S0-01 Szintézis és verifikáció (Programozás elmélet)

Tartalom

1. Programozási alapfogalmak

A Programozás elméletben adatközpontú megközelítést alkalmazunk. Ezzel a megközelítéssel mind a feladat, mind a program, stb. definícióját adatok halmazán, állapottér meghatározásával írjuk le.

Állapottér

Egy adat **típus-értékhalmaza** az adat lehetséges értékeiből áll. **Állapotnak** hívjuk azt az érték-együttest, amikor egy feladat vagy egy program *minden* adata felvesz a saját típus-értékhalmazából egy-egy értéket.

Formálisan:

Legyenek $A_1, ..., A_n (n \in \mathbb{N}^+)$ típusérték-halmazok, és a halmazokat azonosító (egyedi, páronként különböző) $v_1, ..., v_n$ címkék (változók)

Minden cimke egy adatot (változó) jelöl. Egy állapot $\{v_1 : a_1, ..., v_n : a_n\}$ (cimkézett értékek halmaza), ahol minden változó felvesz egy értéket a hozzátartozó típus-értékhalmazból. ($a_i \in A_i$)

Az összes így képzett állapot halmazát **állapottérnek** nevezzük:

$$A := \{\ \{v_1 : a_1, ..., v_n : a_n\} \mid a_i \in A_i\ (i = 1, ..., n)\ \}$$

Feladat

A feladat egy kapcsolat (leképezés) a bemenet és az eredmény között: $F \subseteq A \times A$

Figyeljük meg, hogy az F reláció csak része az $A \times A$ descartes-szorzatnak. Mivel azonban a kiinduló állapotok csak az állapottér egy részhalmazát képzik, illetve az ezekhez rendelt állapotok ugyancsak az állapottérnek egy részhalmazát képzik, így érthető, hogy a feladat nem feltétlenül érvényes a teljes állapottéren.

Példa:

Osztója reláció

Kérdés: d osztója-e n-nek, melyet az l logikai változó jelöl.

$$A = (n : \mathbb{Z}, d : \mathbb{Z}, l : \mathbb{L}) \tag{1}$$
$$F \subseteq A \times A \tag{2}$$
$$\text{ahol} \tag{3}$$
$$D_f = \{\{n, d, l\} \in A \mid d \neq 0\} \tag{4}$$
$$\forall a \in D_F : F(a) = \{\{n, d, l\} \in A \mid l = d|n\} \tag{5}$$

Tehát láthatjuk, hogy az F értelmezési tartományába csak azok az állapotok tartoznak bele, ahol az osztó nemnulla. Az l változó értéke $F(a)$-ban pedig aszerint alakul, hogy az d osztja-e n-et.

Előfeltétel és utófeltétel

Válasszunk egy P paraméterhalmazt, mellyel az F feladat felbontható két reláció kompozíciójára:

$$F = F_1 \circ F_2$$

ahol $F_1 \subseteq A \times P$ és $F_2 \subseteq P \times A$, úgy hogy: $\forall a \in D_F : F(a) = F_2(F_1(a))$

Ekkor definiálni tudjuk a feladat elő- és utófeltételét:

$\forall p \in P :$

- $Ef_P : A \to \mathbb{L}$ melyre $[Ef_P] = F_1^{(-1)}(p)$
- $Uf_P : A \to \mathbb{L}$ melyre $[Uf_P] = F_2(p)$

Jelölés magyarázat:

Ha adott egy $Q : A \to \mathbb{L}$ állítás, annak az igazsághalmaza: $[Q] := \{q \in Q \mid Q(a)\ igaz\}$. Tehát Ef olyan állítás, mely a feladat minden kezdőállapotára igazat ad, illetve az Uf olyan állítás, mely az $F(a)$ állaptokra ad igaz értéket.

Program

Egy programot sokféleképpen lehet megadni. (program-gráf, strukogram, automaták, utasítások, pszeudo- vagy programnyelv).

A program alaptermészete az, hogy különböző végrehajtásokat okoz. Tehát egy program a lehetséges végrehajtásainak összessége. Egy ilyen végrehajtás sorozatos állapot-változásokat idéz elő, ennél fogva leírható egy állapotsorozattal.

