Report for Lab Tasks- LTSA:

Course Code: CSE430

Course Title: Software Testing & Quality Assurance

Section: 01

Submitted To:

Dr. Shamim H Ripon
Professor
Department of Computer Science & Engineering

Submitted By:

Umme Mukaddisa

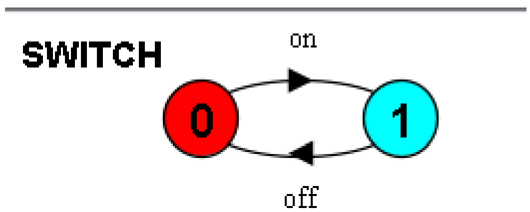
ID: 2022-3-60-317

Problem-01 (A toggle switch that alternates between on & off)**:**

Code:

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

Output:



Explanation:

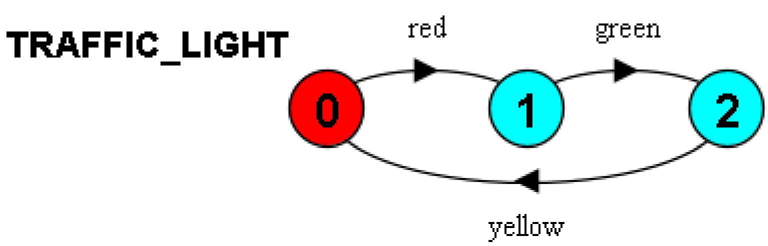
There are two states: **0** (OFF) & **1** (ON). When we press the switch in state **0**, it turns ON & changes from **0** → **1**. When we press it again in state **1**, it turns OFF & changes from **1** → **0**.

Problem-02 (Simple Traffic Light Controller)**:**

Code:

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

Output:



Explanation:

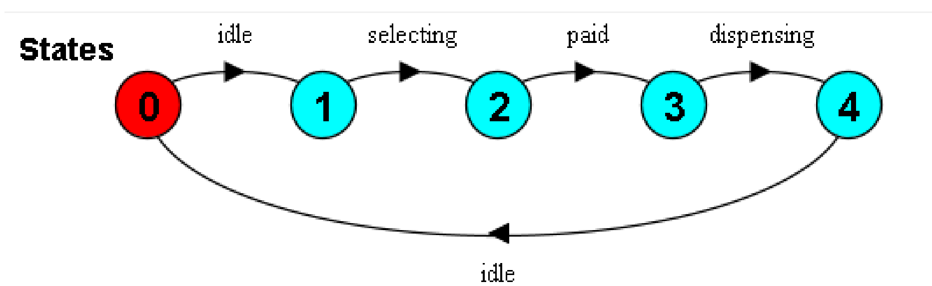
The lights change in a fixed cycle — when it's Red, all cars stop & state is 1 then it goes to Green, all cars move & state is 2. Then from Green to Yellow, cars moving slowly & state is 0. Then from Yellow back to Red — repeating continuously.

Problem-03 (Drinks Machine)**:**

Code:

```
States = (idle -> selecting -> paid -> dispensing -> idle ->States).
```

Output:



Explanation:

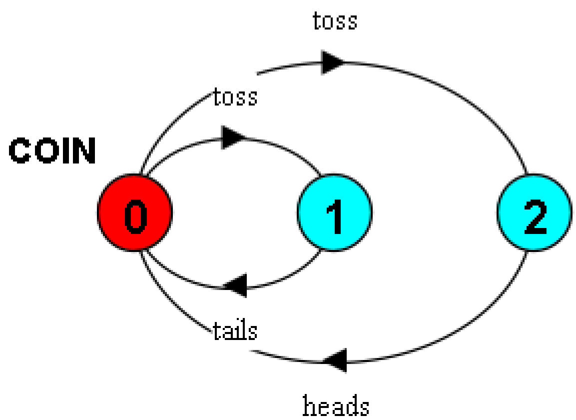
There are two main actions: select drink & cancel. When a drink is selected, the machine waits for payment. If cancel is pressed before payment or dispensing, it returns to the idle state & refunds any inserted money. Otherwise, it continues to dispense the selected drink.

Problem-04 (Coin Toss Machine using non-deterministic choice):

Code:

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

Output:



Explanation:

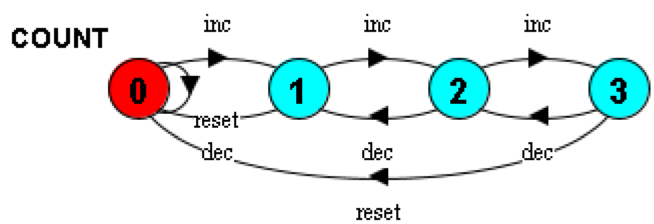
There are three states: COIN, HEADS & TAILS. In the COIN state, a toss can lead to either HEADS or, TAILS — this represents a non-deterministic choice (an unpredictable outcome of an action). After showing the result, the machine returns to the COIN state, ready for the next toss.

