Programozáselmélet - Párhuzamos programok

Készítette: Borsi Zsolt

Továbbra is szekvenciális programokkal foglalkozunk, de bevezetünk három új programkonstrukciót. Ezek definícióját úgy adjuk meg, hogy a velük képzett relációk megfeleljenek a program korábban kimondott definíciónak. Azaz, a három új programkonstrukció segítségével további programokat tudunk előállítani. Mivel célunk továbbra is az, hogy belássuk programok helyességét (tehát hogy igazoljuk hogy adott program megold egy adott feladatot), az új programkonstrukcióknak is megadjuk a levezetési szabályait.

1. Multiprogramozási modell

Három új programkonstrukció egyike a párhuzamos blokk, ami lehetővé teszi párhuzamosan végrehajtott programok reprezentálását. Egy párhuzamos program végrehajtása alatt a párhuzamos blokkot alkotó programok elemi utasításainak valamilyen ütemezés szerinti sorbarendezését és egy processzoron (magon) való egymás utáni végrehajtását értjük. A multiprogramozási modellben az elemi programok végrehajtása (így az értékadás is) atomi módon történik, mint ahogy a logikai feltételek kiértékelése sem szakítható meg.

2. Új programkonstrukciók

- Atomi utasítás: $[S]$
  Megszakítás nélkül hajtjuk végre. S nem tartalmazhat ciklust vagy várakoztató utasítást.

- Várakoztató utasítás: $\textbf{await}\ B\ \textbf{then}\ S\ \textbf{ta}$
  Amennyiben a β őrfeltétel igaz, az S program atomi utasításként a β kiértékelésével egyszerre hajtódik végre. S nem tartalmazhat ciklust vagy várakoztató utasítást. Ha β kiértékelhető de hamis, az utasítást tartalmazó folyamat blokkolttá válik; a végrehajtás nem tud továbblépni a várakoztató utasításon, amíg egy másik folyamat nem „segít" és változtatja meg az állapotot úgy hogy β teljesüljön.

- Párhuzamos blokk: $\textbf{parbegin}\ S_1 \| \dots \| S_n\ \textbf{parend}$
  Befejeződik, ha minden komponensprogramja terminál. Végrehajtása valamely ágnak kiválasztását és az ott lévő komponens első utasításának, illetve a kapott maradék programnak egymás utáni végrehajtását jelenti.

Definíció: *Legyen A tetszőleges állapottér, S program az A állapottér felett.*
$\forall a \in A : [S](a) = S(a)$

Definíció: *Legyen A közös állapottere az S programnak és a β logikai függvénynek (őrfeltételnek).*

$$\forall a \in A : (\mathbf{await}\ \beta\ \mathbf{then}\ S\ \mathbf{ta})(a)\ = \begin{cases} [S](a) & \text{, ha } a \in \mathcal{D}_\beta \wedge \beta(a) \\ (\mathbf{skip}; \mathbf{await}\ \beta\ \mathbf{then}\ S\ \mathbf{ta})(a) & \text{, ha } a \in \mathcal{D}_\beta \wedge \neg\beta(a) \\ \{< a, \mathbf{fail} >\} & \text{, ha } a \notin \mathcal{D}_\beta \end{cases}$$

Amikor az n komponensből álló párhuzamos blokkot végrehajtjuk, akkor egy adott állapotból indulva bármely S_i komponenssel ($i \in [1..n]$) megkezdődhet a párhuzamos program végrehajtása. Amennyiben az S_i programot atomiként kell végrehajtani (mert vagy elemi program, vagy a [ ] atomi művelet programkonstrukcióval nem megszakíthatóvá lett téve), akkor annak végrehajtása után a $\mathbf{parbegin}\ S_1\|\ldots\|S_{i-1}\|S_{i+1}\|\ldots\|S_n\ \mathbf{parend}$ párhuzamos blokkot kell elvégezni.
Amennyiben S_i végrehajtása megszakítható, akkor S_i program felbontható egy u atomi utasításra és egy T programra. Vagyis S_i felfogható az $u;T$ szekvenciaként, ahol u az S_i első (nem megszakítható) utasítása, T pedig az S_i maradék része. Ekkor a párhuzamos blokk egy végrehajtását kapjuk az u utasítás majd azt követően a
$\mathbf{parbegin}\ S_1\|\ldots\|S_{i-1}\|T\|S_{i+1}\|\ldots\|S_n\ \mathbf{parend}$ párhuzamos blokkot elvégezve.
Mivel a párhuzamos blokk végrehajtását bármely komponenst kiválasztva elkezdhetjük, a párhuzamos programok szükségképpen nem-determinisztikusak. Az összes végrehajtási sorozatot megkapjuk a párhuzamos blokk elvégzését az elsőként az S_i komponens aktiválásával kapott végrehajtási sorozatok összességeként.