Példa:

$A = (n : [-5, ..., \infty))$

```
while n != 10:
    n = n + sgn(n)
```

Ennek a lehetséges végrehajtásai a következők:

```
 0: <0,0,0,0,0, ... >             nem terminál (divergál)
 4: <4,5,6,7,8,9,10>              terminál
10: <10>                          terminál
13: <13,14,15,16, ... >           nem terminál (divergál)
-2: <-2,-3,-4,-5, fail >          abortál
```

A program lehet **nem-determinisztikus** (nem mindig ugyanazt az állaposorozatot adja a kezdőállapból indítva), lehetnek **segédváltozói**, lehet **véges** vagy **végtelen hosszú**, illetve leállhat **hibás** állapotban vagy speciálisan **holtpontban**.

Alap-állapottér

A program alap-állapottere a **program interfésze**, mellyel a program a környezetével kommunikál (kezdőállapotát a környezet adja, végállapotát a környezet kapja). Illetve ez írja le a program **alap-változóit**, melyek a program működése során végig léteznek.

Fromális specifikáció

- Legyen A az **alap-állapottér** ($\text{fail} \notin A$)
- Legyen $\overline{A}$ azon (véges komponensű) állapotterek úniója, melyeknek A altere:

$$\overline{A} := \bigcup_{A \leq B} B$$

- Jelöljük H^{**}-gal a H elemeiből képzett összes sorozatot (lehet végtelen is).
- Jelöljük H^{*}-gal a H elemeiből képzett összes véges sorozatot.

Ezek alapján a A feletti programnak hívjuk azt az

$$S \subseteq A \times (\overline{A} \cup \text{fail})^{**}$$

relációt, melyre fennállnak a következők:

1. $D_S = A$

   a program értelmezési tartománya az alap-állapottér.

2. $\forall a \in A : \forall \alpha \in S(a) : \ |\alpha| \geq 1 \land \alpha_1 = a$

   az állapotsorozat legalább egy hosszú és a első állapota mindig az alap-állapottérből való.

3. $\forall \alpha \in R_S : \forall i \ (1 \leq i < |\alpha|) : \alpha_i \neq \text{fail}$

   csak az utolsó elem lehet `fail` a végrehajtási állapotsorozatban

4. $\forall \alpha \in R_S : |\alpha| < \infty \longrightarrow \alpha_{|\alpha|} \in A \cup \{\text{fail}\}$

   a véges végrehajtások utolsó állapota vagy a `fail` vagy alap-állapottérbeli állapot.

Megoldás

Egy program akkor old meg egy feladatot (a program helyes a feladat szempontjából), ha végrehajtásai a feladat kezdőállapotaiból indulva a feladat megfelelő célállapotaiban állnak meg. Ekkor a feladat állapottere és a program alap-állapottere azonos kell legyen. Ilyenkor a program végrehajtásai között találjuk a feladat kezdőállapotából induló végrehajtásokat, amelyekről azt kell eldönteni, hogy terminálnak-e (hibátlan és véges hosszú) és végállapotuk a feladat által megkívánt valamelyik célállapot lesz-e.

Megoldás minősített esetei

- **Parciális helyesség**: Ha a program a feladat kezdőállapotaiból indulva leáll, akkor ezt a feladatnak megfelelő célállapotban teszi. (A leállást nem követeljük meg.) Jelölés: $\{Ef\}S\{Uf\}$ (S minden Ef-beli állapotból induló véges végrehajtása egy Uf-beli állapotba jut)
- **Leállás**: A feladat kezdőállapotából indulva leáll a program. (Hány vagy legfeljebb hány lépés múlva következik ez be? Kizárható-e a végtelen vagy a hibás működés, ez utóbbiba beleértve a holtpont helyzet kialakulását is.)
- **Teljes helyesség**: A program a feladat kezdőállapotaiból indulva a feladatnak megfelelő célállapotban (hibátlanul) áll le. Jelölés: $\{\{Ef\}\}S\{\{Uf\}\}$ (S minden Ef-beli állapotból induló végrehajtása egy Uf-beli állapotba jut.)
- **Gyengén teljes helyesség**: Ilyenkor a helyesség-vizsgálat során figyelmen kívül hagyjuk a holtpont kialakulását.

2. Elemi programok és program konstrukciók definíciói

Elemi programok

Üres program

Az üres program gyakorlatilag az identitásfüggvény. ($S := skip$)

$$skip(\sigma) =< \sigma >$$

(Azaz az üres program egyetlen állapota a kezdőállapot.)

