Párhuzamos folyamatok szintézise

Szerkesztette: *Szlávi Péter*

Előszó

Ez a módszertani anyag a párhuzamos program szintézisének egy lehetséges „mechanikus” módszerét mutatja be. Az első részben magát a módszer ismertetjük, röviden kitérve a legfontosabb szükséges párhuzamossággal kapcsolatos alapfogalmakra is. A második rész néhány konkrét feladatot dolgoz föl, amelyből pontosan kiderülnek a módszer „finomságai”, legapróbb részletei. Igyekeztünk az egyes feladatokat didaktikusan úgy építeni föl, hogy a „bonyodalmak” lépésről-lépésre növekedjenek, s így mindig éppencsak egy „újdonságra” kelljen figyelni. A következő részt az egyik feladat egy lehetséges implementációjának szenteltük, hogy lássuk –az elmélet után– milyen gyakorlati problémákkal találkozhatunk egy konkrét programozási nyelven történő megvalósítás közben..

Az anyag elméleti része *G.R. Endrews 'A Method for Solving Syncronization Problems'* cikke alapján készült. Az egyes párhuzamossággal kapcsolatos alapfogalmaknál utalunk olyan irodalmakra, amelyekben utánanézhet az Olvasó a pontos definícióknak és más kapcsolódó fogalmaknak. A feladatok *Varga László* professzor úr gyűjteményéből valók. A precíz kidolgozás is az Ő iránymutatása szerint történt. Köszönet illeti még *Kozma László* docens urat is, hogy kritikai észrevételeivel nagyban hozzájárult, hogy mentes legyen következetlenségektől és bosszantó elírásoktól. Erre annál is inkább nagy szükség volt, mert az eredeti cikktől bizonyos jelölésekben, értelmezési kérdésekben eltér a jelen anyag.

Mindezek ellenére maradhattak benne elírások, amelyekért elnézést kér az anyag

összeszerkesztője.

Tartalom

Párhuzamos programok szintézise

1. Bevezetés

Egy párhuzamos program az *több szekvenciális program*, azaz *folyamat* és megosztott, *közösen használt objektumok* együttese. Az objektumokon keresztül történik a folyamatok kommunikációja.

Program PárhuzamosProgramNév:

Változó *... közösen használt objektumok, kezdőértékükkel ...* {invariáns állítás}[1]

Folyamat FolyamatNév1(*... folyamat paraméterek...***):**
Ciklus amíg *... aktívság feltétele ...*
... tevékenységek ...
Ciklus vége
F olyamat vége.

Folyamat FolyamatNév2(*... folyamat paraméterek...***):**
Ciklus amíg *... aktívság feltétele ...*
... tevékenységek ...
Ciklus vége
Folyamat vége.

... további folyamatok, ill. azok által felhasznált (közös) eljárások és függvények ...

... inicializálási tevékenységek ...

Program vége.

A későbbiekben számunkra nem lesz fontos a *teljes* párhuzamos program leírása. Csak azt vizsgáljuk, hogy mi történik a folyamatok elindulását követően, hogyan kommunikálhatnak, ill. hogyan kommunikáljanak úgy, hogy a párhuzamosság hibás működést ne vigyen bele az egyébként, külön-külön helyesen működő folyamatok együttműködésébe. Vagyis a *szinkronizáció* megvalósítása érdekel bennünket. (Ez indokolja, hogy „szintaktikusan" nem lesznek teljesek a későbbi programjaink.)

A szinkronizáció két típusát használjuk föl sűrűn az alábbiakban: (1) a *kölcsönös kizárás*t[2] és (2) a *feltételes szinkronizáció*t[3]. Az (1) lényege, hogy egy ún. *kritikus szakasz*ban futó programok tevékenységei *nem átfedőek*. A (2)-é pedig, hogy egy *feltétel várakoztatja* a folyamatot a szinkronizáció érdekében.

2. A levezetési eljárás

Minden ilyen célú „mechanikus" átírással szembeni elvárás a *biztonságosság*, emellett praktikusan az „életképesség", azaz a józan ész egyszerűsítő észrevételeit is magában foglalja.

---

[1] Nemcsak az állításokat, hanem a megjegyzéseket is kapcsos zárójelek közé tesszük majd a programban hűen az algoritmikus nyelvi konvenciókhoz. [PB]

[2] *Kölcsönös kizárás.* A folyamatok által közösen használt erőforrások között vannak olyanok, amelyeket egyidőben csak egy folyamat használhat. Amíg valamely folyamat a végrehajtás olyan szakaszában van, melyben egy oszthatatlan erőforrással manipulál, addig kizárólag csak ő tartózkodhat ebben a szakaszban. [PA&Sz]

[3] *Szinkronizáció*. Az együttműködő folyamatok egymásnak üzeneteket, jelzéseket küldenek annak érdekében, hogy a címzettel tudassák, beléphet egy adott –a közös működés szempontjából problematikus– szakaszba. [PA&Sz]

Az eljárás első megközelítése:

Legyen a programszöveg a bizonyítást lehetővé tevő állításokkal megtűzdelve. Ha **S** egy atomi tevékenység, akkor egy rávonatkozó állítás az alábbi alakú lesz:

{P} S {Q}

Értelmezése: ha az **S** végrehajtása előtt a **P** predikátum igaz volt, és az **S** végrehajtása befejeződött, akkor a **Q** is tejesül. A **P**-t *előfeltétel*nek (Ef), a **Q**-t *utófeltétel*nek (Uf) nevezik. Ekkor –tehát– az **S** mint *predikátum-transzformátor* működik.

A párhuzamos program levezetése során két követelménynek kell teljesülnie:

- A folyamatok önállóan –mint szekvenciális programok– *helyes*ek legyenek.
- A folyamatok kapcsolata legyen *interferencia-mentes*, ami a következőt jelenti. Minden egyes $\mathbf{F_i}$ folyamat, amely a $\mathbf{\{P_i\}\ F_i\ \{Q_i\}}$ predikátumtranszformációt végzi el, és bármely másik folyamatbeli **S** atomi tevékenység esetén a $\mathbf{P_i}$ igaz marad az **S** végrehajtása során. Másképpen fogalmazva **S** *nem interferál* a $\mathbf{\{P_i\}\ F_i\ \{Q_i\}}$ tétellel, ha

  1. $\mathbf{\{P_i \wedge Ef(S)\}\ S\ \{P_i\}}$ és

  2. bármely $\mathbf{F_i'}$ részfolyamatára $\mathbf{F_i}$-nek, amely az $\mathbf{\{Ef(F_i')\}}$ halmazról képez le, akkor teljesül a $\mathbf{\{Ef(F_i') \wedge Ef(S)\}\ S\ \{Ef(F_i')\}}$ [4]

Tehát a párhuzamosság specialitása a bizonyítással szemben megkövetelné, hogy *minden* atomi tevékenység interferálását, *bármely* feltétellel, ellenőrizzük, de ez *polinomiális költségű* lenne. Mármost, jellemezzük a **ROSSZ** predikátummal az elkerülendő állapotokat, és a programban biztosítani tudjuk a **¬ROSSZ** állandóságát (invarianciáját) a megosztott változókra vonatkozólag, akkor az interferenciamentesség bizonyítását *lineáris költségű*vé tettük. Így tehát az elkövetkezőkben éppen ezt fogjuk tenni.

