Major shortcoming of Control Flow Automaton

```c
void setuid(int uid);
void log(void);
void system(void);

int main(void) {
    setuid(1000);
    log(...);
    system(...);
    setuid(0); // back to root
    log(...);
}

void log(void) {
    write(...);
}
```

In these codes, `system(...)` is never going to occur unless `uid` is set to ‘1000’.

However, in the model (Finite State Automata) for these codes ...

![Diagram showing the state transitions](image)

- `system(...)` is acceptable after `setuid(0)`!
- `IDS` only observes generated sequences of system calls.
• **Push Down Automaton (PDA) on input**
  1. Change state
  2. Check and pop top of stack
  3. Push new material onto stack

\[
\alpha / F / G
\]

\(\alpha\): input  
\(F\): top of stack (popped)  
\(G\): pushed onto stack

specify preconditions for taking the edge or changing states

• **Renewed Automata (PDA) for the codes :**

\[
\begin{align*}
\text{main0} & \quad \downarrow \text{setuid}(1000); / \varepsilon / \varepsilon \\
\text{main1} & \quad \varepsilon / \varepsilon / 2 \\
\text{main2} & \quad \downarrow \text{system}(\ldots); / \varepsilon / \varepsilon \\
\text{main3} & \quad \downarrow \text{setuid}(0); / \varepsilon / \varepsilon \\
\text{main4} & \\
\text{main5} & \quad (\text{the end of main function})
\end{align*}
\]

\[
\begin{align*}
\text{log0} & \\
\text{log1} & \quad \downarrow \text{write}(\ldots); / \varepsilon / \varepsilon \\
\end{align*}
\]

: Less ambiguity!  
Now, the loop isn’t going to happen.  
By PDA, if taking ‘\(\varepsilon / \varepsilon / 2\)’ edge so that the state is changed into ‘log 0’,  
‘\(\varepsilon / 5 / \varepsilon\)’ edge from state ‘log1’ to state ‘main5’ cannot be accepted.
- Problem with PDA
  - Sometimes we don’t know which way the program goes.

```java
foo(...)
{
    if(...)
        func1();
    else
        func2();
}
```

In this case, which edge should we accept?
**Correct:** change ‘G’ (pushed onto stack)
Ex) `foo0 → func2 : ε / ε / 2`

- **Another way:** copy the states and keep track of them respectively.
  However, there might be too many states to follow the flow!

- **In Dyck Model...**

  ![Diagram](image.png)

  : modifying the state machine

  Also, **modifying the corresponding code:**

  ```
  setuid(1000);
  **null(2);** // push 2 onto the stack
  log(...);
  **null(-2);** // check and pop 2 from the stack
  system(...);
  setuid(0);
  **null(5);** // push 5 onto the stack
  log(...);
  **null(-5);** // check and pop 5 from the stack
  ```