Értékadás

Az értékadás megváltoztatja egy változó értékét, így új állapotot idéz elő. ($S := v := f(v)$)

$$
\begin{aligned}
(v := f(v))(\sigma) =&< \sigma, \sigma' > && (6)\\
&\text{ahol} \quad \sigma' \in f(\sigma) \quad \text{ha } \sigma \in D_f && (7)\\
&\text{és} \quad \sigma' \in \text{fail} \quad \text{ha } \sigma \notin D_f && (8)
\end{aligned}
$$

Program konstrukciók

Szekvencia

A szekvenciával két program összefűzését érhetjük el.

Legyen S_1 és S_2 közös állapotterű (A) programok.

$$
\begin{aligned}
(S_1; S_2)(\sigma) = &\{\alpha \mid \alpha \in S_1(\sigma) \cap \overline{A}^{\infty}\} && (9)\\
&\cup \{\alpha \mid \alpha \in S_1(\sigma) \text{ és } |\alpha| < \infty \text{ és } \alpha_{|\alpha|} = \text{fail}\} && (10)\\
&\cup \{\alpha \otimes \beta \mid \alpha \in S_1(\sigma) \cap \overline{A}^{\star} \text{ és } \beta \in S_2(\alpha_{|\alpha|})\} && (11)\\
& && (12)
\end{aligned}
$$

Tehát:

1. Ha az S_1 nem terminál, akkor a szekvencia is csak a végtelen hosszú S_1 végrehatás lesz.
2. Ha az S_1 abortál, akkor a szekvencia is csak az abortált S_1 végrehajtás lesz.
3. Ha az S_1 hiba nélkül áll le, akkor ahhoz az állapotsorozathoz fűzzük hozzá az S_2 végrehajtását az S_1 utolsó állapotából. (a csatlakozásnál a duplikátumokat redukáljuk)

Elágazás

Az elágazással feltételek alapján változtathatjuk a sorozatot.

Legyenek $S_1, ..., S_n$ programok és $\pi_1, ..., \pi_n$ feltételek, amelyeknek közös alap-állapottere az A.

$$(\text{if } \pi_1 \rightarrow S_1, ..., \pi_n \rightarrow S_n \text{ fi})(\sigma) = \tag{13}$$

$$\bigcup_{\substack{i=1 \\ \sigma \in D_{\pi_i} \wedge \pi_i(\sigma)}}^{n} S_i(\sigma) \ \cup \ \begin{cases} < \sigma, \text{fail} > & \text{ha} \quad \exists i \in [1..n] : \sigma \notin D_{\pi_i} \ \vee \\ & \quad \vee \ \forall i \in [1..n] : \sigma \in D_{\pi_i} \wedge \neg \pi_i(\sigma) \\ \emptyset \text{ különben} & \end{cases} \tag{14}$$

Tehát:

1. Minden olyan S_i végrehajtás úniója ahol σ megfelel a π_i feltételnek
2. Ha van egy olyan feltétel, ahol a σ nincs benne a feltétel értelmezési tartományában, vagy a σ egyik feltételnek sem tesz eleget, akkor a végrehajtás abortál.

Ciklus

A ciklussal ismétlődő sorozatokat állíthatunk elő.

Legyen S_0 program és π feltétel, amelyeknek közös alap-állapottere az A.

$$(\text{while } \pi \text{ do } S_0 \text{ od})(\sigma) = \begin{cases} (S_0; \text{while } \pi \text{ do } S_0 \text{ od})(\sigma) & \text{ha } \sigma \in D_\pi \wedge \pi(\sigma) \\ < \sigma > & \text{ha } \sigma \in D_\pi \wedge \neg\pi(\sigma) \\ < \sigma, \text{fail} > & \text{ha } \sigma \notin D_\pi \end{cases} \tag{15}$$

Tehát:

1. Amíg π teljesül, addig szekvenciálisan összefűzzük az S_0-t a “rekurzívan” hívott ciklussal
2. Ha nem teljesül a feltétel, akkor gyakorlatilag egy `skip`-et hajtunk végre
3. Ha a σ nincs benne a π értelmezési tartományában, akkor a program abortál

Atomi utasítás

Atomi utasításnak párhuzamos/konkurens programok esetén tulajdonítunk fontos szerepet. Az atomi utasítás nem tartalmazhat sem ciklust, sem várakozó utasítást. Ekkor az utasítást egyszerre, megszakítás nélkül kell végrehajtani.

$$[S](\sigma) = S(\sigma)$$

Elsőre furának tűnhet, de szemantikai értelemben valóban nincs különbség.

