1. Stable and Unstable States

Definition:

- **Stable State**: The system remains in the current state until an external input changes.
- **Unstable State**: The system transitions to another state due to the current state not being stable for the given inputs.

Example:

Consider an asynchronous machine controlling a traffic light at a pedestrian crossing.

- **Stable State**: When there is no pedestrian, the light remains **green** for cars.
- **Unstable State**: When a pedestrian presses the button, the system transitions to a state where the light changes to **red**.

Flow Table Representation:

Present State	Input = 0 (No Pedestrian)	Input = 1 (Pedestrian Button Pressed)
S1 (Green)	S1 (Stable)	S2 (Unstable, transitioning to red)
S2 (Red)	S3 (Yellow, unstable)	S2 (Stable)
S3 (Yellow)	S1 (Green, unstable)	S3 (Stable)

- **Key Insight**: Stability ensures the system remains predictable.

Applications:

- Elevator controls.
- Microwave ovens waiting for input.

2. Flow Table and Races

A **flow table** is a compact representation of state transitions.

Definition of Races:

- A **race** occurs when multiple state variables change simultaneously, potentially causing an unpredictable next state.

Example:

Consider an **automatic door sensor** that detects people entering or leaving.

Current State	Input = Entering (1)	Input = Leaving (0)
S1 (Closed)	S2 (Opening)	S1 (Closed)
S2 (Opening)	S3 (Open)	S1 (Closed)
S3 (Open)	S3 (Open)	S4 (Closing)
S4 (Closing)	S1 (Closed)	S1 (Closed)

Potential Race Condition:

If the door is transitioning between **S2 (Opening)** and **S3 (Open)**, a person leaving at the same time might force a transition to **S4 (Closing)** too early.

Solution:

- Insert **intermediate states** or use **race-free state assignment** to ensure transitions occur sequentially.

3. State Reduction

Goal:

Simplify the system by merging states with identical behaviors.

Example: Vending Machine

A vending machine dispenses a soda after a total of 10 cents. Assume it accepts only 5-cent coins (N) or 10-cent coins (D).

Present State	Input = N	Input = D	Output
S1 (0 cents)	S2 (5)	S3 (10)	0
S2 (5 cents)	S3 (10)	S3 (10)	0
S3 (10 cents)	S3 (10)	S3 (10)	1

Reduction:

- S3 produces the same output for any transition, so it can be merged with similar states beyond 10 cents.
- New Flow Table:

Present State	Input = N	Input = D	Output
S1 (0 cents)	S2 (5)	S3 (10)	0
S2 (5 cents)	S3 (10)	S3 (10)	0

Real-world Implication:

This reduction minimizes the number of states and logic gates needed to design the vending machine.

4. State Assignment

Definition:

Assign binary codes to states to enable implementation using logic circuits.

Objective:

- Minimize the number of bit changes during transitions to avoid hazards.
- Optimize the design to reduce circuit complexity.

Example: Elevator Controller

An elevator with three floors (**G, 1, 2**) can be represented as:

State	Description	Binary Code
S1	Ground Floor	00
S2	1st Floor	01
S3	2nd Floor	10

Transition Table:

Present State	Input = Up	Input = Down
S1 (00)	S2 (01)	S1 (00)
S2 (01)	S3 (10)	S1 (00)
S3 (10)	S3 (10)	S2 (01)

Optimized Binary Codes:

- **Gray Coding**: Assign binary codes to ensure only one bit changes during transitions.
    - S1 = 00
    - S2 = 01
    - S3 = 11

This reduces glitches and minimizes race conditions.

5. Avoiding Races and Hazards

Definition:

- **Critical Races**: Simultaneous changes in state variables leading to unintended states.
- **Hazards**: Transient fluctuations in outputs due to circuit delays.

Example: Traffic Light Controller

Consider transitions from **Green → Yellow → Red**. Without careful state assignment, simultaneous state changes might result in the lights turning **off** momentarily.

Solutions:

1. **Use intermediate states**:
    - Insert "transitional states" to ensure a smooth progression.
2. **Design hazard-free logic**:
    - Use redundant logic to prevent glitches during state changes.

Practical Applications

1. Elevator Controls:

- Manage states like "door opening," "door closing," and "moving" efficiently with reduced hazards.

2. Traffic Signals:

- Ensure smooth transitions between lights with stable states and reduced glitches.

3. Industrial Automation:

- Conveyor belt systems use ASMs to manage states like "load," "move," and "unload."

4. Consumer Electronics:

- Devices like washing machines use ASMs to sequence operations (wash, rinse, spin).

Conclusion

The **analysis of ASMs** involves understanding the system's behavior (stable/unstable states), optimizing it through **state reduction** and **state assignment**, and ensuring robust design by avoiding **races and hazards**. Each step enhances the machine's reliability and efficiency in real-world applications like traffic systems, vending machines, and consumer devices.