PÁRHUZAMUS PROGRAMOK

Párhuzamos program fogalma

- Egy párhuzamos programon olyan folyamatok együttesét értjük, amelyek közös használatú erőforrásokat használhatnak, azaz egymással kommunikálhatnak
  - Folyamat: végrehajtás alatt álló program.
  - Közös használatú erőforrások lehetnek közös adatok (memóriaváltozó vagy akár egy adatbázis), vagy kommunikációs csatornák
- Működése ütemezhető egyetlen processzorra, de lehetőség nyílik több processzor esetén egyrészt egy program végrehajtási idejének csökkentésére, másrészt a processzorok hatékonyabb kihasználására.

Párhozamos programok kategóriái

- Időzítés szerint:
  - szinkron – részben szinkron – aszinkron
- Kommunikáció fajtája szerint:
  - diszjunkt – közös erőforrás – üzenetváltásos
- Hibakezelési szint szerint:
  - hibátlan hardver feltételezés
  - processzor hibák kezelése (megállási hiba, bizánci hiba),
  - kommunikációs hiba (üzenet elvesztése vagy többszöri elküldése) figyelem kívül hagyása vagy kezelése.
- Megoldandó problémák köre szerint

Párhuzamossággal kapcsolatos fogalmak

- Atomi (megszakíthatatlan) művelet
- Sorba rendezhetőség (Összefésülés)
- Nem-determinisztikusság
- Folyamatok sebessége
- Pártatlan (fair) ütemezés

- Közös változók, üzenetek
- Interferencia
- Kritikus szakasz
- Kölcsönös kizárás
- Holtpont
- Kiéheztetés

Párhuzamos programok modelljei

- Különféle modellek:
  - Multiprogramozott rendszerek
  - Állapot-átmenet automaták csoportja
  - Feltételes értékadások halmaza
- Most a modell megválasztásának szempontja a helyesség ellenőrzés. De mit kell ellenőrizni?
  - Parciális helyesség. Befejeződés. Teljes helyesség.
  - Hibamentesség. Interferencia mentesség. Holtpontmentesség. Kiéheztetés elkerülése.

Párhuzamos blokk

- *Szintaxis:*

  parbegin $S_1 || \ldots || S_n$ **parend**

  ahol $S_1, \ldots, S_n$ tetszőleges programok párhuzamosan hajtódnak végre és a blokk akkor ér véget, ha mindegyik S_i $(i=1...n)$ véget ért.
- *Szemantika:*

  $\forall i \in [1..n]$: Ha $\langle \sigma, S_i \rangle \rightarrow \langle \sigma_i, T_i \rangle$ akkor

  $\langle \sigma$, **parbegin** $S_1 \| \ldots \| S_i \| \ldots \| S_n$ **parend**$\rangle \rightarrow$

  $\langle \sigma_i$, **parbegin** $S_1 \| \ldots \| T_i \| \ldots \| S_n$ **parend**$\rangle$
- *Más jelölések:*

  cobegin $S_1 || \ldots || S_n$ **coend**

  $[S_1 || \ldots || S_n]$

Végrehajtási sebesség

- Minden be nem fejeződött folyamat pozitív végrehajtási sebességgel bír, feltéve, hogy nem egy várakozó utasításnál ($\alpha \rightarrow S$) várakozik.
  - A várakozó utasítás egy őrfeltétellel rendelkező utasítás, amelyik addig nem hajtódhat végre, amíg az őrfeltétel hamis. Ez a párhuzamos folyamatok szinkronizálásának eszköze. Segítségével lehet kikényszeríteni azt, hogy egy folyamat végrehajtása csak akkor folytatódjon, ha az őrfeltétel (amit más folyamatok tesznek igazzá) teljesül. Holtpont helyzetet okozhat.

Pártatlan ütemezés

- Semmilyen feltételezésünk sincs a komponensek egymáshoz viszonyított sebességéről, de feltételezzük, hogy pártatlanul ütemezve hajtódnak végre, azaz ha egy művelet végrehajtása végtelen sokszor megtörténhet, akkor végre is fog hajtódni.

Sorba rendezés

- A párhuzamos program bármelyik végrehajtása (a tényleges processzorszámtól függetlenül) megkapható az atomi (nem megszakítható) műveleteinek sorba rendezésének (összefésülésének) végrehajtásaként.
- Mivel azonban a sorba rendezés nem egyértelmű, a párhuzamos programok szükségképpen nem-determinisztikusak.

