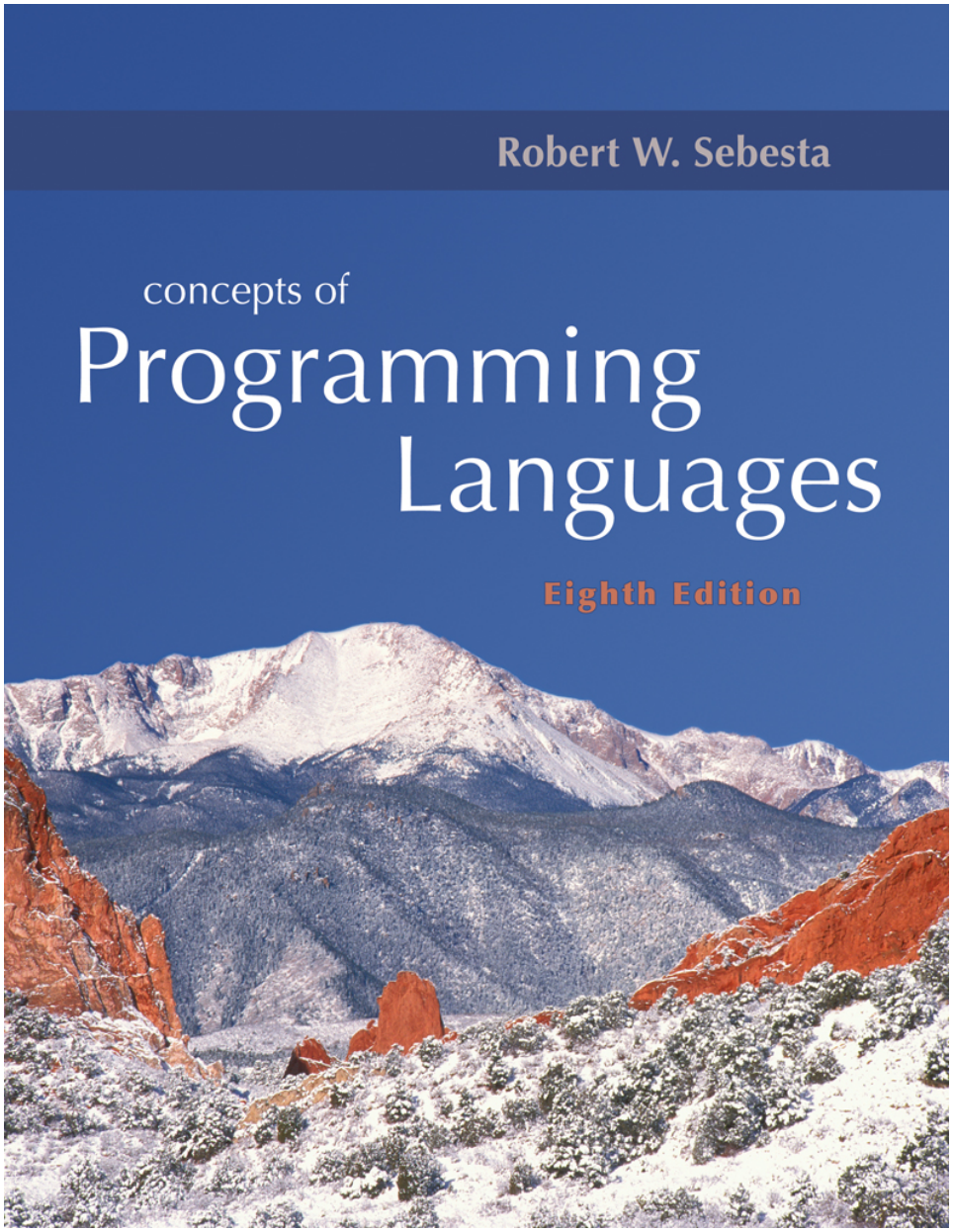
Bölüm 10

Eşzamanlılık (Concurrency)



ISBN 0-321-49362-1

Eşzamanlılık Nedir?

- Kodun iki veya daha fazla parçasının aynı anda eş zamanlı çalıştırılmasıdır
- Programlama dillerindeki eş zamanlılık kavramı ile bilgisayar donanımındaki paralel çalışma birbirinden bağımsız kavramlardır.
- Eğer çalışma zamanında üst üste gelme durumu varsa donanım işlemlerinde paralellik oluşur.
- Bir programdaki işlemler eğer paralel olarak işlenebiliyorsa program eş zamanlıdır denilir.
- Eş zamanlılık kavramının karşıtı ise belirli bir sıraya göre dizilmiş ardışıl işlemlerdir.