Várakozó utasítás

A várakozó utasítás párhuzamos/konkurens programok esetén szinkronizációra használható.

$$(\text{await } \beta \text{ then } S \text{ ta})(\sigma) = \begin{cases} <\sigma, \text{fail}> & \text{ha } \sigma \notin D_\beta \\ S(\sigma) & \text{ha } \sigma \in D_\beta \wedge \beta(\sigma) \\ (\text{await } \beta \text{ then } S \text{ ta})(\sigma) & \text{ha } \sigma \in D_\beta \wedge \neg\beta(\sigma) \end{cases}$$

A harmadik esetben nem párhuzamos program esetén nincs, ami megváltoztassa a σ értékét, így ez holtpontot okozhat. A várakozó utasítás esetén a β kiétékelése és az S program atomi műveletként hajtódik végre. Az S nem tartalmaz sem ciklust, sem várakozó utasítást.

Párhuzamos blokk

A párhuzamos blokkokkal leírhatjuk, hogy melyik programrészek futhatnak párhuzamosan.

Legyen $S_1, ..., S_n$ a párhuzamosan végrehajtott program ún. programágai. Az ütemező ezek közül választhat egyet végrehajtásra.

Amennyiben az ütemező az i-edig ágnak adja a vezérlést:

$$(\text{parbegin } S_1 \| ... \| S_i \| ... \| S_n \text{ parend})(\sigma) = \tag{16}$$

$$\begin{cases} (\text{parbegin } S_1 \| ... \| S_{i-1} \| S_{i+1} \| ... \| S_n \text{ parend})(\sigma) & \text{ha } S_i = \text{skip} \\ (u; \text{parbegin } S_1 \| ... \| S_{i-1} \| T_i \| S_{i+1} \| ... \| S_n \text{ parend})(\sigma) & \text{ha skip} \neq S_i = u; T_i \end{cases} \tag{17}$$

3. Nem-determinisztikus strukturált programok formális verifikációja

A helyesség-vizsgálati módszerek menete

A strukturált programok helyesség bizonyításának lényege, hogy belássuk, a program megoldja az adott feladatot. Hoare egy olyan deduktív módszert javasolt, mely:

- az elemi programok esetében közvetlen választ ad a fenti kérdésre

- összetett programok esetén pedig visszavezeti a helyesség belátását az összetétel komponens programjainak vizsgálatára. (a komponens programok számára kijelöl egy-egy feladatot. Ha a komponens programok egyenként megoldják ezeket a feladatokat, akkor azokból konstruált program megoldja az eredeti feladatot.)

Nevezetes programszerkezetek helyességének szabályai

Jelöljük egy feladat specifikációját (A, Q, R)-el, ahol A az alap-állapottér, Q az előfeltétel, és R az utófeltétel.

Üres program

- Az üres program megoldja az (A, R, R) specifikációjú feladatot.

$$\{\{R\}\}\ \text{skip}\ \{\{R\}\}$$

- Az üres program akkor oldja meg az (A, Q, R) specifikációjú feladatot, ha $Q \Rightarrow R$

$$\frac{Q \Rightarrow R}{\{\{Q\}\}\ \text{skip}\ \{\{R\}\}}$$

Értékadás

- A $v := f(v)$ értékadás az $(A, v \in D_f \wedge \forall e \in f(v) : R^{v \leftarrow e}, R)$ specifikációjú feladatot oldja meg.

$$\{\{v \in D_f \wedge \forall e \in f(v) : R^{v \leftarrow e}\}\}\ v := f(v)\ \{\{R\}\}$$

Ez a következtetés talán némi magyarázatra szorul. A gondolat az egész mögött az, hogy végezzük el az utófeltételben a helyettesítést és az így kapott állítás lesz az előfeltétele az értékadásnak.

Nezzünk erre egy példát. Tegyük fel, hogy az értékadás: $x := 5$, illetve az utófeltétel: $R\ :=\ 0 < x < y$

Ekkor $R^{x \leftarrow 5} = 0 < 5 < y = 5 < y$ az előfeltétel, hiszen annak, hogy a program lefutása után $0 < x < y$ fennáljon egyedül az a feltétele, hogy $5 < y$, mivel az x-et a program meghatározza. A definíció ezt még megszorítja azzal, hogy:

1. a v természetesen f értelmezési tartományában kell legyen
2. f-nek esetleg több állapota is lehet melyet v-ből képez, így az összes lehetséges állítás konjukcióját kell venni.

