1 Alapfogalmak

Halmaz: *"Azonos tulajdonságú" elemek összessége.*
Halmaz jelölése: Latin ABC nagybetűi (általában).
Halmaz elemeinek jelölése: Latin kisbetűk (általában).
Halmaz megadása:

a) *elemeinek felsorolásával*, pl. $A = \{1, 2, 3, 5, 7\}$;
b) *az elemeit jellemző közös tulajdonság megadásával*:

$$A = \{x \mid P(x) \text{ igaz}\},$$

ahol $P(x)$ egy x-től függő tulajdonság (állítás). Pl.

$$A = \{x \mid 1 \leq x \leq 7, x \text{ prímszám}\}.$$

Feltevés: *Adott a objektum és A halmaz vonatkozásában el tudjuk dönteni, hogy a eleme-e az A halmaznak, vagy sem. Jelölés: $a \in A$ (a eleme A-nak), $a \notin A$ (a nem eleme A-nak).*
Részhalmaz: $B \subseteq A$, ha $\forall b \in B \Rightarrow b \in A$.
Halmazműveletek:

$$A \cup B = \{x \mid x \in A, \text{ vagy } x \in B\},$$

$$A \cap B = \{x \mid x \in A \text{ és } x \in B\}.$$

1.1 Definíció: *$A_1, A_2, \ldots, A_n$ tetszőleges halmazok direkt, vagy Descartes féle szorzata:*

$$A_1 \times A_2 \times \ldots \times A_n = \{(a_1, \ldots, a_n) \mid a_i \in A_i,\ i = 1, \ldots n\}.$$

Jelölés: $\times_{i=1}^{n} A_i$. □
Megjegyzés: *A direkt szorzat elemei rendezett elem n-esek.*
További jelölések:

$\mathbb{N}$ - természetes számok halmaza
$\mathbb{N}_0$ - nemnegatív egész számok halmaza ($\mathbb{N}_0 = \mathbb{N} \cup \{0\}$)
$\mathbb{Z}$ - egész számok halmaza
$\mathbb{Q}$ - racionális számok halmaza ($\mathbb{Q} = \left\{ \frac{p}{q} \mid p, q \in \mathbb{Z},\ q \neq 0 \right\}$)
$\mathbb{C}$ - komplex számok halmaza ($\mathbb{C} = \{a + bi \mid a, b \in R\}$, "$i = \sqrt{-1}$")
$\emptyset$ - üres halmaz
$[a..b]$ - az $[a, b] \subset R$ intervallum egész számainak halmaza

$$[a..b] := [a, b] \cap \mathbb{Z}$$

$\subset$ - valódi részhalmaz
$\subseteq$ - részhalmaz
$|A|$ - az A halmaz számossága (elemeinek száma)
$\wedge$ - logikai "és"
$\vee$ - logikai "vagy"

1.1 Relációk

1.2 Definíció: *Legyenek A és B tetszőleges halmazok. Tetszőleges $S \subseteq A \times B$ részhalmazt (bináris) relációnak nevezünk. Az $a \in A$ és $b \in B$ elemek S relációban állnak egymással (jelölés aSb) akkor és csak akkor, ha $(a, b) \in S$.* □
Megjegyzés: *A definíció rövidebben: $aSb \iff (a, b) \in S$.*

1.3 Definíció: *Reláció értelmezési tartománya:*

$$D_S = \{a \in A \mid \exists b \in B : (a,b) \in S\}.$$

1.4 Definíció: *Reláció értékkészlete:*

$$R_S = \{b \in B \mid \exists a \in A : (a,b) \in S\}.$$

1.5 Definíció: *Reláció értéke (metszete) egy adott helyen:*

$$S(a) = \{b \in B \mid (a,b) \in S\}.$$

1.6 Definíció: *Az S relációt determinisztikus, vagy parciális függvénynek nevezzük, ha*

$$|S(a)| \leq 1 \quad (\forall a \in A).$$

Megjegyzés: *Determinisztikus reláció esetén $S(a)$ vagy üres, vagy egyelemű halmaz.*
1.7 Definíció: *Az S relációt függvénynek nevezzük, ha*

$$|S(a)| = 1 \quad (\forall a \in D_S).$$

1.8 Definíció: *A $H \subseteq A$ halmaz S reláció szerinti képe (metszete):*

$$S(H) = \{b \in B \mid \exists a \in H : (a,b) \in S\}.$$

1.1 Feladat: *Igazoljuk, hogy*

$$S(H) = \cup_{a \in H} S(a).$$

1.9 Definíció: *Az $S^{(-1)}$ reláció az $S \subseteq A \times B$ reláció inverze, ha*

$$S^{(-1)} = \{(b,a) \in B \times A \mid (a,b) \in S\}.$$

1.10 Definíció: *A $H \subseteq B$ halmaz S reláció szerinti inverz képe:*

$$S^{(-1)}(H) = \{a \in A \mid S(a) \cap H \neq \emptyset\}$$

1.2 Feladat: *Igazoljuk, hogy az inverz kép az inverz reláció szerinti kép!*
1.11 Definíció: *Az $R \subseteq A \times C$ reláció a $P \subseteq A \times B$ és $Q \subseteq B \times C$ relációk kompozíciója (szorzata) (jelölés: $R = Q \bigcirc P$), ha*

$$R = \{(a,c) \in A \times C \mid \exists b \in B : (a,b) \in P \wedge (b,c) \in Q\}.$$

1.2 Sorozatok és projekciók

1.12 Definíció: *Legyen A ($A \neq \emptyset$) tetszőleges halmaz. Ekkor*

$$\alpha = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$$

($\alpha_i \in A$) egy A-beli véges sorozat. □
1.13 Definíció: *Az $\alpha = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$ véges sorozat hossza $|\alpha|$.*□
Megjegyzés: *Tulajdonképpen $|\alpha| = n$, illetve a sorozat, mint halmaz számossága.*
1.14 Definíció: *Az A halmazbeli véges sorozatok halmaza A^*.* □
Megjegyzés: *A^* az $\langle \alpha_1 \rangle$, $\langle \alpha_1, \alpha_2 \rangle$, $\langle \alpha_1, \alpha_2, \alpha_3 \rangle$, ... sorozatok összessége.*
1.15 Definíció: *A-beli végtelen sorozat:*

$$\alpha = \langle \alpha_1, \alpha_2, \ldots, \alpha_k, \ldots \rangle,$$

ahol $\alpha_i \in A$ ($\forall i$).

1.16 Definíció: *A-beli végtelen sorozatok halmaza* A^{∞}. □
1.17 Definíció: *Az A-beli véges és végtelen sorozatok halmaza* A^{**}. □
Megjegyzés:

$$A^{**} = A^{*} \cup A^{\infty}$$

1.18 Definíció: $\alpha \in A^{**}$ sorozat értelmezési tartománya D_{α}, ahol

$$D_{\alpha} = \begin{cases} [1..|\alpha|], & \text{ha } \alpha \in A^{*} \\ \mathbb{N}, & \text{ha } \alpha \in A^{\infty} \end{cases}$$

Megjegyzés: *A sorozat egy* $\mathbb{N} \to A$ *tipusú függvény értékkészlete.*
1.19 Definíció: *Az* $\alpha^1, \alpha^2, \ldots, \alpha^{n-1} \in A^{*}$ *és* $\alpha^n \in A^{**}$ *sorozatok egymásután írása, konkatenációja kon* $\left(\alpha^1, \alpha^2, \ldots, \alpha^{n-1}, \alpha^n\right)$. □

Ha $\alpha^i = \left\langle \alpha_1^i, \alpha_2^i, \ldots, \alpha_{k_i}^i \right\rangle$ $(1 \le i \le n-1)$ és $\alpha^n = \langle \alpha_1^n, \alpha_2^n, \ldots \rangle$, akkor a konkatenáció alakja:

$$\langle \underbrace{\alpha_1^1, \ldots, \alpha_{k_1}^1}_{\alpha^1}, \underbrace{\alpha_1^2, \ldots, \alpha_{k_2}^2}_{\alpha^2}, \ldots, \underbrace{\alpha_1^{n-1}, \ldots, \alpha_{k_{n-1}}^{n-1}}_{\alpha^{n-1}}, \underbrace{\alpha_1^n, \alpha_2^n, \ldots}_{\alpha^n} \rangle.$$

Példa: $A = \{a, \ldots, z\}$, $\alpha^1 = \langle o, k, o, s \rangle$, $\alpha^2 = \langle m, i, n, t, a, t, o, r \rangle$, $\alpha^3 = \langle d, a, i \rangle$, $\alpha^4 = \langle k, o, s \rangle$.

$$kon\left(\alpha^1, \alpha^2, \alpha^3, \alpha^4\right) = \langle o, k, o, s, m, i, n, t, a, t, o, r, d, a, i, k, o, s \rangle$$

1.20 Definíció: $\alpha \in A^{**}$ *sorozat redukáltja az a sorozat, amelyet úgy kapunk, hogy az* α *sorozat minden azonos elemből álló véges részsorozatát a részsorozat egyetlen elemével helyettesítjük. Jelölése:* $red\,(\alpha)$.
Példa: $A = \{a, \ldots, z\}$, $\alpha = \langle e, n, n, i, k, e, l, l, e, n, e \rangle$

$$red\,(\alpha) = \langle e, n, i, k, e, l, e, n, e \rangle\,.$$

1.21 Definíció: *Utolsó elem függvény* $\tau : A^{*} \to A$, *ahol* $\tau\,(\alpha) = \alpha_{|\alpha|}$ $(\alpha \in A^{*})$. □
Megjegyzés: $\tau\,(\alpha) = \alpha_n$, *ha* $\alpha = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$. *Az utolsó elem függvényt csak véges sorozatokra értelmezzük!*
Példa: $A = \{a, \ldots, z\}$, $\alpha = \langle u, b, u, l, k, u, t, y, u, l \rangle$, $|\alpha| = 10$, $\alpha_{10} = l$, $\tau\,(\alpha) = l$.
1.22 Definíció: *Legyen* $m, n \in \mathbb{N}$, $m \le n$, $1 \le i_1 < i_2 < i_3 < \ldots < i_m \le n$, $A = \times_{i=1}^{n} A_i$ *és* $B = \times_{j=1}^{m} A_{i_j}$. *A* $pr_B : A \to B$ *függvényt projekciónak nevezzük (A ortogonális projekciója B-re), ha*

$$pr_B\,(a) = (a_{i_1}, a_{i_2}, \ldots, a_{i_m}) \quad (\forall a = (a_1, a_2, \ldots, a_n) \in A)\,.$$

Példa: $A_1 = A_2 = R$, vetítés x, vagy y tengelyre.

A definícióban szereplő B-t az A alterének nevezzük. Ha $m < n$, akkor valódi altér. Csak $m \neq 0$ lehetséges $\Rightarrow \emptyset$ nem altere egyetlen direkt szorzatnak.
1.23 Definíció *(Projekció kiterjesztése "terek" direkt szorzataira): Legyen A és B mint előbb,* $(a_1, a_2) \in A \times A$. *Ekkor*

$$pr_B\,((a_1, a_2)) = (pr_B\,(a_1), pr_B\,(a_2)) \in B \times B.$$

1.24 Definíció *(Projekció kiterjesztése "sorozatterekre"): Legyen A és B mint előbb,* $\alpha = \langle \alpha_1, \alpha_2, \ldots, \alpha_i, \ldots \rangle \in A^{**}$. *Ekkor*

$$pr_B\,(\alpha) = \langle pr_B\,(\alpha_1), pr_B\,(\alpha_2), \ldots, pr_B\,(\alpha_i), \ldots \rangle \in B^{**}.$$

Megjegyzés: *Más formában ugyanez:*

$$pr_B\,(\alpha) = \beta \in B^{**}, \text{ ahol } \beta_i = pr_B\,(\alpha_i) \quad (\forall i \in D_{\beta} = D_{\alpha})\,.$$

2 A programozás alapfogalmai

Öt alapvető fogalmat definiálunk:

1. *Állapottér* (a számítógép memória modellje)
2. *Feladat* (a programozási feladat modellje)
3. *Program* (a programfutás modellje)
4. *Programfüggvény* (a programfutás eredménye)
5. *Feladat megoldása* (a feladat és a program viszonyának modellezése).

2.1 Definíció: *Legyenek $A_1, A_2, \ldots, A_n$ tetszőleges véges, vagy megszámlálhatóan végtelen halmazok. Az $A = A_1 \times A_2 \times \ldots \times A_n$ halmazt állapottérnek nevezzük, az A_i halmazokat pedig tipusértékhalmazoknak.* □

Megjegyzés:

1. Az állapottér a számítógép memória modellje. Az állapottér egy eleme a memória egy adott állapotának felel meg.

2. Az állapottér A_i komponensei az egyes jellemzők lehetséges értékeinek halmazai. Az állapottér komponensei között ugyanaz a (tipusérték) halmaz többször is szerepelhet.

3. A tipusértékhalmaz elnevezés azt jelzi, hogy ezek a halmazok bizonyos közös tulajdonságú (azonos tipusú) elemekből állnak.

2.2 Definíció: *Legyen A állapottér. Az $F \subseteq A \times A$ relációt (programozási) feladatnak nevezzük.* □

Megjegyzés: *A feladat egy "leképezés" az állapottéren: az állapottér minden D_F-beli pontjára megmondjuk, hogy hova kell belőle eljutni.*

Az állapottér megadása nem egyszerű.

Példa: P pont koordinátái R^2-ben, polárkoordináta rendszerben.

Egy program futásakor a számítógép memóriájának tartalma dinamikusan változik. Egy film tkp. fényképek sorozata. Ehhez hasonlóan a program futását az állapotok, vagyis állapottérbeli sorozatok megadásával modellezzük.

2.3 Definíció: *Az $S \subseteq A \times A^{**}$ relációt programnak nevezzük, ha*

1. $D_S = A$,
2. $\forall a \in A : \forall \alpha \in S(a) : \alpha_1 = a$,
3. $\forall \alpha \in R_S : \alpha = red(\alpha)$.

A definíció jelentése a következő:

1. triviális;

2. A program futását jellemző $\alpha \in A^{**}$ sorozat mindig abból az $a \in A$ pontból indul el, amelyhez hozzárendeltük;

3. $R_S = \{\alpha \in A^{**} \mid \exists a \in A : (a, \alpha) \in S\}$, bármelyik $\alpha \in R_S$ olyan sorozat, amelyben az állapotok nem ismétlődnek.

Összefoglalva: A program egy reláció, amelynek értékkészlete az állapottéren értelmezett sorozatokból áll. Ezek a sorozatok egy-egy konkrét program végrehajtást jellemeznek.

Kérdés: *Az $F \subseteq A \times A$ feladat és az $S \subseteq A \times A^{**}$ program viszonya?*

Az F feladat reláció az $a \in A$ "kezdeti memóriaállapothoz" egy (vagy több) olyan $b \in A$ "végső memóriaállapotot" (tkp. megoldást) rendel, amelyre igaz, hogy $(a, b) \in F$.

Az S program reláció ugyanehhez az a kezdeti állapothoz egy (vagy több) $\alpha \in A^{**}$ sorozatot rendel. Ha ez a sorozat véges ($\alpha \in A^*$), akkor a végső állapot, amely a sorozat utolsó eleme, a programfutás eredménye. Ennek a végső állapotnak a feladat b "megoldásához" való viszonyát kell jellemezni. A közbülső memóriaállapotok ebből a szempontból nem érdekesek, ill. ezeket nem vizsgáljuk.

A viszony jellemzése két lépésben történik.

