Concurrency & Concurrent Processes

Outline

- Concurrent Process
- Bernstein Conditions for Concurrency
- Concurrency Representation
  - Begin---end, **and** *par begin* ---par end
  - Fork and Join
  - Semaphores

Concurrency (Concurrent Processes)

Concurrent Processes

- Two Processes are said to be concurrent if they can be executed simultaneously **(in Parallel)** in a multiprogramming environment.

- In a uniprocessor environment, two processes are said to be concurrent if the order of execution doesn't matter, i.e., either of the processes can be executed first.

- **Example: Consider a Process p with four statements.**
  - **S1 & S2 are concurrent**
  - **S1 & S3 are sequential (non-concurrent)**
  - **S2 & S3 are also sequential**
  - **S2 & S4 are also sequential (Transitive concurrency)**
  - **S3 & S4 are also sequential**

Process P1:

```
S1 :    a = b + c;
S2 :    d = b – d;
S3 :    e = a * d;
S4 :    f = e / 2;
```

Bernstein Concurrency Conditions

- Two Processes (statements) are concurrent if the following three conditions are satisfied:
- **Let S1 and S2 are two processes:**
  - **R (S1) ∩ W (S2) = ϕ**
  - **W (S1) ∩ R (S2) = ϕ**
  - **W (S1) ∩ W(S2) = ϕ**
- **R(S1):** Read the set of processes/statements S1. It is the set of all variables that are read-referenced in the process/statement S1.
  - **E.g., R(S1) = {b, c}**
- **W(S1):** Write the set of processes/statements S1. It is the set of all those variables on which write operation is performed in the process/statement.
  - **E.g., W(S1) = {a}**
- ***Note: Along with the Bernstein condition, transitive dependencies are also to be observed to identify concurrency.***

Representation of Concurrency

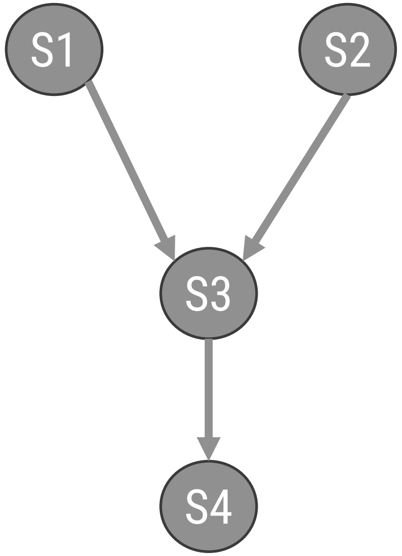
- **Two Methods:**
  - Precedence Graph
  - Using Concurrent Statements (Languages)
    - Begin---end, **and** *par begin* ---par end
    - Fork and Join
    - Semaphores

Representation of Concurrency: Precedence Graph

- ***It is a directed graph with a vertex (statements) & edges (between dependent statements).***

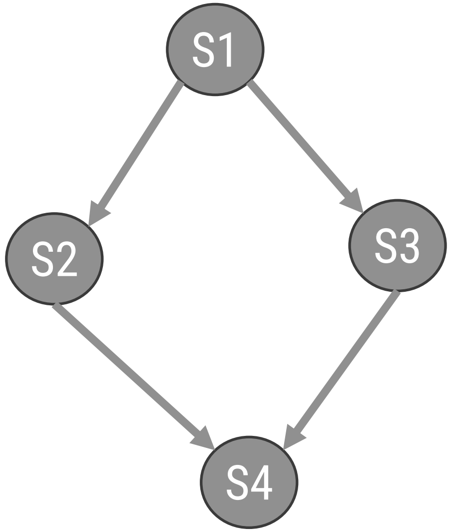
- ***Example:***

S1 :	a = b + c;
S2 :	d = b − d;
S3 :	e = a * d;
S4 :	f = e / 2;



Example:

S1 :	a = b + c;
S2 :	d = a * 2;
S3 :	e = e + a;
S4 :	f = d / e;