- A $v := f(v)$ értékadás akkor oldja meg az (A, Q, R) specifikációjú feladatot, ha $Q \Rightarrow v \in D_f \wedge \forall e \in f(v) : R^{v \leftarrow e}$

$$\frac{Q \Rightarrow v \in D_f \wedge \forall e \in f(v) : R^{v \leftarrow e}}{\{\{Q\}\}\ v := f(v)\ \{\{R\}\}}$$

Megjegyzés: speciális esetekben egyszerűsödhet az előfeltétel. Ha $D_f = A$, akkor a $v \in D_f$ feltétel elhagyható, ha pedig az értékadás determinisztikus, akkor a $\forall e \in f(v) : R^{v \leftarrow e}$ helyett elég $R^{v \leftarrow f(v)}$-t írni.

Szekvencia

Legyen S_1 és S_2 programok szekvenciája az A alap-állapottéren az $(S_1; S_2)$.

Ha S_1 az (A, Q, Q') feladatot és S_2 az (A, Q', R) feladatot oldja meg, akkor a szekvencia megoldja az (A, Q, R) feladatot.

$$\frac{\{\{Q\}\}\ S_1\ \{\{Q'\}\}\ \wedge\ \{\{Q'\}\}\ S_2\ \{\{R\}\}}{\{\{Q\}\}\ S_1; S_2\ \{\{R\}\}}$$

Elágazás

Legyen az $S_1, ..., S_n$ programokból és a $\pi_1, \ldots, \pi_n : A \to \mathbb{L}$ feltételekből álló elágazás az A alap-állapottéren az if $\pi_1 \to S_1, ..., \pi_n \to S_n$ fi

Ha minden Q-beli állapotra minden elágazás feltétel értelmes, és legalább az egyik teljesül is, továbbá minden S_i programág megoldja az $(A, Q \wedge \pi_i, R)$ feladatot, akkor az elágazás megoldja az (A, Q, R) feladatot.

$$\frac{\substack{Q \subseteq D_{\pi_1} \cap \ldots \cap D_{\pi_n} \\ Q \Rightarrow \pi_1 \vee \ldots \vee \pi_n \\ \forall i \in [1..n]:\ \{\{Q \wedge \pi_i\}\}\ S_i\ \{\{R\}\}}}{\{\{Q\}\}\ \text{if } \pi_1 \to S_1, ..., \pi_n \to S_n \text{ fi } \{\{R\}\}}$$

Ciklus

Tekintsük a while π do S_0 od ciklust az A alap-állapottéren, ahol S_0 program a ciklusmag, a $\pi : A \to \mathbb{L}$ a ciklusfeltétel. Továbbá legyen az ún. *invariáns állítás* egy $I : A \to \mathbb{L}$ logikai függvény.

Ha:

1. Minden Q-beli állapot egyben I-beli (azaz $[Q] \subseteq [I]$, másként: $Q \Rightarrow I$)
2. Az I-beli állapotokra értelmes a π
3. A π-t nem kielégítő (I-beli) állapotok R-beliek. (Magyarul a ciklusból kilépve igaz lesz R)
4. Továbbá az S_0 megoldja az $(A, I \wedge \pi, I)$ feladatot. (tehát a ciklumag megőrzi az invariánst)

Akkor a ciklus parciális értelemben megoldja az (A, Q, R) feladatot.

$$\begin{array}{l}
Q \Rightarrow I \qquad (18)\\
I \subseteq D_\pi \qquad (19)\\
I \wedge \neg\pi \Rightarrow R \qquad (20)\\
\{I \wedge \pi\}\ S_0\ \{I\} \qquad (21)\\
\hline
\{Q\} \text{ while } \pi \text{ do } S_0 \text{ od } \{R\} \qquad (22)
\end{array}$$

A teljes helyességhez szükségünk van arra, hogy a ciklus leálljon. Ehhez be kell vezetnünk egy ún. *variáns függvényt*, vagy *termináló függvényt*: $t : A \to W$, ahol W-n létezik egy $< \subseteq W \times W$ rendezési reláció. Emellett $W_< \subseteq W$ egy jólrendezett halmaz, azaz teljesen rendezett (bármely két elem összehasonlítható), és bármely (nemüres) részhalmazának van minimuma. A legtöbb esetben $\mathbb{N} = W_< \subseteq W = \mathbb{Z}$ választással szoktunk élni.