Neden Kullanılır?

- Veriler coğrafi olarak farklı yerlerde bulunabilir
- Farklı makineler arası işlemler yapmaya izin verir
- Bir parça kod bir çok client prosesine hizmet vermesi gerekebilir
- Hızı artırır
- Programlamayı kolaylaştırır

Eşzamanlılık Kavramları

- Eş zamanlı prosesler:
  Dönüşümlü çalışma veya
  Fiziksel eş zamanlılık şeklinde uygulanabilir
- Dönüşümlü çalışma:
  Tekli işlemci üzerinde çoklu program çalıştırma
- Fiziksel eşzamanlılık:
  Çoklu işlemci üzerinde tekli veya çoklu program çalıştırma
- Process, thread, task:
  Hesaplamanın sıralı yürütülen birimleridir
- Program parçalarının büyüklüğü:
  Çok küçük olursa: işlem yükü artar
  Çok büyük olursa: düşük eşzamanlılık olur

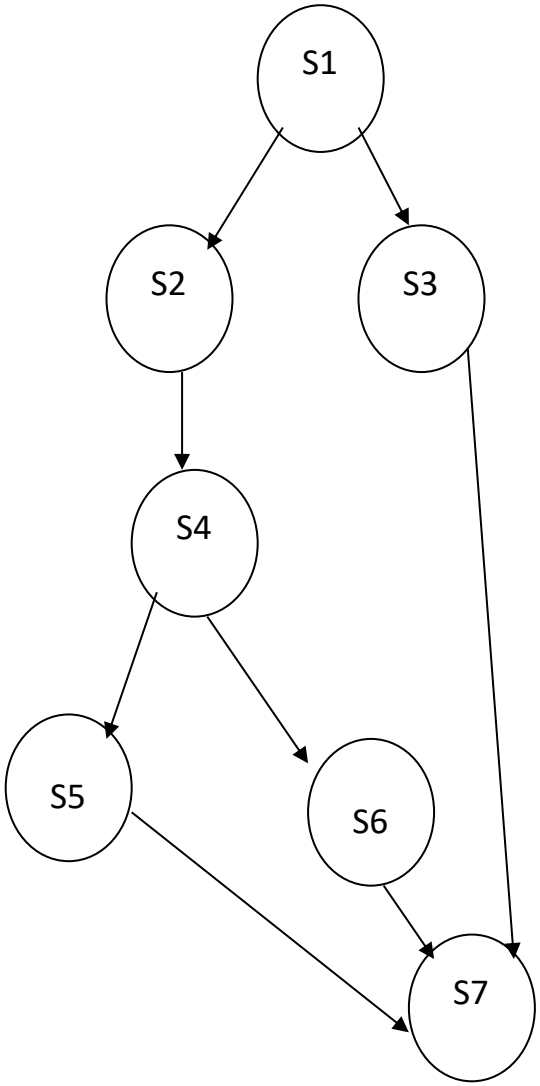
Öncelik Grafı

```
a:=x + y;
b:= z + 1;
c:= a -  b;
w:= c + 1;
```

Burada c:=a –b yi hesaplamak için öncelikle a ve b'ye değer atanması gerekmektedir. Benzer biçimde w:=c + 1 ifadesinin sonucu da c'nin hesaplanmasına bağlıdır. Diğer taraftan a:=x + y ve b:= z + 1 deyimleri birbirine bağlı değildir. Bu yüzden bu iki deyim birlikte çalıştırılabilir.

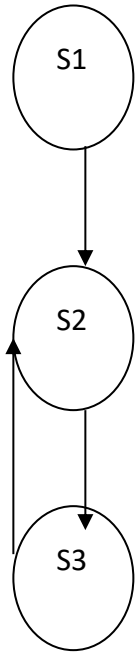
Buradan anlaşılıyor ki bir program parçasında değişik deyimler arasında bir öncelik sıralaması yapılabilir. Bu sıralamanın grafik olarak gösterimine **öncelik grafı** denir. Bir öncelik grafı, her bir düğümü ayrı bir deyimi ifade eden, döngüsel olmayan yönlendirilmiş bir graftır.

