

**CSE 509 : COMPUTER SYSTEMS SECURITY**  
**SPRING' 09 : LECTURE NOTES**  
**Date 02/27/2009**

- Chetan J. Bharadwaj

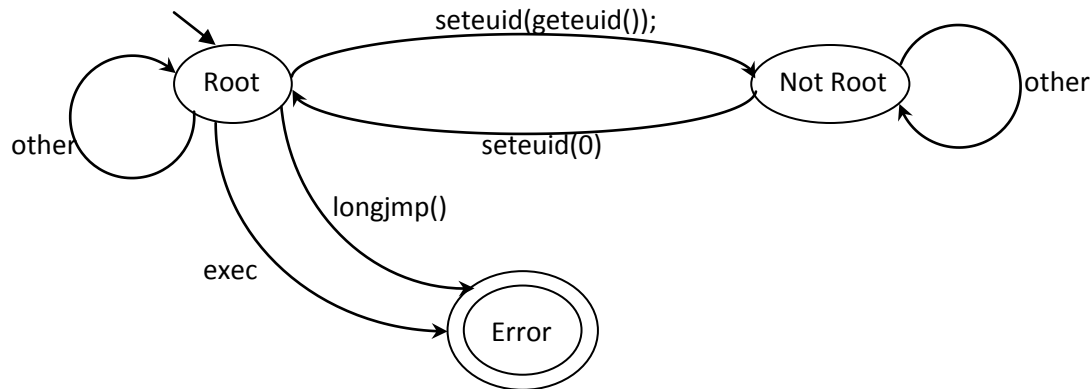
**Static analysis: MECA**

- 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):**

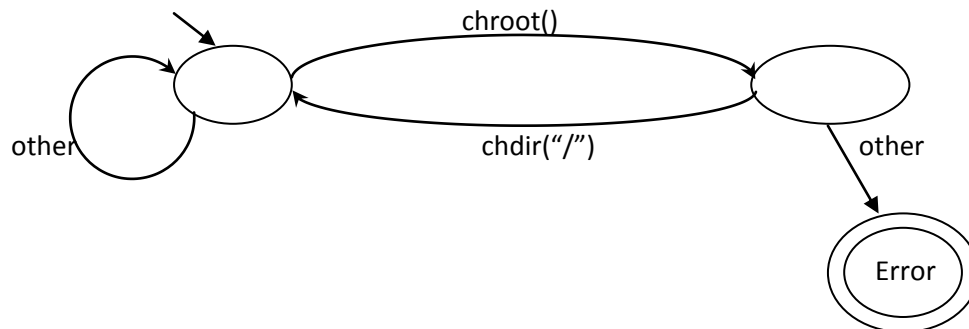
Rule (1): Cannot call *longjmp()* as root

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



**FSA for Rule(1) & (2)**

Rule (3): *chroot()* must always be followed by *chdir()*

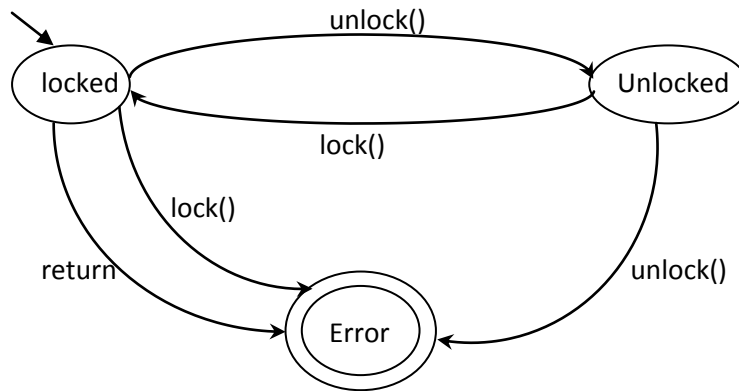


**FSA for Rule(3)**

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:**



Note: Wrapper functions can be added to ensure lock/unlock related race conditions.

**Model Checking:** [MOPS (*MOdelchecking Programs for Security properties*)]

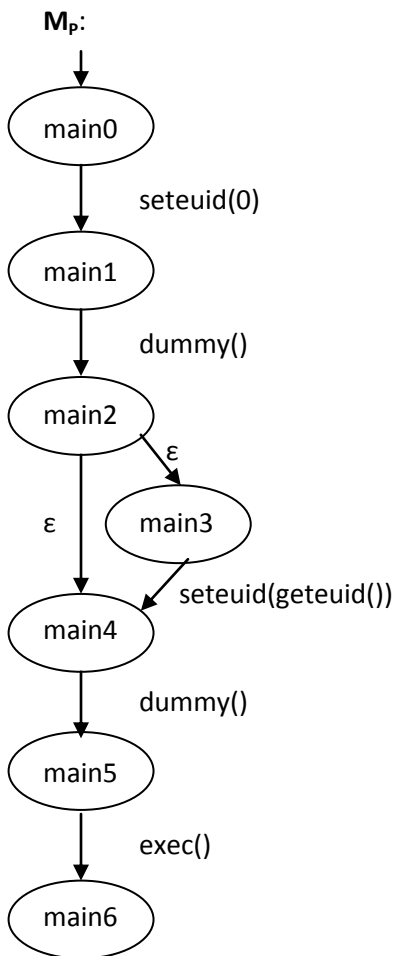
It involves state machine implementations as models of error. If the model reaches / traversal reaches the accept state, then it implies there is **Error** in the system.