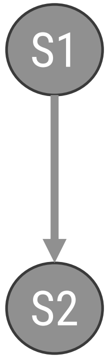


Representation of Concurrency: Using Concurrent Statements

- ***1. begin---end and par begin----par end***

Sequential

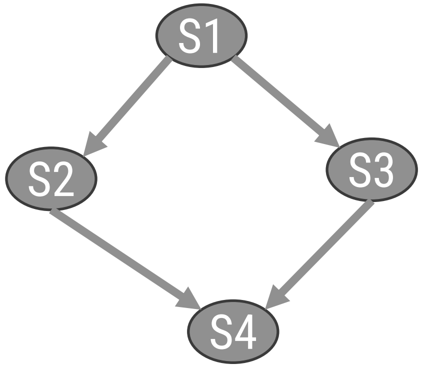
```
begin
      S1;
      S2;
end
```



Concurrent

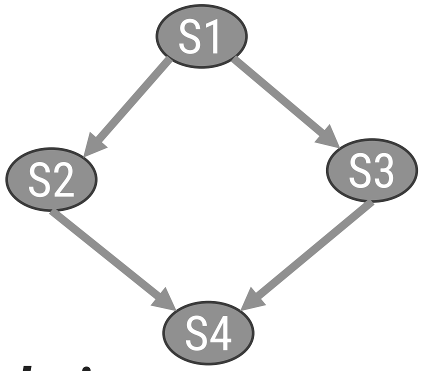
```
par begin
      S1;
      S2;
par end
```

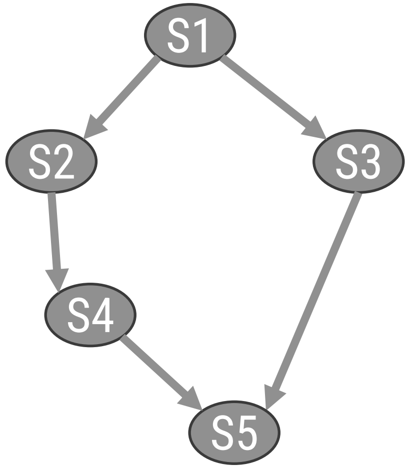
- ***Example: Write concurrent statements for the following graph.***



Representation of Concurrency: Using Concurrent Statements

- ***Ques. Write concurrent statements for the following graph.***





- ***Solution:***

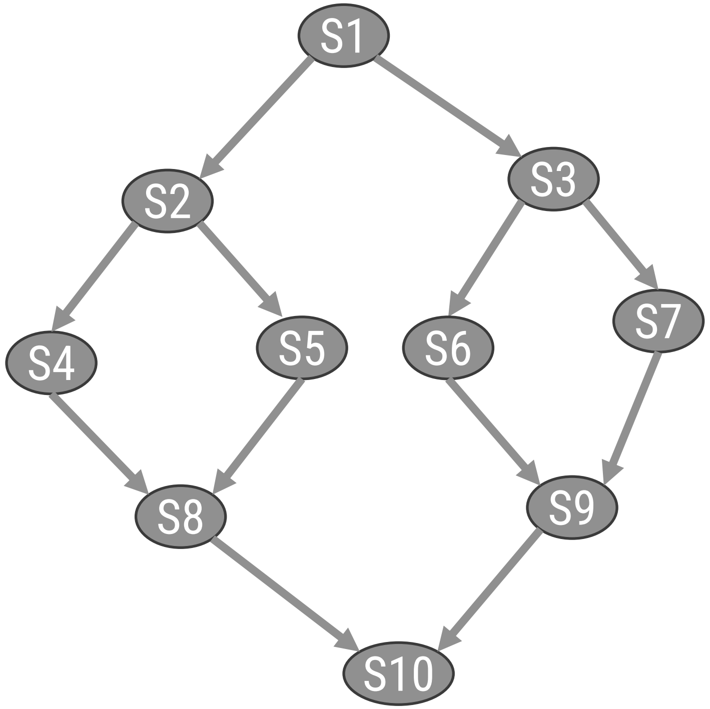
```
begin
        S1;
        par begin
                S2;
                S3;
        par end
        S4;
end
```

```
begin
        S1;
        par begin
                S3;
                begin
                        S2;
                        S4;
                end
        par end
        S5;
end
```

```
begin
        S1;
        par begin
                begin
                        S2;
                        S4;
                End
                S3;
        par end
        S5;
end
```

