Static analysis: MECA

- Based on control flow analysis for checking system rules so as to ensure correctness.
  E.g. If you are at state X whether transition to state Y is allowed or not is checked by MECA approach
- Concentration is on handling many general bugs (rule violations) as compared to format string bug detection which limits its scope mainly to a specific kind of bug.

Examples for safe programming rules and corresponding FSA (Finite State Automaton):

Rule (1): Cannot call `longjmp()` as root

Rule (2): Do not call exec with root privileges

Rule (3): `chroot()` must always be followed by `chdir()`
The reading assignment paper deals with:

- Interrupt mechanism (enable/disable checks)
- Memory management & Optimizations (FLASH)
- Locking / Unlocking system
- Thread implementations
- Deadlock avoidance
- Reference counting updates
- Dependency analysis of function calls like Assertions

**Locking Mechanism FSA:**

![Locking Mechanism State Diagram]

Note: Wrapper functions can be added to ensure lock/unlock related race conditions.
**Model Checking**: [MOPS (MOdelchecking Programs for Security properties)]

It involves state machine implementations as models of error. If the model reaches / traversal reaches the accept state, then it implies there is *Error* in the system.

Let $P \rightarrow \text{Program} \\
M_p \rightarrow \text{Machine Program} \\
M_s \rightarrow \text{Machine Specification}$

Let $P$:

```c
Int main(...)
{
    Seteuid(0);
    dummy();
    If(....) { //two paths possible
        Seteuid(geteuid());
    }
    dummy();
    exec(...);
}
```

Converting program($P$) into corresponding model [Control Flow Automaton] 
(Based on Root / Not Root FSA)

![Diagram](image-url)
**M₅: (Root/ Not Root FSA)**

Let, \( L(M) \) → Language generated or accepted by M

**Goal:** \( L(M₅) \) Intersection \( L(P) \) = \( \emptyset \)

If we prove \( L(M₅) \) Intersection \( L(M₆) \) = \( \emptyset \), then it is as good as proving the above statement.

---

**Product of two Machines, M₅ and M₆:**

- main0, root
  - seteuid(0)
  - main1, root
    - dummy()
    - main2, root
      - \( \epsilon \)
      - main3, root
        - seteuid(geteuid())
        - main4, root
          - dummy()
          - main5, root
            - exec()
            - main6, error (Accept State)
        - main4, root
          - dummy()
          - main5, root
            - exec()
            - main6, not root

If accept state is reached → there is error/bug in the system.
**Metal Extensions:**

1) Test Conditions
2) Mention Data
3) Track Data State
4) Not sound (Disadvantage)

Based on (2) and (3): Parameterize for each lock($l_i$) $\rightarrow$ Separate state machine for each lock($l_i$)
Sample program and related state machine representation:

```c
// Initial state unlocked
:
lock(l);
if(....) {
    unlock(l);
}
If(islocked()) {
    unlock(l);
}
```

![State machine diagram]