming a continuous loop with three states.

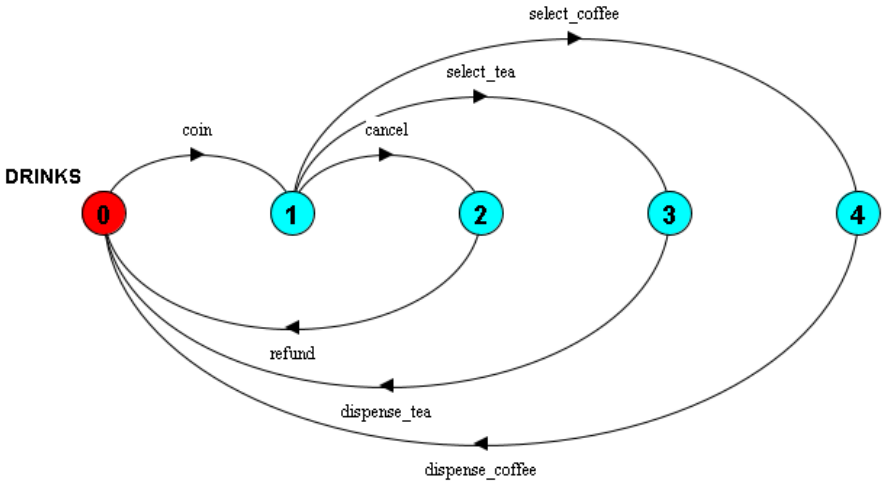
- Three states connected in a circular sequence: red → green → yellow → red.
- The process repeats endlessly with no branches or alternatives.

Lab Task 3

Extend the Drinks Machine model with a cancel option.

```
DRINKS = ( coin ->
         ( select_coffee -> dispense_coffee -> DRINKS
         | select_tea  -> dispense_tea  -> DRINKS
         | cancel  -> refund  -> DRINKS  )
          ).
```

Output:



Analysis & Observation: The DRINKS model describes a vending machine process. After inserting a coin, the user can choose between select_coffee, select_tea, or cancel. Each choice leads to a different outcome: dispensing a drink or refunding the coin. After each sequence, the machine resets and waits for the next customer.

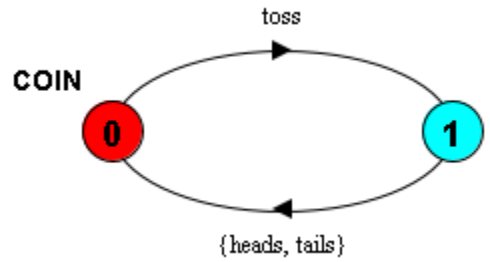
- About five states with multiple branching paths.
- After coin, the user can pick coffee, tea, or cancel.
- Each drink option leads to dispensing; cancel leads to a refund.

Lab Task 4

Simulate a Coin Toss Machine using non-deterministic choice.

```
COIN = ( toss ->
      ( heads -> COIN
      | tails -> COIN )
       ).
```

Output:



Analysis & Observation: The COIN model represents a coin toss with two possible outcomes. After the toss action, the process can non-deterministically go to either heads or tails, then return to the start for another toss. This cycle continues indefinitely.

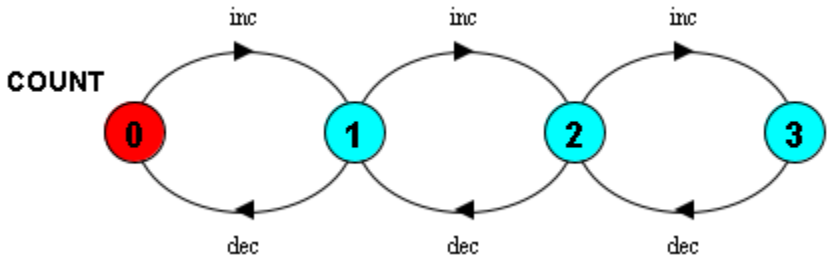
- Two states with branching after the toss action.
- toss can lead to either heads or tails, showing non-deterministic behavior.

Lab Task 5

Create a Counter model with guard conditions to restrict underflow and overflow.

```
COUNT(N=3) = COUNT[0],

COUNT[i:0..N] = ( when(i<N)  inc -> COUNT[i+1]
                | when(i>0)  dec -> COUNT[i-1] ).
```

Output:



Analysis & Observation: The COUNT model defines a counter that moves between numbered states from COUNT[0] to COUNT[3]. The inc action increases the count (if it's below the maximum), and the dec action decreases it (if it's above zero). The guards (when(i<N) and when(i>0)) control which actions are allowed at each state.