Representation of Concurrency: Using Concurrent Statements

- ***Ques. Write concurrent statements for the following graph.***



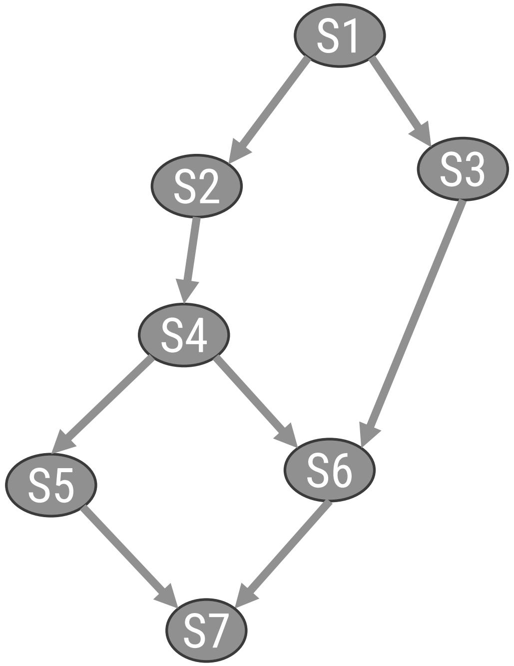
Solution:
begin
S1;
par begin
begin
S2;
par begin
S4;
S5;
par end
S8;
end

begin
S3;
par begin
S6;
S7;
par end
S9;
end
par end
S10;
end

- ***Solution:***

Representation of Concurrency: Using Concurrent Statements

- ***Ques. Write concurrent statements for the following graph.***



- ***Solution:***
  - *Not Possible with only begin—end, and par begin – par end, method.*

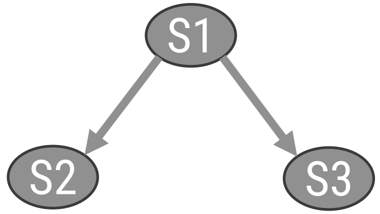
Representation of Concurrency: Using Concurrent Statements

- ***2. Fork & Join Construct:***
- ***Fork:***
  - *E.g.,*

Syntax for using Fork:

```
Fork:  L1
       -----
       -----
       -----
L1:
       ---
       ---
       ---
```

Concurrent




```
      S1;
Fork:  L1
      S2;
      Goto Next
L1: S3;

Next:
```

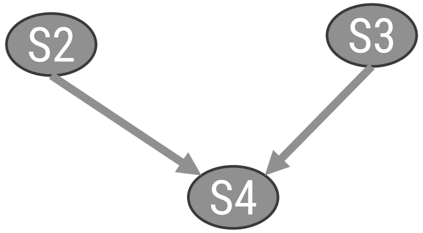
  - ***L1 (Level):*** *Used to jump. When we create a level (L1), two concurrent processes are created.*
  - *First one below the **Fork**, & second one is start from level **L1**.*

Representation of Concurrency: Using Concurrent Statements

- ***Fork* & *join* Construct:**
- ***Join:***
  - *E.g.,*

Syntax for using Join:

```
        initialize counter
        -----
        -----
    Join Counter
        -----
        -----
        -----
```

