EECS 219C: Formal Methods

Modeling for Verification: Example Used in Lecture

Sanjit A. Seshia

EECS, UC Berkeley

Example: Interrupt-Driven S/W

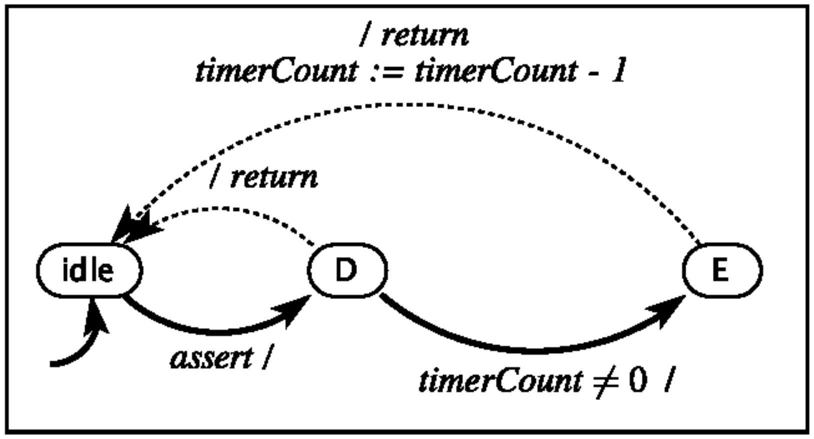
```
    volatile uint timerCount = 0;
    void ISR(void) {
       … disable interrupts
D →    if(timerCount != 0) {
E →       timerCount--;
       }
       … enable interrupts
    }
    int main(void) {
       // initialization code
       SysTickIntRegister(&ISR);
       … // other init
       timerCount = 2000;
A →    while(timerCount != 0) {
B →     … code to run for 2 seconds
       }
C → … whatever comes next
    }
```

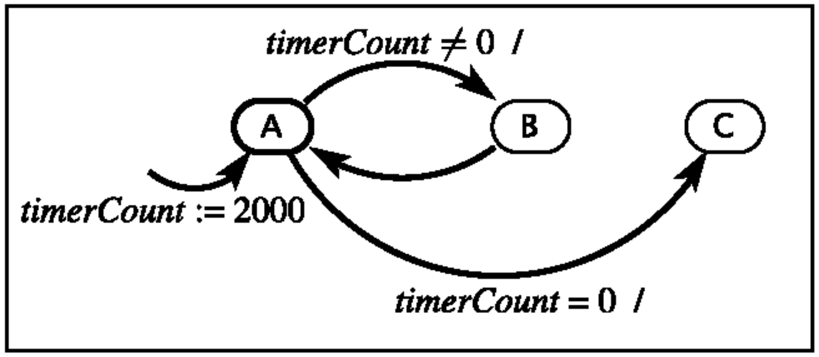
Question: Assuming interrupt can occur infinitely often, is position C always reached?

State machine model

```
     volatile uint timerCount = 0;
     void ISR(void) {
        … disable interrupts
D →     if(timerCount != 0) {
E →        timerCount--;
        }
        … enable interrupts
     }
     int main(void) {
        // initialization code
        SysTickIntRegister(&ISR);
        … // other init
        timerCount = 2000;
A →     while(timerCount != 0) {
B →      … code to run for 2 seconds
        }
C →  … whatever comes next
     }
```