Megyjegyzés: a definíció bonyolultságát az indokolja, hogy a ciklus után a t értéke eggyel kisebb, mint a $W_<$ minimuma, tehát a t értelmezési tartománya bővebb kell legyen, mint $W_<$.

A parciális helyességen túl tehát azt kell még biztosítani, hogy S_0 mellett a t szigorúan monoton csökkenő, azaz bármely $a \in [I \wedge \pi]$ állapotból az S_0 olyan $b \in A$ állapotba jut, melyre $t(b) < t(a)$. Amennyiben ez teljesül, akkor a ciklus megoldja az (A, Q, R) feladatot.

$$\begin{array}{l}
Q \Rightarrow I \qquad (23)\\
I \subseteq D_\pi \qquad (24)\\
I \wedge \neg\pi \Rightarrow R \qquad (25)\\
I \wedge \pi \Rightarrow t \in W_< \qquad (26)\\
\forall c_0 \in W : \{\{I \wedge \pi \wedge t = c_0\}\}\ S_0\ \{\{I \wedge\ t < c_0\}\} \qquad (27)\\
\hline
\{\{Q\}\} \text{ while } \pi \text{ do } S_0 \text{ od } \{\{R\}\} \qquad (28)
\end{array}$$

4. Párhozamos programok formális verifikációja

A helyesség-vizsgálati módszerek menete

A párhuzamos programok helyesség-vizsgálatánál hasonlóan járunk el, mint nem párhuzamos estben. A gyengén teljes helyesség vizsgálatához ez elegendő is. A teljes helyességhez a holtpont-mentességet is vizsgálni kell majd.

Nevezetes programszerkezetek helyességének szabályai

Atomi utasítás

Ahogy már korábban láttuk, az atomi utasítás szemantikája: $[S](\sigma) = S(\sigma)$.

Emiatt a szemantikából közvetlenül adódik a helyesség:

$$\frac{\{\{Q\}\}\ S\ \{\{R\}\}}{\{\{Q\}\}\ [S]\ \{\{R\}\}}$$

Megjegyzés: A interferencia-mentesség és holtpont-mentesség jelen esetben biztosítva van, mivel az S atomi végrehajtású.

Várakozó utasítás

A várakozó utasítás (await β then S ta) szemantikájánál láttuk, hogy 3 eset lehetséges:

1. ha $\sigma \notin D_\beta$, akkor abortál
2. ha nem teljesül a feltétel, akkor nincs változás
3. ha $\sigma \in D_\beta$ és $\beta(\sigma)$ igaz, akkor végrehajtjuk az S-et.

A helyesség kapcsán csak a harmadik pont érdekes, ami meg közvetlenül adódik.

$$\frac{\{\{Q \wedge \beta\}\}\ S\ \{\{R\}\}}{\{\{Q\}\}\ \text{await } \beta \text{ then } S \text{ ta}\ \{\{R\}\}}$$

Megjegyzés: A interferencia-mentesség és holtpont-mentesség jelen esetben biztosítva van, mivel β kiértékelése és S is atomi végrehajtású.

Párhuzamos blokk

A párhuzamos blokk esetében azt tudjuk mondani, hogy ha $S_1, ..., S_n$ interferencia-mentesek és a belőük képzett párhuzamos blokk holtpont-mentes, akkor:

$$\{\{Q_1\}\}\ S_1\{\{R_n\}\}, ..., \{\{Q_n\}\}\ S_n\{\{R_n\}\} \tag{29}$$

$$Q \Rightarrow Q_1 \wedge ... \wedge Q_n \tag{30}$$

$$R_1 \wedge ... \wedge R_n \Rightarrow R \tag{31}$$

$$\{\{Q\}\}\ \text{parbegin } S_1 \| ... \| S_n \text{ parend}\ \{\{R\}\} \tag{32}$$

5. Az interferencia-mentesség és a holtpont-mentesség vizsgálata

Interferencia-mentesség

Annotáció

Az S program **segédállításokkal** és S változóit nem értintő **extra műveletekkel** kiegészített változatát az S program annotációjának nevezzük és S^*-gal jelöljük.

$$\frac{\{\{Q\}\}\ S^*\ \{\{R\}\}}{\{\{Q\}\}\ S\ \{\{R\}\}}$$

Parciális helyességi tételek interferencia-mentessége

kritikus utasítás: Értékadás, vagy értékadást is tartalmazó atomi művelet.