Multiprogramozott végrehajtás

- Tekintsük az alábbi párhuzamos programot:

```
parbegin a := a+1 || a:= a-1 parend
```

- Ennek az úgynevezett multiprogramozott végrehajtása – amikor az elemi utasításokat egyetlen processzorra ütemezzük úgy, hogy az értékadásokat nem engedjük megszakítani – két végrehajtási sorozatot eredményez a kétféle sorba rendezhetőség miatt.
- Amennyiben a kezdőállapot az $a=0$, akkor a multiprogramozott végrehajtások garantáltan az $a=0$ végállapotba vezetnek.

Valódi párhuzamos végrehajtás

```
parbegin a := a+1 || a:= a-1 parend
```

- A valódi párhuzamos végrehajtás során külön lépés egy változó értékének lekérdezése, külön a visszatöltése, és külön egy érték megváltoztatása, és így húsz különböző végrehajtást jöhet létre.

```
parbegin
   load a; add 1; store a;
||
   load a; add -1; store a;
parend
```

- Amennyiben a kezdőállapot az $a=0$, akkor a végrehajtások eredménye lehet: $a=-1, a=0, a=1$.

Manna-Puneli elv

- Alakítsuk át az egy utasításban egy kritikus hivatkozás elv betartásával az előző programot (legyenek r1 és r2 az egyes folyamatok saját változói, amik nem közös változók):

```
parbegin
   r1 := a; a := r1+1 ;
||
   r2 := a; a := r2-1 ;
parend
```

- Minden programhoz megadható egy vele ekvivalens, az egy utasításban egy kritikus hivatkozás elvét betartó program.
- Ez utóbbi multiprogramozott végrehajtásai „megegyeznek” az eredeti program a valódi párhuzamos végrehajtásaival.

Helyesség és a sorba rendezés

- Néha (pl: cseréljük ki az x és y változók értékeit) az értékadások atomi voltának feltételezése csupa nem kívánt végrehajtást eredményez, azaz az alábbi program nem oldja meg a feladatot.

  parbegin x := y || y := x **parend**

- De a nem kívánt végrehajtásokat sem e programnak az egy kritikus hivatkozást betartó átírása,

  parbegin r1 := y; x := r1; || r2 := x; y := r2; **parend**

  (r1 és a r2 nem közös változók.)

- sem a valódi párhuzamos végrehajtása nem tudja kizárni.

Megjegyzés

- Általában egy tetszőleges párhuzamos program multiprogramozott végrehajtásai nem azonosak a valódi párhuzamos végrehajtásokkal.
- Ha azonban az értékadásokat és a logikai feltételek kiértékelését atomi (azaz nem megszakítható) műveleteknek tekintjük, akkor igen.
- Ez a kitétel nem olyan erős korlátozás, mert ha ugyanis lényeges, hogy egy értékadás oszthatatlanul hajtódjék végre (mert például közös változót használ), akkor ezt szinkronizálással biztosítani lehet.

Helyesség és szinkronizáció

- A változó cserés probléma egy helyes megoldása szinkronizációval.

```
l1 := false; l2 := false;
parbegin
    r1 := y;
    l1 := true;
    await l2;
    x := r1;
||
    r2 := x;
    l2 :=true;
    await l1;
    y := r2 ;
parend
```


Atomi műveletek

- Az atomi (primitív) művelet egy nem megszakítható lépés, amely blokkol minden más folyamatot. Jele: <u>, ahol u egy művelet.
- Vannak ún. hardver által támogatott atomi műveletek (logikai feltétel kiértékelése, **skip** végrehajtása, változó lekérdezése ill. megváltoztatása), de kijelölhetünk atominak explicit módon is egy programszakaszt (több egymást követő utasítást) a várakozó (vagy szinkronizációs) utasítás segítségével.

Várakozó utasítás

- *Szintaxis:*

  await α **then** S **ta**

  amely atomi műveletként hajtódik végre és az S nem tartalmazhat ciklust, sem várakozó utasítást.

- *Szemantika:*
  - Ha σ állapotban teljesül az α feltétel és <σ , S> → <σ' , S'> akkor <σ , **await** α **then** S **ta**> → <σ' , S'>
  - Ha σ állapotban nem teljesül az α feltétel akkor <σ , **await** α **then** S **ta**> → <σ , **await** α **then** S **ta**>

Megjegyzés

- A várakozó utasítás definíció szerint atomi művelet.
  - <α→ S> ~ **await** α **then** S **ta**
- Speciális várakozó utasítás.
  - <α→**skip**> ~ **await** α **then** SKIP **ta**
    ~ **await** α
- A várakozó utasítás segítségével egy utasítás sorozatot is atomivá tehetünk, csak körültekintően járjunk el (pl. nem szerencsés egy ciklust atomivá tenni).
  - <S> ~ **await** true **then** S **ta**