Az eljárás lépései

L1. *Problémadefiníció* – a folyamatok és szinkronizációjuk „azonosítása", megtervezése, vagyis a szükséges változók bevezetése az invariáns állítások leírásához ...

L2. *Megoldásvázlat* – a folyamatok „megtűzdelése" az osztott változók inicializálását elvégző értékadásokkal, hogy az invariancia –legalábbis– kezdetben biztosítva legyen ...

L3. *Az invariancia (további) biztosítása* – minden, az invarianciát érintő atomi értékadó utasítást „védünk" várakoztató feltételekkel, ha kell, hogy az utófeltétel teljesíthesse az invarianciát ...

L4. *Az atomi tevékenységek implementálása* – az atomi értékadások kódolása szekvenciális utasításokká ...

További előzetes megjegyzések:

M1. **Lf(S,Q)** jelölés – **S** *leggyengébb előfeltétele* **Q**-ra vonatkozólag az a feltétel, amelyből kiindulva **S** után **Q** igaz.

Pl. a) ha **S: x:=e {Q(x)}**, akkor **Lf(S , Q(x)) ≡ Q(e)**

b) ha **S: S1;S2 {Q}**, akkor **Lf(S1;S2 , Q) ≡ Lf(S1 , Lf(S2 , Q))**

[4] *Interferenciamentesség*. Egyszerűbben fogalmazva: **S** végrehajtása az $\mathbf{F_i}$ folyamat után nem változtatja meg $\mathbf{P_i}$=Igaz érvényességét, ill. az $\mathbf{F_i}$ folyamat bármely részfolyamata előtti végrehajtása az **S**-nek sem változtatja meg a $\mathbf{Ef(F_i')}$=Igaz érvényességét. [PA&Sz]

M2. **< S >** jelölés – **S** *oszthatatlanul* hajtható végre.

<várj amíg nem B majd S> jelölés egy „őrzött" tevékenységre utal, ami azt jelenti, hogy a **B** *igazzá válásáig*[5] legyen várakozás, majd **S** oszthatatlanul hajtódik végre. A várakozás egy *sor*ban[6] történik.

M3. Tegyük föl, hogy **K**, **L** predikátumok függetlenek más folyamatoktól, és igaz a **{K} <S> {L}** .

Állítás. Tegyük föl (továbbá), hogy **I** invariánsa **S**-nek, akkor a **<várj amíg nem B majd S>**-nek is invariánsa lesz. S így teljesülni fog az alábbi is: **K∧I∧B ≡ Lf(S,L∧I)** .

Így ez „receptül" is szolgálhat a **B** megvalósítására: **B** legyen olyan, hogy

- **S** *teminá*ljon, és
- a végállapot az **L∧I**-t kielégítse, továbbá
- a *leggyengébb* legyen az indokolatlan megvárakoztatás elkerülése érdekében.

3. Programozás szemaforokkal

Három fontos programozási technikát fogunk az alábbiakban bemutatni:

1. a *változók cseréje*,
2. a *bináris szemaforok hasítása* és
3. a *stafétabot továbbadása.*

Mindenek előtt gondoljuk meg, mik is azok a szemaforok és hogyan használhatók párhuzamos folyamatok szinkronizálására!

A. Szemafor[7]

A szemafor egy *absztrakt adattípus*, amelyhez két jellegzetes művelet tartozik: a **P** és a **V**[8]. Méghozzá azzal az elvárással, hogy az adott szemaforra vonatkozó befejezett **P**-k végrehajtási száma soha nem haladhatja meg a befejezett **V**-k számát. Így invariánsnak veendő a **V** és a **P** végrehajtási száma különbségének nem negativitása.

A **P** egy védett szakaszba *belépés* előtt hajtandó végre; a **V** *kilépéskor*. Józan ésszel is belátható, hogy kilépéskor nem kell „akadékoskodni", nem úgy, mint belépéskor, hiszen ekkor meg kell akadályozni, hogy túl sok folyamat kerüljék a védett szakaszban.

A szemafor szokásos definícióját és megvalósítását formalizáljuk az alábbiakban:

Értékhalmaz: **s**∈**Z** [9]

Asszociált műveletek[10]*:* **P,V**: **Z**→**Z** , **P(s)**: s:=s-1, **V(s)**: s:=s+1

---

[5] Itt eltérünk a cikk koncepciójától, ui. abban a **B** feltétel *igazzá válásáig* történik a várakozás.

[6] *Sor*. Az a típuskonstrukciós eszköz, amely strukturálisan valamely típus iteráltja, és funkcionálisan két jellegezetes művelettel definiálható: az elemsorozat két, ellentétes végével operáló *'Sorból'* és *'Sorba'* műveletekkel. [EST]

[7] L. PA&Sz

[8] A **P**-t és **V**-t magyarul hívhatnánk: **Várj**-nak, ill. **Jelezz**-nek.

[9] Valójában, mint típus: **Z**×**Sor**(Folyamat) értékhalmazzal rendelkezik.

[10] A későbbiek miatt célszerű lenne kiegészíteni a szemafor típusát még egy függvénnyel is, ami a szemafornál való várakozás tényét tesztelné: W:**Z**→{Igaz,Hamis}, W(**s**):=Igaz, ha az **s** szemafornál nem üres a várakozók sora, különben Hamis. Tradícionális okok miatt ezt az értelmezésbeli bővítést most nem tesszük meg.

Típusinvariáns: **SZEM(s) ≡ s≥0** [11].