Definíció: *Legyen A közös alap-állapottere az $S_1 \ldots S_n$ programoknak.*

$$\forall a \in A : (\mathbf{parbegin}\ S_1 \parallel \cdots \parallel S_i \parallel \cdots \parallel S_n\ \mathbf{parend})(a)\ = \bigcup_{i=1}^{n} B_i(a)$$

ahol

$$B_i(a) = \begin{cases} (S_i; \mathbf{parbegin}\ S_1 \parallel \cdots \parallel S_{i-1} \parallel S_{i+1} \parallel \cdots \parallel S_n\ \mathbf{parend})(a) \\ \qquad \text{ha } S_i \text{ nem megszakítható az } a \text{ állapotból indulva} \\ (u_i; \mathbf{parbegin}\ S_1 \parallel \cdots \parallel S_{i-1} \parallel T_i \parallel S_{i+1} \parallel \cdots \parallel S_n\ \mathbf{parend})(a) \\ \qquad \text{ha } S_i(a) = (u_i; T_i)(a) \text{ ahol } u_i \text{ nem megszakítható az } a \text{ állapotból indulva} \end{cases}$$

Lásd a fejezet végén lévő kiegészítés részt, hogy mit értünk a különböző programszerkezetek első utasításaként egy adott a állapotból indulva.

3. Az új programkonstrukciók levezetési szabályai

Tétel: *Atomi művelet levezetési szabálya*
Ha
$Q \implies lf(S, R)$ *akkor* $Q \implies lf([S], R)$

Tétel: *Várakozó utasítás levezetési szabálya*
Ha

1. $Q \implies \beta \vee \neg\beta$
2. $Q \wedge \beta \implies lf(S, R)$

akkor $Q \implies lf(\mathbf{await}\ \beta\ \textit{\textbf{then}}\ S\,\mathbf{ta}, R)$

Tétel: *Párhuzamos blokk levezetési szabálya*
Legyenek $Q, Q_1, \dots Q_n$ *és* $R, R_1, \dots R_n$ *logikai függvények. Ha*

1. $Q \implies Q_1 \wedge \dots \wedge Q_n$ *és*
2. $R_1 \wedge \dots \wedge R_n \implies R$ *és*
3. $\forall i \in [1..n] : Q_i \implies lf(S_i, R_i)$ *és*
4. *a* $Q_1 \implies lf(S_1, R_1), \dots, Q_n \implies lf(S_n, R_n)$ *teljes helyességi formulák interferencia-mentesek és*
5. *a párhuzamos blokk holtpontmentes*

akkor $Q \implies lf(\mathbf{parbegin}\ S_1 \| \dots \| S_n\ \mathbf{parend}, R)$

A párhuzamos blokk levezetési szabályának első három feltétele szemléletesen azt fejezi ki, hogy

- ha Q előfeltétel teljesül, akkor a párhuzamos blokk bármely S_i komponensprogramjának végrehajtása elkezdhető, mert teljesül a Q_i előfeltétele (ezt nevezhetjük „belépési feltételnek”).
- amikor a párhuzamos blokk terminál (az összes S_i komponens befejeződött, elérte utófeltételét) akkor jó helyen terminált: ott ahol R utófeltétel igaz (ezt nevezhetjük „kilépési feltételnek”).
- minden S_i komponens önmagában teljesen helyes a Q_i előfeltételével és R_i utófeltételével adott feladatra nézve.