- Four states: COUNT[0], COUNT[1], COUNT[2], and COUNT[3].
- At COUNT[0], only inc is possible; at COUNT[3], only dec is possible.
- Middle states allow both inc and dec.

Lab Task 6

Build a Maker–User synchronization system using shared actions.

```
MAKER = ( make -> ready -> MAKER ).
USER  = ( ready -> use -> USER ).
||SYSTEM = ( MAKER || USER ).
```

Output:

Analysis & Observation: The MAKER model describes a simple production cycle. It performs the make action, then signals ready, and returns to the start to repeat the process. This shows a continuous workflow where something is made and then prepared for use.

- Two actions: make followed by ready, looping back to the start.
- Models a process that repeatedly produces and prepares items.

Lab Task 7

Implement a Client–Server model using relabelling to synchronize request and reply actions.

```
CLIENT = ( call -> wait -> CLIENT ).
SERVER = ( request -> service -> reply -> SERVER ).
||CS = ( CLIENT || SERVER ) / { call/request, reply/wait }.
```

Output:

Analysis & Observation: The composed system CS connects the CLIENT and SERVER processes through relabelling, allowing them to interact. The call action from the client synchronizes with the server's request, and the server's reply synchronizes with the client's wait. Other actions, like service, occur independently.

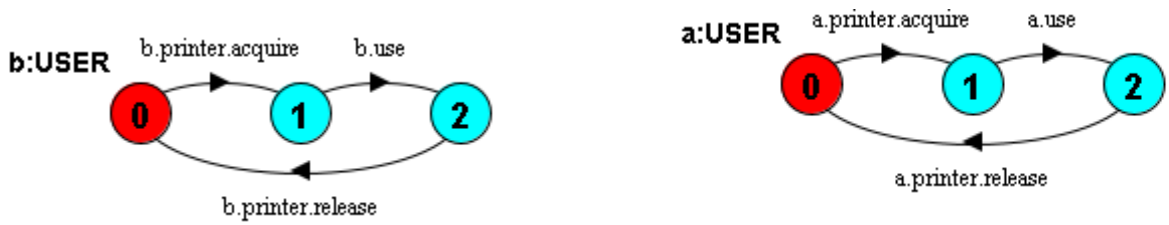
- Two synchronization points: call/request and reply/wait.
- service happens independently between these synchronized steps.

Lab Task 8

Construct a Shared Printer model ensuring only one user can use the printer at a time.

```
RESOURCE = ( acquire -> release -> RESOURCE ).
USER = ( printer.acquire -> use -> printer.release -> USER ).
||SHARED = ( a:USER || b:USER || {a,b}::printer:RESOURCE ).
```

Output:



Analysis & Observation: The RESOURCE model represents a shared item that can be acquired and released. Two USER processes interact with this shared printer resource. Each user must acquire the printer before using it and release it afterward. The composition enforces that only one user can hold the printer at a time.

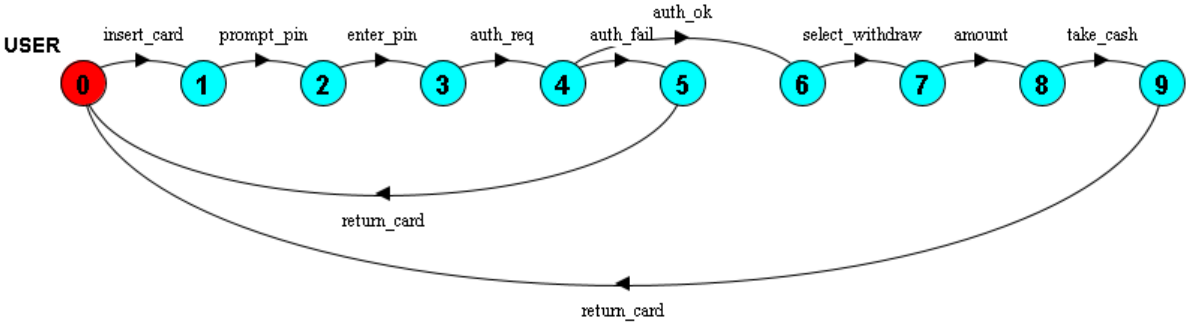
- Six total states showing different combinations of user and printer statuses.
- Only one user can acquire the printer at once — the other must wait.
- Models mutual exclusion, ensuring safe sharing of a single resource.