Gondoljuk meg, hogy a típusinvariáns fenntartása érdekében, milyen feltétellel kell ellátnunk a **P**-t és a **V**-t:

1) **Lf(P(s), SZEM(s)) ≡Lf(s:=s-1, s≥0) ≡ s>0.** Így:

P(s): <várj amíg s≤0 majd s:=s-1>

(Vegyük észre, hogy eszerint a szemafor egy *sor*kezelő mechanizmust is feltételez. Erre később még visszatérünk.)

2) **Lf(V(s), SZEM(s)) ≡Lf(s:=s+1, s≥0) ≡ s+1≥0 ≡ s≥-1.**

Az nyilvánvaló, hogy **SZEM(s) ⇒ s≥-1**, ezért **V**-nél nincs szükség várakoztató feltételre:

V(s): <s:=s+1>

B. Bináris szemafor

A bináris szemafor is szemafor, de esetében a végrehajtott **V**-k száma legfeljebb eggyel haladhatja meg a **P**-k számát. Azért *bináris*, mert reprezentálásához elegendő a 0, 1 érték. Formálisabban:

Értékhalmaz: **b**∈{0,1}

Asszociált műveletek: **P,V**: {0,1}→{0,1}, **P(b)**: b:=b-1, **V(b)**: b:=b+1

Típusinvariáns: **BSZEM(b) ≡ b∈{0,1}**

A fentihez hasonló gondolatmenettel belátható, hogy

P(b): <várj amíg b≤0 majd b:=b-1>
V(b): <várj amíg b≥1 majd b:=b+1>

Ha garantálható legalább az, hogy a **P**-k és a **V**-k „párosával" hajtódnak végre, **P**-**V** sorrendben (ami egy-egy folyamat körültekintő meggondolásával elérhető), akkor a **V** őrfeltétele elhagyható[12]:

V(b): <b:=b+1>

3.1. Kritikus szakasz – változók cseréje

L1. Megadjuk a probléma definícióját.

Legyenek **F(1..N)** folyamatok, amelyek egy kritikus szakaszban közösen használnak egy erőforrást, a kritikus szakaszokon kívül csak lokális változókat használnak. (S mindezt szakadatlanul teszik.) A **bent(1..N)** a folyamatokhoz rendelt változó, amely 1, ha a megfelelő folyamat a kritikus szakaszban van, egyébként 0. Nyilvánvaló elvárás, hogy

KSz: bent(1)+...+bent(N)≤1

Itt formálisan nem jeleztük, hogy **bent(i)**∈{0,1}.

[11] Sőt precízebben: **SZEM(s) ≡ s.Állás≥0 ∧ (0<SorHossz(s.Várók)⇔0=s.Állás)**

[12] Sőt további egyszerűsítésekre is mód van bizonyos ellenőrzött körülmények között. Nevezetesen a **b:=b+1** helyett **b:=1**, ill. **b:=b-1** helyett **b:=0**.

L2. A megoldásvázlat és inicializálások.

Változó **bent: Tömb(1..N: Egész) (1..N: 0)** {a KSz triviálisan igaz}

Folyamat F(i: 1..N):
Ciklus
... a nem kritikus szakasz ...
{bent(i)=0} <bent(i):=1> {bent(i)=1}
... a kritikus szakasz ...
{bent(i)=1} <bent(i):=0> {bent(i)=0}
... a nem kritikus szakasz ...
Ciklus vége
Folyamat vége.

Lényeg tehát, hogy az i. folyamat az i. **bent**-elemet használja a kommunikációra, de a **bent** vektort egyszerre *csak egy* folyamat használhatja.

*L3. Mivel **KSz** kezdetben kielégül, úgy kell „gardírozni" az atomi tevékenységeket, hogy a **KSz** később is fennálljon, azaz invariáns lehessen. Ehhez vizsgáljuk:*

{bent(i)=0} <bent(i):=1> {bent(i)=1}.

M3 alapján [13]

$$Lf(\ bent(i):=1,\ bent(i)=1 \wedge bent(1) + ... + bent(i) + ... + bent(N) \le 1\ ) \equiv \quad (1)$$

$$\equiv 1=1 \wedge bent(1) +... + 1 +... + bent(N) \le 1 \equiv \quad (2)$$

$$\equiv bent(1) +... + bent(i-1) + 0 + bent(i+1) + ... + bent(N)=0$$

(1) ekvivalencia igaz az M1a) miatt, (2) igaz, mivel $\forall j$: $bent(j) \in \{0,1\}$, továbbá a *bent(i):=1* értékadás előtt *bent(i)=0* kellett teljesüljön; majd a reláció mindkét oldalából 1-t kivontunk. Tehát az őrző feltétellel kiegészített atomi tevékenység:

<várj amíg bent(1) +... + bent(N) ≠ 0 majd bent(i):=1>

Hasonlóan vizsgáljuk:

{bent(i)=1} <bent(i):=0> {bent(i)=0}.

M3 alapján

$$Lf(\ bent(i):=0,\ bent(i)=0 \wedge bent(1) + ... + bent(i) + ... + bent(N) \le 1\ ) \equiv \quad (1)$$

$$\equiv 0=0 \wedge bent(1) +... + 0 +... + bent(N) \le 1 \Leftarrow KSz \quad (2)$$

(1) ekvivalencia M1a) miatt igaz, (2) implikáció triviálisan igaz. Tehát az őrfeltétel nélkül is megfelelő az atomi tevékenység:

<bent(i):=0>

Mindezek után a megoldás precízebb változata:

Változó **bent: Tömb(1..N: Egész) (1..N: 0)** {a KSz triviálisan igaz}

[13] M3-beli jelölések: K = bent(i)=0, L = bent(i)=1, I = bent(1) + ... + bent(i) + ... + bent(N)≤1

Folyamat F(i: 1..N):
Ciklus
... a nem kritikus szakasz ...
{bent(i)=0 ∧ KSz} <várj amíg $\Sigma_{j=1..N}$ bent(j)≠0 majd bent(i):=1> {bent(i)=1 ∧ KSz}
... a kritikus szakasz ...
{bent(i)=1 ∧ KSz} <bent(i):=0> {bent(i)=0 ∧ KSz}
... a nem kritikus szakasz ...
Ciklus vége
Folyamat vége.

L4. Kódolás szemaforokkal.

Egy új változót vezetünk be, ami majd helyettesítheti a **bent(1..N)**-t.

Definíció: mutex[14] := 1 - $\Sigma_{j=1..N}$ bent(j).

Állítás: mutex *bináris szemafor*.