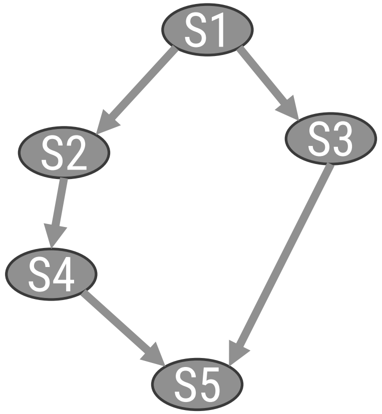


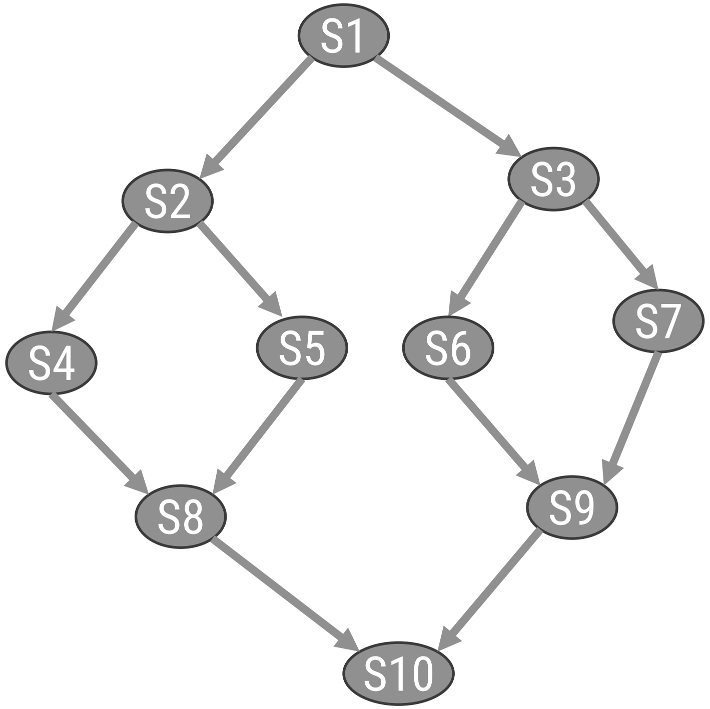

```
     C =2;
Fork:  L1
     S2;
     Goto L2

L1: S2;

L2: join C
      S4;
```

  - *On a visit to the Join statement, the counter variable is decremented by one every time.*
  - *If the counter variable becomes zero after decrement, then the join statement allows the control to execute the next below mentioned statements.*
  - *If the counter is not zero, block the control.*

Representation of Concurrency: Using Concurrent Statements

- ***Ques. Write concurrent statements (using Fork and Join) for the following graph.***





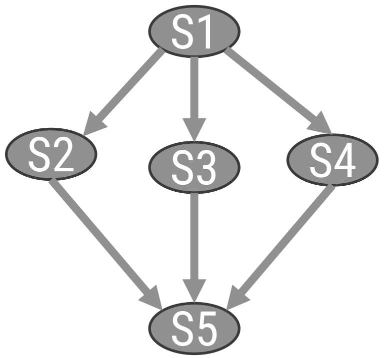
Solution: ?

- ***Solution:***

```
C=2;
        S1;
        Fork: L1
        S3;
        Goto L2
        L1:   S2;
              S4;
        L2: join C
              S5;
```

Representation of Concurrency: Using Concurrent Statements

- ***Ques. Write concurrent statements (using Fork and Join) for the following graph.***

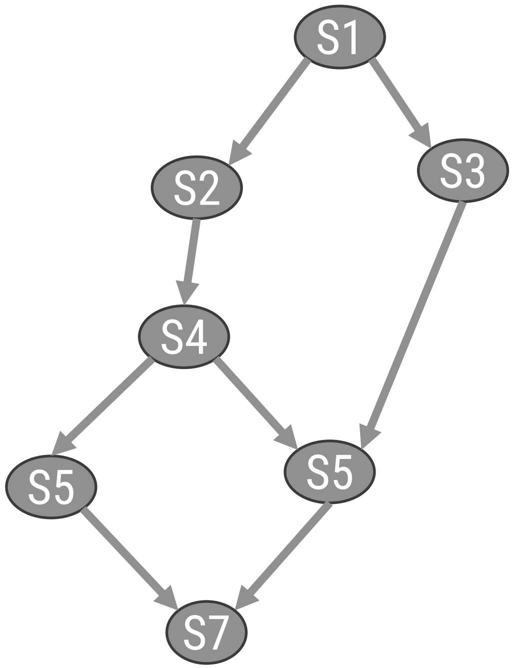


- ***Solution:***

```
C=3;
    S1;
    Fork: L1
    S2;
    Goto L3

    L1:  Fork: L2;
         S3;
         Goto L3

    L2: S4;

    L3: join C
         S5;
```