Egy u kritikus utasítás nem interferál a $\{Q\}\ S^*\ \{R\}$ parciális helyességi tétellel, ha:

1. u nem sérti meg R-t, azaz $\{R \wedge \mathrm{pre}_u\}\ u\ \{R\}$
2. u nem sérti meg az S^*-beli egyik utasítás előfeltételét (pre_s) sem, azaz $\{\mathrm{pre}_s \wedge \mathrm{pre}_u\}\ u\ \{\mathrm{pre}_s\}$ (az egyik pre_s éppen Q lesz.)

Ezek alapján a $\{Q_k\}\ S_k^*\ \{R_k\}\ k \in [1..n]$ parciális helyességi tételek interferencia-mentesek, ha $\forall i, j \in [1..n] : i \neq j$ folyamatpárra az S_i^*-nak egy kritikus utasítása sem interferál a $\{Q_j\}\ S_j^*\ \{R_j\}$ parciális helyességi tétellel.

Gyengén teljes helyességi tételek interferencia-mentessége

A parciális helyességi tételből kiindulva meg tudjuk határozni a gyengén teljes helyességi tételek interferencia-mentességét.

Tehát a fentiekhez hasonlóan, egy u kritikus utasítás nem interferál a $\{\{Q\}\}\ S^*\ \{\{R\}\}$ gyengén teljes helyességi tétellel, ha:

1. nem interferál a $\{Q\}\ S^*\ \{R\}$ parciális helyességi tétellel
2. és minden S^*-beli ciklus összes s utasítására: $\{t = c_0 \wedge \mathrm{pre}_s \wedge \mathrm{pre}_u\}\ u\ \{t \leq c_0\}$

Illetve hasonlóan a $\{\{Q_k\}\}\ S_k^*\ \{\{R_k\}\}\ k \in [1..n]$ gyengén teljes helyességi tételek interferencia-mentesek, ha $\forall i, j \in [1..n] : i \neq j$ folyamatpárra az S_i^*-nak egy kritikus utasítása sem interferál a $\{\{Q_j\}\}\ S_j^*\ \{\{R_j\}\}$ gyengén teljes helyességi tétellel.

Holtpont-mentesség

Blokkolt állapot: Egy párhuzamos program valamelyik folyamata blokkolt állapotba kerül, ha van benne egy várakozó utasítás, melynek előfeltétele Q és fennáll a $Q \wedge \neg\beta$.

Holtpont-állapot: Egy párhuzamos program holtpont állapotban van, ha legalább egy folyamata blokkolt, míg a többi vagy befejeződött, vagy azok is blokkoltak.

Holtpont-mentesség: Egy program egy állításra nézve holtpont-mentes, ha nincs olyan állapot, mely kielégíti az állítást viszont abból indítva a program holtpont-állapotba jut.

Általános párhuzamos program

Az S párhuzamos program szerkezetének legfelső szintjén szekvenciálisan követik egymást (tetszőleges sorrendben) párhuzamos blokkok (S_k) és várakozó utasítások (w_j). Ezen felül a párhuzamos blokkok komponensei $S_l^{(k)}$ is ugyanilyen szerkezetűek.

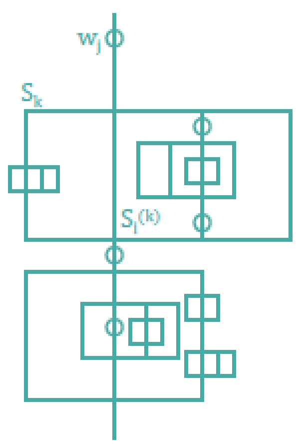

Figure 1: Általános párhuzamos program

Owicki-Gries holtpont-mentességi kritérium

Egy program akkor várakozik, ha:

- vagy valamelyik await utasításnál várakozik ($\text{pre}(w_j) \wedge \neg\beta_j$)
- vagy valamelyik parbegin-parend blokkban várakozik. ($D'(S_k)$)

$$D(S) = \Big(\bigvee_{j=1}^{n} (\mathrm{pre}(w_j) \wedge \neg\beta_j)\Big) \vee \Big(\bigvee_{k=1}^{m} D'(S_k)\Big)$$

Egy program a parbegin-parend blokkban akkor várakozik, vagy van legalább egy várakozó ága, miközben a többi végetért, vagy azok is várakoznak.

$$D'(S_k) = \Big(\bigvee_{i=1}^{l} D(S_i^{(k)})\Big) \wedge \Big(\bigwedge_{i=1}^{l} (\mathrm{post}(S_i^{(k)}) \vee D(S_i^{(k)}))\Big)$$

Ha $D(S)$ hamis, akkor az S program holtpont-mentes.