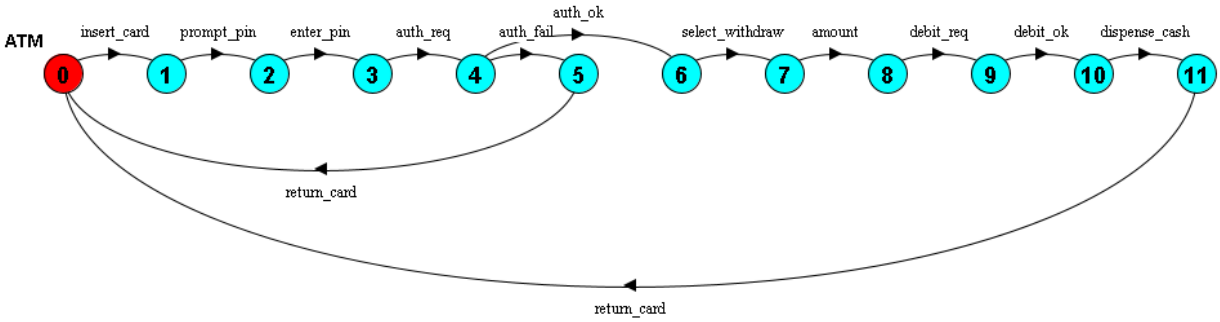
Lab Task 9

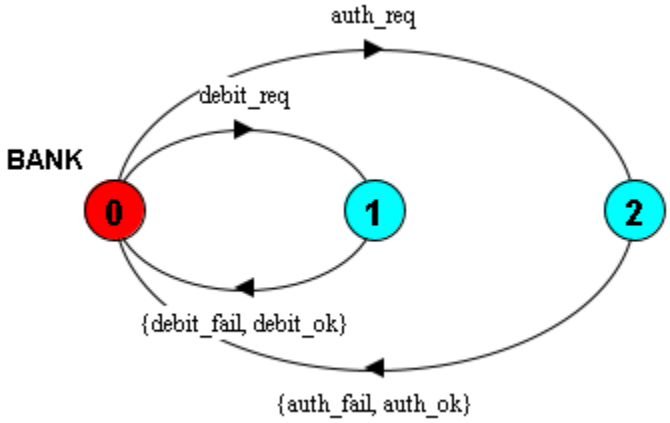
Analyze the ATM system model for safety: verify that cash is never dispensed without successful authorization.

```
USER = (
    insert_card -> prompt_pin -> enter_pin -> auth_req ->
           ( auth_ok -> ( cancel -> USER
           | select_withdraw -> amount -> take_cash -> return_card ->
USER )
           | auth_fail ->
            return_card -> USER ) ).

ATM = (
    insert_card ->prompt_pin ->enter_pin ->auth_req ->(auth_ok ->(
cancel -> ATM
            | select_withdraw -> amount -> debit_req ->  ( debit_ok
-> dispense_cash -> return_card -> ATM
                | debit_fail -> show_error -> return_card -> ATM ) )
           | auth_fail ->return_card -> ATM ) ).


BANK = ( auth_req -> ( auth_ok -> BANK | auth_fail -> BANK )
           | debit_req -> ( debit_ok -> BANK | debit_fail -> BANK )
).
```

Output:







Analysis & Observation: The ATM_SYSTEM models an ATM transaction involving three components: USER, ATM, and BANK. The process includes inserting a card, entering a PIN, authentication, and optionally withdrawing cash. The system handles both success and failure paths for authentication and debit requests, with synchronization on key actions like auth_req, debit_req, and return_card.

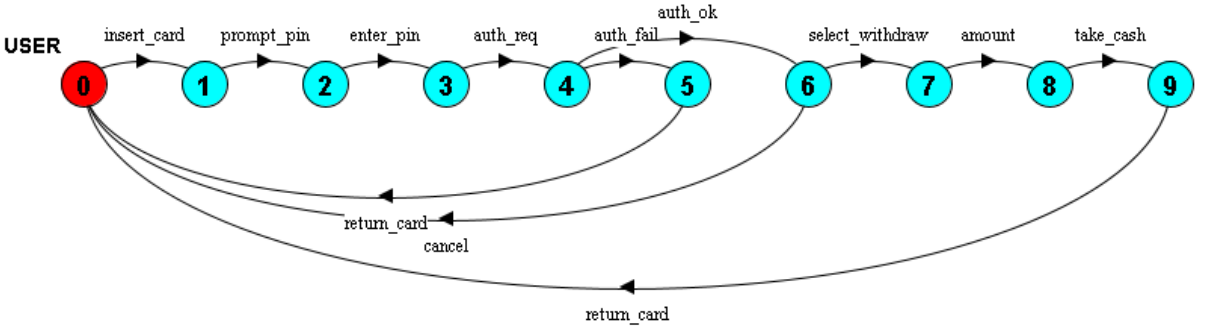
- Around 15–20 states with branching for different outcomes (success or failure).
- Key synchronization points: authentication requests, debit requests, and card returns.
- Safety checks ensure cash is only dispensed after successful authentication and debit approval.

