TITLE: LOGIC FOR CS ASSIGNMENT

Name:K.Gnana Pratheek Reddy

Roll Number:B231038CS

Overview of the Code

The code consists of the following components:

1. **FiniteRun Function**: A generic function that checks if a system can reach a final state within k steps.
2. **State Transition Systems**:

  - **2-Bit Counter**: A 2-bit counter that increments from 00 to 11 and wraps around.
  - **3-Bit Counter**: A 3-bit counter that increments from 000 to 111 and wraps around.
  - **4-Bit Counter**: A 4-bit counter that increments from 0000 to 1111 and wraps around.
  - **Vending Machine**: A simple vending machine with two states (Idle and Dispense).
  - **Mutex Protocol**: A mutual exclusion protocol with two processes, each having three states (Idle, Waiting, and Critical).
3. **Testing**: Each system is tested with different values of k to determine if the final state can be reached.

Explanation of the FiniteRun Function

The FiniteRun function is the core of the code. It checks whether a system can reach a final state within k steps. Here’s a detailed explanation of its logic:

Parameters:

- M: A tuple containing:
    - I: The initial condition.
    - T: The transition relation.
    - F: The final condition.
- k: The maximum number of steps to check.

Steps:

1. **Check if the Initial State Satisfies the Final Condition**:
    - The solver checks if the initial state I already satisfies the final condition F.
    - If true, the function returns True.
2. **Create Variables for Each Step**:
    - A list of Z3 integer variables (states) is created to represent the state at each step.

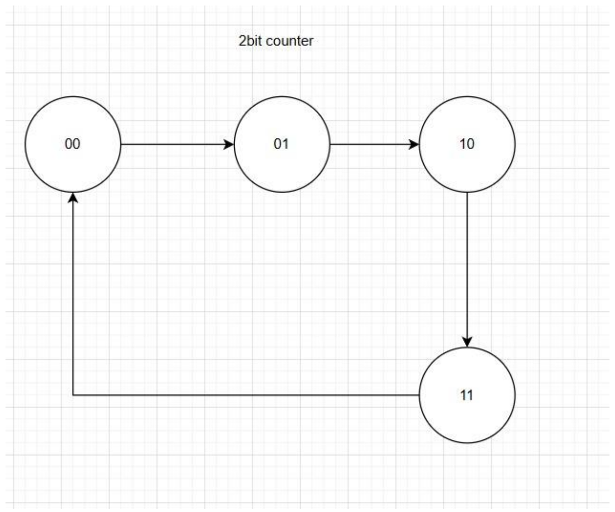
    - For example, step_0 represents the initial state, and step_k represents the state after k steps.

3. **Add Initial State Constraint**:
    - The initial state is constrained to the value specified in I.
4. **Add Transition Constraints**:
    - For each step i, the transition relation T is applied to move from step_i to step_{i+1}.
    - This ensures that the system follows the defined transitions.
5. **Add Final State Constraint**:
    - The final state (step_k) is constrained to the value specified in F.
6. **Check Satisfiability**:
    - The solver checks if the constraints are satisfiable (i.e., if there exists a sequence of transitions that reaches the final state within k steps).

    - If satisfiable, the function returns True; otherwise, it returns False.

State Transition Systems

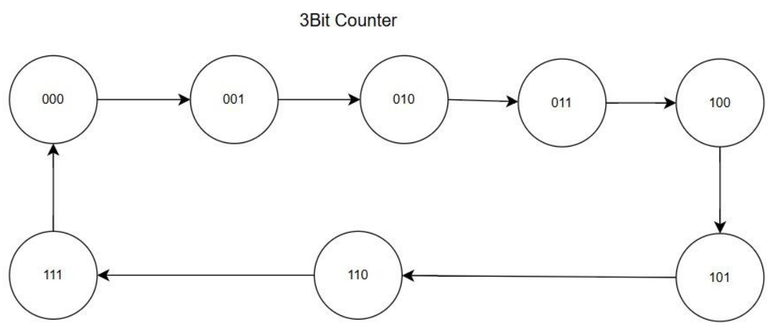
3.1. 2-Bit Counter

- **States**: 00 (0), 01 (1), 10 (2), 11 (3).
- **Transitions**:
    - 00 → 01 → 10 → 11 → 00 (wraps around).
- **Final Condition**: Reach 11 (3).
- 

3.2. 3-Bit Counter

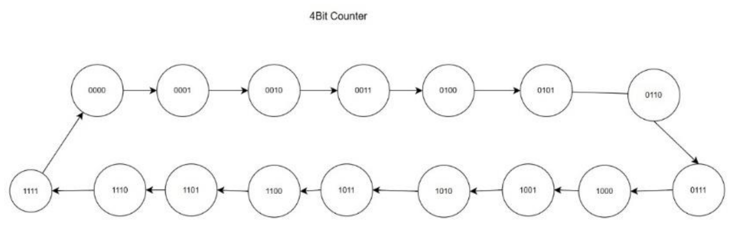
- **States**: 000 (0) to 111 (7).
- **Transitions**:

    - Increments by 1 at each step, wrapping around after 111.
- **Final Condition**: Reach 111 (7).