6. A kölcsönös kizárás és megvalósítása

Kritikus szakasz: Az interferencia vizsgálat szempontjából kritikusak egy folyamat azon utasításokból álló szakaszai (utasítás-szekvenciái) amelyek más párhuzamosan futó folyamatokkal közösen használnak egy erőforrást. Az ilyen kritikus szakaszoknak a működését az interferencia mentesség érdekében össze kell hangolni.

Kölcsönös kizárás

A kölcsönös kizárás egy adott erőforrás (erőforrás csoport) párhuzamos programbeli használatának egy lehetséges módja. Ennek során amíg egy folyamat egy adott közös erőforrás kritikus szakaszában van, addig a többi folyamat ugyanezen erőforrás kritikus szakaszában nem tartózkodhat: valami mást csinál vagy a kritikus szakaszba történő belépésre várakozik.

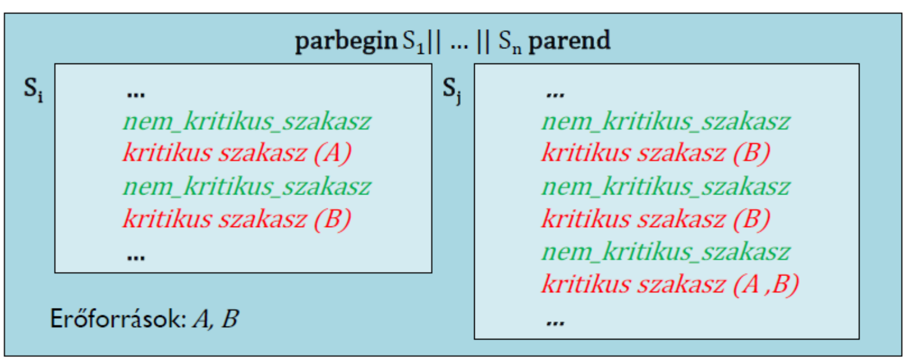

Figure 2: Kölcsönös kizárás

Kölcsönös kizárás megvalósítása

A kölcsönös kizárás megvalósításához a kritikus szakaszokat különleges belépő és kilépő szakaszokkal egészítjük ki, amelyek diszjunktak a folyamat többi részétől:

$$\big(\mathrm{var}(\mathrm{BK}_i) \cup \mathrm{var}(\mathrm{KK}_i)\big) \cap \big(\mathrm{var}(\mathrm{NK}_i) \cup \mathrm{var}(\mathrm{KS}_i)\big) = \emptyset$$

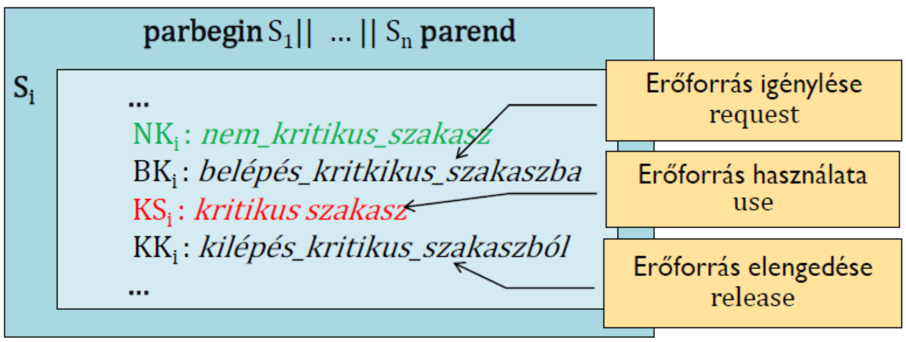

Figure 3: Kölcsönös kizárás megvalósítása

A kölcsönös kizárás akkor teljesül, ha:

- az $\{\mathrm{Ef}_i\}\ S_i^*\ \{\mathrm{Uf}_i\}$ annotációk interferencia-mentesek.
- és minden $i, j \in [1..n] : i \neq j$-re fennáll, hogy $\mathrm{pre}_{\mathrm{KS}_i} \wedge \mathrm{pre}_{\mathrm{KS}_j} \equiv \downarrow$

7. További források

- Előadás diasor