Especificação de Concorrência

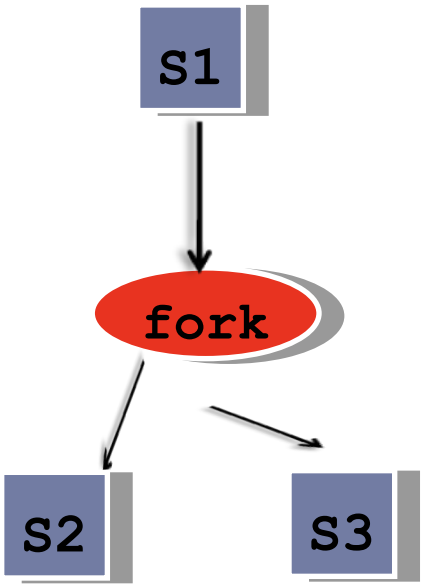
Professora: Lúcia Drummond

Construção FORK e JOIN

- Conway[1963]
- Dennis e Van Horn[1966]

Construção FORK e JOIN

- Uma das primeiras notações de linguagens para especificar concorrência.




```
    S1;

    fork L;

    S2;
    •
    •
    •
L:  S3;
```

Construção FORK e JOIN

- A instrução `fork L` produz duas execuções concorrentes no programa:
  - Uma execução começa no comando de *label* `L`;
  - Enquanto a outra é a continuação da execução do comando seguindo a instrução `fork`.

Construção FORK e JOIN

- A instrução `join` fornece o meio de recombinar duas computações concorrentes em uma.
- Cada uma das duas computações deve pedir para ser unida com a outra.
- Já que computações podem executar em diferentes velocidades, uma pode executar o `join` antes da outra.

Construção FORK e JOIN

- A computação que executa o `join` primeiro termina, enquanto a outra continua.

- Se houvessem três computações para serem unidas:
  - as duas primeiras a executarem o `join` terminam,
  - enquanto a terceira continua.

- É necessário saber o número de computações que devem ser unidas, para que se possa terminar todas, menos a última.

Construção FORK e JOIN

- A instrução `join` tem um parâmetro para especificar o número de computadores para unir.

- Assim, a execução da instrução `join` tem o seguinte efeito:

```
Count := Count - 1;
if (Count != 0) then
    quit;
```

Onde `Count` é uma variável inteira não negativa e `quit` é uma instrução que resulta no término da execução. (join é atômico!)

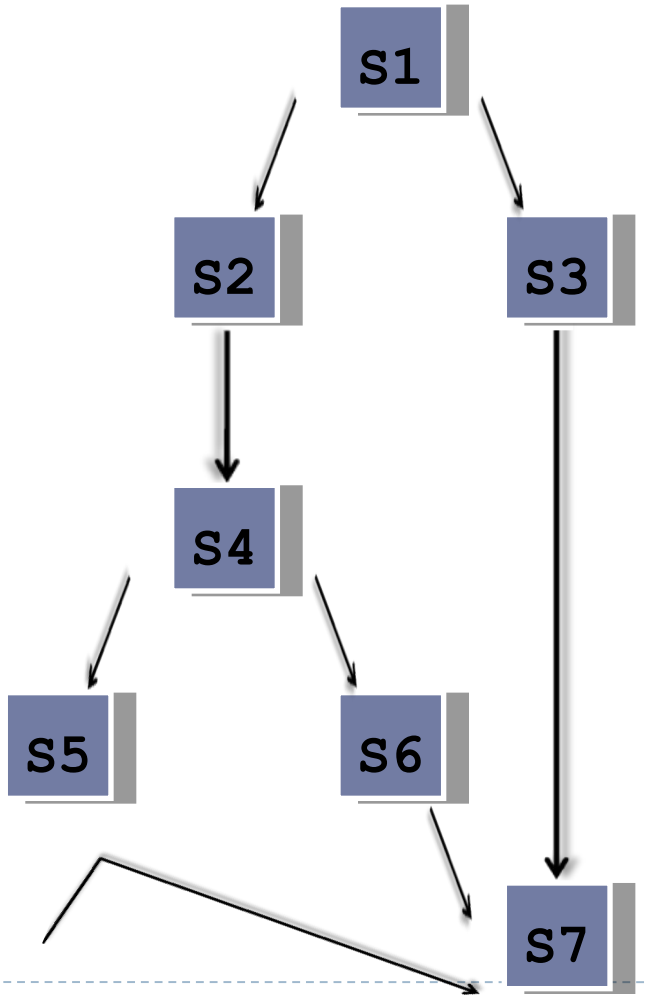
Construção FORK e JOIN

- Exemplo

```
     Count := 2;
     fork L1;
     read(a);
     goto L2;
L1:  read(b);
L2:  join Count;
     c := a + b;
     write(c);
```