2.4 Definíció: *A $p(S) \subseteq A \times A$ relációt az $S \subseteq A \times A^{**}$ program programfüggvényének nevezzük, ha*

1. $D_{p(S)} = \{a \in A \mid S(a) \subseteq A^*\}$,
2. $p(S)(a) = \{b \in A \mid \exists \alpha \in S(a) : \tau(\alpha) = b\}$.

A definíció jelentése a következő:

1. Csak olyan pontokban vizsgáljuk azt, hogy hova jut el a program, amelyekben a futás véges (a program nem száll el);

2. Tetszőleges $b \in A$ ponthoz, amelyre $(a,b) \in p(S)$, létezik véges, a program által előállított $\alpha \in A^*$ sorozat ($(a,\alpha) \in S$), amelynek utolsó eleme (utolsó programállapota) éppen b.

A programfüggvény a program futásának eredményét jellemzi. A függvény jelző nem igazán korrekt, ui. nem feltétlenül függvény, de a hagyomány ezt követeli.

Megjegyzés:

$$p(S)(a) = \{\tau(\alpha) \mid \alpha \in S(a)\} = \tau(S(a)) = (\tau \bigcirc S)(a).$$

2.5 Definíció: *Az $S \subseteq A \times A^{**}$ program megoldja az $F \subseteq A \times A$ feladatot, ha*

1. $D_F \subseteq D_{p(S)}$,

2. $\forall a \in D_F : p(S)(a) \subseteq F(a)$.

A definíció jelentése a következő:

1. A

$$D_F = \{a \in A \mid \exists b \in A : (a,b) \in F\} \subseteq D_{p(S)} = \{a \in A \mid S(a) \subseteq A^*\}$$

feltétel miatt az állapottér azon pontjaihoz, ahol a feladat értelmezve van, a program csak véges sorozatokat rendelhet;

2. A

$$p(S)(a) = \{b \in A \mid \exists \alpha \in S(a) : \tau(\alpha) = b\} \subseteq F(a) = \{c \in A \mid (a,c) \in F\}$$

feltétel azt fejezi ki, hogy a sorozatok végpontjait a feladat hozzárendeli a kezdőponthoz. Tehát a program által generált "véges" sorozatok végpontjai a feladat megoldásai.

Ezzel a program és a feladat viszonyát jellemeztük.

2.1 Példa: Legyen $A = \mathbb{Z}$ és az $x \to y(x) = \left(x^2-1\right)^2 + 1$ leképezés kiszámítása a "feladat". Ekkor

$$F = \left\{(x,y) \mid y = \left(x^2-1\right)^2 + 1\right\} \subset A \times A,$$

$D_F = A$ és $\forall a \in A$ esetén $F(a) = \left\{\left(a^2-1\right)^2 + 1\right\}$. Tekintsük a következő "programot":

$$x \xrightarrow{f_1} x^2 - 1 \xrightarrow{f_2} \left(x^2-1\right)^2 + 1.$$

A fenti formába átírva ez a következőképpen néz ki:

$$S = \left\{(x,\alpha) \mid \alpha = \left\langle x, x^2-1, \left(x^2-1\right)^2+1\right\rangle\right\} \subset A \times A^* \subset A \times A^{**}.$$

Igazoljuk, hogy S program. Világos, hogy $D_S = A$ (1. tulajdonság) és $\forall \alpha \in S(a) = \left\{\left\langle a, a^2-1, \left(a^2-1\right)^2+1\right\rangle\right\}$ esetén $\alpha_1 = a$ (2. tulajdonság). A 3. tulajdonság belátásához vegyük észre, hogy

$$\alpha \in R_S \Leftrightarrow \exists a \in A : \alpha = \left\langle a, a^2-1, \left(a^2-1\right)^2+1\right\rangle.$$

Egy α sorozat akkor és csak akkor redukált, ha $\alpha_i \neq \alpha_{i+1}$ $(\forall i)$. Esetünkben

$$a = a^2 - 1 \Leftrightarrow a = \frac{1 \pm \sqrt{5}}{2} \notin A$$

és

$$a^2 - 1 = \left(a^2-1\right)^2 + 1 \Leftrightarrow a = \pm\sqrt{\frac{1 \pm \sqrt{-3}}{2} - 1} \notin A.$$

Tehát az $\alpha \in R_S$ sorozatok redukáltak. Ezzel a 3. tulajdonságot is beláttuk. Tehát S program. A programfüggvény definíciója (Miért?):

$$p(S) = \{(a,b) \mid S(a) \subset A^*,\ \exists \alpha \in S(a) : \tau(\alpha) = b\}.$$

Minthogy $\forall a \in D_F = D_S = A$ esetén $S(a) = \left\{\left\langle a, a^2-1, \left(a^2-1\right)^2+1\right\rangle\right\}$, azért $\alpha \in S(a)$ esetén $\tau(\alpha) = \left(a^2-1\right)^2+1$. Tehát $D_{p(S)} = A$ (Miért?) és $p(S)(a) = \left\{\left(a^2-1\right)^2+1\right\}$. Igazoljuk, hogy az S program megoldja az F feladatot. Világos, hogy $D_F = A \subseteq D_{p(S)} = A$ (1. tulajdonság). A fentiek alapján ugyancsak világos, hogy

$$\forall a \in D_F : p(S)(a) = \left\{\left(a^2-1\right)^2+1\right\} \subseteq F(a) = \left\{\left(a^2-1\right)^2+1\right\}.$$

Kérdés: Miért nem program az $S' = \left\{(x,\alpha) \mid \alpha = \left\langle x, \left(x^2-1\right)^2+1\right\rangle\right\} \subseteq A \times A^{**}$ reláció?

2.1 Specifikáció

A fejezetben programok helyességével, ill. részleges (parciális) helyességével foglalkozunk. Ehhez szükségünk van a következő logikai fogalmakra és eredményekre.
Jelölés: (Logikai értékek halmaza) $\mathbb{L} = \{igaz, hamis\} = \{i, h\}$.
3.1 Definíció: *A halmazon értelmezett (logikai) állítás: $Q : A \to \mathbb{L}$ függvény.*
3.2 Definíció: *Legyen Q az A halmazon értelmezett állítás. A Q állítás igazsághalmaza*

$$[Q] = \{a \in A \mid Q(a) = igaz\}.$$

3.3 Definíció: *Legyenek Q_1 és Q_2 az A halmazon értelmezett állítások. A Q_1 és Q_2 állítások ekvivalensek, ha $[Q_1] = [Q_2]$. Jelölés: $Q_1 \equiv Q_2$.*
3.4 Definíció: *Legyen $R \subseteq A$ tetszőleges részhalmaz. $P(R)$ olyan állítást jelöl, amelyre $[P(R)] = R$.*
Következmény: *Tetszőleges Q állításra igaz, hogy $Q \equiv P([Q])$.*
3.5 Definíció: *Legyenek P és Q az A halmazon értelmezett állítások. A következő logikai műveleteket definiáljuk, un. igazságtáblával:*

1. *$P \wedge Q$ (konjunkció/és/logikai szorzás):*

P	i	i	h	h
Q	i	h	i	h
$P \wedge Q$	i	h	h	h

A $P \wedge Q$ állítás igaz $\iff$ P és Q is igaz;

2. *$P \vee Q$ (diszjunkció/vagy/logikai összeadás):*

P	i	i	h	h
Q	i	h	i	h
$P \vee Q$	i	i	i	h

A $P \vee Q$ állítás igaz $\iff$ P és Q közül legalább az egyik igaz;

3. *$\neg Q$ (negáció/ tagadás):*

Q	i	h
$\neg Q$	h	i

A $\neg Q$ állítás igaz $\iff$ Q hamis, az állítás hamis$\iff$ Q igaz;

4. *$P \Rightarrow Q$ (implikáció/ következés/ha P, akkor Q):*

P	i	i	h	h
Q	i	h	i	h
$P \Rightarrow Q$	i	h	i	i

A $P \Rightarrow Q$ állítás hamis$\iff$ P igaz és Q hamis.
3.1 Állítás: *Legyenek P és Q az A halmazon értelmezett állítások. Ekkor*
(i) $[P \wedge Q] = [P] \cap [Q]$;

(ii) $[P \vee Q] = [P] \cup [Q]$*;*
(iii) $[\neg Q] = A \setminus [Q]$*;*
(iv) Ha $P \Rightarrow Q$*, akkor*$[P] \subseteq [Q]$*.*

Egy S program akkor és csak akkor oldja meg az F feladatot, ha

$$D_F \subseteq D_{p(S)} \wedge \{a \in D_F \Rightarrow p\,(S)\,(a) \subseteq F\,(a)\}\,.$$

A megoldás definíciójának közvetlen ellenőrzése helyett elégséges feltételt adunk meg a program helyességének ellenőrzésére. Ezt az eredményt specifikáció tételnek nevezzük. A tétel megfogalmazásához két fogalmat vezetünk be:

1. Leggyengébb előfeltétel
2. Paramétertér

Legyen $R : A \to \mathbb{L}$ egy olyan feltétel (*utófeltétel*), amelyet az S program eredményétől megkövetelünk, azaz teljesülése esetén az eredményt helyesnek tekintjük. Ehhez keresünk egy olyan $Q : A \to \mathbb{L}$ *előfeltételt*, amelyet a program kezdőállapotára rovunk ki és amelynek teljesülése esetén a program terminál és végállapotára (eredményére) fennáll az R feltétel:

$$[Q] \subseteq D_{p(S)} \wedge \{a \in [Q] \Rightarrow p\,(S)\,(a) \subseteq [R]\}\,.$$

A legbővebb igazsághalmazzal rendelkező előfeltételt leggyengébb előfeltételnek nevezzük.
3.6 Definíció: *Legyen* $S \subseteq A \times A^{**}$ *program,* R *az* A *állapottéren értelmezett állítás. Az* S *program* R *utófeltételhez tartozó leggyengébb előfeltétele az* $lf\,(S, R)$ *állítás, amelyre*

$$[lf\,(S, R)] = \left\{a \in D_{p(S)} \mid p\,(S)\,(a) \subseteq [R]\right\}.$$

Megjegyzés: *Egy feladat megoldásakor olyan programot keresünk, amelyik bizonyos feltételeket kielégítő pontokban terminál. Ha a számunkra kedvező végállapotokra megadjuk a program leggyengébb előfeltételét, akkor a programfüggvény nélkül tudjuk jellemezni a program működését. Az* R *utófeltétel megválasztásától függően* $[lf\,(S, R)]$ *és* D_F *különbözhet. Ezért a program helyességének ez a jellemzése csak részleges (parciális).*
3.1 Példa: Legyen $A = \mathbb{N}_0$, $F = \{(x, y) \mid y = 2x + 1\} \subseteq A \times A$, $S = \{(a, \alpha) \mid \alpha = \langle a, 2a + 1\rangle\}$. Ekkor $p\,(S)\,(a) = \{2a + 1\}$ és $D_{p(S)} = A$. Legyen az utófeltétel $R : y = 13$, ahol y a program eredményét jelöli. Az R utófeltétel igazsághalmaza: $[R] = \{13\}$. A $p\,(S)\,(a) = \{2a + 1\} \subseteq [R] = \{13\}$ feltételből $a = 6$ adódik. Tehát $[lf\,(S, R)] = \{6\}$ és $lf\,(S, R) : a = 6$. Ha az S programot az "$y := 2a + 1$" formában adjuk meg, akkor az eredményt a következőképpen is megkaphatjuk:

$$lf\,(S, R) = lf\,(\text{"}y := 2a + 1\text{"}, y = 13) = \{2a + 1 = 13\} = \{a = 6\}\,.$$

Kérdés: Program lesz-e S, ha $\mathbb{N}_0$ helyett $\mathbb{Z}$-t választunk A gyanánt?
3.2 Példa (A 2.1 Példa folytatása): Legyen $A = \mathbb{Z}$,

$$F = \left\{(x, y) \mid y = \left(x^2 - 1\right)^2 + 1\right\}$$

és $S = \left\{(x, \alpha) \mid \alpha = \left\langle x, x^2 - 1, \left(x^2 - 1\right)^2 + 1\right\rangle\right\}$. Ekkor

$$p\,(S)\,(a) = \left\{\left(a^2 - 1\right)^2 + 1\right\}$$

és $D_{p(S)} = A$. Legyen az S program utófeltétele $R : y \geq 1 \wedge y \leq 17$, ahol y a program eredménye. Az R utófeltétel igazsághalmaza: $[R] = \{1, 2, \ldots, 17\} = [1..17]$. A

$$p\,(S)\,(a) = \left\{\left(a^2 - 1\right)^2 + 1\right\} \subseteq [R] = [1..17]$$

feltételből $-2 \leq a \leq 2$ adódik. Tehát $[lf\,(S,R)] = [-2..2]$, amelyhez választhatjuk az $lf\,(S,R) : -2 \leq a \leq 2$ állítást, mint leggyengébb előfeltételt.

3.1 Feladat: Adjunk meg más előfeltételt is!

3.3 Példa (A 3.2 Példa folytatása): Legyen az S program új utófeltétele: $R : y = 18 \vee y = 19$. Az $\left\{\left(a^2-1\right)^2+1\right\} \subseteq [R] = \{18,19\}$ tartalmazási feltételnek egész a-ra nincs megoldása. Tehát csak $[lf\,(S,R)] = \emptyset$ és $lf\,(S,R) \equiv h$ lehetséges. Ha viszont az $R : y \in \mathbb{N}$ feltételt választjuk, akkor $[R] = \mathbb{N}$, $[lf\,(S,R)] = \mathbb{Z} = A$ és $lf\,(S,R) \equiv i$.

3.1 Tétel *(Dijkstra): Legyen $S \subseteq A \times A^{**}$ program, R és Q az A halmazon értelmezett állítások, és $HAMIS$ az azonosan hamis állítás. Ekkor*

1. $lf\,(S, HAMIS) = HAMIS$,
2. *Ha $Q \Rightarrow R$, akkor $lf\,(S,Q) \Rightarrow lf\,(S,R)$,*
3. $lf\,(S,Q) \wedge lf\,(S,R) = lf\,(S, Q \wedge R)$,
4. $lf\,(S,Q) \vee lf\,(S,R) \Rightarrow lf\,(S, Q \vee R)$.

Bizonyítás:

1. Indirekt: Tegyük fel, hogy $\exists a \in [lf\,(S, HAMIS)]$. Ekkor definíció szerint $a \in D_{p(S)}$ és $p\,(S)\,(a) \subseteq [HAMIS] = \emptyset$. Ez ellentmondás.

2. Tegyük fel, hogy $a \in [lf\,(S,Q)]$. Ekkor $p\,(S)\,(a) \subseteq [Q]$. Minthogy $[Q] \subseteq [R]$, azért $p\,(S)\,(a) \subseteq [R]$ és $a \in [lf\,(S,R)]$.