Ugyanis: KSz ⇒ mutex ∈ {0,1}

P,V –korábbi értelmezésével összhangban– definiálhatjuk így:

P(mutex) := <várj amíg mutex=0 majd mutex:=0; bent(i):=1> (1)

V(mutex) := < mutex:=1; bent(i):=0 > (2)

s ekkor a jelen helyzetben is mint bináris szemaforoperátorok funkcionálnak, hiszen:

0=mutex ⇔ 0 = 1 - $\Sigma_{j=1..N}$ bent(j) ⇔ $\Sigma_{j=1..N}$ bent(j)=1 ⇒ $\Sigma_{j=1..N}$ bent(j)≠0.

Azaz a '**mutex=0**' feltétel elvivalens a korábbi programban szereplő '**$\Sigma_{j=1..N}$ bent(j)≠0**' őrfeltétellel.

□

Mivel a **bent(1..N)**-nek megszünt a szerepe (ami abból látszik, hogy bár módosul, de értékére sehol sincs szükség), ezért bűntetlenül elhagyható (a **P(mutex)** és **V(mutex)**-ből is). A végleges megoldás tehát így fest:

Változó mutex: BinSzemafor (1)[15] {a KSz triviálisan igaz}

Folyamat F(i: 1..N):
Ciklus
... a nem kritikus szakasz ...
P(mutex)
... a kritikus szakasz ...
V(mutex)
... a nem kritikus szakasz ...
Ciklus vége
Folyamat vége.

Az itt bemutatott módszer **változók cseré**jével operált. Alkalmazásának feltételei:

V1. Szemantikusan különböző őrök *változók* diszjunkt halmazára vonatkozhatnak; a *változók* csak atomi utasításokban szerepelhetnek.

V2. Minden őrző „*kifejezés=0*" alakúvá tehető. (Most mutex=0.)

V3. Minden őrzött atomi utasítás csökkenti a *kifejezés* értékét a transzformált őrzőben. (Most mutex:=0.)

V4. Minden nem őrzött atomi utasítás növeli a *kifejezés* értékét a transzformált őrzőben. (Most mutex=1.)

---

[14] Az elnevezésről: ***mut***ual ***ex***clusion=kölcsönös kizárás

[15] Ez az a pont, ahol először tudatosodhat, hogy egy szemafor *kezdőértéké*nek szerepe éppen az, hogy megmondja: *hány* folyamat lehet az általa felügyelt kritikus szakaszban.

3.2. Termelő-fogyasztó probléma – bináris szemaforok hasítása

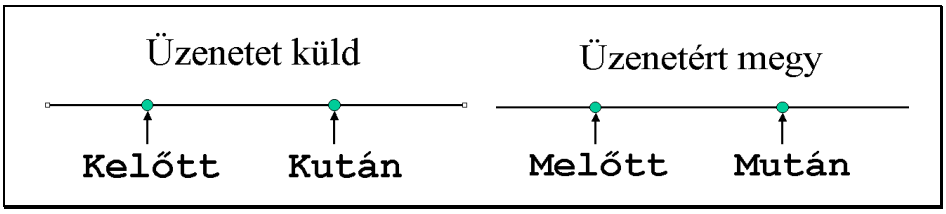
L1. Megadjuk a probléma definícióját.

A termelők üzenetet küldenek, hogy a fogyasztók azokat megkapják. A (termelő és fogyasztó) folyamatok egyetlen, megosztott pufferen keresztül kommunikálnak. A puffer minden időpillanatban legfeljebb egy üzenetet tartalmazhat. Mindehhez két művelet áll rendelkezésre: **Elküld** (Deposit), **Értemegy** (Fetch)[16]. A műveletek alkalmazásának vannak feltételei:

1. először az **Elküld** művelet hajtódjék végre;
2. felváltva következzenek az **Elküld** és **Értemegy** műveletek annak érdekében, hogy elkerülhető legyen két probléma: mielőtt a címzett értemenne az üzenetnek, más üzenet foglalja el helyét, ill. csak egyszer lehessen a kurrens üzenetért menni.

Megjegyezzük: a puffer (az „üzenettábla”) valójában egy *típuskonstrukció*, amely az üzenet típusából képezhető két művelet hozzávételével.

Jelöljük a termelő **Elküld** műveletének elkezdései számát *Kelőtt*-tel, befejezései számát *Kután*-nal, ill. a fogyasztó **Értemegy** műveletének elkezdései számát *Melőtt*-tel, a befejezései számát *Mután*-nal.



Mivel a puffer legfeljebb egy üzenetet fogadhat be, és csak létező üzenet vehető ki belőle, ezért állandóan teljesülnie kell, hogy

TF: Kelőtt≤Mután+1 ∧ Melőtt≤Kután

L2. A megoldásvázlat és inicializálások.

```
Változó     puf: T
            Kelőtt,Kután, Melőtt,Mután: Egész (0)          {TF igaz}

Folyamat  Termelő(i: 1..M):
    Változó
          üzenet: T
    Ciklus
(Ti1)       Üzenetgyártás(üzenet)
(Ti2)       Elküld(üzenet)
    Ciklus vége
Folyamat vége.

Folyamat  Fogyasztó(i: 1..N):
    Változó
          üzenet: T

    Ciklus
(Fi1)       Értemegy(üzenet)
(Fi2)       Üzenetetfeldolgoz(üzenet)
    Ciklus vége
Folyamat vége.

Eljárás  Elküld(üzenet: T):
    < Kelőtt:=Kelőtt+1 >; puf:=üzenet; < Kután:=Kután+1 >
Eljárás vége.
```

[16] Szinkronizáció üzenetátadással (mégpedig üzenőtáblával).

Eljárás Értemegy(üzenet: T):
< Melőtt:=Melőtt+1 >; üzenet:=puf; < Mután:=Mután+1 >
Eljárás vége.

(Állítások közbeiktatásától most el lehet tekinteni, mivel nyilvánvaló az interferenciamentesség.)

A későbbiekben manuálisan nyomon követjük a programot, ezért jelöltük meg az egyes utasításokat rövid szimbólumokkal: *(Ti1)..(Fi2)*.

*L3. Gondoskodni kell arról, hogy a **TF** állítás igaz maradhasson a működés közben is.* Ehhez vizsgálnunk kell az atomi tevékenységeket mind az **Elküld** eljárásban, mind az **Értemegy**-ben. Azaz:

< Kelőtt:=Kelőtt+1 >, < Kután:=Kután+1 > , ill.
< Melőtt:=Melőtt+1 >, < Mután:=Mután+1 >.