Kröger modellje

- Minden program
  $\Pi \equiv$ **initial** Pre;
  parbegin $\Pi_1 \| \ldots \| \Pi_p$ **parend**
  alakú, ahol
  - Pre (kezdeti feltétel) egy predikátum kalkulusbeli formula
  - p≥1 természetes szám
  - minden $1 \le i \le p$-re Π_i vagy egy <cikl. párh. komp.>, vagy egy <acikl. párh. komp.>
  - < cikl. párh. komp.> ::= **loop** <ut. sor.> **end**
  - <acikl. párh. komp.>::= <ut. sor.>; <címke> : **stop**
  - <ut. sor.> ::=<utasítás> | <utasítás>; <ut. sor.>
  - <utasítás>::=<címke> : <címkézetlen utasítás>

- <címkézetlen utasítás>::=<$\mathcal{A}$ eleme> |
  if <feltétel> **then** <ut.sor.> **fi** |
  if <feltétel> **then** <ut.sor.> **else** <ut.sor.> **fi** |
  while <feltétel> **do** <ut.sor.> **od**
- <feltétel> ::= < $\mathcal{L}_p$ formulája> $\mathcal{L}_p$ elsőrendű logika nyelve

- Alap utasítások: $\mathcal{A} = \mathcal{E} \cup \mathcal{S}$ ahol
  - $\mathcal{E}$ az elemi utasítások halmaza
    (Nincs előre rögzítve, de később megállapodás szerint ide tartozónak tekintik az értékadásokat)
  - $\mathcal{S}$ = { **await** <feltétel> **then** a **ta** | a∈$\mathcal{E}$ } ∪
    { **await** <feltétel> }

Címkék

- Szigorú címkézési konvenciót vezet be a definíció. Például külön van címkéje egy ciklusnak és külön a ciklus első utasításának.
- Nincs két ugyanolyan címkéjű utasítás a Π programban
- Jelölések:
  - $\mathcal{M}_\Pi$ jelöli a Π program összes címkéjét
  - $\mathcal{M}_{\Pi_i}$ jelöli a Π_i program összes címkéjét
  - $\alpha_0^{(i)}$ jelöli a Π_i program kezdő címkéjét
  - $\alpha_e^{(i)}$ jelöli a Π_i program kilépési pontjának címkéjét

Atomi műveletek

- Alap utasítások: $\mathcal{A}$
- Elágazás (**if**) utasítás feltételének kiértékelése az elágazás megfelelő ágába történő lépéssel együtt.
- Ciklus (**while**) utasítás feltételének kiértékelése a ciklus magba illetve a ciklus utáni utasításra való lépéssel együtt.

Sorba rendezés

- Egy párhozamos program végrehajtása diszkrét lépésekben, az alábbi összefésüléssel történik.
  - Egy ütemező nyilván tarja minden párhuzamos komponens soron következő címkézett utasítását, és kiválasztja ezek közül azt, amelyiket végre kell hajtani.
  - Az ütemező mindig egy atomi műveletet hajt végre, amely után a soron következő címkézett utasításhoz lép.
  - Kivétel a várakozó (**await**) utasítás, amelynek végrehajtása után csak akkor lép az ütemező a következő utasításhoz (és várakozó utasítás esetleges **then** ág is csak ebben az esetben hajtódik végre), ha a várakozó utasítás feltétele teljesül.

Példa

initial a=0 ∧ b=0;
parbegin
  loop α_0: a:=a+1; α_1: **await** b≠0 ; α_2: a:=a+3; **end**
||
  loop β_0: a:=2*a; β_1: **await** a≠1 ; β_2: b:=b+1; **end**
parend

lépés	művelet	a	b
		0	0
1.	β_0: a:=2*a	0	0
2.	α_0: a:=a+1	1	0
	holtpont		

lépés	művelet	a	b
		0	0
1.	α_0: a:=a+1	1	0
2.	β_0: a:=2*a	2	0
3.	β_1: **await** a≠1	2	0
4.	β_2: b:=b+1	2	1
5.	α_1: **await** b≠0	2	1
…	nincs holtpont		

Műveleti szemantika

- Bármely S utasítássorozat rendelkezik az alábbi három tulajdonsággal:
  - entry(S)$\in \mathcal{M}_S$
  - trans(S)$\subseteq \mathcal{M}_S \times \mathcal{F}(\mathcal{L}_p) \times \mathcal{M}_S$
  - exits(S) $\subseteq \mathcal{M}_S \times \mathcal{F}(\mathcal{L}_p)$
- Definíció szerint: ha S egy párhuzamos program egyik komponense (S $\equiv \Pi_i$), akkor

S	entry	trans	exits
S_1 ; α: **stop** S_1 egy ut. sor.	entry(S_1)	trans(S_1)∪ $\{(\beta, C, \alpha) \mid (\beta, C) \in exits(S_1)\}$	$\{(\alpha, true)\}$
loop S_1 **end** S_1 egy ut. sor.	entry(S_1)	trans(S_1)∪ $\{(\beta, C, entry(S_1)) \mid (\beta, C) \in exits(S_1)\}$	∅