3.

$$\begin{aligned} [lf\,(S,Q)] &= \left\{a \in D_{p(S)} \mid p\,(S)\,(a) \subseteq [Q]\right\} \\ [lf\,(S,R)] &= \left\{a \in D_{p(S)} \mid p\,(S)\,(a) \subseteq [R]\right\} \end{aligned}$$

miatt

$$\begin{aligned} [lf\,(S,Q)] \cap [lf\,(S,R)] &= \left\{a \in D_{p(S)} \mid p\,(S)\,(a) \subseteq [Q] \wedge p\,(S)\,(a) \subseteq [R]\right\} \\ &= \left\{a \in D_{p(S)} \mid p\,(S)\,(a) \subseteq [Q \wedge R]\right\} \\ &= [lf\,(S, Q \wedge R)]\,. \end{aligned}$$

4. Legyen $a \in [lf\,(S,Q) \vee lf\,(S,R)]$. Ekkor $a \in [lf\,(S,Q)]$, vagy $a \in [lf\,(S,R)]$. Felhasználjuk, hogy $Q \Rightarrow Q \vee R$ és $R \Rightarrow Q \vee R$. Ha $a \in [lf\,(S,Q)]$, akkor a 2. állítás alapján $a \in [lf\,(S, Q \vee R)]$. Ha $a \in [lf\,(S,R)]$, akkor a 2. állítás alapján ismét $a \in [lf\,(S, Q \vee R)]$. Q.E.D.

Megjegyzés:

1. Az 1. állítás a "csoda kizárása".
2. A 2. állítást monotonitási tulajdonságnak nevezzük.

3.7 Definíció: *Legyen $F \subseteq A \times A$ feladat. A B halmazt a feladat paraméterterének nevezzük, ha van olyan $F_1 \subseteq A \times B$ és $F_2 \subseteq B \times A$ reláció, hogy $F = F_2 \bigcirc F_1$.* $\square$

Megjegyzés:

1. *A paramétertér általában az állapottér egy olyan altere, amelytől a feladat függ.*
2. *Triviális paraméterteret mindig tudunk választani:*

$$F_1 = id_A = \{(a,a) \mid a \in A\}\,, \quad B = A, \quad F_2 = F.$$

3.2 Tétel *(Specifikáció tétele): Legyen $F \subseteq A \times A$ feladat, B az F egy paramétertere, $F_1 \subseteq A \times B$, $F_2 \subseteq B \times A$, $F = F_2 \bigcirc F_1$. Legyen $b \in B$ és legyenek Q_b és R_b olyan állítások, amelyek igazsághalmazai*

$$[Q_b] = \{a \in A \mid (a,b) \in F_1\} = F_1^{(-1)}\,(b)\,,$$

$$[R_b] = \{a \in A \mid (b,a) \in F_2\} = F_2\,(b)\,.$$

Ha $\forall b \in B$ esetén $Q_b \Rightarrow lf\,(S, R_b)$, akkor az S program megoldja az F feladatot.

Bizonyítás: Két tulajdonságot kell belátnunk: 1. $D_F \subseteq D_{p(S)}$ és 2. $\forall a \in D_F : p\,(S)\,(a) \subseteq F\,(a)$. Legyen $a \in D_F$ tetszőleges. Ekkor $\exists b \in B : a \in [Q_b]$. A tétel feltevése miatt $[Q_b] \subseteq [lf\,(S, R_b)] =$

$\left\{a \in D_{p(S)} \mid p(S)(a) \subseteq [R_b]\right\} \subseteq D_{p(S)}$. Tehát $D_F \subseteq D_{p(S)}$. A $b \in F_1(a)$ tartalmazási feltétel miatt $F_2(b) \subseteq F_2(F_1(a))$ és

$$p(S)(a) \subseteq [R_b] = F_2(b) \subseteq F_2(F_1(a)) = F(a).$$

Q.E.D.

Megjegyzés: *A tétel csak elégséges és nem megfordítható. A tétel jelentősége sokrétű: (i) Adott S program esetén csak a $Q_b \Rightarrow lf(S, R_b)$ $(\forall b \in B)$ tulajdonságot kell belátnunk a program helyességéhez; (ii) Adott Q_b és R_b esetén olyan S programot kell keresnünk, amelyik kielégíti a $Q_b \Rightarrow lf(S, R_b)$ $(\forall b \in B)$ feltételt; (iii) Az F feladat szerencsés paraméterezése segítheti a program helyességének ellenőrzését, vagy akár a megfelelő program keresését.*

3.4 Példa (A 2.1 Példa módosítása): Legyen $A = \mathbb{Z}$,

$$F = \left\{\left(x, \left(x^2+1\right)^2 - 1\right) \mid x \in A\right\}$$

és

$$S = \left\{(x, \alpha) \mid \alpha = \left\langle x, x^2+1, \left(x^2+1\right)^2 - 1\right\rangle\right\} \subseteq A \times A^{**}.$$

A 2.1 Példához hasonlóan beláthatjuk, hogy S program és megoldja az F feladatot. Alkalmazzuk azonban a specifikáció tételét! Legyen $B = \mathbb{N}$, $F_1 = \left\{\left(x, x^2+1\right) \mid x \in A\right\}$ és $F_2 = \left\{\left(u, u^2-1\right) \mid u \in \mathbb{N}\right\}$. Ekkor $F = F_2 \bigcirc F_1$ és B paramétertér. Legyen $b \in \mathbb{N} = B$ rögzített. Definíció szerint kapjuk, hogy

$$[Q_b] = \{a \in \mathbb{Z} \mid (a, b) \in F_1\} = \left\{a \in \mathbb{Z} \mid a^2 + 1 = b\right\}$$

és $Q_b : a^2 + 1 = b$. Hasonlóképpen adódik, hogy

$$[R_b] = \{a \in \mathbb{Z} \mid (b, a) \in F_2\} = \left\{a \in \mathbb{Z} \mid a = b^2 - 1\right\}$$

és $R_b : a = b^2 - 1$. Ugyancsak definíció szerint kapjuk, hogy $p(S)(a) = \left\{\left(a^2+1\right)^2 - 1\right\}$,

$$[lf(S, R_b)] = \left\{a \in A \mid \left\{\left(a^2+1\right)^2 - 1\right\} \subseteq \left\{b^2 - 1\right\}\right\} = \left\{a \in A \mid a^2 + 1 = b\right\}$$

és $lf(S, R_b) : a^2 + 1 = b$. Ha $b \in B$ olyan, hogy $Q_b = i$, akkor $a^2 + 1 = b$ és $lf(S, R_b) = i$. Ha $Q_b = h$, akkor $a^2 + 1 \neq b$ és $lf(S, R_b) = h$. Tehát minden $b \in B$ esetén $Q_b \Rightarrow lf(S, R_b)$ (tkp. $Q_b \equiv lf(S, R_b)$). A specifikáció tétele miatt az S program megoldja az F feladatot (azaz S helyes program).

2.1.1 A változó fogalma

A változók használata egyszerűsítéseket tesz lehetővé.

3.8 Definíció: *Legyen $A = A_1 \times A_2 \times \ldots \times A_n$ állapottér. A $pr_{A_i} : A \to A_i$ projekciós függvényeket változóknak nevezzük:*

$$pr_{A_i}(a) = a_i \quad (\forall a = (a_1, a_2, \ldots, a_n) \in A).$$

A változók jelölése: $v_i = pr_{A_i}$.

Megjegyzés: *A változókra fennáll, hogy*

$$D_{v_i} = A = D_{pr_{A_i}}, \quad R_{v_i} = A_i = R_{pr_{A_i}}.$$

Az $R_{v_i} = A_i$ összefüggés miatt beszélünk a változó tipusáról, amely azonos A_i típusával.

Legyen $B = A_{i_1} \times \ldots \times A_{i_m}$. Ekkor a $pr_B : A \to B$ projekciót a $pr_B = (v_{i_1}, \ldots, v_{i_m})$, azaz a $pr_B(a) = (v_{i_1}(a), \ldots, v_{i_m}(a))$ előírással adhatjuk meg. Tetszőleges $f \bigcirc pr_B = f \bigcirc (v_{i_1}, \ldots, v_{i_m})$ alakú összetett függvényt, ha ez nem okoz félreértést, az $f(v_{i_1}, \ldots, v_{i_m})$ alakban is írunk.

A specifikáció tételében szerepelnek a $Q_b, R_b : A \to \mathbb{L}$ állítások, illetve ezek $[Q_b] = \{a \in A \mid (a, b) \in F_1\}$ és $[R_b] = \{a \in A \mid (b, a) \in F_2\}$ igazsághalmazai. Tegyük fel, hogy $\widetilde{F}_1 : A \times B \to \mathbb{L}$ és $\widetilde{F}_2 : A \times B \to \mathbb{L}$ olyan predikátumok, amelyekre $Q_b \equiv \widetilde{F}_1(a, b)$ és $R_b \equiv \widetilde{F}_2(a, b)$. Legyen $v_i : A \to A_i$ az A állapottér i-edik

változója ($i = 1, \ldots, n$), $p_j : B \to B_j$ a $B = B_1 \times \ldots \times B_m$ paramétertér j-edik változója ($j = 1, \ldots, m$). Ekkor $a = (v_1(a), \ldots, v_n(a))$ és $b = (p_1(b), \ldots, p_m(b))$ miatt írhatjuk, hogy

$$Q_b = Q_{p_1,\ldots,p_m} = \widetilde{F}_1(a, b) = \widetilde{F}_1(v_1, \ldots, v_n, p_1, \ldots, p_m)$$

és

$$R_b = R_{p_1,\ldots,p_m} = \widetilde{F}_2(a, b) = \widetilde{F}_2(v_1, \ldots, v_n, p_1, \ldots, p_m).$$

3.5 Példa: Legyen $A = \mathbb{Z}\times\mathbb{Z}$, $x, y : A \to \mathbb{Z}$ változó. Az $R = (x > 0 \wedge y > 0)$ jelölés az $(x(a) > 0 \wedge y(a) > 0 \wedge a \in A)$ állítást jelöli és

$$[R] = \{(a_1, a_2) \in A \mid a_1 > 0 \wedge a_2 > 0\}.$$

Ha π predikátum ($A \to \mathbb{L}$ tipusú logikai függvény), akkor a $\pi = (x > y)$ azt jelenti, hogy

$$\pi(a) = \begin{cases} \text{igaz, ha } x(a) > y(a) \\ \text{hamis, ha } x(a) \leq y(a). \end{cases}$$

3.6 Példa: Legyen a feladat két egész szám maximumának meghatározása. Legyen az állapottér $A = \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$, az állapottér változói pedig legyenek x, y, z. A feladat paramétertere legyen $B = \mathbb{Z} \times \mathbb{Z}$ a paramétertér változói pedig legyenek x', y',

$$F_1 = \{((x, y, z), (x', y')) \mid x = x' \wedge y = y'\},$$

$$F_2 = \{((x', y'), (x, y, z)) \mid z = \max\{x', y'\}\},$$

$$Q_b = Q_{x',y'} = (x = x' \wedge y = y'),$$

$$R_b = R_{x',y'} = (z \geq x' \wedge z \geq y' \wedge (z = x' \vee z = y')).$$

Tehát olyan S programot kell találnunk, amelyikre $\forall b \in B$ esetén fennáll

$$\begin{aligned} x(a) &= x'(b) \wedge y(a) = y'(b) \Rightarrow \\ &= lf(S, (z(a) \geq x'(b) \wedge z(a) \geq y'(b) \wedge (z(a) = x'(b) \vee z(a) = y'(b)))). \end{aligned}$$

Ezt rövidebben is írhatjuk:

$$x = x' \wedge y = y' \Rightarrow lf(S, (z \geq x' \wedge z \geq y' \wedge (z = x' \vee z = y'))).$$

3.2 Feladat: Kétféleképpen is igazoljuk, hogy az

$$S = \left\{((x, y, z), \alpha) \mid \alpha = \left\langle (x, y, z), (x + 1, y, z), \left(x, y, \frac{x + y}{2} + \frac{|x - y|}{2}\right)\right\rangle\right\}$$

program megoldja a 3.6 Példa feladatát! Miért nem program S, ha elhagyjuk az α sorozat második tagját?

2.2 Elemi programok

4.1 Definíció: *Az $S \subseteq A \times A^{**}$ program elemi, ha*

$$\forall a \in A : S(a) \subseteq \{\langle a\rangle, \langle a, a, a, \ldots\rangle, \langle a, b\rangle \mid b \neq a\}.$$

A definíció alapján könnyen látható, hogy egy elemi program tényleg program. Speciális elemi programok a kövekezők:

4.2 Definíció: *Üres, vagy skip program:*

$$\forall a \in A : SKIP(a) = \{\langle a\rangle\}.$$

A SKIP program nem csinál semmit.

4.3 Definíció: *A törlődés, vagy abort program:*

$$\forall a \in A : ABORT\,(a) = \{\langle a, a, a, \ldots\rangle\}\,.$$

Az ABORT program soha nem terminál (soha nem fejeződik be).

4.1 Feladat: Mely feladatok megoldásai lehetnek a SKIP és ABORT programok?

4.4 Definíció: *Legyen $F \subseteq A \times A$. Az S programot általános értékadásnak nevezzük, ha*

$$\begin{aligned} S \quad = \quad & \{(a, red\,(\langle a, b\rangle)) \mid a, b \in A \wedge a \in D_F \wedge b \in F\,(a)\} \cup \\ & \{(a, \langle a, a, a, \ldots\rangle) \mid a \in A \wedge a \notin D_F\}\,. \end{aligned}$$

4.5 Definíció: *Legyen $S \subseteq A \times A^{**}$ általános értékadás program.*

a) Ha $D_F = A$, akkor az S programot értékkiválasztásnak nevezzük és $a :\in F\,(a)$-val jelöljük.

b) Ha az F reláció függvény, akkor az S programot értékadásnak nevezzük és $a := F\,(a)$-val jelöljük.

c) Ha $D_F \subset A$, akkor S parciális értékkiválasztás.

d) Ha $D_F \subset A$ és F determinisztikus (F parciális függvény), akkor S parciális értékadás.

4.6 Definíció: *Identitásfüggvény:*

$$id_A = \{(a, a) \mid a \in A\} \subseteq A \times A.$$

4.1 Tétel *(elemi programok programfüggvénye):*

1. $p\,(SKIP) = id_A$,
2. $p\,(ABORT) = \emptyset$,
3. $p\,(a := F\,(a)) = F$,
4. $p\,(a :\in F\,(a)) = F$.

Bizonyítás: Házi feladat.