Az elsőhöz tartozó meggondolásaink a következők. M3 alapján

Lf(Kelőtt:=Kelőtt+1, Kelőtt≤Mután+1 ∧ Melőtt≤Kután) ≡ (1)

≡ Kelőtt+1≤Mután+1 ∧ Melőtt≤Kután (2)

≡ Kelőtt≤Mután

(1) ekvivalencia M1a) miatt igaz. A (2)-beli második tényezőnek nincs szerepe, hisz a transzformáció az igazságtartalmát nem változtatja meg, továbbá az invariancia miatt előtte is igaz volt. Így az őrző feltétellel kiegészített atomi tevékenység:

<várj amíg Kelőtt>Mután majd Kelőtt:=Kelőtt+1>

Hasonlóan az előzőhöz

Lf(Kután:=Kután+1, Kelőtt≤Mután+1 ∧ Melőtt≤Kután) ≡ (1)

≡ Kelőtt≤Mután+1 ∧ Melőtt≤Kután+1 (2)

≡ Igaz

(1) ekvivalencia M1a) miatt igaz, (2) TF igazságából következik. Így azután nincs szükség a második atomi tevékenység mellé őrfeltételt rendelni:

< Kután:=Kután+1 >

A másik kettő atomi utasítás esetén is mechanikusan járható ez az út. Vagyis az M3 alapján

Lf(Melőtt:=Melőtt+1, {Kelőtt≤Mután+1 ∧ Melőtt≤Kután}) ≡ (1)

≡ Kelőtt≤Mután+1 ∧ Melőtt+1≤Kután (2)

≡ Melőtt<Kután

(1) ekvivalencia M1a) miatt igaz. A (2)-beli első tényezőnek ez esetben sincs szerepe, hisz a transzformáció az igazságtartalmát nem változtatja meg, továbbá az invariancia miatt előtte is igaz volt. Így az őrző feltétellel kiegészített atomi tevékenység:

<várj amíg Melőtt≥Kután majd Melőtt:=Melőtt+1>

Másrészt

Lf(Mután:=Mután+1, Kelőtt≤Mután+1 ∧ Melőtt≤Kután) ≡ (1)

≡ Kelőtt≤Mután+2 ∧ Melőtt≤Kután+1 (2)

≡ Igaz

(1) ekvivalencia M1a) miatt igaz, (2) ismét a TF igazságából következik. Így azután nincs szükség a második atomi tevékenység mellé őrfeltételt rendelni:

< Mután:=Mután+1 >

Az eredményeket így foglalhatjuk össze:

Eljárás Elküld(üzenet: T):
(K1) **<várj amíg Kelőtt>Mután majd Kelőtt:=Kelőtt+1>;**
(K2) **puf:=üzenet;**
(K3) **< Kután:=Kután+1 >**
Eljárás vége.

Eljárás Értemegy(üzenet: T):
(M1) **<várj amíg Melőtt≥Kután majd Melőtt:=Melőtt+1>;**
(M2) **üzenet:=puf;**
(M3) **< Mután:=Mután+1 >**
Eljárás vége.

Mielőtt továbblépnénk „játszunk el" egy fiktív gondolatkísérlet néhány lépését a jobb megértés érdekében! Legyen 2 termelő (M=2) és 3 fogyasztó (N=3). Az események nyomonkövetésére talán legalkalmasabb az alábbi táblázat átnézése. A változók értékeit az időegység elején és végén is közöljük, így minden időegységhez két sor tartozik. A megváltozást megvastagítva kiemeltük, a folyamat állapotokat a fenti programban az egyes utasításikhoz rendelt szimbólumokkal jelezzük, s csak a megvaáltozásnál írjuk ki. Pl. ha az **F2** fogyasztó folyamat az *(F21)* **Értemegy** eljárás *(M1)* **<várj amíg Melőtt≥Kután majd Melőtt:=Melőtt+1>** utasítás végrehajtásánál tart, akkor ezt így rövidítjük: F21:M1.

Idő	**Elküld**		**Értemegy**		**Folyamatok állapota**				
	Kelőtt	**Kután**	**Melőtt**	**Mután**	**F1**	**F2**	**F3**	**T1**	**T2**
0	**0**	**0**	**0**	**0**	**F11:M1**	**F21:M1**	**F31:M1**	**T11**	**T21**
	0	0	0	0					

F1, F2, F3 megakad az Értemegy 1. atomi utasításán (M1).

1	0	0	0	0	F11:M1	F21:M1	F31:M1	T11:**K1**	T21:**K1**
	1	0	0	0					

T1, T2 közül (pl.) a T1 „győz" az Elküld 1. atomi utasításánál (K1), így T2 elakad itt (K1).

2	1	0	0	0				T11:**K2**	T21:K1
	1	0	0	0					
3	1	0	0	0				T11:**K3**	
	1	**1**	0	0					

F1, F2, F3 közül (pl.) F1 „győz", M1-t „befejezi", a többiek tovább várakoznak.

4	1	1	0	0	F11:**M1**			T11:**K1**	
	1	1	**1**	0					
5	1	1	1	0	F11:**M2**				
	1	1	1	0					
6	1	1	1	0	F11:**M3**				
	1	1	1	**1**					

T1, T2 közül (pl.) a T2 „győz", s így „befejezheti" K1-t, T1 várakozni kényszerül K1-nél.

7	1	1	1	1					T21:**K1**
	2	1	1	1					
8	2	1	1	1					T21:**K2**
	2	1	1	1					
9	2	1	1	1					T21:**K3**
	2	**2**	1	1					

F1, F2, F3 közül (pl.) a ... „győz” ...

10	...	...	...	...					

L4. Kódolás szemaforokkal.

Vegyük észre, hogy az alábbi két fogalom bevezetésével csökkenthető a szükséges változók száma:

Definíció: üres := Mután - Kelőtt + 1, ill. tele := Kután - Melőtt.

(Azaz az **üres**=1, ha a puffer üres, máskülönben 0; ill. a **tele**=1, ha a pufferben van üzenet, máskülönben 0.)

Állítás: az üres és a tele két *bináris szemafor*.