- 

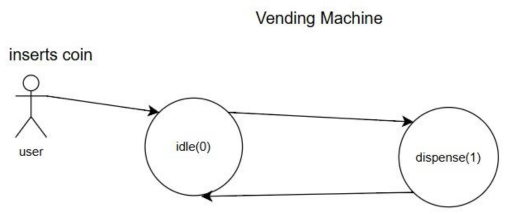
3.3. 4-Bit Counter

- **States**: 0000 (0) to 1111 (15).
- **Transitions**:
    - Increments by 1 at each step, wrapping around after 1111.
- **Final Condition**: Reach 1111 (15).

- 

3.4. Vending Machine

- **States**:
  - Idle (0): The machine is waiting for coins.
  - Dispense (1): The machine is dispensing an item.
- **Transitions**:
  - Idle → Dispense: When coins are inserted.
  - Dispense → Idle: After dispensing.



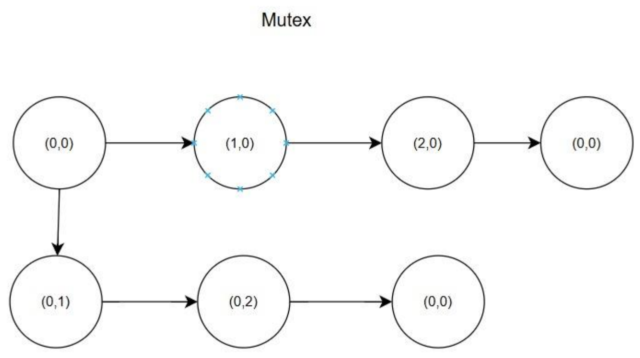
-

3.5. Mutex Protocol

- **States**:
  - Each process (p1 and p2) can be in one of three states:
    - Idle (0): The process is not using the resource.

        - Waiting (1): The process is waiting to enter the critical section.
        - Critical (2): The process is using the resource.

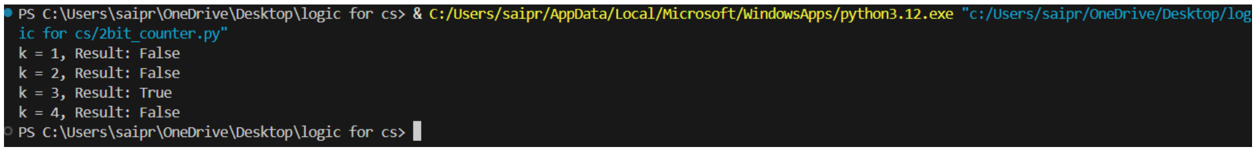
- **Transitions**:
    - Processes can move from Idle to Waiting and then to Critical.
    - After using the resource, processes return to Idle.
- **Final Condition**: Both processes are in the Critical state simultaneously (this is an unsafe state and should not be reachable).



Testing and Results

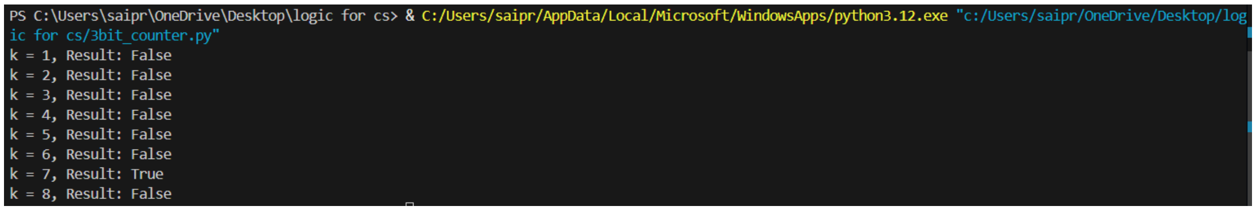
Each system is tested with different values of k to determine if the final state can be reached. Here are the results for each system:

4.1. 2-Bit Counter

- For k = 1: False (cannot reach 11 in 1 step).
- For k = 2: False.
- For k = 3: True (reaches 11 in 3 steps).
- For k = 4: False.
- 