A következőkben elemi programok leggyengébb előfeltételeit határozzuk meg, Legyen R egy tetszőleges utófeltétel. Ekkor $p\,(SKIP)\,(a) = \{a\}$ miatt

$$[lf\,(SKIP, R)] = \{a \in A \mid p\,(SKIP)\,(a) \subseteq [R]\} = \{a \in A \mid \{a\} \subseteq [R]\} = [R]$$

és

$$lf\,(SKIP, R) = R.$$

Hasonlóan $p\,(ABORT)\,(a) = \emptyset$ miatt

$$lf\,(ABORT, R) = hamis.$$

Az általános értékadás leggyengébb előfeltétele négy esetben:

1. $F : A \to A$ függvény, $D_F = A$.

$$[lf\,(a := F\,(a)\,, R)] = \{a \in A \mid F\,(a) \subseteq [R]\} = F^{(-1)}\,([R])\,.$$

2. $F : A \to A$ függvény, $D_F \subset A$.

$$[lf\,(a := F\,(a)\,, R)] = \{a \in A \mid F\,(a) \subseteq [R]\} \cap D_F = F^{(-1)}\,([R]) \cap D_F.$$

3. $F \subseteq A \times A$, $D_F = A$, F nem determinisztikus.

$$[lf\,(a :\in F\,(a)\,, R)] = \{a \in A \mid F\,(a) \subseteq [R]\} = F^{(-1)}\,([R])\,.$$

4. $F \subseteq A \times A$, $D_F \subset A$, F nem determinisztikus.

$$[lf\,(a :\in F\,(a)\,, R)] = \{a \in A \mid F\,(a) \subseteq [R]\} \cap D_F = F^{(-1)}\,([R]) \cap D_F.$$

Minthogy definíció alapján $F^{(-1)}([R]) = [R \bigcap F]$, azért az 1. és 3. esetekben a leggyengébb előfeltétel $R \bigcap F$.
Az értékadást változókkal is leírjuk. Legyen $A = \times_{i=1}^{n} A_i$, $F \subseteq A \times A$ és

$$F(a) = F_1(a) \times F_2(a) \times \ldots \times F_n(a) \quad (a \in D_F),$$

ahol $F_i \subseteq A \times A_i$. Legyenek az állapottér változói $x_1, x_2, \ldots, x_n$. Ekkor az $a := F(a)$ program jelölése:

$$x_1, \ldots, x_n := F_1(x_1, \ldots, x_n), \ldots, F_n(x_1, \ldots, x_n).$$

A jelölésből elhagyhatjuk az x_i-t és $F_i(x_1, \ldots, x_n)$-t, ha $F_i(x_1, \ldots, x_n) = x_i$. Ha valamely F_j nem függ valamelyik x_i-től, akkor ezt is elhagyhatjuk a jelölésből. Ha a fenti értékadásban csak egy komponens szerepel, akkor egyszerű, egyébként pedig szimultán értékadásról beszélünk.
4.1 Példa: Legyen $A = \underset{x}{\mathbb{Z}} \times \underset{l}{\mathbb{L}}$, x a $\mathbb{Z}$, l pedig az $\mathbb{L}$ állapottér komponenshez tartozó változó. Legyenek az értékadás komponensei: $\forall a = (a_1, a_2) \in A$:

$$\begin{aligned} F_1(a_1, a_2) &= a_1, \text{ azaz } F_1 = pr_{\mathbb{Z}}, \text{ és} \\ F_2(a_1, a_2) &= (a_1 > 0). \end{aligned}$$

Ekkor az $a := F(a)$ értékadás változókkal felírva:

$$x, l := x, (x > 0).$$

A jelölés fent leírt egyszerűsítéseit elvégezve az

$$l := (x > 0)$$

egyszerű értékadást kapjuk.

2.3 Programkonstrukciók

Olyan műveletekkel foglalkozunk, amelyekkel meglévő programjainkból újakat állíthatunk elő. Sokféle konstrukció képzelhető el, de mi csak háromféle konstrukciós műveletet fogunk megengedni:
- szekvencia (sorozat),
- elágazás,
- ciklus.

A struktúrált programozás alaptétele (Böhm-Jacopini, 1966) kimondja: bármely program megadható ekvivalens struktúrált program formájában is, amelyben csak a fenti három konstrukciós művelet szerepel. Két program ekvivalens, ha ugyanazon input értékekre ugyanazon output értékeket számolják ki.

Az első konstrukciós technika a *szekvencia*, vagy *sorozat*. Itt két program közvetlen egymásután való végrehajtásáról van szó. Legyen S_1 és S_2 a két program, $a \in A$ és $\alpha \in S_1(a) \cap A^*$ az S_1 program által előállított véges sorozat. Az S_2 program az S_1 program $\tau(\alpha)$ végállapotából indul és a $\beta \in S_2(\tau(\alpha))$ véges, vagy végtelen sorozatot állítja elő. Tehát az $a \in A$ állapotból kiindulva a két program egymásutáni végrehajtása a $kon(\alpha, \beta)$ egyesített sorozatot eredményezi. Ez azonban nem redukált, mert $\tau(\alpha) = \beta_1$. Ezért az egyesített sorozat redukáltját tekintjük a szekvencia eredményének. Legyen $\alpha \in A^*$, $\beta \in A^{**}$ és

$$\chi_2(\alpha, \beta) = red(kon(\alpha, \beta)).$$

5.1 Definíció: *Legyenek $S_1, S_2 \subseteq A \times A^{**}$ programok. Az $S \subseteq A \times A^{**}$ relációt az S_1 és S_2 programok szekvenciájának nevezzük, és $(S_1; S_2)$-vel jelöljük, ha $\forall a \in A$:*

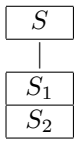
$$\begin{aligned} S(a) &= \{\alpha \in A^{\infty} \mid \alpha \in S_1(a)\} \cup \\ &\quad \cup \{\chi_2(\alpha, \beta) \in A^{**} \mid \alpha \in S_1(a) \cap A^* \wedge \beta \in S_2(\tau(\alpha))\}. \end{aligned}$$

Megjegyzés:

1. Ha a két program értékkészlete csak véges sorozatokat tartalmaz, akkor a szekvencia is csak véges sorozatokat rendel hozzá az állapottér pontjaihoz.

2. Determinisztikus programok szekvenciája is determinisztikus reláció (függvény).

A programkonstrukciókat többféleképpen is ábrázolhatjuk. Ezek egyike a *struktogram.* Legyen S_1, S_2 program. Az $S = (S_1; S_2)$ szekvencia struktogramja:



5.1 Feladat: Adjuk meg az $S = (S_1; S_2)$ szekvencia által megoldott feladatot, ha S_1 és S_2 általános értékadás program!

A második konstrukciós technika az *elágazás* (feltételes utasítás, case szerkezet), ahol más-más programot hajtunk végre feltételtől függően.

5.2 Definíció: *Legyenek $\pi_1, \ldots, \pi_m : A \to \mathbb{L}$ feltételek, $S_1, \ldots, S_m$ programok A-n. Ekkor az $IF \subseteq A \times A^{**}$ relációt az S_i-kből képezett π_i-k által meghatározott elágazásnak nevezzük, és $(\pi_1 : S_1, \ldots, \pi_m : S_m)$-vel jelöljük, ha $\forall a \in A$:*

$$IF(a) = (\cup_{i=1}^{m} w_i(a)) \cup w_0(a),$$

ahol $\forall i \in [1..m]$:

$$w_i(a) = \begin{cases} S_i(a), \text{ ha } a \in [\pi_i] \\ \emptyset, \text{ különben} \end{cases}$$

és

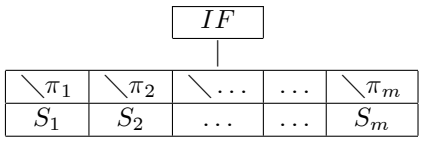
$$w_0(a) = \begin{cases} \langle a, a, a, \ldots \rangle, \text{ ha } a \notin \cup_{i=1}^{m} [\pi_i] \\ \emptyset, \text{ különben.} \end{cases}$$

Megjegyzés:

1. Ha a feltételek nem diszjunktak, akkor az elágazás bármelyik olyan S_i ág választását megengedi, amelyre π_i igaz.

2. Az elágazásoktól a gyakorlatban azt is megköveteljük, hogy a feltételek diszjunktak legyenek és $\cup_{i=1}^{m} [\pi_i] = A$ teljesüljön (Miért?).

Az $IF = (\pi_1 : S_1, \ldots, \pi_m : S_m)$ elágazás struktogramja:



5.2 Feladat: Milyen Pascal szerkezetnek felel meg a $(\pi : S; \neg\pi : SKIP)$ elágazás?

A harmadik konstrukciós technika a *ciklus*, amelyben egy meglévő programot egy adott feltételtől függően valahányszor végrehajtunk. Legyen S a program és π az adott feltétel. Ezután a következőképpen járunk el:

1. Ha az $a \in A$ kiinduló pontban π nem igaz, akkor az S programot nem hajtjuk végre és a ciklusból kilépünk.

2. Ha π igaz, akkor az S programot végrehajtjuk és eredményül kapunk egy α sorozatot.

3. Ha α véges és a $\tau(\alpha)$ pontban π nem igaz, akkor kilépünk a ciklusból.

4. Ha α véges és a $\tau(\alpha)$ pontban π igaz, akkor a $\tau(\alpha)$ pontból indulva megismételjük az előző lépéseket a 2. ponttól.

A vázolt eljárással - n ismétlést (iterációt) feltéve- az $\alpha^1, \alpha^2, \ldots, \alpha^n$ sorozatokat kapjuk, amelyekre fennáll, hogy $\alpha^1, \ldots, \alpha^{n-1} \in A^*$, $\alpha^1 \in S(a)$, $\alpha^2 \in S\left(\tau\left(\alpha^1\right)\right)$, ...,$\alpha^n \in S\left(\tau\left(\alpha^{n-1}\right)\right)$. Ha α^n végtelen, akkor sem S, sem a ciklus nem fejeződik be. Ha α^n véges, akkor két eset lehetséges. Ha π igaz a $\tau(\alpha^n)$ pontban, akkor folytatjuk a ciklust a $\tau(\alpha^n)$ pontból kiindulva. Ha π nem igaz, akkor a ciklust befejezzük. Ez

esetben a ciklus a $kon\left(\alpha^1,\ldots,\alpha^n\right)$ egyesített sorozatot eredményezi. Ez a sorozat a szekvencia esetéhez hasonlóan nem redukált, mert $\tau\left(\alpha^1\right)=\alpha_1^2$, $\tau\left(\alpha^2\right)=\alpha_1^3$, stb. Ezért az egyesített sorozat redukáltját tekintjük a ciklus, vagy iteráció eredményének.

Szükségünk van a következő jelölésekre: Legyen $\alpha^1,\alpha^2,\ldots,\alpha^{n-1}\in A^*$ és $\alpha^n\in A^{**}$. Ekkor

$$\chi_n\left(\alpha^1,\alpha^2,\ldots,\alpha^n\right)=red\left(kon\left(\alpha^1,\alpha^2,\ldots,\alpha^n\right)\right).$$

Ha $\alpha^i\in A^*$ $(i\in\mathbb{N})$, akkor

$$\chi_\infty\left(\alpha^1,\alpha^2,\ldots\right)=red\left(kon\left(\alpha^1,\alpha^2,\ldots\right)\right).$$

5.3 Definíció: *Legyen π feltétel és S program A-n. A $DO\subseteq A\times A^{**}$ relációt az S-ből a π feltétellel képezett ciklusnak nevezzük, és (π,S)-sel jelöljük, ha*

1. $\forall a\notin[\pi]$: $DO(a)=\{\langle a\rangle\}$,

2. $\forall a\in[\pi]$:

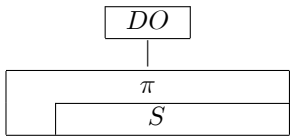
$$\begin{aligned}DO(a) \;=\; & \left\{\alpha\in A^{**}\mid\exists n\in\mathbb{N}:\alpha=\chi_n\left(\alpha^1,\alpha^2,\ldots,\alpha^n\right)\wedge\alpha^1\in S(a)\wedge\right.\\ & \wedge\forall i\in[1..n-1]:\alpha^{i+1}\in S\left(\tau\left(\alpha^i\right)\right)\wedge\tau\left(\alpha^i\right)\in[\pi]\wedge\\ & \left.\wedge\left(\alpha^n\in A^\infty\vee\left(\alpha^n\in A^*\wedge\tau\left(\alpha^n\right)\notin[\pi]\right)\right)\right\}\cup\\ & \cup\left\{\alpha\in A^\infty\mid\alpha=\chi_\infty\left(\alpha^1,\alpha^2,\ldots\right)\wedge\alpha^1\in S(a)\wedge\right.\\ & \left.\wedge\forall i\in\mathbb{N}:\alpha^{i+1}\in S\left(\tau\left(\alpha^i\right)\right)\wedge\tau\left(\alpha^i\right)\in[\pi]\right\}.\end{aligned}$$

Megjegyzés:

1. Determinisztikus programból képezett ciklus is determinisztikus.

2. A ciklus értékkészlete tartalmazhat végtelen sorozatot akkor is, ha az S program csak véges sorozatokat generál (soha nem jutunk ki a π feltétel igazsághalmazából).

A $DO=(\pi,S)$ ciklus struktogramja:



Igazoljuk a következő eredményt.

5.1 Tétel: *A szekvencia, az elágazás és a ciklus program.*

Bizonyítás: Mindhárom program $A\times A^{**}$ tipusú reláció és mindhárom esetben $D_S=A$. A másik két tulajdonság igazolása a következő:

1. Szekvencia: $a\in A$, $\alpha\in S(a)$. Ha $\alpha\in S_1(a)$, akkor S_1 program volta miatt $\alpha_1=a$ és $\alpha=red(\alpha)$. Ha $\alpha=\chi_2\left(\alpha^1,\alpha^2\right)$, ahol $\alpha^1\in S_1(a)$ és $\alpha^2\in S_2\left(\tau\left(\alpha^1\right)\right)$, akkor χ_2 definíciója miatt α redukált és $\alpha_1=\alpha_1^1=a$.

2. Elágazás: $a\in A$, $\alpha\in IF(a)$. Ekkor

$$\alpha\in\cup_{i=0}^m w_i(a).$$

Ha $\alpha\in w_0(a)$, akkor $\alpha=\langle a,a,\ldots\rangle$ kielégíti a két kritériumot. Ha $\exists i\in[1..m]:\alpha\in w_i(a)$, akkor $\alpha\in S_i(a)$ és mivel S_i program, $\alpha_1=a$ és $\alpha=red(\alpha)$.

3. Ciklus: $a\in A$, $\alpha\in DO(a)$. Ha $a\notin[\pi]$, akkor $\alpha=\langle a\rangle$, ami teljesíti a program követelményeit. Ha $a\in[\pi]$, akkor két eset lehetséges:

(a) Ha $\alpha\in A^{**}$ és $\exists n\in\mathbb{N}:\alpha=\chi_n\left(\alpha^1,\alpha^2,\ldots,\alpha^n\right)$, $\alpha^1\in S(a)$, akkor χ_n definiciója miatt α redukált és $\alpha_1=\alpha_1^1=a$.

(b) Ha $\alpha\in A^\infty$ és $\alpha=\chi_\infty\left(\alpha^1,\alpha^2,\ldots\right)$, $\alpha^1\in S(a)$, akkor χ_n definíciója miatt α redukált és $\alpha_1=\alpha_1^1=a$.

Q.E.D.