Problem-05 (A counter model with guard conditions to restrict underflow & overflow) **:**

Code:

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

Output:



Explanation:

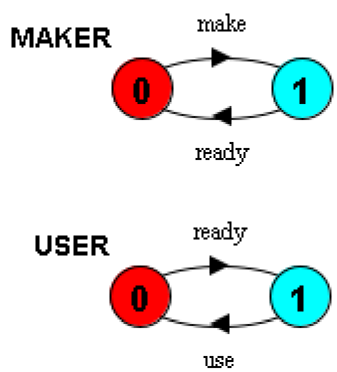
The counter has four states (0 to 3) and only allows increments if below max or decrements if above zero, preventing invalid counts. It shows a safe bounded counter with controlled transitions.

Problem-06 (A maker-User synchronization system using shared actions)**:**

Code:

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

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

Output:



Explanation:

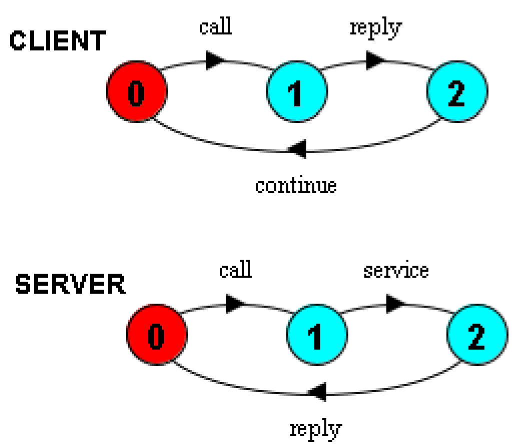
The maker repeatedly performs a make action followed by signaling ready. This syncs with a user who waits for ready before proceeding, modeling coordinated production and usage.

Problem-07 (A client–Server model using relabelling to synchronize request and reply actions)**:**

Code:

```
CLIENT = (call -> wait -> continue -> CLIENT).
SERVER = (request -> service -> reply -> SERVER).

||CLIENT_SERVER = (CLIENT || SERVER) / {
    call/request,
    reply/wait
}.
```

Output:



Explanation:

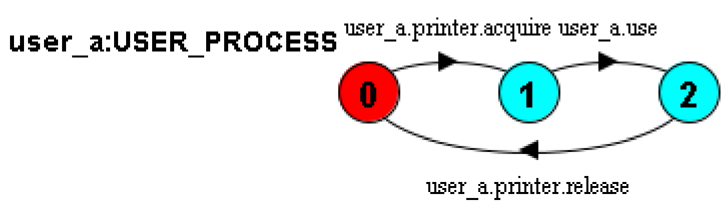
The client and server communicate via relabelling, synchronizing call with request and reply with wait. The server independently handles service between these synchronized steps, modeling client-server interaction.

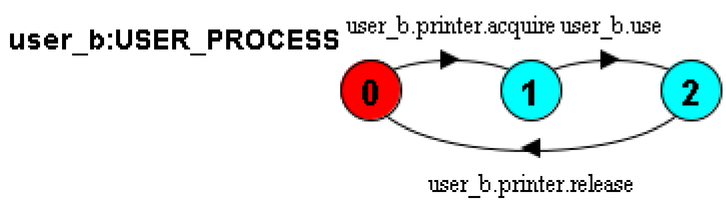
Problem-08 (A shared Printer model ensuring only one user can use the printer at a time)**:**

Code:

```
PRINTER = (acquire -> release -> PRINTER).
USER_PROCESS = (printer.acquire -> use -> printer.release ->
USER_PROCESS).

||SHARED_PRINTER = (
    user_a:USER_PROCESS ||
    user_b:USER_PROCESS ||
    {user_a, user_b}::printer:PRINTER
).
```

Output:





Explanation:

Two users share access to a printer resource with mutual exclusion. Only one user can acquire and use the printer at a time, ensuring no conflicts during printing.