Öncelik Grafı (devam)



S2 ve S3 deyimleri, S1 tamamlandıktan sonra işletilebilir.
S4, S2 tamamlandıktan sonra işletilebilir.
S5 ve S6, S4 tamamlandıktan sonra işletilebilir.
S7, sadece S5,S6 ve S3 tamamlandıktan sonra işletilebilir.
Bu örnekte S3 deyimi; S2, S4, S5 ve S6 deyimleri ile eş zamanlı olarak çalışabilir.

Öncelik Grafı (devam)



Bu grafta görüldüğü gibi, S3 sadece S2 tamamlandıktan sonra işletilebilir.
S2 deyimi ise sadece S3 tamamlandıktan sonra işletilebilir.
Burada açıkça görülmektedir ki bu iki kısıtlamanın her ikisi aynı anda giderilemez.
Yani bir programın akışını ifade eden **öncelik grafı döngü içermemelidir.**

Eşzamanlılığı Belirleme

- İki deyim arasındaki eş zamanlılık için şu şartların tümünün sağlanması gerekir:

  1.R(S1) ∩ W(S2)= {}

  2.W(S1) ∩ R(S2)={}

  3.W(S1) ∩ W(S2)={}

S1: a:=x+y
S2:b:=z+1
S3:c:=a-b

R(S1)={x, y}
R(S2)={z}
R(S3)={a, b}
W(S1)={a}
W(S2)={b}
W(S3)={c}

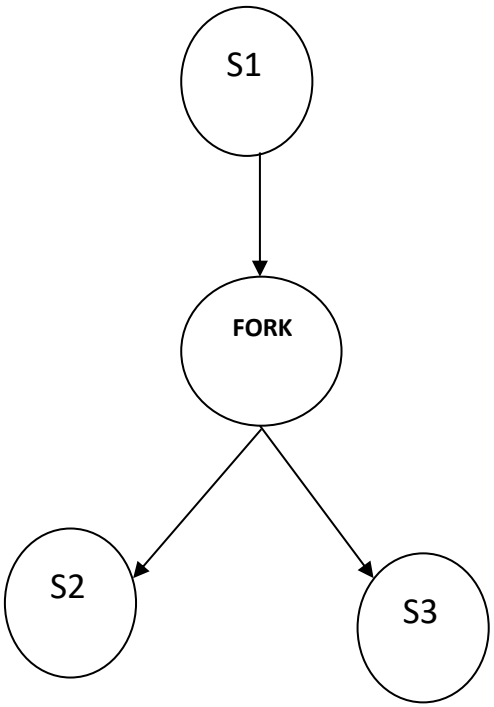
S1 ve S2 deyimleri eş zamanlı olarak çalışabilir mi?
Koşul 1.R(S1)∩ W(S2)= {x,y}∩{b}={}
Koşul 2.W(S1) ∩ R(S2)={a}∩ {z}= {}
Koşul 3.W(S1) ∩ W(S2)={a}∩ {b}= {}

S1 ve S3 deyimleri eş zamanlı olarak çalışabilir mi?
Koşul 1.R(S1) ∩ W(S3)= {x,y} ∩{c}={}
Koşul 2.W(S1) ∩ R(S3)={a} ∩ {a,b}= {a}
Koşul 3.W(S1) ∩ W(S3)={a} ∩ {c}= {}

S2 ve S3 deyimleri eş zamanlı olarak çalışabilir mi?
Koşul 1.R(S2) ∩ W(S3)= {z}∩{c}={}
Koşul 2.W(S2) ∩ R(S3)={b}∩ {a,b}= {b}
Koşul 3.W(S2) ∩ W(S3)={b}∩ {c}= {}

FORK ve JOIN Yapıları

- FORK (Bölme) ve JOIN (Birleşme) yapıları eş zamanlılığı tanımlayan ilk programlama dili notasyonlarından biridir.