variables: *timerCount*: `uint`
input: *assert*: pure
output: *return*: pure



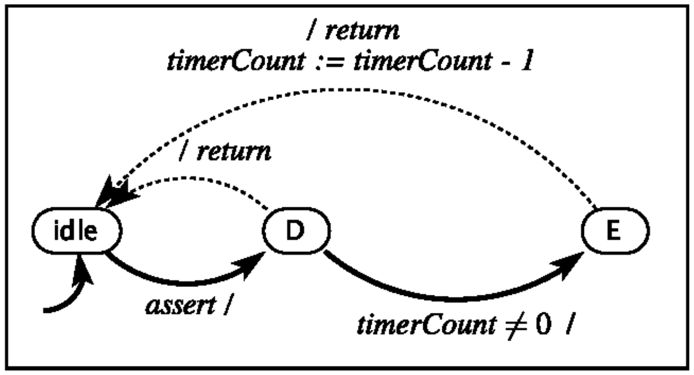


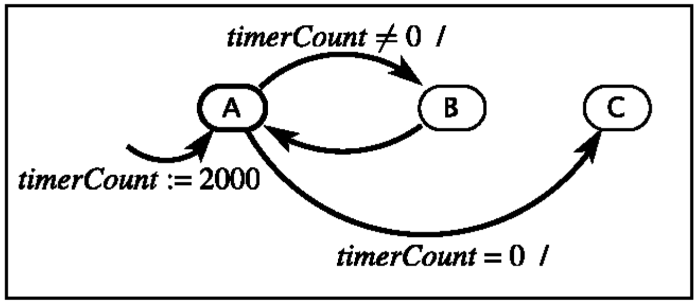
Which form of composition is the right thing to do here?

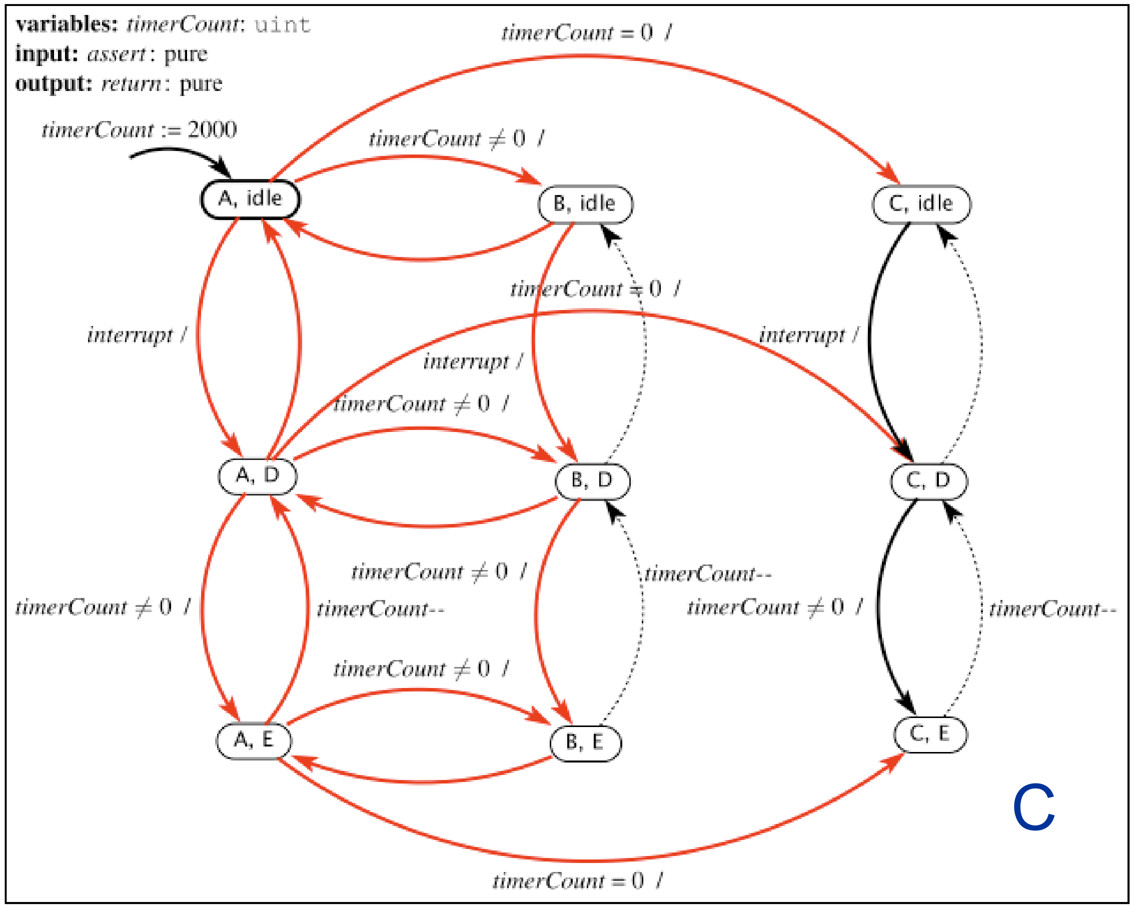
Asynchronous Composition

$$S_C = S_A \times S_B$$

variables: *timerCount*: `uint`
input: *assert*: pure
output: *return*: pure







This has transitions that will not occur in practice, such as A,D to B,D. Interrupts have priority over application code.