Ugyanis: TF $\Rightarrow$ üres, tele$\in\{0,1\}$

P,V –korábbi értelmezésével összhangban– definiálhatjuk így:

P(üres) := **<**várj amíg üres=0 majd üres:=0**>** (1)

V(üres) := **<** üres:=1 **>** (2)

P(tele) := **<**várj amíg tele=0 majd tele:=0**>** (3)

V(tele) := **<** tele:=1 **>** (4)

s ekkor a jelen helyzetben is mint bináris szemaforok funkcionálnak, hiszen P(üres) (1)-beli definíciója megfelelő, mivel triviális átalakításokkal megkapható a fenti őrző feltétel:

0=üres $\wedge$ 0= Mután - Kelőtt + 1 $\wedge$ Kelőtt > Mután (ez éppen az ‘Elküld’ őrzője.).

továbbá P(tele) (3)-beli definíciója is hasonlóan megfelelő:

0=tele $\wedge$ 0= Kután - Melőtt $\wedge$ Melőtt = Kután

(mivel a > eleve lehetetlen, így) $\wedge$ Melőtt $\geq$ Kután (ez éppen az ‘Értemegy’ őrzője.)

□

Mivel

- az (1)-beli *üres:=0* (az *üres* eggyel csökkentése) a *Kelőtt:=Kelőtt+1 (definíció szerint)*,
- a (2)-beli *üres:=1* (az *üres* eggyel növelése) *Mután:=Mután+1*,
- a (3)-beli *tele:=0* (a *tele* eggyel csökkentése) a *Melőtt:=Melőtt+1,*
- a (4)-beli *tele:=1* (az *tele* eggyel növelése) *Kután:=Kután+1* értékadással ekvivalens,

ezért elhagyva a fölöslegessé vált **Kelőtt**, **Kután**, **Melőtt** és **Mután** változókat, az alábbi végleges megoldást kapjuk:

```
Változó     puf: T
            üres, tele: BinSzemafor (1,0)      {TF-fel ekvivalens invariáns: üres+tele∈{0,1} igaz}

Folyamat    Termelő(i: 1..M):
   ... nem változik ...
Folyamat vége.

Folyamat    Fogyasztó(i: 1..N):
   ... nem változik ...
Folyamat vége.

Eljárás     Elküld(üzenet: T):
   P(üres); puf:=uzenet; V(tele)
Eljárás vége.
```

```
Eljárás      Értemegy(üzenet: T):
    P(tele); uzenet:=puf; V(üres)
Eljárás vége.
```

Belegondolva az eredményben semmi meglepő nincsen. Egyetlen szemafor nem lehet elegendő (amint esetleg első „villanásra” az ember érezné), hisz amikor az üzenet elküldésbe belefog az aktuális folyamat, az üzenet tényleges megérkezéséig nem igaz sem az, hogy a puffer üres lenne (s így másik folyamat is üzenetküldésbe foghatna), sem az, hogy tele van (s így más folyamat folyamodhatna tartalmáért). Kettő szemafor viszont éppen elegendő…

Összegezve a fentieket egy fontos általános gondolat fogalmazható meg a **hasított bináris szemafor**okról. Tudniillik: **b** hasított bináris szemaforként használható, ha $\mathbf{b}=\mathbf{b_1,...,b_N}$: BinSzemaforok (most: üres, tele), továbbá teljesíti a **HASÍT(b):** $\Sigma_{\mathbf{i=1..N}}\ \mathbf{b_i}\in\{0,1\}$ feltételt.

3.3. Egy osztott adatbázis olvasási és írási m űveletei – a stafétabot továbbadása

L1. A probléma:

Egy adatbázison osztozik két típusú folyamat. Az *Olvasó*k olvashatnak belőle, akár egyszerre többen is, az *Író*k írhatják, de ők csak „egymagukban”. Ezt az elvárást fejezi ki az alábbi állítás:

$$\mathbf{ÍO:\ (oDb=0 \vee íDb=0) \wedge íDb\le 1}$$

ahol az **oDb** az olvasó(folyamato)k számát, az **íDb** az írók számát jelöli.

L2. A megoldásvázlat és inicializálások.

```
Változó      oDb,íDb: Egész (0)                                  {ÍO igaz}

Folyamat  Olvasó(i: 1..M):
        Ciklus
            < oDb:=oDb+1 >;  Olvas;  < oDb:=oDb-1 >
        Ciklus vége
Folyamat vége.

Folyamat  Író(i: 1..N):
        Ciklus
            < íDb:=íDb+1 >;  Ír;  < íDb:=íDb-1 >
        Ciklus vége
Folyamat vége.
```

*L3. Gondoskodni kell arról, hogy az **ÍO** állítás igaz maradhasson* (volt is és legyen is) *a működés közben is.* Ehhez vizsgálnunk kell az atomi tevékenységeket. Azaz:

$$\mathbf{< oDb:=oDb+1 >}\ ,\ \mathbf{< oDb:=oDb-1 >}\ \text{, ill.}$$
$$\mathbf{< íDb:=íDb+1 >}\ ,\ \mathbf{< íDb:=íDb-1 >}.$$

Az elsőhöz tartozó meggondolásaink a következők. M3 alapján

$\mathrm{Lf}(\ oDb:=oDb+1,\ (oDb=0 \vee íDb=0) \wedge íDb\le 1\ ) \equiv$ (1)

$\equiv (oDb+1=0 \vee íDb=0) \wedge íDb\le 1 \equiv$ (2)

$\equiv (oDb=-1 \wedge íDb£1) \vee (íDb=0 \wedge íDb\le 1) \equiv$ (3)

$\equiv íDb=0$

(1) ekvivalencia M1a) miatt igaz, (2) azonos átalakítás és a disztributivitás miatt igaz, (3) után az 1. tag az *oDb<0* lehetetlensége miatt azonosan hamis, így elhagyható; a 2. tag egyszerűsíthető a nyilvánvaló implikáció miatt. Így kapjuk, hogy

```
<várj amíg íDb≠0 majd oDb:=oDb+1>
```

Másrészt

Lf(oDb:=oDb-1, (oDb=0 ∨ íDb=0) ∧ íDb≤1) ≡ (1)

≡ (oDb-1=0 ∨ íDb=0) ∧ íDb≤1 ≡ (2)

≡ (oDb=1 ∧ íDb≤1) ∨ (íDb=0 ∧ íDb≤1) ≡ IGAZ (3)

(1) ekvivalencia M1a) miatt igaz, (2) azonos átalakítás és a disztributivitás miatt igaz, (3) igaz, mivel *oDb>0 ∧ ÍO* igaz volt az atomi utasítás előtt (l. L2 programban!). Így kapjuk, hogy