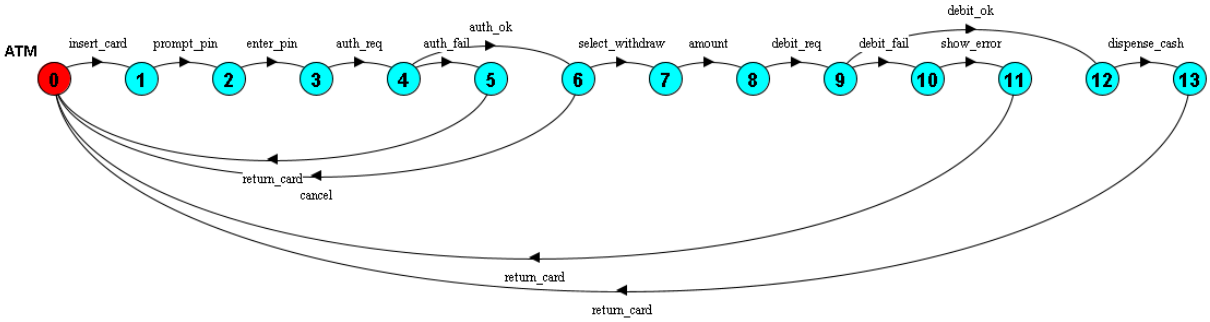
Lab Task 10

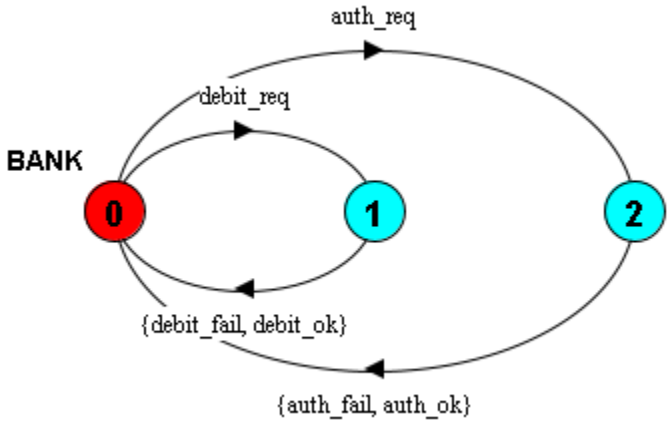
Add a progress property to confirm that the card is always returned after any transaction.

```
USER = (
    insert_card -> prompt_pin -> enter_pin -> auth_req ->
           ( auth_ok -> ( cancel -> USER
           | select_withdraw -> amount -> take_cash -> return_card ->
USER )
           | auth_fail ->
            return_card -> USER ) ).

ATM = (
    insert_card ->prompt_pin ->enter_pin ->auth_req ->(auth_ok ->(
cancel -> ATM
            | select_withdraw -> amount -> debit_req ->  ( debit_ok
-> dispense_cash -> return_card -> ATM
                | debit_fail -> show_error -> return_card -> ATM ) )
           | auth_fail ->return_card -> ATM ) ).


BANK = ( auth_req -> ( auth_ok -> BANK | auth_fail -> BANK )
           | debit_req -> ( debit_ok -> BANK | debit_fail -> BANK )
).


||ATM_SYSTEM = ( USER || ATM || BANK ).


progress RETURN_CARD = {return_card}

||ATM_WITH_PROGRESS = (ATM_SYSTEM || SAFE_WITHDRAWAL) >>
{return_card}.
```

Output:







Analysis & Observation: The progress property CARD_RETURNED checks that the action card_returned occurs on every possible execution path in the ATM_SYSTEM. The composed system ATM_WITH_PROGRESS verifies that the system does not get stuck or loop indefinitely without returning the card.

- LTSA analyzes all outcomes—successful transactions, authentication failures, debit failures.
- Confirms that card_returned always happens eventually on every path.
- Ensures no deadlocks or infinite loops prevent card return.