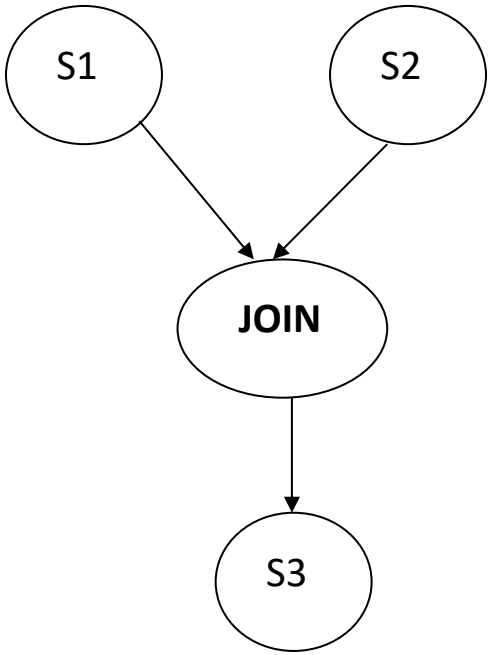

```
S1;
FORK L
S2;
…
…
L:S3;
```

Burada eş zamanlı işlemlerden birisi L etiketi ile gösterilen deyimlerden başlarken diğeri FORK komutunu izleyen deyimlerin işlenmesi ile devam eder.

FORK L deyimi işletildiği zaman S3'de yeni bir hesaplama başlar. Bu yeni hesaplama S2'de devam eden eski hesaplama ile eş zamanlı olarak işletilir.

FORK ve JOIN Yapıları

- JOIN yapısı iki eş zamanlı hesaplamayı tekrar birleştirir.

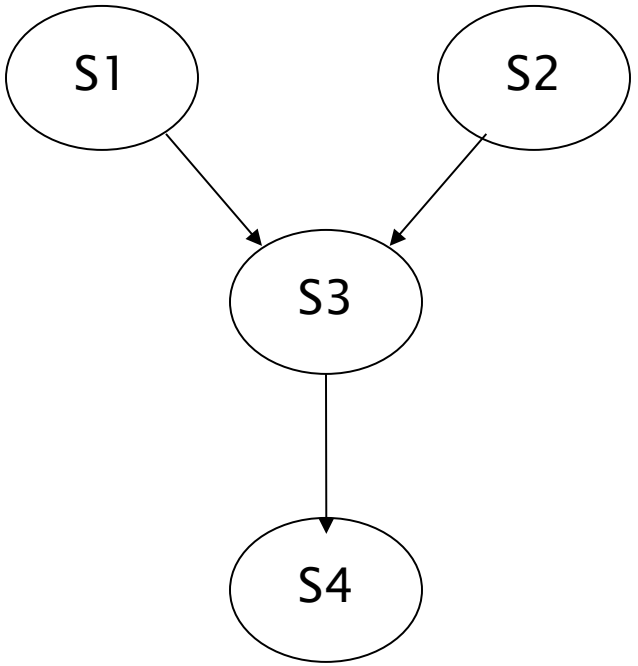



```
kolSayisi:=2;
FORK L1;
…
…
S1;
GOTO L2;
L1:S2;
L2:JOIN kolSayisi;
S3;
```

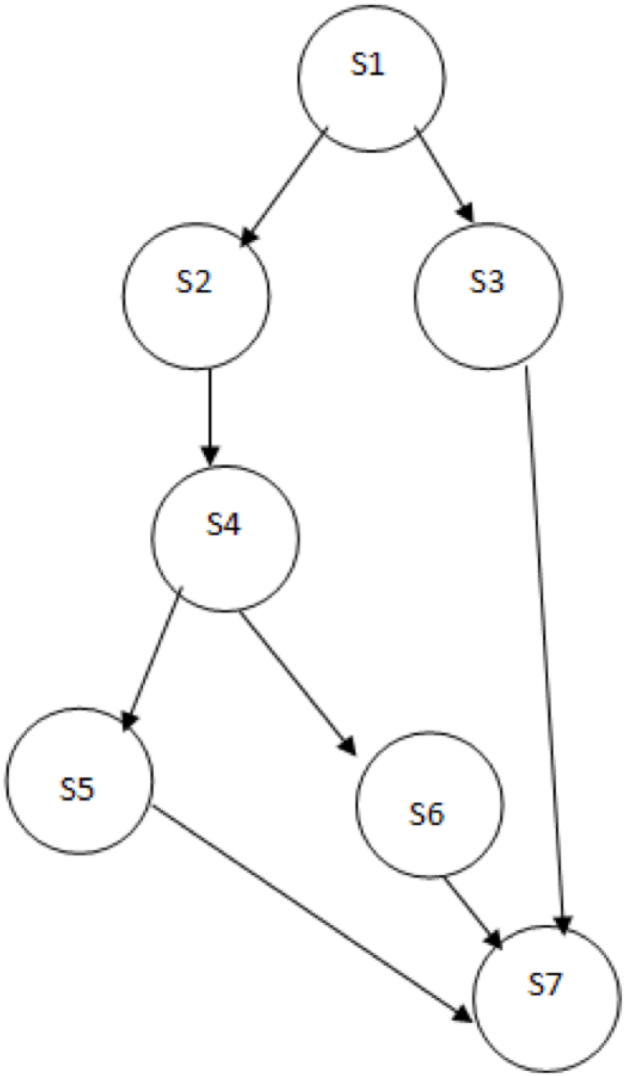
Fork/Join Örnek

```
S1:     A := X + Y
S2:     B := Z + 1
S3:     C := A – B
S4:     W :=C + 1
```

```
Count :=  2;
FORK   L1;
A := X + Y;
GOTO L2;
L1:     B := Z + 1;
L2:     JOIN Count;
C := A – B;
W := C + 1;
```