Let  $P \rightarrow$  Program  
 $M_p \rightarrow$  Machine Program  
 $M_s \rightarrow$  Machine Specification

Let **P** :

```
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)

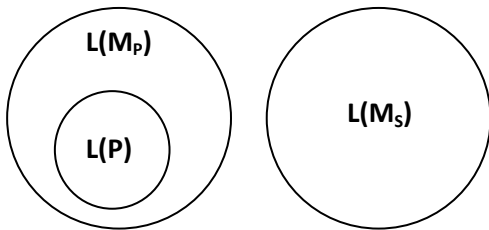


**M<sub>S</sub>: (Root/ Not Root FSA)**

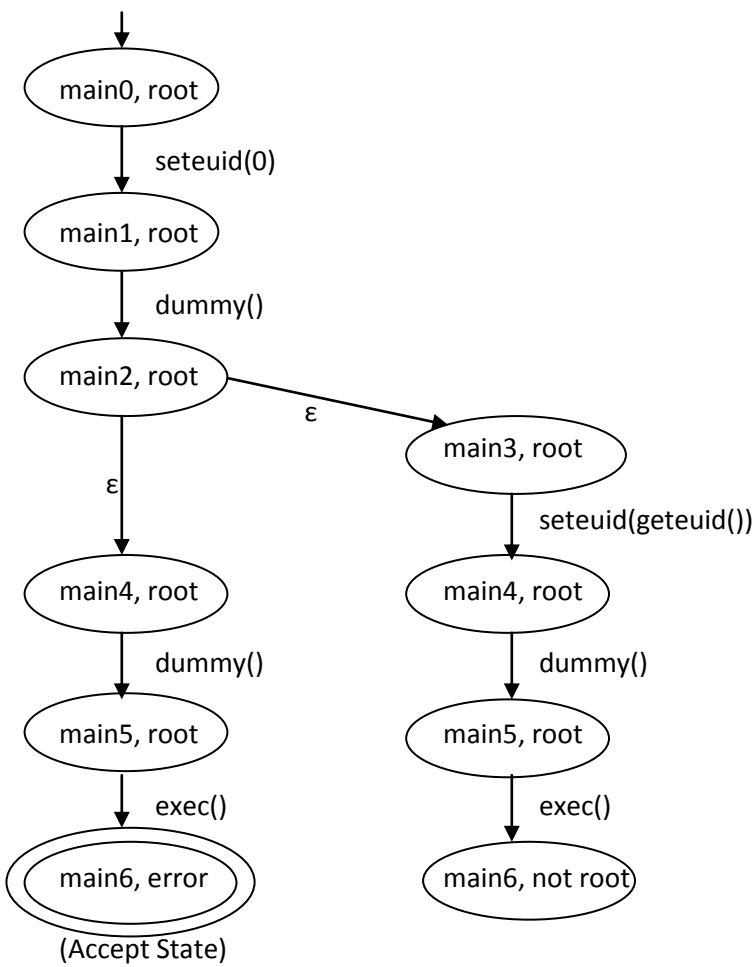
Let, L(M) → Language generated or accepted by M

**Goal:** L(M<sub>S</sub>) Intersection L(P) = ∅

If we prove L(M<sub>S</sub>) Intersection L(M<sub>P</sub>) = ∅, then it is as good as proving the above statement.



**Product of two Machines, M<sub>S</sub> and M<sub>P</sub>:**

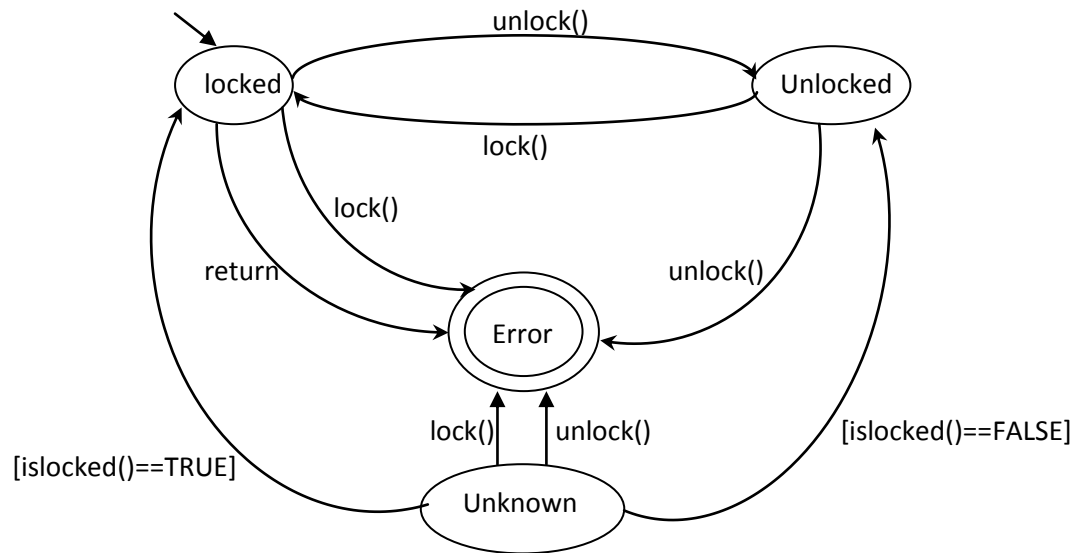


If accept state is reached → there is error/bug in the system.

**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;) → Separate state machine for each lock(l;)



Sample program and related state machine representation:

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