4. Interferencia-mentesség

Ezzel a fogalommal azt akarjuk kifejezni, hogy ha valamit már beláttunk az S_i komponens esetén, az a bizonyítás nem veszti érvényét egy S_i komponenssel párhuzamosan végrehajtott S_j komponensben lévő u utasítás elvégzésével. Például, ne legyen az hogy miután már beláttuk hogy S_i egy ciklusának magjában a termináló függvény értéke csökken, a bizonyításunkat tönkreteszi u végrehajtása, azáltal hogy meg tudja növelni az adott ciklus termináló függvényének értékét. Különben érvényét vesztené az a bizonyításunk, melyben beláttuk hogy a ciklus terminál.
Mivel értéket megváltoztatni, és így egy logikai állítást hamissá tenni csak az értékadás tud, így u utasításként azon utasításokat kell kell figyelembe venni, amik vagy értékadások vagy atomi módon végrehajtott programként értékadást tartalmaznak. Nevezzük őket kritikus utasításnak!

Jelölés: *Teljes helyességi formula Ha valamilyen S program és Q, R logikai függvények esetén teljesül hogy $Q \implies lf(S, R)$, akkor nevezzük az $Q \implies lf(S, R)$ alakot az S egy teljes helyességi formulájának! Mivel az interferencia-mentesség igazolásához azt akarjuk megmutatni hogy egy bizonyítás nem veszti érvényét, S program helyett egy hozzá tartozó $Q \implies lf(S, R)$ teljes helyességi formulát veszünk figyelembe (tehát azt amellyel bizonyítottuk hogy S program a Q feltételnek eleget tevő állapotokból indulva biztos hogy helyesen terminál, és a végállapotokban teljesül R).*

Definíció: *A $Q_1 \implies lf(S_1, R_1), \ldots, Q_n \implies lf(S_n, R_n)$ teljes helyességi formulák interferencia-mentesek, ha bármely i és j esetén ($i, j \in [1..n]$ és $i \neq j$) az S_i komponens egyik kritikus utasítása sem interferál a $Q_j \implies lf(S_j, R_j)$ teljes helyességi formulával.*

Definíció: *Azt mondjuk hogy az S_i komponens u kritikus utasítása (aminek előfeltételét jelölje pre_u) nem interferál az $Q_j \implies lf(S_j, R_j)$ teljes helyességi formulával, ha*

- $pre_u \wedge R_j \implies lf(u, R_j)$
  Azaz, u végrehajtása nem rontja el S_j utófeltételét: ha R_j igaz volt a végrehajtás előtt akkor utána is igaz lesz.

- $pre_u \wedge pre_s \implies lf(u, pre_s)$ *bármely S_j-beli s utasítás (aminek előfeltételét jelölje pre_s) esetén*
  Azaz u végrehajtása igaznak tartja meg S_j bármely s utasításának előfeltételét: ha s elvégezhető volt u végrehajtása előtt akkor utána is az lesz.

- $pre_u \wedge t = t_0 \implies lf(u, t \leq t_0), \forall t_0 \in \mathbb{Z}$ *esetén ahol t egy S_j-beli ciklus termináló függvénye*

Azaz u végrehajtása S_j bármely ciklusának t terminálófüggvényének értékét nem növeli meg.

5. Holtpont

Legyen $A = (a{:}\mathbb{Z}, b{:}\mathbb{Z})$ és tekintsük az A alap-állapottér feletti következő programot:

```
a:=0; b:=0;
parbegin
   while IGAZ do
      α0: a:=a+1;
      α1: await b≠0 then
            α2:     a:=a+3
          ta;
   od
||
   while IGAZ do
      β0: a:=2*a;
      β1: await a≠1 then
            β2:   b:=b+1
          ta;
   od
parend
```

Ez a program az állapottér bármely állapotából indulva, sorrendben a β_0 majd α_0 címkével jelzett utasításokat végrehajtva, az $\{a{:}1, b{:}0\}$ állapotba kerülve holtpontba jut.