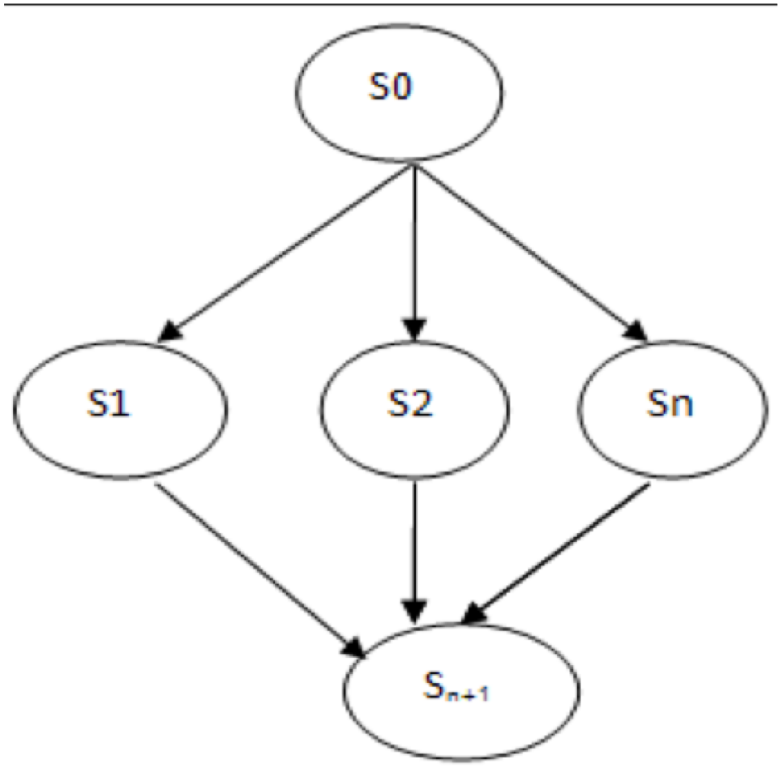
Fork/Join Örnek




```
S1;
kolSayisi:=3;
FORK L1;
S2;
S4;
FORK L2;
S5;
GOTO L3;
L2: S6;
GOTO L3;
L1: S3;
L3: JOIN kolSayisi;
S7;
```