Representation of Concurrency: Using Concurrent Statements

- ***Ques. Write concurrent statements (using Fork and Join) for the following graph.***



- ***Solution:***

Representation of Concurrency: Concurrent Statements (Semaphore)

- ***3. Semaphore:***
  - *Semaphores are a special type of variable that **cannot be used (modified)** in arithmetic & logical operations. These variables can only be altered with predefined operations called **wait(P)** and **signal (V).***

- ***Types of Semaphore***
  - ***Boolean (value- 0 or 1)***
  - ***Counting***

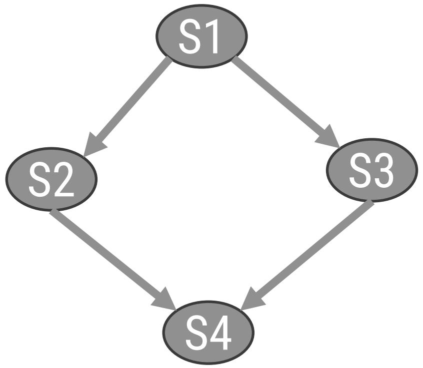
- ***Example:***

```
a : Semaphore
------
------
------ wait(a)   or P(a)
------
-----signal(a)   or V(a)
-----
----
```

- ***Wait(a):*** *It decrements the value of a and allows control to go to the next statement if a is 0. Otherwise, wait or block the control.*
- ***Signal(a):*** *No wait, it only increments the value by 1, passes the control to go to the next statement.*
- ***Semaphore can be initialized*** *using built-in functions as*
  - *a, b: counting/Boolean semaphores*
  - *Initialize(a, b =0)*

Representation of Concurrency: Concurrent Statements (Semaphore)

- ***Ques. Write concurrent statements using Semaphores for the following graph.***



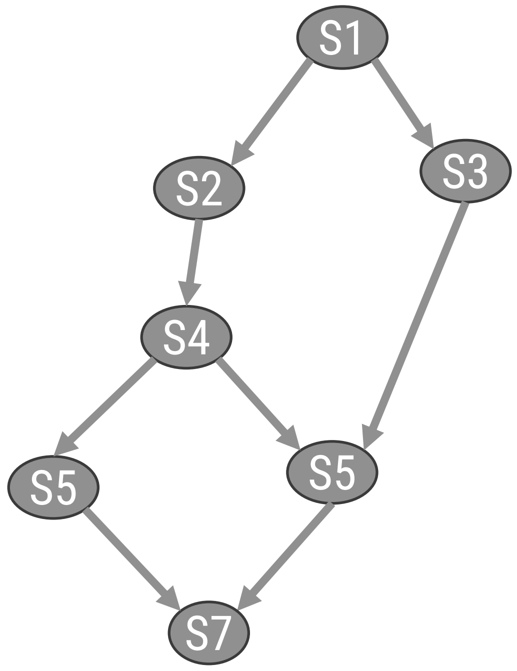
- ***Solution:***

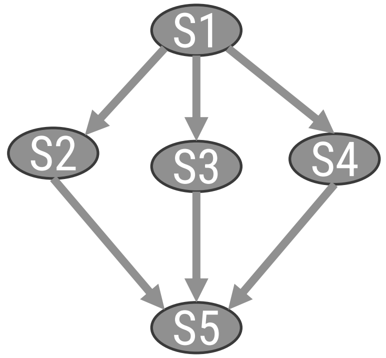
Solution:

a, b, c, d: Boolean semaphores
Initialize (a, b c, d, =0)
par begin
    begin S1; V(a); V(b); end
    begin P(a); S2; V(c); end
    begin P(b); S3; V(d); end
    begin P(c); P(d); S4; end
par end

Representation of Concurrency: Concurrent Statements (Semaphore)

- ***Ques. Write concurrent statements using Semaphores for the following graphs.***





- ***Solution:***

Thank You