Definíció: *Holtpont*

- *Egy várakoztató utasítás egy adott állapotban blokkolt, ha az őrfeltétele hamis az állapotban.*
- *Egy szekvenciális program blokkolt, ha egy általa közvetlenül tartalmazott várakoztató utasítás blokkolt.*
- *Egy párhuzamos blokk holtpontban van, ha minden még be nem fejeződött komponense blokkolttá vált.*

6. A holtpontmentesség egy elégséges feltétele

Elégséges feltételt szeretnénk arra adni, hogy a programunk garantáltan nem juthat holtpontba, nincs olyan végrehajtása amely során holtponthelyzet állna fent. Hangsúlyozzuk, hogy egy általános párhuzamos program alatt olyan programot értünk ami az eddig megengedett programkonstrukciókból épül fel (nagy valószínűséggel szekvencia) de tartalmazza az új programkonstrukciók valamelyikét is.
Tekintsünk egy ilyen programot. A holtpontmentesség szempontjából számunkra a várakoztató utasítások és a párhuzamos blokkok az érdekesek. Tegyük fel hogy az általános programunkban m olyan várakoztató utasítás szerepel ami nincs egy párhuzamos blokkon belül, a párhuzamos blokkok száma pedig legyen n.
A k-adik párhuzamos blokkot jelöljük így (tehát n_k komponense van a $k$adik párhuzamos blokknak):
T_k: **parbegin** $S_1^k \| \dots \| S_{n_k}^k$ **parend**, ahol a komponensek akár újabb várakoztató utasításokat tartalmazhatnak.

S :

$\vdots$

A_1: **await**β_1 then S_1 ta;

$\vdots$

A_i: **await**β_i then S_i ta;

$\vdots$

T_1: **parbegin** $S_1^1 \| \dots \| S_{n_1}^1$ **parend**

$\vdots$

A_j: **await**β_j then S_j ta;

$\vdots$

T_n: **parbegin** $S_1^n \| \dots \| S_{n_n}^n$ **parend**

$\vdots$

A_m: **await**β_m then S_m ta;

$\vdots$

Sorra vesszük a lehetséges holtpont helyzeteket:

- Az S szekvenciális folyamat blokkolt, ha valamely (nem párhuzamos blokkon belüli) várakoztató utasításánál várakozunk. Ezt úgy írjuk le hogy az A_j ($j \in [1..m]$) várakoztató utasítást készek vagyunk elvégezni (a $pre(A_j)$ előfeltétele teljesül) de a β_j őrfeltétel $hamis$.

- A szekvenciális S program valamely párhuzamos blokkja blokkolt. Tegyük fel hogy a T_k párhuzamos blokkról van szó. Azaz T_k minden S_i^k ($i \in [1..n_k]$) komponensprogramja terminált vagy blokkolt, de legalább az egyik blokkolt.

Definiáljuk $D(S)$-et és $D1(T_k)$-et a következőképpen: (A $D1$ alkalmazásakor tudjuk hogy az argumentuma egy párhuzamos blokk.)

- $$D(S) = \left[\bigvee_{j=1}^{m}(pre(A_j) \wedge \neg\beta_j)\right] \vee \left[\bigvee_{k=1}^{n} D1(T_k)\right]$$

  A formula azt fejezi ki hogy van benne (de nem párhuzamos blokkon belüli, hanem közvetlenül a „főprogramban") blokkolt várakoztató utasítás, vagy valamely párhuzamos blokkja blokkolt.

- $$D1(T_k) = \left[\bigwedge_{j=1}^{n_k}(post(S_i^k) \vee D(S_i^k))\right] \wedge \left[\bigvee_{i=1}^{n_k} D(S_i^k)\right]$$

  A T_k párhuzamos blokk blokkolt, ha minden S_i^k komponense blokkolt vagy véget ért, miközben van legalább egy komponense ami blokkolt.

Tétel: *Amennyiben $D(S)$ azonosan $HAMIS$, nem lehet az S programnak olyan végrehajtása hogy holtpont előfordulhatna; S holtpontmentes.*

7. Kiegészítés: S program felbontása első utasításra és az azt követő maradék programra

Itt megadjuk hogy írható fel egy a kezdőállapotot tekintve az S program (ahol S nem elemi program és nem is atomiként végrehajtható) az $u;T$ szekvenciaként, ahol u az S nem megszakítható első utasítása, T pedig a maradék program.