Parbegin/Parend Yapısı

```
S0;
Parbegin
S1;
S2;
…
…
Sn;
Parend
Sn1;
```



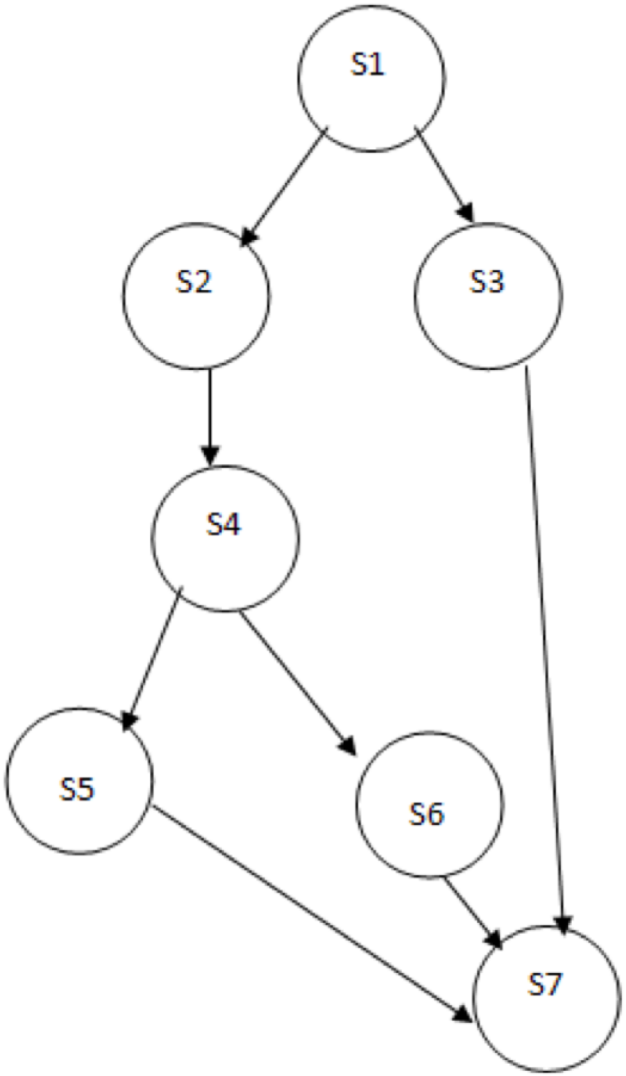
Parbegin/Parend Örnek

```
Begin
     PARBEGIN
A := X + Y;
B := Z + 1;
     PAREND;
C := A – B;
W := C + 1;
End
```

Öncelik graflarını siz çizin!

```
Begin
     S1;
     PARBEGIN
S3
BEGIN
     S2;
     S4;
     PARBEGIN
          S5;
          S6;
     PAREND;
 End;
      PAREND;
      S7;
End;
```

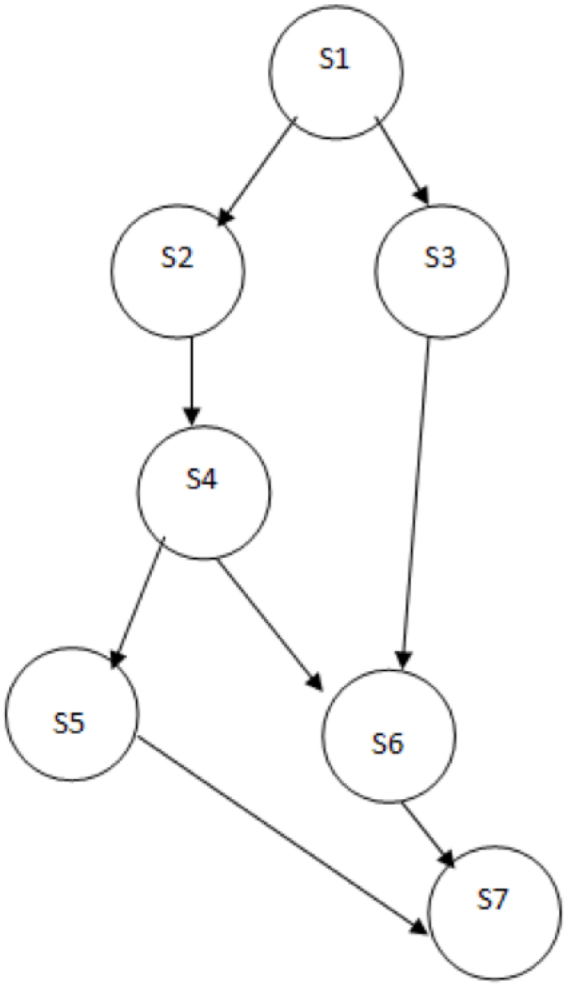
Parbegin/Parend Örnek




```
S1;
Parbegin
   S3;
   Begin
      S2;
      S4;
      Parbegin
         S5;
         S6;
      Parend
   End
Parend
S7;
```

Parbegin/Parend mi? Fork/Join mi?

```
S1;
Kol1:=2;
FORK L1;
S2;
S4;
Kol2:=2;
FORK L2;
S5;
GOTO L3;
L1:S3;
L2: JOIN kol1;
S6;
L3: JOIN kol2;
S7;
```



Bu öncelik grafını sadece parbegin-parend yapısı kullanarak gerçekleştiremeyiz.