2.4 A programkonstrukciók programfüggvényei

A három programkonstrukció programfüggvényeit határozzuk meg.
6.1 Tétel: *Legyen A állapottér, S_1, S_2 programok A-n, $S = (S_1; S_2)$ a belőlük képezett szekvencia. Ekkor*

$$p(S) = p(S_2) \bigcirc p(S_1).$$

Bizonyítás: Emlékeztetünk arra, hogy a programfüggvény definíciója:

$$p(S) = \{(a,b) \mid S(a) \subseteq A^* \wedge \exists \alpha \in S(a) : \tau(\alpha) = b\}.$$

Esetünkben (feltéve, hogy minden sorozat véges):

$$\begin{gathered}(a,a') \in p(S_2) \bigcirc p(S_1) \Longleftrightarrow \\ \exists b \in A : (a,b) \in p(S_1) \wedge (b,a') \in p(S_2) \Longleftrightarrow \\ \exists \alpha \in S_1(a) : \tau(\alpha) = b \wedge \exists \beta \in S_2(b) : \tau(\beta) = a' \Longleftrightarrow \\ \chi_2(\alpha,\beta) \in S(a) \wedge \tau(\chi_2(\alpha,\beta)) = a' \Leftrightarrow \\ (a,a') \in p(S).\end{gathered}$$

Q.E.D.
6.2 Tétel: *Legyen A állapottér, $S_1, S_2, \ldots, S_m$ programok A-n, valamint $\pi_1, \pi_2, \ldots, \pi_m : A \to \mathbb{L}$ feltételek A-n, $IF = (\pi_1 : S_1, \ldots, \pi_m : S_m)$. Ekkor*
1. $D_{p(IF)} = \left\{a \in A \mid a \in \cup_{i=1}^m [\pi_i] \wedge \forall j \in [1..m] : a \in [\pi_j] \Rightarrow a \in D_{p(S_j)}\right\}$
2. $\forall a \in D_{p(IF)}$:

$$p(IF)(a) = \cup_{i=1}^m pw_i(a),$$

ahol

$$pw_i(a) = \begin{cases} p(S_i)(a), & \text{ha } a \in [\pi_i] \\ \emptyset, & \text{különben} \end{cases}$$

Bizonyítás:

$$\begin{gathered}a \in D_{p(IF)} \Longleftrightarrow IF(a) \subseteq A^* \Longleftrightarrow \\ \exists i \in [1..m] : a \in [\pi_i] \wedge \cup_{i=1}^m w_i(a) \subseteq A^* \Longleftrightarrow \\ \exists i : a \in [\pi_i] \wedge \forall j \in [1..m] : a \in [\pi_j] \Rightarrow a \in D_{p(S_j)}.\end{gathered}$$

Legyen $a \in D_{p(IF)}$. Ekkor

$$p(IF)(a) = \tau(\cup_{i=1}^m w_i(a)) = \cup_{i=1}^m pw_i(a).$$

Q.E.D.
Megjegyzés: *Ha az elágazásfeltételek lefedik az egész állapotteret, akkor mindig van olyan π_i állítás, amelyik igaz.*

Szükségünk van a következő fogalmakra.
6.1 Definíció: *Legyen $R \subseteq A \times A$ és $n \in \mathbb{N}$. Az R reláció n-edik hatványa ($n \geq 1$):*

$$\begin{aligned}R^{(n)} \quad = \quad & \{(a,a') \in A \times A \mid \exists a_0, a_1, \ldots, a_n \in A : \\ & (\forall i \in [1..n] : (a_{i-1}, a_i) \in R) \wedge a_0 = a \wedge a_n = a'\}.\end{aligned}$$

Az R reláció 0-adik hatványa (tkp. id_A):

$$R^{(0)} = \{(a,a') \in A \times A \mid a = a'\}.$$

Megjegyzés: $R^{(n)} = R \bigcirc R^{(n-1)}$, $R^{(1)} = R$ *és* $(id_A)^{(n)} = id_A$. *Értelemszerűen*

$$\begin{aligned}R^{(n)}(a) \quad = \quad & \{a' \in A \mid \exists a_0, a_1, \ldots, a_n \in A : \\ & (\forall i \in [1..n] : (a_{i-1}, a_i) \in R) \wedge a_0 = a \wedge a_n = a'\},\end{aligned}$$

$$R^{(1)}(a) = R(a), \quad R^{(0)}(a) = \{a\}.$$

6.2 Definíció: *Az $R \subseteq A \times A$ reláció lezártján az $\overline{R} \subseteq A \times A$ relációt értjük, amelyre*

$$D_{\overline{R}} = \{a \in A \mid \nexists \alpha \in A^{\infty} : (a, \alpha_1) \in R \wedge \forall i \in \mathbb{N} : (\alpha_i, \alpha_{i+1}) \in R\}$$

és $\forall a \in D_{\overline{R}}$:

$$\overline{R}(a) = \left\{ b \in A \mid b \notin D_R \wedge \exists k \in \mathbb{N}_0 : b \in R^{(k)}(a) \right\}.$$

Megjegyzés: *A reláció lezártja olyan pontokban van értelmezve, amelyekből kindulva a relációt nem lehet végtelen sokszor egymás után alkalmazni (a pontból nem indulhat végtelen "lánc", csak véges). Ezekhez a $D_{\overline{R}}$-beli pontokhoz olyan pontokat rendel, amelyek nincsenek benne az eredeti reláció értelmezési tartományában. Tkp. a véges láncok kivezetnek D_R-ből. Tehát $D_R \cap R_{\overline{R}} = \emptyset$.*

Vizsgáljuk meg az R reláció lezárását. Az eredeti D_R értelmezési tartományt felbonthatjuk két diszjunkt halmaz uniójára:

$$D_R = D_R^* \cup D_R^{\infty},$$

ahol D_R^* azon pontok halmaza, amelyből csak véges lánc indulhat, D_R^{∞} pedig azoké, amelyekből indul végtelen lánc. Bontsuk fel az A állapotteret három diszjunkt halmaz uniójára:

$$A = (A \setminus D_R) \cup D_R^* \cup D_R^{\infty}.$$

Legyen $a \in A \setminus D_R$. Minthogy $\nexists b \in A : (a, b) \in R$, azért $a \in D_{\overline{R}}$. Következésképpen

$$D_{\overline{R}} = (A \setminus D_R) \cup D_R^*.$$

Tekintsük az $\overline{R}(a)$ halmaz ($a \in D_{\overline{R}}$)

$$\left\{ b \in A \mid b \notin D_R \wedge b \in R^{(0)}(a) \right\}$$

részhalmazát: $b \in R^{(0)}(a) \Longleftrightarrow a = b \notin D_R$. Ez csak $a \in A \setminus D_R$ esetén lehetséges, amikor $b = a$. Tehát

$$\overline{R}(a) = \begin{cases} \{a\}, & \text{ha } a \in A \setminus D_R \\ \left\{ b \in A \mid b \notin D_R \wedge \exists k \in \mathbb{N} : b \in R^{(k)}(a) \right\}, & \text{ha } a \in D_R^* \end{cases}$$

Vegyük észre, hogy itt $\mathbb{N}_0$ helyett $\mathbb{N}$-et írhattunk és

$$\overline{R}(a) = id_A(a) \quad (a \in A \setminus D_R).$$

Összegezve: a lezárás egyrészt leszűkíti az R reláció értelmezési tartományát, másrészt kiterjeszti a relációt az $A \setminus D_R$ halmazra, ahol az identitás relációval lesz azonos.

Legyen A tetszőleges állapottér, $S \subseteq A \times A^{**}$ program, π feltétel A-n és $DO = (\pi, S)$. Vizsgáljuk a DO ciklus programfüggvényét!
Emlékeztető: Az S program programfüggvénye:

$$\begin{aligned} D_{p(S)} &= \{a \in A \mid S(a) \subseteq A^*\}, \\ p(S)(a) &= \{b \in A \mid \exists \alpha \in S(a) : \tau(\alpha) = b\} \\ &= \tau(S(a)) = (\tau \bigcirc S)(a). \end{aligned}$$

Esetünkben

$$a \in D_{p(DO)} \Longleftrightarrow DO(a) \subseteq A^*$$

csak két esetben lehetséges:

$$DO(a) = \{a\}, \quad \text{ha } a \notin [\pi],$$

$$\begin{aligned} DO(a) = \big\{ & \alpha \in A^* \mid \exists n \in \mathbb{N} : \alpha = \chi_n\left(\alpha^1, \alpha^2, \ldots, \alpha^n\right) \wedge \alpha^1 \in S(a) \wedge \\ & \wedge \forall i \in [1..n-1] : \alpha^{i+1} \in S\left(\tau\left(\alpha^i\right)\right) \wedge \tau\left(\alpha^i\right) \in [\pi] \wedge \\ & \wedge \tau(\alpha^n) \notin [\pi] \big\}, \quad \text{ha } a \in [\pi]. \end{aligned}$$

Vegyük észre, hogy $\tau(\alpha) = \tau(\alpha^n)$! Ezért a fenti két esetben

$$p(DO)(a) = \{a\}, \quad \text{ha } a \notin [\pi]$$

és

$$\begin{aligned} p(DO)(a) \quad = \quad & \{\tau(\alpha^n) \in A \mid \exists n \in \mathbb{N} : \exists \alpha^1, \alpha^2, \ldots, \alpha^n : \alpha^1 \in S(a) \wedge \\ & \wedge \forall i \in [1..n-1] : \alpha^{i+1} \in S\left(\tau\left(\alpha^i\right)\right) \wedge \tau\left(\alpha^i\right) \in [\pi] \wedge \\ & \wedge \tau(\alpha^n) \notin [\pi]\}, \quad \text{ha } a \in [\pi]. \end{aligned}$$

Az utóbbi kifejezést a $b \in S(c) \Rightarrow \tau(b) \in \tau(S(c)) = p(S)(c)$ összefüggés miatt felírhatjuk a következő formában is:

$$\begin{aligned} p(DO)(a) \quad = \quad & \{\tau(\alpha^n) \in A \mid \exists n \in \mathbb{N} : \exists \tau\left(\alpha^1\right), \ldots, \tau(\alpha^n) : \tau\left(\alpha^1\right) \in p(S)(a) \wedge \\ & \wedge \forall i \in [1..n-1] : \tau\left(\alpha^{i+1}\right) \in p(S)\left(\tau\left(\alpha^i\right)\right) \wedge \tau\left(\alpha^i\right) \in [\pi] \wedge \\ & \wedge \tau(\alpha^n) \notin [\pi]\}, \quad \text{ha } a \in [\pi]. \end{aligned}$$

Vegyük észre, hogy $\tau(\alpha^n) \in (p(S))^{(n)}(a)$, $a \in [\pi]$, $a \in D_{p(S)}$ és a $\tau(\alpha^n)$-hez vezető $\tau\left(\alpha^1\right), \ldots, \tau\left(\alpha^{n-1}\right)$ lánc minden tagjára $\tau\left(\alpha^i\right) \in [\pi]$ teljesül. Indokolt bevezetnünk a következő fogalmat.

6.3 Definíció: *A $p(S)$ reláció megszorítása a $[\pi]$ igazsághalmazra:*

$$p(S)_{|[\pi]} = p(S) \cap ([\pi] \times A).$$

Ekkor

$$D_{p(S)_{|[\pi]}} = D_{p(S)} \cap [\pi].$$

Könnyen látható, hogy az $a \in D_{p(S)} \cap [\pi]$ pontban $p(DO)(a)$ tulajdonképpen a $p(S)_{|[\pi]}$ reláció lezárása. Bontsuk fel $p(S)_{|[\pi]}$ értelmezési tartományát

$$D_{p(S)_{|[\pi]}} = D^*_{p(S)_{|[\pi]}} \cup D^\infty_{p(S)_{|[\pi]}}$$

alakban. Értelemszerűen

$$D_{p(DO)} = [\neg\pi] \cup D^*_{p(S)_{|[\pi]}}.$$

A $p(S)_{|[\pi]}$ reláció lezárásának értelmezési tartománya:

$$D_{\overline{p(S)_{|[\pi]}}} = \left(A \setminus D_{p(S)_{|[\pi]}}\right) \cup D^*_{p(S)_{|[\pi]}} = [\neg\pi] \cup \left([\pi] \setminus D_{p(S)}\right) \cup D^*_{p(S)_{|[\pi]}}.$$

Az $a \in [\neg\pi]$ esetben

$$\overline{p(S)_{|[\pi]}}(a) = \{a\} = p(DO)(a).$$

Ezt figyelembevéve a következő eredményt kapjuk.

6.3 Tétel: *Legyen A tetszőleges állapottér, $S \subseteq A \times A^{**}$ program, π feltétel A-n, $DO = (\pi, S)$. Ekkor $D_{p(DO)} = [\neg\pi] \cup D^*_{p(S)_{|[\pi]}}$ és*

$$p(DO)(a) = \overline{p(S)_{|[\pi]}}(a) \quad \left(a \in D_{p(DO)}\right).$$

2.5 Levezetési szabályok

A programkonstrukciók és a specifikáció kapcsolatát vizsgáljuk. Emlékeztetünk arra, hogy egy S program R utófeltételhez tartozó leggyengébb előfeltételének neveztük azt az $lf(S, R)$ állítást, amelyre

$$[lf(S, R)] = \left\{a \in D_{p(S)} \mid p(S)(a) \subseteq [R]\right\}.$$

Legyen Q és R két állítás A-n, S program. A

$$\{Q\}\, S\, \{R\}$$

szimbólum azt jelöli, hogy $a \in [Q]$ esetén $p(S)(a) \subseteq [R]$. A Q állítást (predikátumot) előfeltételnek nevezzük. Világos, hogy $[Q] \subseteq [lf(S,R)]$ és $Q \Rightarrow lf(S,R)$.

7.1 Tétel *(A szekvencia levezetési szabálya): Legyen $S = (S_1; S_2)$ szekvencia, Q, R és Q' állítások A-n. Ha*

(1) $Q \Rightarrow lf(S_1, Q')$ és (2) $Q' \Rightarrow lf(S_2, R)$,

akkor $Q \Rightarrow lf(S,R)$.

Bizonyítás: Legyen $q \in [Q]$. Ekkor (1) miatt $q \in D_{p(S_1)}$ és $p(S_1)(q) \subseteq [Q']$. A (2) feltétel miatt $[Q'] \subseteq D_{p(S_2)}$ és $\forall q' \in Q' : p(S_2)(q') \subseteq [R]$. Ezért $p(S_2) \bigcirc p(S_1)(q) \subseteq [R]$, azaz $q \in [lf(S,R)]$. Q.E.D.

7.1 Következmény: *A specifikáció tételéből (Ha $\forall b \in B$ esetén $Q_b \Rightarrow lf(S, R_b)$, akkor az S program megoldja az F feladatot.) és 7.1 Tételből következik, hogy ha S_1 és S_2 olyan programok, amelyekre a paramétertér minden pontjában $Q_b \Rightarrow lf(S_1, Q'_b)$ és $Q'_b \Rightarrow lf(S_2, R_b)$ teljesül, akkor $(S_1; S_2)$ megoldja a Q_b, R_b párokkal megadott feladatokat.*

7.2 Tétel *(A szekvencia levezetési szabályának megfordítása): Legyen $S = (S_1; S_2)$ szekvencia, Q és R olyan állítások A-n, amelyekre $Q \Rightarrow lf(S,R)$. Ekkor $\exists Q' : A \to \mathbb{L}$ állítás, hogy*

(1) $Q \Rightarrow lf(S_1, Q')$ és (2) $Q' \Rightarrow lf(S_2, R)$.

Bizonyítás: Legyen $Q' = lf(S_2, R)$. Ekkor (2) teljesül. Tfh. (1) nem teljesül. Ekkor $\exists q \in [Q] : q \notin [lf(S_1, lf(S_2, R))]$. Két eset lehetséges:

a) $q \notin D_{p(S_1)}$, ami ellentmondás a $[Q] \subseteq D_{p(S)} \subseteq D_{p(S_1)}$ feltétellel;

b) $p(S_1)(q) \not\subseteq [lf(S_2,R)]$. Legyen $r \in p(S_1)(q) \setminus [lf(S_2,R)]$. Két eset lehetséges:

$r \notin D_{p(S_2)}$, ami ellentmondás a $q \in D_{p(S_2) \bigcirc p(S_1)}$ feltétellel.

$p(S_2)(r) \not\subseteq [R]$: Legyen $s \in p(S_2)(r) \setminus [R]$. Ekkor $s \in p(S)(q)$ és $s \notin [R]$, ami ellentmond a $p(S)(q) \subseteq [R]$ feltételnek. Q.E.D.

