Zápis paralelismu pomocí cobegin – coend a precedenčních grafů

- Možnost zápisu paralelně probíhajících entit.
- Např. mají-li procedury p1 a p2 běžet paralelně, lze to zapsat takto:

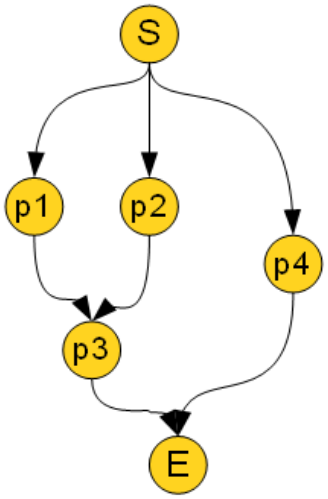
```
cobegin
   p1 || p2
coend
```

- Pokud mají entity probíhat sekvenčně (sériově, po sobě), lze použít konstrukce begin – end.
- Např. fakt, že procedura p2 se má spustit až po proceduře p1 lze znárodnit takto:

```
begin
   p1
   p2
end
```

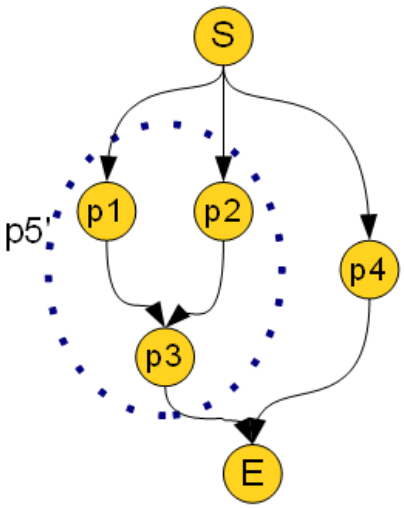
Takto lze pomocí pseudokódu zapsat, jak se budou provádět operace v programu a jejich závislost. Tuto závislost lze také znázornit pomocí tzv. precedenčního grafu.

Následující graf zobrazuje 4 procesy (p1 až p4). Proces p1 a p2 mohou běžet paralelně. Proces p3 může běžet až po ukončení p1 a p2. Proces p4 může běžet nezávisle na všech ostatních (tj. paralelně se všemi ostatními).



Pokud bychom chtěli posloupnost těchto procesů zapsat pomocí konstrukcí cobegin-coend a begin-end, postupovali bychom takto:

1. V celém grafu nalezneme největší části, které současně běží paralelně nebo sériově (v tomto případě `p4` běží paralelně s pseudoprocesem `p5'` (`p1`, `p2` a `p3`). (viz následující obrázek)

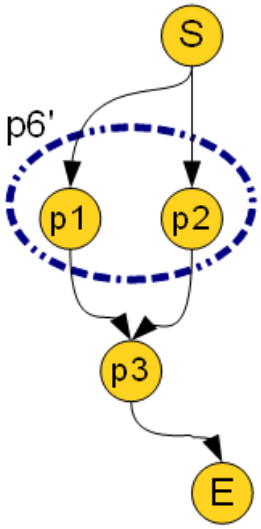


Pro tento graf můžeme napsat pseudokód:

```
cobegin
    p4 || p5'
coend
```

2. Dále budeme postupovat jako v bodě jedna, akorát nebudeme hledat v celém grafu, ale jen v podgrafu označeném jako `p5'`. Zde nalezneme sekvenčně běžící pseudoproces `p6'` (procesy `p1` a `p2`) a proces `p3` (viz následující graf).

d



Pro tento graf můžeme napsat pseudokód:

```
begin
   p6'
   p3
end
```

Spojíme-li kód z bodu 1 s tímto, získáme pseudokód:

```
cobegin
   p4 || begin
      p6'
      p3
   end
coend
```

3. Nyní nám zbývá již jen „zpracovat“ pseudoproces `p6'`. V pseudoprocesu `p6'` běží procesy `p1` a `p2` paralelně, můžeme tedy rovnou psát pseudokód:

```
cobegin
       p4 || begin
          cobegin
               p1 || p2
          coend
          p3
   end
coend
```

Nyní zkusíme obrácený postup. Z pseudokódu se pokusíme vygenerovat graf procesů. Mějme tedy pseudokód:

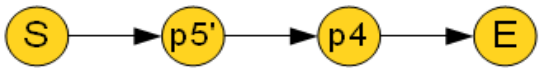
```
begin
    cobegin
        p1||begin
           p2
           p3
        end
    coend
    p4
end
```

V pseudokódu vidíme „na nejvyšší úrovni“ sériově běžící pseudoproces `p5'` (procesy `p1`, `p2` a `p3`) a proces `p4`. Pseudokód tedy můžeme přepsat do podoby:

```
begin

    p5'
    p4
end
```

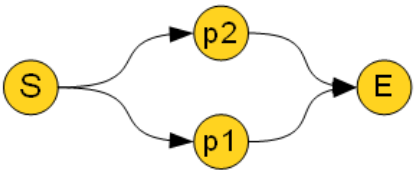
Graf procesů:



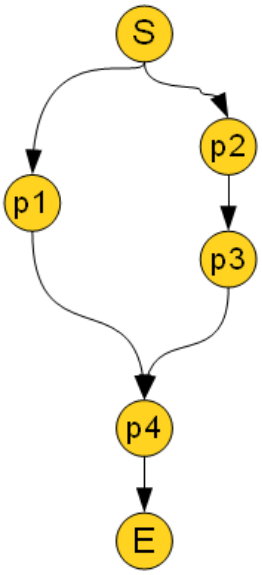
Dále budeme zkoumat pseudoproces `p5'`. V tomto pseudoprocesu běží paralelně proces `p1` a pseudoproces `p6'` (sekvenční `p2` a `p3`). Pseudokód pseudoprocesu `p5'` bude tedy vypadat takto:

```
cobegin
     p1||p6'
coend
```

Graf procesů:



A nakonec prozkoumáme pseudoproces `p6'`. V tomto pseudoprocesu fungují sekvenčně procesy `p2` a `p3`. Pseudokód a graf p6' je zřejmý, vytvoříme tedy rovnou celý graf:



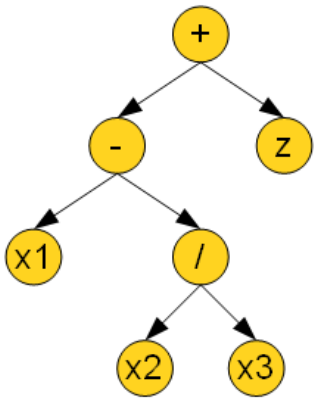
Př. Následující matematický zápis spočtěte s využitím maximálního paralelismu.

$$(a + b) - (c - d) / (e + f) + z$$

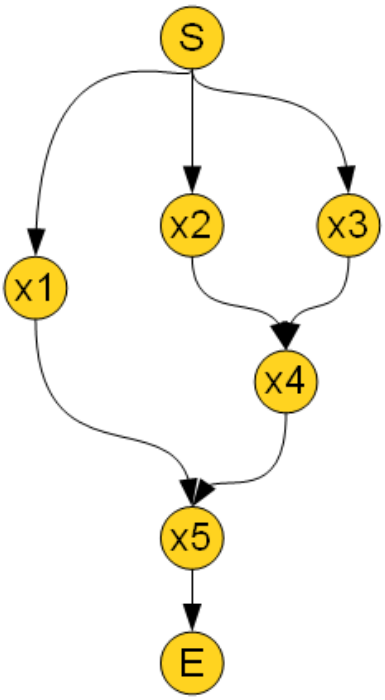
Nejprve si nahradíme obsah závorek jinými proměnnými (např. indexovaná proměnná x):

$$x1 - x2 / x3 + z$$

Nyní k tomuto výrazu sestrojíme derivační strom:



Výpočty, které probíhají na stejných „patrech“ tohoto stromu lze počítat paralelně. Můžeme tedy sestrojit precedenční graf tohoto výrazu (derivační strom ani nemusíme vždy sestrojit, občas postačí metoda „kouknu a vidím“): Pozn. x4 = x2/x3; x5 = x1 – x4 +z



Zápis v pseudokódu:

```
begin
    cobegin
        x1=a+b; ||
        begin
             cobegin
                  x2=c-d; || x3=e+f;
             coend
        end
    coend
    x5=x1-x4+z;
end
```