Construção FORK e JOIN

- Grafo de precedência




```
      S1;
      Count := 3;
      fork L1;
      S2;
      S4;
      fork L2;
      S5;
      goto L3;
L2:   S6;
      goto L3;
L1:   S3;
L3:   join Count;
      S7;
```

Construção FORK e JOIN

- Programa que copia sequencialmente um arquivo para outro:

```
var F, G: file of T;
    R, S: T
    Count: integer;
begin
  reset(F);
  read(F,R);
  while not eof(F) do
   begin
     Count := 2;
     S := R;
     fork L1;
     write(G,S);
     goto L2;
L1:  read(F,R);
L2:  join Count;
   end
end;
```

PARBEGIN / PAREND

- Dijkstra(1965) → Construção de nível mais alto.

```
S0; PARBEGIN S1; S2; …; Sn PAREND; Sn + 1
```

PARBEGIN / PAREND

- Exemplo 1

```
parbegin
   read(a);
   read(b);
parend;
c := a + b;
write(c);
```

PARBEGIN / PAREND

- Exemplo 2 - O programa que copia um arquivo F para outro arquivo G.

```
var F, G: file of T;
    R, S: T
begin
  reset(F);
  read(F,R);
  while not eof(F) do
   begin
     S := R;
     parbegin
       write(G,S);
       read(F,R);
     parend
   end
  write(G,R);
end;
```

PARBEGIN / PAREND

- Exemplo 3

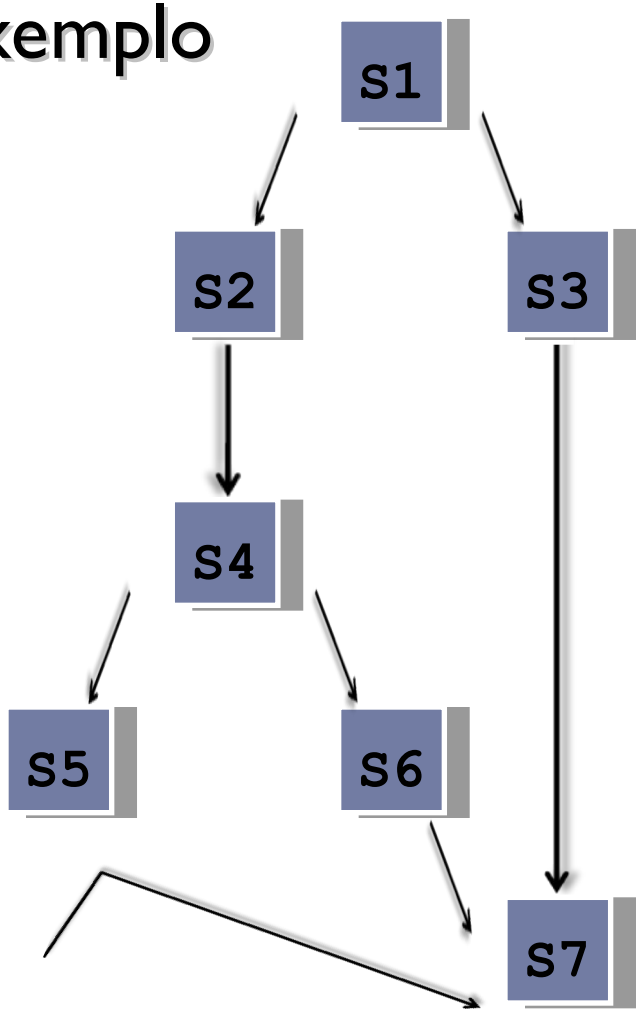
```
S1;
parbegin
   S3;
   begin
      S2;
      S4;
      parbegin;
         S5;
         S6;
      parend;
   end;
parend;
S7;
```

PARBEGIN / PAREND

- Os comandos `parbegin` / `parend` não são poderosos o suficiente para modelar todos os grafos de precedência possíveis.

PARBEGIN / PAREND

- Exemplo




```
     S1;
     Count1 := 3;
     fork L1;
     S2;
     S4;
     Count2 := 2;
     fork L2;
     S5;
     goto L3;
L1:  S3;
L2:  join Count1;
     S6;
L3:  join Count2;
     S7;
```

PARBEGIN / PAREND

```
var A, B, C, D, E, F, G: Semaphores;
begin
  pargbegin
    begin S1; V(A); V(B); end;
    begin P(A); S2; S4; V(C); V(D); end;
    begin P(B); S3; V(E); V(B); end;
    begin P(C); S4; V(F); end;
    begin P(D); P(E); S6; V(G); end;
    begin P(F); P(G); S7; end;
  parend;
end;
```