< oDb:=oDb-1 >

Az előbbiekhez hasonlóan adódik, M3 alapján

Lf(íDb:=íDb+1, (oDb=0 ∨ íDb=0) ∧ íDb≤1) ≡ (1)

≡ (oDb=0 ∨ íDb+1=0) ∧ íDb+1≤1 ≡ (2)

≡ (oDb=0 ∨ íDb=-1) ∧ íDb≤0 ≡ (3)

≡ oDb=0 ∧ íDb=0

(1) ekvivalencia M1a) miatt igaz, (2) azonos átalakítás miatt igaz; (3) után az 1. tényező az *íDb≥0* miatt redukálható, ugyanezen indokkal alakítható a 2. tényező. Így kapjuk, hogy

<várj amíg oDb≠0 vagy íDb≠0 majd íDb:=íDb+1>

Az <oDb:=oDb-1>-re vonatkozó meggondolásokkal haladva

Lf(íDb:=íDb-1, (oDb=0 ∨ íDb=0) ∧ íDb≤1) ≡ (1)

≡ (oDb=0 ∨ íDb-1=0) ∧ íDb-1≤1 ≡ (2)

≡ (oDb=0 ∨ íDb=1) ∧ íDb≤2 ≡ IGAZ (3)

(1) ekvivalencia M1a) miatt igaz, (2) azonos átalakítás miatt igaz, (3) igaz, mivel *íDb>0 ∧ ÍO* igaz volt az atomi utasítás előtt (l. L2 programban!). Így kapjuk, hogy

< íDb:=íDb-1 >

Az eredményeket így foglalhatjuk össze:

```
Változó      oDb,íDb: Egész (0)                                  {ÍO igaz}

Folyamat  Olvasó(i: 1..M):
        Ciklus
              <várj amíg íDb≠0 majd oDb:=oDb+1 >
              Olvas
              < oDb:=oDb-1 >
        Ciklus vége
Folyamat vége.

Folyamat  Író(i: 1..N):
        Ciklus
              <várj amíg oDb≠0 vagy íDb≠0 majd íDb:=íDb+1 >
              Ír
              < íDb:=íDb-1 >
        Ciklus vége
Folyamat vége.
```

L4. Kódolás.

Itt a két őrt álló feltételnek van közös része, ezért a *változók cseréje* módszer nem használható az atomi utasítások implementálására. (Nem teljesül a V1. feltétel.) Ennek az a magyarázata, hogy *egyetlen*

szemafor nem képes elválasztani a két szemantikailag különböző őrző feltételt. Új módszer kell tehát: az ún. *stafétabot továbbadása*.

Ennek lényege röviden a következő. Kétféle atomi utasítás van:

F1: < S_i >, illetve **F2: <várj amíg nem B_j majd S_j>**

Ezek implementálhatók bináris szemaforok hasításával. Mégpedig így:

1. Legyen **e** *bináris* szemafor 1 kezdőértékkel. Ez kontrollálja egy (tetszőleges) atomi utasításba való belépést.
2. Legyen $\mathbf{c_j}$ *bináris* szemafor, $\mathbf{d_j}$ számláló[17] a várakozók számát adminisztrálandó (kezdetben 0 értékkel), minden egyes szemantikusan eltérő őrző feltételhez.

F1-nek megfeleltetjük az

```
F1:   P(e)
      {I}
      Si
      {I}
      SIGNAL
```

F2-nek pedig az

```
F2:   P(e)
      {I}
      Ha nem Bj akkor dj:=dj+1; V(e); P(cj);   ← itt adja át a stafétabotot mielőtt sorba állna
      {I∧Bj}
      Sj
      SIGNAL
```

ahol az *I* a szinkronizációs invariánst, és a *SIGNAL* a következő programdarabot jelöli:

```
SIGNAL:  Elágazás
           B1 és d1>0  esetén  {I∧B1} d1:=d1-1; V(c1)     ← itt aktivizál egy várakozót
           . . .
           BN és dN>0 esetén  {I∧BN} dN:=dN-1; V(cN)     ← itt aktivizál egy várakozót
             egyéb    esetben {I} V(e)                   ← itt aktivizál egy várakozót
         Elágazás vége
```

Néhány megjegyzést kell fűznünk a e programdarabokhoz:

1. Az F2-beli „… V(e); P(c_j) …” nél garantáltan elakad B_j teljesüléséig, mivel 0 kezdőértékkel indult a c_j szemafor.
2. Ha létezne a W függvény, akkor F2 egyszerűsödne (a kékkel jelölt utasítás elhagyásával), hisz nem muszáj külön adminisztrálnunk a darabszámot:

```
F2:   P(e)
      {I}
      Ha nem Bj akkor V(e); P(cj);
      {I∧Bj}
      Sj
      SIGNAL
```

3. A *SIGNAL* betét is rövidebben lenne megfogalmazható:

[17] Visszautalunk egy korábbi megjegyzésünkre, amelyben célszerűnek állítottunk egy **W** függvényt. Ez adna választ arra, hogy a szemafornál van-e várakozó vagy sem. Ilyen **W** birtokában nem lenne szükség a $\mathbf{d_j}$ változókra.

```
SIGNAL: Elágazás
          B1 és W(c1) esetén  {I∧B1} V(c1)
          . . .
          BN és W(cN) esetén  {I∧BN} V(cN)
            egyéb esetben     {I} V(e)
        Elágazás vége
```

4. A *SIGNAL*-beli elágazás egy *nem determinisztikusan választó alternatív szerkezet*, amely az igaz-értékű ágak közül választ egyet nem determinisztikusan. Természetesen az *egyéb* feltétel csak rövidíti a „megelőző” feltételek nem teljesülését megfogalmazó feltételt.

5. A feltételek első része (B_j) akkor igaz, ha az S_j végrehajtható, míg a második tényezője ($d_j>0$) akkor, ha van az adott atomi tevékenységre várakozó.

6. Az elágazó ágak utolsó dolguk, hogy átadják (egyszerre persze csak egyikük) a stafétabotot azzal, hogy a $V(c_j)$-vel szabadra állítják a saját szemaforukat.