7.3 Tétel *(Az elágazás levezetési szabálya): Legyen $IF = (\pi_1 : S_1, \ldots, \pi_m : S_m)$ elágazás, Q és R állítások A-n. Ha $\forall i \in [1..m] : Q \wedge \pi_i \Rightarrow lf(S_i, R)$, akkor*

$$Q \wedge (\vee_{i=1}^{m} \pi_i) \Rightarrow lf(IF, R).$$

Bizonyítás: Legyen $q \in [Q]$ és tfh. $\exists i \in [1..m] : q \in [\pi_i]$. Ekkor $q \in D_{p(IF)}$, ui.

$$\forall j \in [1..m] : q \in [\pi_j] \Rightarrow lf(S_j, R) \Rightarrow q \in D_{p(S_j)}.$$

Mivel $\forall j \in [1..m] : q \in [\pi_j] \Rightarrow p(S_j)(q) \subseteq [R]$, azért

$$p(IF)(q) = \cup_{\substack{j \in [1..m] \\ q \in [\pi_j]}} p(S_j)(q) \subseteq [R].$$

Tehát $q \in [lf(IF,R)]$. Q.E.D.

7.2 Következmény: *Legyen adott az F feladat specifikációja (A, B, Q, R). Ekkor, ha $\forall b \in B$ paraméterre és $\forall S_i$ programra $Q_b \wedge \pi_i \Rightarrow lf(S_i, R_b)$, és $\forall b \in B$ paraméterhez van olyan π_i feltétel, amelyre $b \in [\pi_i]$, akkor az IF program megoldja a Q_b, R_b párokkal definiált feladatot.*

7.4 Tétel *(Az elágazás levezetési szabályának megfordítása): Legyen $IF = (\pi_1 : S_1, \ldots, \pi_m : S_m)$ elágazás, Q és R olyan állítások A-n, amelyekre*

$$Q \wedge (\vee_{i=1}^{m} \pi_i) \Rightarrow lf(IF, R).$$

Ekkor $\forall i \in [1..m] : Q \wedge \pi_i \Rightarrow lf(S_i, R)$.

Bizonyítás: Indirekt: tfh. $\exists i \in [1..m] : [Q \wedge \pi_i] \not\subseteq [lf(S_i,R)]$. Legyen $q \in [Q \wedge \pi_i] \setminus [lf(S_i,R)]$. Két eset lehetséges:

$q \notin D_{p(S_i)}$, ami ellentmond a $q \in D_{p(IF)}$ feltevésnek.

$p(S_i)(q) \not\subseteq [R]$. Ekkor $p(S_i)(q) \subseteq p(IF)(q) \subseteq [R]$ miatt ellentmondás. Q.E.D.

Megjegyzés: *Nem elég az elágazás levezetési szabályának teljesülését vizsgálni. Ellenőrizni kell, hogy a feltételek lefedik-e a feladat értelmezési tartományát.*

A ciklus levezetési szabálya bonyolultabb. A cél:

DO véges lépésben terminál**j**on.

Olyan P feltételt keresünk, amely:

1. Igaz a ciklus/iteráció megkezdése előtt;
2. Igaz az iteráció alatt;
3. Igaz az iteráció befejezése után.

A ciklus befejezésekor $\neg\pi$, tehát $P\wedge\neg\pi$ igaz. Ha $P\wedge\neg\pi \Rightarrow R$, akkor a ciklus helyességét igazoltuk. A P feltétel elnevezése: *invariáns feltétel/tulajdonság.*

Bevezetünk továbbá egy $t : A \to \mathbb{Z}$ egész értékű függvényt, amely

1. A program változóitól függ;
2. Korlátot ad a szükséges iterációk számára;
3. Minden végrehajtott iteráció legalább 1-el csökkenti t értékét;
4. t alulról korlátos: $t > 0$ terminálás előtt.

A t függvény elnevezése: *korlátozó/terminálló függvény.* A korlátozó függvény biztosítja, hogy a ciklusnak terminálnia kell.

7.5 Tétel *(A ciklus levezetési szabálya): Legyen P állítás A-n, $DO = (\pi, S)$ és $t : A \to \mathbb{Z}$. Ha*

(1) $P \wedge \pi \Rightarrow t > 0$,

(2) $P \wedge \pi \Rightarrow lf\,(S, P)$,

(3) $P \wedge \pi \wedge \forall t = t_0 \Rightarrow lf\,(S, t < t_0)$,

akkor $P \Rightarrow lf\,(DO, P\wedge\neg\pi)$.

Megjegyzés: *Ha $Q \Rightarrow P$ és $P\wedge\neg\pi \Rightarrow R$, akkor $Q \Rightarrow lf\,(DO, R)$.*

A 7.5 Tétel bizonyítása: Legyen g a $p\,(S)$ leszűkítése $[\pi]$-re, azaz $g = p\,(S) \cap ([\pi] \times A)$.

1. állítás: Legyen $q \in [P \wedge \pi]$. Ekkor

$$\forall k \in \mathbb{N}_0 : g^{(k)}\,(q) \subseteq [P]\,.$$

Az állítást teljes indukcióval igazoljuk: $k = 0$ esetén $g^{(0)}\,(q) = q \in [P]$; Tfh. $g^{(k)}\,(q) \subseteq [P]$. Legyen $r \in g^{(k)}\,(q)$ tetszőleges. Két eset lehetséges:

$r \in [P\wedge\neg\pi]$. Ekkor a ciklus terminál, $g\,(r) = r \in [P]$.

$r \in [P \wedge \pi]$. Ekkor (2) miatt $r \in D_{p(S)}$ és $g\,(r) = p\,(S)\,(r) \subseteq [P]$.

Tehát $g^{(k+1)}\,(q) \subseteq [P]$.

2. állítás: Legyen $q \in [P \wedge \pi]$. Ekkor

$$\exists n \leq t\,(q) : g^{(n)}\,(q) \subseteq [\neg\pi]\,.$$

Indirekt bizonyítás: Tfh. $\forall n : g^{(n)}\,(q) \subseteq [\pi]$. A $q \in A$ pontban $P \wedge \pi \wedge t = t\,(q)$ $(t_0 = t\,(q))$ igaz, amiből (3) miatt következik, hogy

$$q \in [lf\,(S, t < t\,(q))] \Longleftrightarrow q \in D_{p(S)} \wedge p\,(S)\,(q) \subseteq [(t < t\,(q))]$$

Legyen $b \in p\,(S)\,(q)$ tetszőleges. Ekkor $t\,(b) < t\,(q)$, ahonnan $\max_{b\in g(q)} t\,(b) < t\,(q)$. Hasonlóan igazoljuk, hogy

$$t_{k+1} = \max_{\widetilde{b}\in g^{(k+1)}(q)} t\left(\widetilde{b}\right) < \max_{b\in g^{(k)}(q)} t\,(b) = t_k.$$

Minthogy

$$\widetilde{b} \in g^{(k+1)}\,(q) \Longleftrightarrow \exists z \in g^{(k)}\,(q) : \left(z, \widetilde{b}\right) \in g$$

és a z pontban $P \wedge \pi \wedge t = t_k$ igaz (mert $g^{(k)}\,(q) \subseteq [\pi]$, $z \in [\pi]$), azért

$$z \in [[lf\,(S, t < t_k)]] \Longleftrightarrow z \in D_{p(S)} \wedge p\,(S)\,(z) \subseteq [(t < t_k)]\,.$$

Minthogy $\widetilde{b} \in p\,(S)\,(z)$, azért $t\left(\widetilde{b}\right) < t_k$. Tehát $t_{k+1} < t_k$ és fennáll, hogy

$$\max_{b\in g^{(t(q)+1)}(q)} t\,(b) < \max_{b\in g^{(t(q))}(q)} t\,(b) < \ldots < \max_{b\in g(q)} t\,(b) < t\,(q)\,.$$

Minthogy itt $t(q)+1$ darab egymástól különböző és $t(q)$-nál kisebb egész számról van szó, szükségképpen

$$\max_{b \in g^{(t(q)+1)}(q)} t(b) < 0$$

következik. Ez ellentmond (1)-nek ($t > 0$).

A két állítás után a tétel bizonyítása a következő. Legyen $q \in [P]$ tetszőleges. Ekkor vagy $q \in [P \wedge \neg\pi]$, vagy $q \in [P \wedge \pi]$.

Ha $q \in [P \wedge \neg\pi]$, akkor a ciklus definíciója alapján $p(DO)(q) = \{q\} \subseteq [P \wedge \neg\pi]$. Tehát $q \in [lf(DO, P \wedge \neg\pi)]$.

Ha $q \in [P \wedge \pi]$, akkor a 2. állítás miatt

$$\exists n \in \mathbb{N}_0 : g^{(n)}(q) \subseteq [\neg\pi].$$

Ez azt jelenti, hogy $q \in D_{p(DO)}$ (n-nél rövidebb "láncok" is indulhatnak q-ból, de azok is $[\neg\pi]$-ben kell, hogy végződjenek). Definíció alapján

$$p(DO)(q) \subseteq [\neg\pi].$$

Az 1. állítás miatt

$$p(DO)(q) \subseteq [P].$$

Tehát

$$p(DO)(q) \subseteq [P \wedge \neg\pi].$$

Következésképpen

$$q \in [lf(DO, P \wedge \neg\pi)].$$

Tehát $P \Rightarrow lf(DO, P \wedge \neg\pi)$. Q.E.D

Vizsgáljunk meg néhány példát. A ciklust a következő "metanyelvi" leírással adjuk meg:

while π
  S
end

A Q előfeltételt, R utófeltételt, a P invariáns állítást és a t korlátozó függvényt belefoglaljuk a fenti leírásba:

$\{Q\}$
$\{P:$ az invariáns állítás$\}$
$\{t:$ korlátozó függvény$\}$
while π
  S
end
$\{R\}$

A ciklus helyességét a következők szerint kell ellenőrizni:

1. Igazoljuk, hogy P igaz a ciklus megkezdése előtt.
2. Igazoljuk, hogy $\{P \wedge \pi\}\, S\, \{P\}$, azaz P tényleg invariáns.
3. Igazoljuk, hogy $P \wedge \neg\pi \Longrightarrow R$, azaz terminálás után az R utófeltétel bekövetkezik.
4. Igazoljuk, hogy $P \wedge \pi \Longrightarrow (t > 0)$.
5. Igazoljuk, hogy $\{P \wedge \pi\}\ (t_0 = t;\ S)\ \{t < t_0\}$.

7.1 Feladat: Formálisan igazoljuk a fenti öt pontot a következő három példára!

7.1 Példa: Az algoritmus kiszámítja az $i = 2^p$ $(i \leq n < 2i)$ mennyiséget.

$\{n > 0\}$
$i := 1;$
$\{P : 0 < i \leq n \wedge (\exists p : i = 2^p)\}$
$\{t : n - i\}$
while $2 * i \leq n$
  $i := 2 * i$
end
$\{R : 0 < i \leq n < 2 * i \wedge (\exists p : i = 2^p)\}$

7.2 Példa: Az algoritmus kiszámítja az n-edik Fibonacci számot $n > 0$ esetén ($f_0 = 1$, $f_1 = 1$ és $f_n = f_{n-1} + f_{n-2}$, ha $n > 1$).

$\{n > 0\}$
$i, a, b := 1, 1, 1;$
$\{P : 1 \leq i \leq n \wedge a = f_i \wedge b = f_{i-1}\}$
$\{t : n - i\}$
while $i < n$
  $i, a, b := i + 1, a + b, a$
end
$\{R : a = f_n\}$

7.3 Példa: Az algoritmus kiszámítja a q hányadost és az r maradékot, amikor x-et y-al osztjuk.

$\{x \geq 0 \wedge y > 0\}$
$q, r := 0, x;$
$\{P : 0 \leq r \wedge 0 < y \wedge q * y + r = x\}$
$\{t : r\}$
while $r \geq y$
  $r, q := r - y, q + 1$
end
$\{R : 0 \leq r < y \wedge q * y + r = x\}$

3 Programszerkezetek analízise

A programok szerkezetét irányított gráffal ábrázoljuk. Feltesszük, hogy minden program (és részprogram) determinisztikus (függvényreláció).

3.1 Gráfelméleti alapfogalmak

8.1 Definíció: *A gráf pontokból és a pontokat összekötő vonalakból álló alakzat. A gráf pontjait szögpontoknak, vagy csúcsoknak nevezzük. A gráf két szögpontját összekötő olyan vonalat, amely nem megy át más szögponton, élnek nevezzük.*

A szögpontok halmazát V (vertex), az élek halmazát E (edge) jelöli. A G gráfot a $G = (V, E)$ pár adja meg. Egy $e \in E$ élt a rendezetlen $[u, v]$ pár ad meg, ahol $u, v \in V$. Az u és v csúcsok az e él végpontjai.

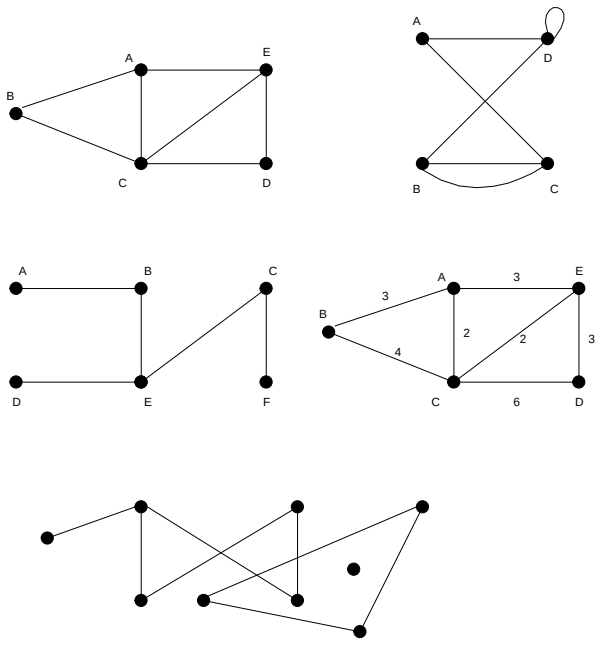
Az $[u, u] \in E$ élt huroknak nevezzük. Az $e, e' \in E$ éleket többszörös éleknek nevezzük, ha ugyanazt a két pontot kötik össze, azaz $e = [u, v]$ és $e' = [u, v]$.

A hurkot és többszös éleket nem tartalmazó gráfokat *egyszerű gráfoknak* nevezzük egyébként pedig *multigráfnak*.

8.2 Definíció: *Az $u \in V$ csúcs $\phi(u)$ fokán az u csúcsot tartalmazó élek számát érjük. Ha $\phi(u) = 0$, akkor az u csúcsot izoláltnak nevezzük.*

8.3 Definíció: *A G gráf üres, ha $E = \emptyset$. Teljes a gráf, ha minden szögpontpár éllel van összekötve.*
8.4 Definíció: *Az $u, v \in V$ csúcsokat összekötő n hosszúságú vonalnak nevezzük az egymáshoz csatlakozó $\{[v_{i-1}, v_i]\}_{i=1}^{n}$ élek sorozatát, ha $v_0 = u$ és $v_n = v$. A vonal zárt, ha $v_0 = v_n$. A vonalat útnak nevezzük, ha a $v_0, v_1, \ldots, v_n$ csúcsok a $v_0 = v_n$ lehetőség kivételével egymástól különböznek. A zárt utat körnek nevezzük.*
8.5 Definíció: *A gráf összefüggő, ha bármely két szögpontját út köti össze.*
Következmény: *Ha egy gráf nem összefüggő, akkor van legalább egy olyan szögpontja, amelyből nem vezet út az összes többi szögpontba.*
8.6 Definíció: *Azok a szögpontok, amelyek egy adott szögpontból úttal elérhetők, a hozzájuk illeszkedő élekkel együtt a gráf egy összefüggő komponensét alkotják.*
8.7 Definíció: Az olyan összefüggő gráfot, amelyben nincsen kör, fának nevezzük.