Utasítássorozat műveleti szemantikája

S	entry	trans	exits
α: a ahol a$\in\mathcal{E}$	α	∅	$\{(\alpha, true)\}$
α: **await** B **then** a	α	∅	$\{(\alpha, B)\}$
α: **if** B **then** S_1 **else** S_2	α	trans(S_1) ∪ trans(S_2) ∪ $\{(\alpha, B, entry(S_1)), (\alpha, \neg B, entry(S_2))\}$	exits(S_1) ∪ exits(S_2)
α: **if** B **then** S_1	α	trans(S_1) ∪ $\{(\alpha, B, entry(S_1))\}$	exits(S_1) ∪ $\{(\alpha, \neg B)\}$
α: **while** B **do** S_1 **od**	α	trans(S_1) ∪ ∪ $\{(\alpha, B, entry(S_1))\}$ ∪ $\{(\beta, C, \alpha) \mid (\beta, C) \in exits(S_1)\}$	$\{(\alpha, \neg B)\}$
α: S_1 ; S_2 ahol S_1 címkézetlen ut. S_2 ut. sor.	α	trans(S_1) ∪ trans(S_2) ∪ $\{(\beta, C, entry(S_2)) \mid (\beta, C) \in exits(S_1)\}$	exits(S_2)

Program állapot

- A $\Pi \equiv$ **initial** Pre; **parbegin** $\Pi_1 \| \ldots \| \Pi_p$ **parend**

  program egy állapota

  $\eta = (\sigma, \lambda_1, \ldots, \lambda_p, \kappa)$

  ahol
  - σ a memória állapot (változók aktuális értékei)
  - $\lambda_i \in \mathcal{M}_{\Pi_i}$ (Π_i címkéje) i=1...p
  - $\kappa \in \{0, 1, \ldots, p\}$
- Ha κ=0 azt jelenti, hogy a Π program terminált, azaz egyik komponense sem ütemezhető mert az mind vagy várakozó utasításnál áll, vagy befejeződött.

Végrehajtási sorozat

- A Π párhuzamos program egy végrehajtási sorozata a program állapotainak $< \eta_0, \eta_1, \ldots >$ végtelen sorozata
  - ahol $\eta_0 = (\sigma_0, \alpha_0^{(1)}, \ldots, \alpha_0^{(p)}, \kappa_0)$ és Pre teljesül a σ_0–ra,
  - a $\eta_i = (\sigma, \lambda_1, \ldots, \lambda_p, \kappa)$-t az a $\eta_{i+1} = (\sigma', \lambda'_1, \ldots, \lambda'_p, \kappa')$ követi, ahol $\kappa \in \{1 \ldots p\}$ esetén
    - $(\lambda_\kappa, C, \lambda'_\kappa) \in trans(\Pi_i)$ és C teljesül a σ-ra, továbbá $\lambda'_i = \lambda_i$ minden $i \in \{1 \ldots p\}$, $i \neq \kappa$-ra
    - a σ' a λ_κ címkéjű műveletnek σ-ra végrehajtott eredménye, ha λ_κ egy $\mathcal{A}$-beli művelet címkéje, különben $\sigma' = \sigma$,
    - a λ'_κ egy nem stop utasítás címkéje, kivéve, ha minden λ'_i címkéjű utasítás ($i \in \{1 \ldots p\}$) vagy **stop**, vagy olyan várakozó utasítás, amelynek feltétele σ'-ben hamis, mert ekkor $\kappa' = 0$.
  - illetve κ=0 esetén $\eta_{i+1} = \eta_i$.

Példa

- **initial** a=0 ∧ b=0;
  parbegin
  loop α_0: a:=a+1; α_1: **await** b≠0 ; α_2: a:=a+3; **end**
  ||
  loop β_0: a:=2*a; β_1: **await** a≠1 ; β_2: b:=b+1; **end**
  parend

η	σ	λ_1	λ_2	κ
η_0	a=0, b=0	α_0	β_0	2
η_1	a=0, b=0	α_0	β_1	1
η_2	a=1, b=0	α_1	β_1	0

η	σ	λ_1	λ_2	κ
η_0	a=0, b=0	α_0	β_0	2
η_1	a=0, b=0	α_0	β_1	2
η_2	a=0, b=0	α_0	β_2	1
η_3	a=1, b=0	α_1	β_2	2
η_4	a=1, b=1	α_1	β_0	1
η_5	a=1, b=1	α_2	β_0	2
η_6	a=2, b=1	α_1	β_1	*
η_6	…			