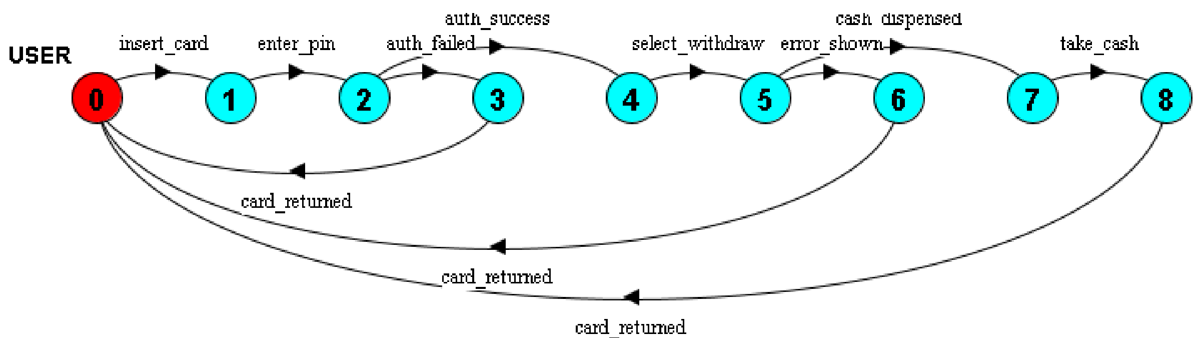
Problem-09 (ATM system model for safety: verify that cash is never dispensed without successful authorization)**:**

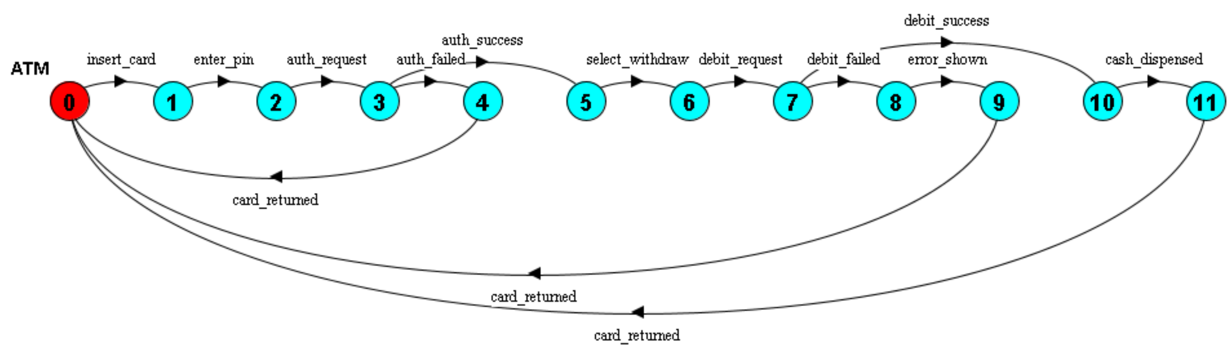
Code:

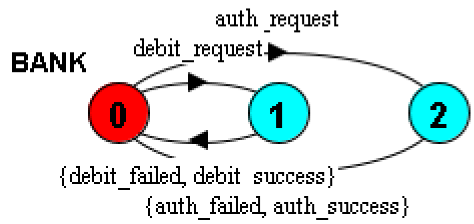
```
USER = (
    insert_card -> enter_pin ->
    (auth_success -> select_withdraw ->
        (cash_dispensed -> take_cash -> card_returned -> USER
        | error_shown -> card_returned -> USER)
    | auth_failed -> card_returned -> USER)
).

ATM = (
    insert_card -> enter_pin -> auth_request ->
    (auth_success -> select_withdraw -> debit_request ->
        (debit_success -> cash_dispensed -> card_returned -> ATM
        | debit_failed -> error_shown -> card_returned -> ATM)
    | auth_failed -> card_returned -> ATM)
).

BANK = (
    auth_request -> (auth_success -> BANK | auth_failed -> BANK)
  | debit_request -> (debit_success -> BANK | debit_failed -> BANK)
).

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

// Safety Property: Cash never dispensed without authorization
property SAFE_WITHDRAWAL = (
```

```
    auth_success -> AUTHORIZED
),
AUTHORIZED = (
    cash_dispensed -> SAFE_WITHDRAWAL
  | auth_success -> AUTHORIZED
  | card_returned -> SAFE_WITHDRAWAL
).

||ATM_WITH_SAFETY = (ATM_SYSTEM || SAFE_WITHDRAWAL).
```

Output:







Explanation:

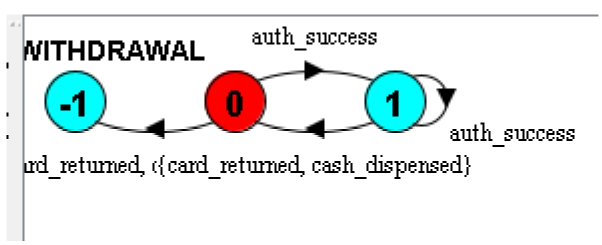
The ATM system models inserting a card, PIN authentication, selecting withdrawal, and bank approval. It includes success and failure paths with synchronization points, ensuring secure and orderly transactions.

Problem-10 (A progress property to confirm that the card is always returned after any transaction)**:**

Code:

```
progress CARD_RETURNED = {card_returned}

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

Output:



Explanation:

This system guarantees that the card_returned action occurs on every path, no matter if authentication or debit fails. It proves the ATM never keeps a customer's card, ensuring proper progress and customer safety.