- Ha $S =$ **await** β **then** S_0 **ta** és $a \in \mathcal{D}_\beta \wedge \neg\beta(a)$, akkor u a **SKIP** program míg T az **await** β **then** S_0 **ta** várakoztató utasítás.
- Ha $S = (S_1; S_2)$ és S_1 végrehajtása nem megszakítható a-ból indulva, akkor u az S_1 és T az S_2.
- Ha $S = (S_1; S_2)$ és S_1 végrehajtása megszakítható a-ból indulva, akkor u az S_1 első (atomi módon végrehajtható) utasítása, míg T a $(T_1; S_2)$ szekvencia (ahol T_1 az S_1 maradék része).

- Ha $S = \textbf{if } \pi_1 \rightarrow S_1 \,\Box \ldots \Box\, \pi_n \rightarrow S_n \textbf{ fi}$ és van olyan π_i feltétel ami nem értelmezett a állapotban vagy mindegyik feltétel hamis a-ban ($\exists i \in [1..n] : a \notin \mathcal{D}_{\pi_i} \vee \forall i \in [1..n] : a \in \mathcal{D}_{\pi_i} \wedge \neg\pi_i(a)$), akkor u az **ABORT** program és T szintén az **ABORT**.

- Ha $S = \textbf{if } \pi_1 \rightarrow S_1 \,\Box \ldots \Box\, \pi_n \rightarrow S_n \textbf{ fi}$ és az a állapotban mindegyik feltétel értelmezett és legalább egy közülük teljesül ($\forall i \in [1..n] : a \in \mathcal{D}_{\pi_i} \wedge \exists i \in [1..n] : \pi_i(a)$), akkor u a **SKIP** program míg T az S_i (feltéve hogy azt a végrehajtást nézzük ami az a állapotból indulva az i-edik ág kiválasztásával történik).

- Ha $S = \textbf{while } \pi \textbf{ do } S_0 \textbf{ od}$ és az a állapotban a ciklusfeltétel nem értelmezett ($a \notin \mathcal{D}_\pi$), akkor u és T is az **ABORT** program.

- Ha $S = \textbf{while } \pi \textbf{ do } S_0 \textbf{ od}$ és a állapotban a ciklusfeltétel hamis ($a \in \mathcal{D}_\pi \wedge \neg\pi(a)$), akkor u és T is az **ABORT** program.

- Ha $S = \textbf{while } \pi \textbf{ do } S_0 \textbf{ od}$ és a ciklusfeltétel teljesül az a állapotban ($a \in \mathcal{D}_\pi \wedge \pi(a)$), akkor u a **SKIP** program és T az ($S_0; \textbf{while } \pi \textbf{ do } S_0 \textbf{ od}$) szekvencia.

- Ha $S = \textbf{parbegin } S_1 \parallel \cdots \parallel S_i \parallel \cdots \parallel S_n \textbf{ parend}$ és az a állapotból indulva nem megszakíthatóan végrehajtható S_i elvégzésével kezdjük meg a párhuzamos blokk végrehajtását, akkor u az S_i és T a $\textbf{parbegin } S_1 \parallel \cdots \parallel S_{i-1} \parallel S_{i+1} \parallel \cdots \parallel S_n \textbf{ parend}$ párhuzamos blokk.

- Ha $S = \textbf{parbegin } S_1 \parallel \cdots \parallel S_i \parallel \cdots \parallel S_n \textbf{ parend}$ és az a állapotból indulva a megszakítható S_i elvégzésével kezdjük meg a párhuzamos blokk végrehajtását, akkor u az S_i első (atomi módon végrehajtható) utasítása, és T az $\textbf{parbegin } S_1 \parallel \ldots \| S_{i-1} \parallel T_i \parallel S_{i+1} \| \cdots \parallel S_n \textbf{ parend}$ párhuzamos blokk, ahol T_i az S_i maradék programja.