Vegyük észre az *e bemeneti szemafor* szerepét! Az *e* megakadályozza, hogy több párhuzamos folyamat több (természetesen különböző) szemafort párhuzamosan kezeljen. Ez azért okozna bajt, mert a szemaforok „összekuszálódása” olyan folyamatokat is útnak indíthatnának, amelyek valójában egymást ki kellene zárniuk. (Emlékezzünk, korábban épp ilyen probléma akadályozta a *változók cseréje* egyszerű módszerének alkalmazását[18])

Jelen esetben a szemaforok egy *hasított bináris szemafor*t alkotnak, mivel egyidejüleg közülük legfeljebb egy lehet 1-értékű, és minden végrehajtási út egy *P*-vel kezdődik és egy *V*-vel végződik. Vagyis a *P* és *V* közti utasítások *kölcsönösen kizárva* hajtódhatnak csak végre. Az *I szinkronizációs invariáns* igaz lesz minden *V* művelet előtt, ugyancsak igaz valahányszor a szemaforok egyike 1-értékű. Továbbá B_j garantáltan teljesül, amikor S_j végrehajtódik, hisz ha B_j nem lenne igaz, akkor c_j szemafornál kényszerül a folyamat várakozni B_j igazzá válásáig. Végezetül a transzformáció nem tartalmaz *holtpont*ot, hiszen c_j jelzi, hogy valamely folyamat várakozik vagy várakozni készül a c_j-nél.

Ezt a sémát alkalmazzuk az „írók-olvasók” probléma megoldására! Vezessük be a következő jelöléseket:

- o – szemafor az *(iDb=0)* feltételre váró olvasóhoz,
- í – szemafor az *(oDb=0 és íDb=0)* feltételre váró íróhoz,
- b – belépési szemafor,
- dO – számláló az *(iDb=0)* feltételre váró olvasóhoz,
- dÍ – számláló az *(oDb=0 és íDb=0)* feltételre váró íróhoz,

A megoldás tehát:

```
Változó   oDb,íDb: Egész (0)                          {ÍO igaz}
          o,í: BinSzemafor(0)
          b: BinSzemafor(1)                           {o+í+b∈{0,1}}
          dO,dÍ: Egész(0)                             {dO≥0 és dÍ≥0}

Folyamat Olvasó(i: 1..M):
     Ciklus
          P(b)
          Ha íDb≠0 akkor dO:=dO+1; V(b); P(o)
          oDb:=oDb+1;
          SIGNAL1
          Olvas
```

[18] L. a *változók cseréje* módszer V1. feltételét.

```
            P(b)
            oDb:=oDb-1
            SIGNAL2
        Ciklus vége
Folyamat vége.
Folyamat  Író(i: 1..N):
        Ciklus
            P(b)
            Ha oDb≠0 vagy íDb≠0 akkor dÍ:=dÍ+1; V(b); P(í)  ← ha itt elakad, akkor csak ott
            íDb:=íDb+1
            SIGNAL3
            Ír
            P(b)
            íDb:=íDb-1
            SIGNAL4
        Ciklus vége
Folyamat vége.
SIGNALi: Elágazás
            íDb=0 és dO>0          esetén   dO:=dO-1; V(o)
            íDb=0 és oDb=0 és dÍ>0 esetén   dÍ:=dÍ-1; V(í)   ← ott: indítható tovább
                  egyéb            esetben  V(b)
            Elágazás vége
```

Megjegyzések:

1. Az atomi utasításokat helyettesítő részeket dőlten szedve emeltük ki.

2. Az *egyéb eset*et a későbbiek miatt részletezve is megadjuk:

$$(\text{íDb}\neq 0 \text{ vagy } \text{dO}=0) \text{ és } (\text{íDb}\neq 0 \text{ vagy } \text{oDb}>0 \text{ vagy } \text{dÍ}=0)$$

Többnyire –mint jelen esetben is– a *SIGNAL* rész redundáns, tartalmaz egyszerűsíthető, elhagyható részeket. A fenti algoritmusban szereplő négy *SIGNAL*-t vesszük sorra, figyelembe véve azt a helyet, ahol ő valójában található.

A $SIGNAL_1$ előtt biztosan igaz, hogy *(íDb=0 és oDb=0)*, tehát az elágazás első feltétele egyszerűen *dO>0*; a második ág hamis, tehát elhagyható; a harmadik ág feltételéből mindössze *dO=0* marad. Azaz

```
            Elágazás
              dO>0      esetén dO:=dO-1; V(o)
              dO=0      esetén V(b)
            Elágazás vége
```

A $SIGNAL_2$ előtt tudjuk, hogy *(íDb=0 és dO=0)* igaz, hisz nincs író folyamat, ami sorba kényszerített volna akár egyetlen olvasót is. Így az első feltétel hamis, elhagyható; a második ág feltétele egyszerűbb: *(dÍ>0 és oDb=0)*; a harmadik ág *(oDb>0 vagy dÍ=0)* marad:

```
            Elágazás
              oDb=0 és dÍ>0     esetén dÍ:=dÍ-1; V(í)
              oDb>0 vagy dÍ=0   esetén  V(b)
            Elágazás vége
```

Hasonlóan kell végigjárni a maradék két *SIGNAL* részletet is. Miután a végeredmény így alakul:

```
Változó     oDb,íDb: Egész (0)                                  {ÍO igaz}
            o,í: BinSzemafor(0)
            b: BinSzemafor(1)                   {o+í+b∈{0,1}, ui. BinSzem hasítás}
            dO,dÍ: Egész(0)                                     {dO≥0 és dÍ≥0}
```

```
Folyamat  Olvasó(i: 1..M):
        Ciklus
                P(b)
                Ha íDb≠0 akkor dO:=dO+1; V(b); P(o)
                oDb:=oDb+1;
                Elágazás
                  dO>0  esetén dO:=dO-1; V(o)
                  dO=0  esetén V(b)
                Elágazás vége

                Olvas

                P(b)
                oDb:=oDb-1
                Elágazás
                  oDb=0 és dÍ>0     esetén  dÍ:=dÍ-1; V(i)
                  oDb>0 vagy dÍ=0 esetén  V(b)
                Elágazás vége
        Ciklus vége
Folyamat vége.

Folyamat  Író(i: 1..N):
        Ciklus
                P(b)
                Ha oDb≠0 vagy íDb≠0 akkor dÍ:=dÍ+1; V(b); P(í)
                íDb:=íDb+1
                V(b)

                Ír

                P(b)
                íDb:=íDb-1
                Elágazás
                    dO>0            esetén  dO:=dO-1; V(o)
                    dÍ>0            esetén  dÍ:=dÍ-1; V(í)
                  dO=0 és dÍ=0    esetén  V(b)
                Elágazás vége
        Ciklus vége
Folyamat vége.
```

Példafeladatok: SzinkronPl.rtf/pdf .