Ha a fának n csúcsa van, akkor pontosan $n - 1$ éle van.
8.8 Definíció: *A G gráfot cimkézettnek nevezzük, ha az éleihez adatokat rendelünk. Ha minden e éléhez egy $w(e) \geq 0$ számot rendelünk, akkor súlyozott gráfról beszélünk.*
8.9 Definíció: *A G gráfot végesnek nevezzük, ha V és E véges halmazok.*
8.10 Definíció: *A $G_s = (V_s, E_s)$ gráf a $G = (V, E)$ gráf részgráfja, ha $V_s \subseteq V$ és $E_s \subseteq E$.*
8.11 Definíció: *A gráf rangja, vagy ciklomatikus száma $V(G) = e - n + p$, ahol e az élek száma, n a szögpontok száma és p az összefüggő komponensek száma.*
Megjegyzés: *A ciklomatikus szám azonos a "lineárisan független" körök maximális számával. Összefüggő gráf esetén $V(G) = e - n + 1$.*



Irányítatlan gráfok

8.12 Definíció: *A $G = (V, E)$ gráfot irányítottnak vagy digráfnak (directed graph) nevezzük, ha minden élét irányítjuk.*

Ekkor E rendezett párok halmaza. Az $e = [u, v] \in E$ élnek u a kezdőpontja és v a végpontja. Egy $u \in V$ csúcspont $\phi_{be}(u)$ bemenő foka, vagy *be-foka* az u szögpontban végződő élek száma. Az u csúcspont $\phi_{ki}(u)$ kimenő foka, vagy *ki-foka* az u pontból induló élek száma.

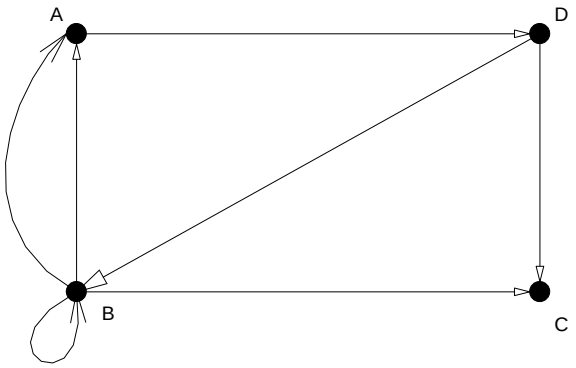
Az $u \in V$ csúcspontot forrásnak nevezzük, ha $\phi_{ki}(u) > 0$, de $\phi_{be}(u) = 0$. Az $u \in V$ csúcspont nyelő, ha $\phi_{ki}(u) = 0$, de $\phi_{be}(u) > 0$.

Az irányított vonal, út és kör definíciója hasonló az eredeti definícióhoz azzal az eltéréssel, hogy az út (és a kör) esetén az élek irányítása meg kell, hogy egyezzen az vonal irányításával.

Az v csúcs elérhető az u csúcsból, ha létezik u-ból induló és v-ben végződő irányított út.

8.13 Definíció: *A $G = (V, E)$ irányított gráf összefüggő, ha az irányítások elhagyásával kapott gráf összefüggő.*

8.14 Definíció: *A $G = (V, E)$ irányított gráf erősen összefüggő, ha bármely $u, v \in V$ csúcsot irányított él köt össze.*



Irányított gráf

A gráfok és relációk szoros kapcsolatban állnak egymással:

1. Legyen $G = (V, E)$ irányított gráf. Ez megfeleltethető egy $R \subseteq V \times V$ relációnak:

$$R = \{(u, v) \mid e = [u, v] \in E\}.$$

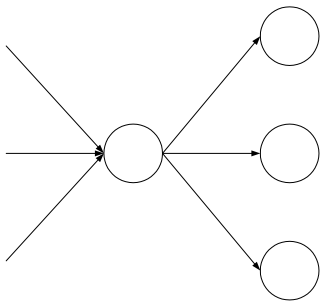
2. Legyen $R \subseteq A \times B$ reláció. Ez megfeleltethető egy (V, E) gráfnak:

$$V = A \cup B, \quad E = \{e = [u, v] \mid (u, v) \in R\}.$$

3.2 Vezérlési gráfok, blokkdiagramok

A program utasításai egy irányított gráfra képezhetők le, amelyben az utasításoknak a csomópontok felelnek meg, a gráf élei pedig az egyes utasítások után a következő lépésben végrehajtható utasításoknak megfeleltetett csomópontokat jelölik ki.

A gráf csomópontjaihoz általában több bemenő- és kimenőél csatlakozik (1. ábra).

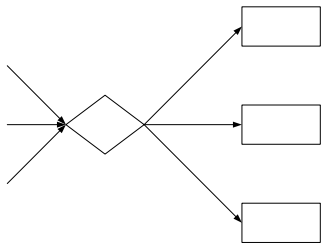


1. ábra

Az utasításoknak (csomópontoknak) kettős funkciójuk lehet:

- transzformációt hajtanak végre az adattéren
- kijelölik a következő végrehajtandó utasítást (kiválasztják azt a kimenőélt, amelyen a vezérlés továbbhalad).

A fenti gráfot célszerűen kétféle tipusú gráfelemre bontjuk (2. ábra):

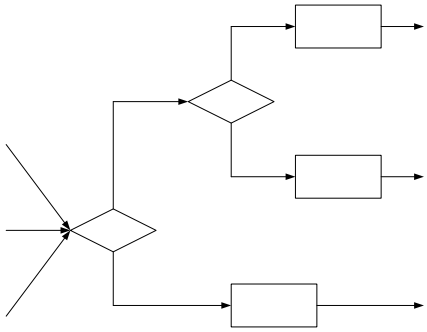


2. ábra

A baloldali négyszög (deltoid) *vezérlési csomópont*, az utasításnak mint elemi függvénynek azokat a funkcióit reprezentálja, melyek az adatteret (állapotteret) érintetlenül hagyják. Feladata kizárólag a következő utasítás (csomópont) kijelölése. A fekvő téglalap az adatteret transzformálja. Az ilyen tipusú csomópontból csak egy kimenőél fut ki. Tehát a következő utasítás egyértelműen meghatározott.

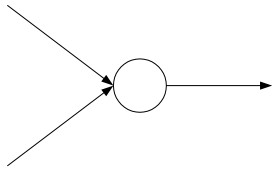
Kikötjük, hogy a vezérlési csomópontokból kizárólag két él futhat ki. Ebben az esetben a vezérlési csomópontokat *döntési (predikátum) csomópontoknak* nevezzük.

Ennek figyelembevételével az előző gráfot (2. ábra) a következőképpen helyettesíthetjük (3. ábra):



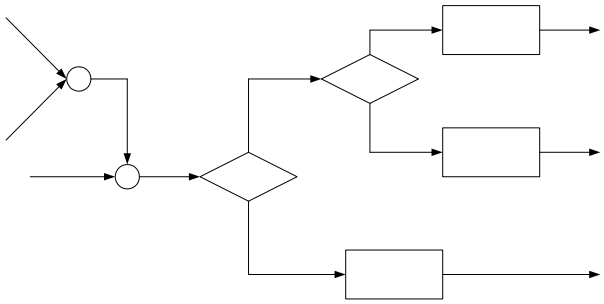
3. ábra

A döntési csomópontok kétirányú elágazást reprezentálnak. Ennek analógiájára bevezetjük a gyűjtő csomópontot, melyen feladata kizárólag az, hogy a kétirányból érkező vezérlést egyesítse (4. ábra):



4. ábra

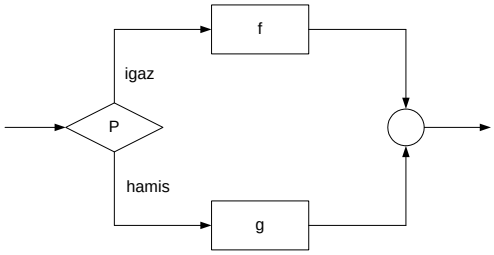
Ez a csomópont nem hajt végre transzformációt az állapottéren. Az új csomópont fogalom segítségével a 3. ábra gráfja átírható a következő alakba (5. ábra):



5. ábra

Ha eltekintünk attól, hogy a csomópontokban ténylegesen milyen döntés, vagy transzformáció játszódik le, tehát csak a gráf szerkezetét tekintjük, akkor az utóbbi tipusú gráfokat (a szűkebb értelemben vett) *vezérlési gráfoknak* nevezzük.

A programokat sok esetben *blokkdiagram* segítségével ábrázoljuk. A blokkdiagramot egy vezérlőgráf és a gráf egyes csomópontjaihoz rendelt *adattranszformációk*, illetve *lekérdezések* (tesztek, predikátumok) határozzák meg. Ezt a hozzárendelést a vezérlési gráf (egy adott programnak megfelelő) interpretációjának nevezzük (6. ábra).



6. ábra

A 6. ábra az $IF = (P : f, \neg P : g)$ elágazásnak felel meg (szöveges leírásban: **if** P **then** f **else** g).

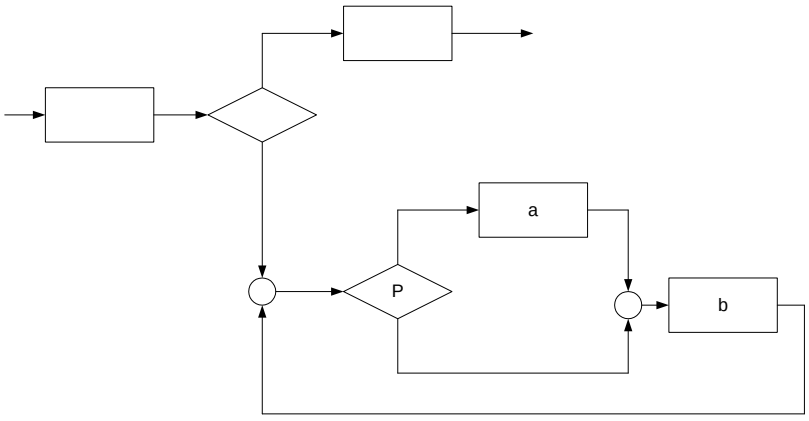
A blokkdiagramban csak adattranszformációs, döntési és gyűjtő csomópontokat engedünk meg.

9.1 Definíció: *A valódi program egy olyan összefüggő irányított gráf, amelyre igazak a következők:*

1. *Véges számú nemzérus bemenőéllel és kimenőéllel rendelkezik;*
2. *Csomópontjait predikátumcsomópontok, függvénycsomópontok és gyüjtőcsomópontok alkotják;*
3. *Minden csomóponton át vezet legalább egy bemenőéllel kezdődő és kimenőéllel végződő útvonal.*

A programgráfot ki szokás egészíteni az indítási (START) és a befejezési (HALT, STOP) csomóponttal.

Példa (nem valódi programra)

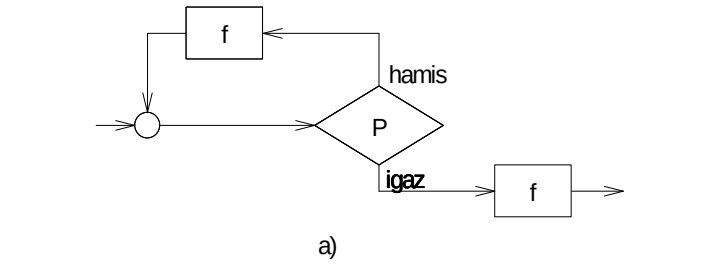


7. ábra

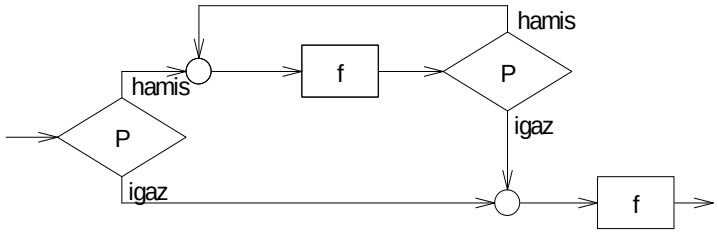
A P, a és b jelű csomópontokon keresztül nem vezet útvonal a bemenőéltől a kimenőélhez.

9.2 Definíció: *Két programot (programgráfot) ekvivalensnek nevezünk, ha a programfüggvények megegyeznek.*

Példa (ekvivalens programgráfokra):



a)



b)

8. ábra

Az a) esetben a programfüggvény:

$$\text{ha } d \in [P], \text{ akkor } p(S)(d) = f(d),$$

egyébként

$$\text{ha } f(d) \in [P], \text{ akkor } p(S)(d) = f^{(2)}(d),$$
$$\vdots$$

azaz a programfüggvény $p(S)(d) = f^{(n+1)}(d)$ feltéve, hogy $f^{(i)}(d) \notin [P]$ $(i = 0, 1, \ldots, n-1)$ és $f^{(n)}(d) \in [P]$. Itt

$$f^{(n)}(d) = f\left(f^{(n-1)}(d)\right), \quad f^{(0)}(d) = d, \quad f^{(1)}(d) = f(d).$$

A b) esetben a programfüggvény:

ha $d \in [P]$, akkor $p(Q)(d) = f(d)$,

egyébként egy (utótesztelő) iteráció kezdődik, amelyből akkor lépünk ki, ha $f^{(i)}(d) \notin [P]$ $(i = 1, \ldots, n-1)$ és $f^{(n)}(d) \in [P]$. Ezután még végrehajtódik az $f\left(f^{(n)}(d)\right)$ leképezés. Tehát $p(Q)(d) = f^{(n+1)}(d)$ feltéve, hogy $f^{(i)}(d) \notin [P]$ $(i = 0, 1, \ldots, n-1)$ és $f^{(n)}(d) \in [P]$.

Igazoltuk, hogy a két program ekvivalens.

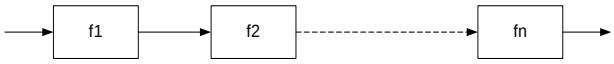
3.3 A struktúrált programszerkezet

Strukturáltnak, vagy D (Dijkstra) gráfnak nevezünk egy vezérlési gráfot, ha

- meghatározott tipusú elemi részekből tevődik össze,
- és a részekből való összeállítás bizonyos szabályoknak tesz eleget.

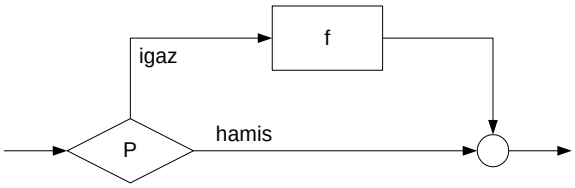
A strukturált gráfok elemi részgráfjai a következők:

1. *Szekvencia* (**begin** $f1$, $f2, \ldots, fn$ **end**, vagy $B(f1, f2, \ldots fn)$):



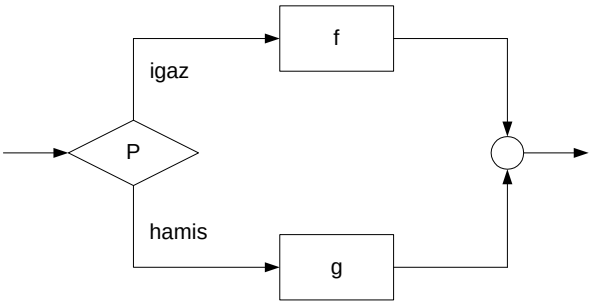
9. ábra

2. *Feltételes elágazás:*
if P **then** f , vagy $IT(P;f)$ szerkezet:



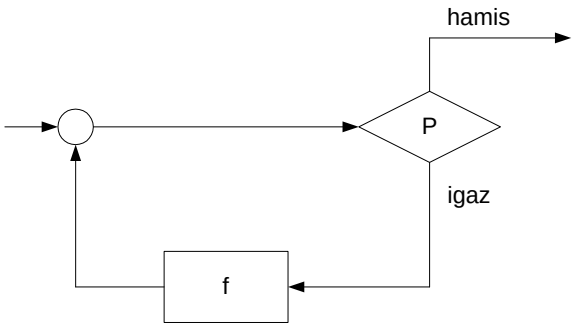
10. ábra

if P **then** f **else** g, vagy $ITE(P;f;g)$ szerkezet:



11. ábra

Az $IT(P;f)$ szerkezet az $ITE(P;f;g)$ szerkezet speciális esete ($g = id$, vagy SKIP).
3. *Iteráció, vagy ciklus* (**while** P **do** f, vagy $WD(P;f)$):



12. ábra

A strukturált gráfokat az alábbi szabályrendszer alapján szerkeszthetjük:
1. Egyetlen csomópontból álló, egy input és egy output éllel rendelkező gráf struktúrált gráf;
2. A fenti elemi gráfok strukturált gráfok;
3. Strukturált gráf minden olyan vezérlési gráf, amelyben a komponens strukturált gráfokat egyetlen csomóponttá zsugorítva strukturált gráfot kapunk.