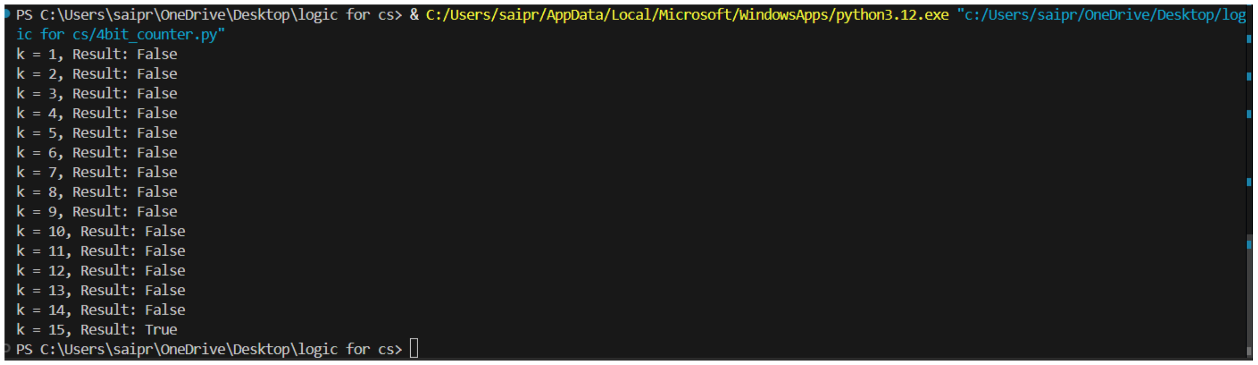
```
PS C:\Users\saipr\OneDrive\Desktop\logic for cs> & C:/Users/saipr/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/saipr/OneDrive/Desktop/logic for cs/2bit_counter.py"
k = 1, Result: False
k = 2, Result: False
k = 3, Result: True
k = 4, Result: False
PS C:\Users\saipr\OneDrive\Desktop\logic for cs>
```

4.2. 3-Bit Counter

- For k = 1 to k = 6: False.
- For k = 7: True (reaches 111 in 7 steps).
- For k = 8: False.
- 

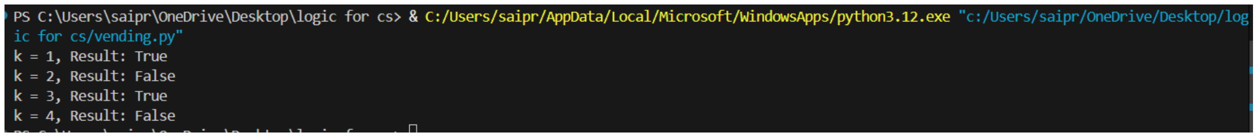
```
PS C:\Users\saipr\OneDrive\Desktop\logic for cs> & C:/Users/saipr/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/saipr/OneDrive/Desktop/logic for cs/3bit_counter.py"
k = 1, Result: False
k = 2, Result: False
k = 3, Result: False
k = 4, Result: False
k = 5, Result: False
k = 6, Result: False
k = 7, Result: True
k = 8, Result: False
```

4.3. 4-Bit Counter

- For k = 1 to k = 14: False.
- For k = 15: True (reaches 1111 in 15 steps).
- 

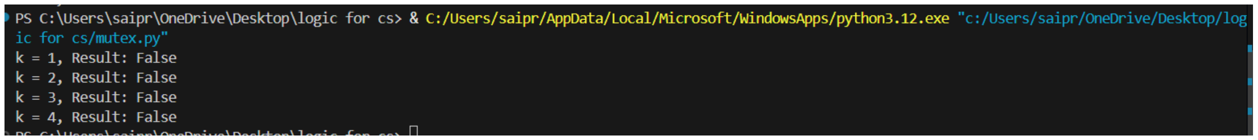
```
PS C:\Users\saipr\OneDrive\Desktop\logic for cs> & C:/Users/saipr/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/saipr/OneDrive/Desktop/log
ic for cs/4bit_counter.py"
k = 1, Result: False
k = 2, Result: False
k = 3, Result: False
k = 4, Result: False
k = 5, Result: False
k = 6, Result: False
k = 7, Result: False
k = 8, Result: False
k = 9, Result: False
k = 10, Result: False
k = 11, Result: False
k = 12, Result: False
k = 13, Result: False
k = 14, Result: False
k = 15, Result: True
PS C:\Users\saipr\OneDrive\Desktop\logic for cs> 
```

4.4. Vending Machine

- For k = 1: True (reaches Dispense in 1 step).
- For k = 2: False.
- For k = 3: True.
- For k = 4: False.
- 

```
PS C:\Users\saipr\OneDrive\Desktop\logic for cs> & C:/Users/saipr/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/saipr/OneDrive/Desktop/log
ic for cs/vending.py"
k = 1, Result: True
k = 2, Result: False
k = 3, Result: True
k = 4, Result: False
```

4.5. Mutex Protocol

- For all tested values of k (1 to 4): False.
- This confirms that the unsafe state (both processes in Critical) is not reachable.
- 

```
PS C:\Users\saipr\OneDrive\Desktop\logic for cs> & C:/Users/saipr/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/saipr/OneDrive/Desktop/log
ic for cs/mutex.py"
k = 1, Result: False
k = 2, Result: False
k = 3, Result: False
k = 4, Result: False
```

5. Conclusion

The code demonstrates how to use the Z3 SMT solver to verify finite state systems. The FiniteRun function is a powerful tool for checking reachability in state transition systems. The results show that the counters and vending machine behave as expected, while the mutex protocol successfully prevents both processes from entering the critical section simultaneously.