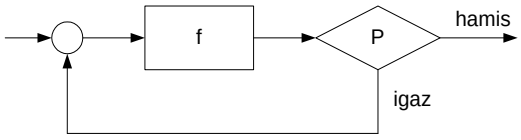
Csakis a fentiek szerint szerkesztett gráfokat tekintjük strukturált, vagy D gráfoknak.
9.3 Definíció: *Egy vezérlőgráf lebontásának azt az eljárást nevezzük, amelynek során a gráfban előforduló három alapszerkezet valamelyikét egy csomóponttal helyettesítjük és ezt az eljárást mindaddig folytatjuk, amíg ilyen helyettesítés lehetséges. Ha a programgráf végül az egy csomópontot tartalmazó blokkra redukálódik, akkor ezt az önmagában álló éllel helyettesítjük.*

9.4 Definíció: *Azt a programot, amelynek a vezérlőgráfja lebontható az önmagában álló irányított élre, strukturált programnak nevezzük.*

Megjegyzések:

1. A lebontás szempontjából az $IT(P; f)$ *és* $ITE(P; f; g)$ *szerkezeteket azonosnak tekintjük.*

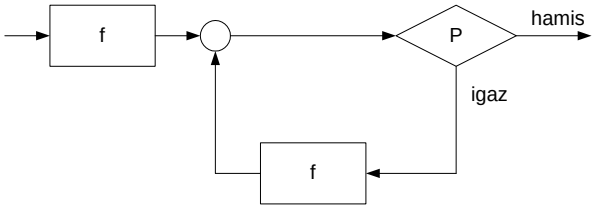
2. A $WD(P; f)$ *ciklus mellett szokták használni a Do* f *until* P*, vagy* $DU(P; f)$ *utótesztelő ciklust is (13. ábra).*



13. ábra

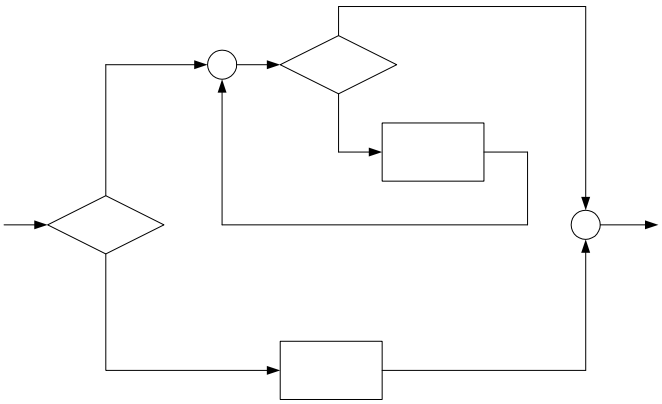
Ez nem strukturált program, de a lebontáskor annak tekintjük.

Feladat: Igazoljuk hogy a 13. ábra programja ekvivalens az alábbi strukturált programmal:



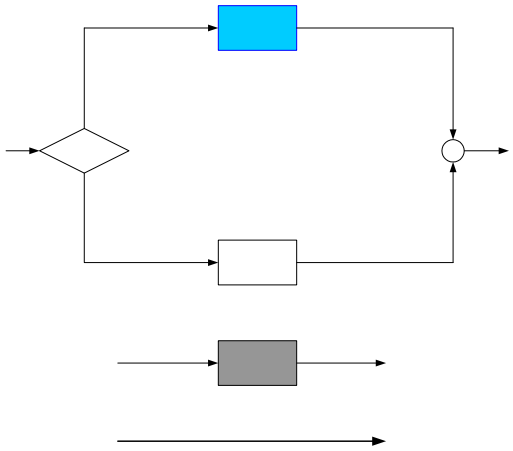
14. ábra

Példa (strukturált program lebontása): Tekintsük az alábbi programgráfot.



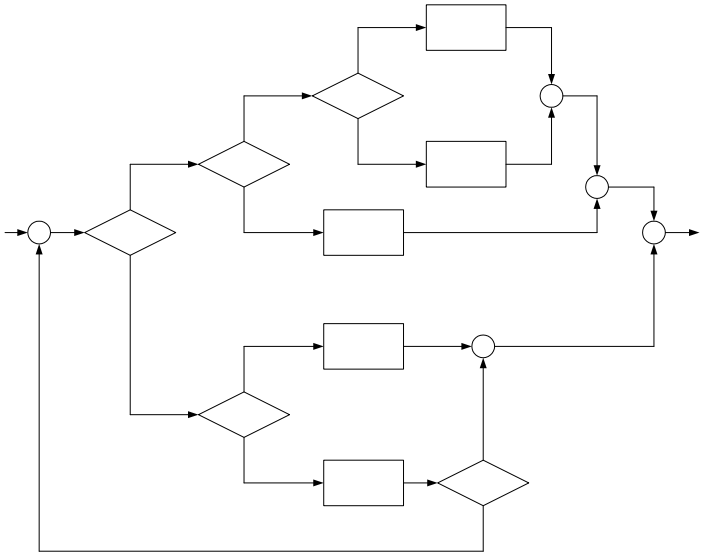
15. ábra

A gráf lebontásának lépései a következők:



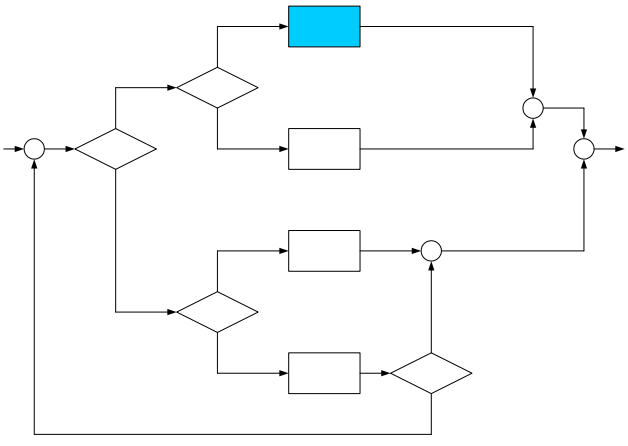
16. ábra

Példa (nem strukturált valódi program)

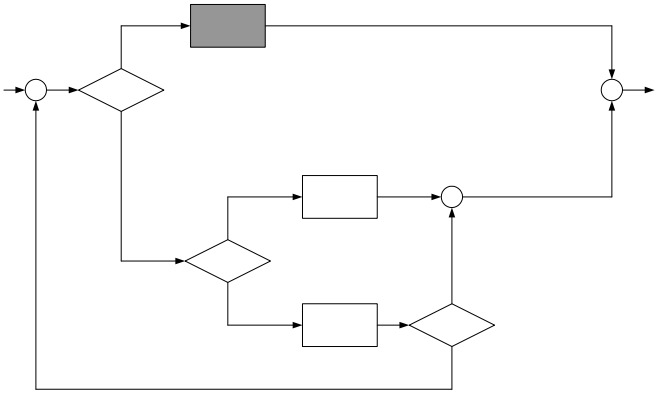


17. ábra

A program lebontása két lépés után elakad (18-19. ábra).



18. ábra

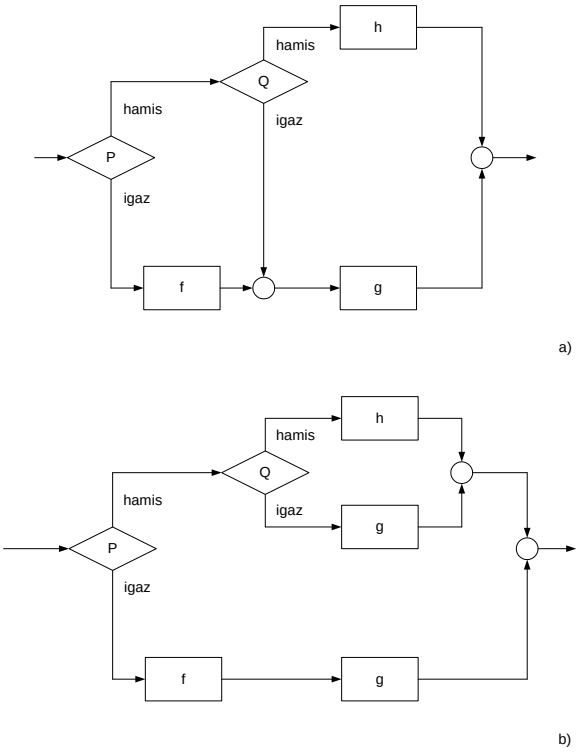


19. ábra

Ez utóbbi már nem bontható tovább mert az alsó, **if then else** alakhoz hasonló szerkezet jobb alsó sarkában predikátum csomópont van.

Kérdés: *Hozzá lehet-e rendelni minden nem strukturált valódi programhoz egy strukturált programot?*

Feladat: Igazoljuk, hogy a 20. a) ábra nem strukturált programja ekvivalens a 20. b) ábra strukturált programjával!



20. ábra

9.1 Lemma: *Ha egy programgráfban az élek száma e, a predikátumcsomópontok száma n_p, a függvénycsomópontok száma n_f és a gyűjtő csomópontok száma n_c, akkor $e = 3n_p + n_f + 1$ és $n_p = n_c$.*
Bizonyítás: Számoljuk meg a csomópontokba befutó élek számát!

függvény csomópont: n_f
predikátum csomópont: n_p
gyűjtő csomópont: $2n_c$
összesen: $n_f + n_p + 2n_c$

Ehhez hozzáadjuk a HALT csomóponthoz vezető 1 élt. Így a bemenő élek száma: $n_f + n_p + 2n_c + 1$.

A program csomópontjaiból kimutató élek száma:
függvény csomópont: n_f
predikátum csomópont: $2n_p$
gyűjtő csomópont: n_c
összesen: $n_f + 2n_p + n_c$

Ehhez hozzáadjuk a START csomópontból kivezető 1 élt. Így a kimenő élek száma: $n_f + 2n_p + n_c + 1$.

Minthogy

$$e = n_f + n_p + 2n_c + 1 = n_f + 2n_p + n_c + 1$$

kapjuk, hogy $n_c = n_p$ és $e = 3n_p + n_f + 1$. Q.E.D.

9.2 Lemma *(Böhm-Jacopini): Egy S program akkor és csak akkor strukturált program, ha felírható $B(g,h)$, vagy $ITE(P;g,h)$, vagy $WD(P;g)$ alakban, ahol P predikátum, g és h strukturált programok.*

Megjegyzések:

1. Az üres programot, amelynek vezérlési gráfja az önmagában álló irányított él, is strukturáltnak tekintjük.

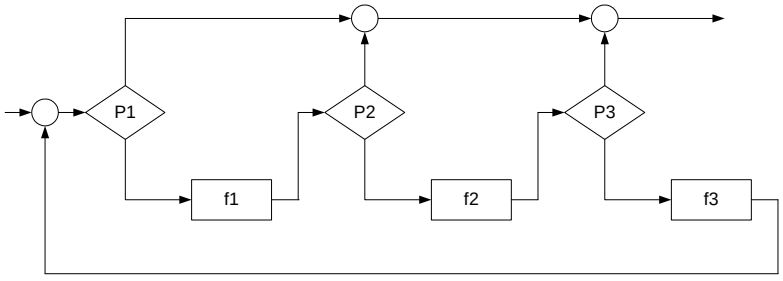
2. A lemma megadja a TOPDOWN programtervezési módszer hátterét is.

9.1 Tétel *(Corrado Böhm-Giuseppe Jacopini, 1966): Bármely valódi programhoz meg lehet konstruálni egy vele ekvivalens strukturált programot.*

3.4 A struktúrálatlanság jellemzői

A következőkben három fontos példát mutatunk nem strukturált programokra.

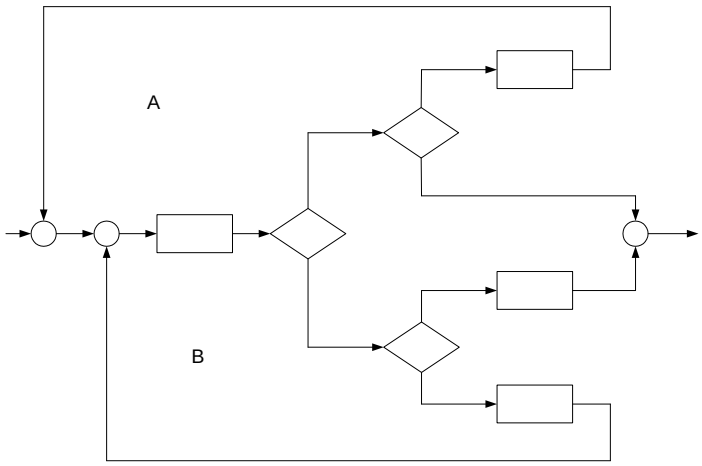
Példa (Böhm-Jacopini Ω_3)



21. ábra

A példa vezérlési gráfja tkp. egy ciklus három kimenőéllel.

Példa (két párhuzamos ciklus)

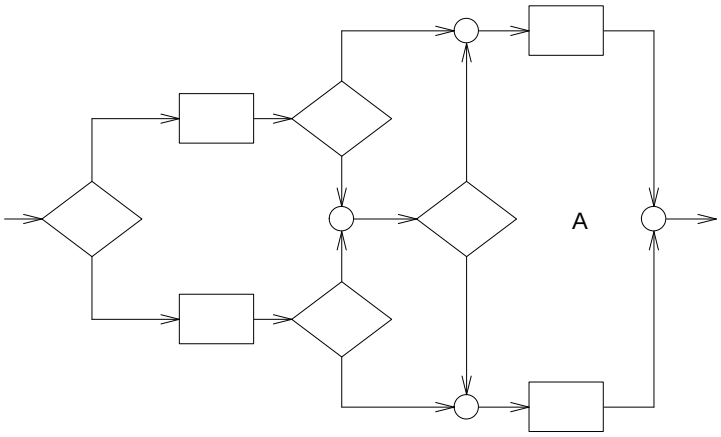


22. ábra

A program két párhuzamos ciklusból áll, amelyek több kimenőéllel rendelkeznek. Az A ciklus ezenkívül több belépőéllel is rendelkezik.

A ciklusba történő rendellenes belépések, ill. kilépések nem strukturáltsághoz vezetnek.

Példa (rendellenes elágazások)

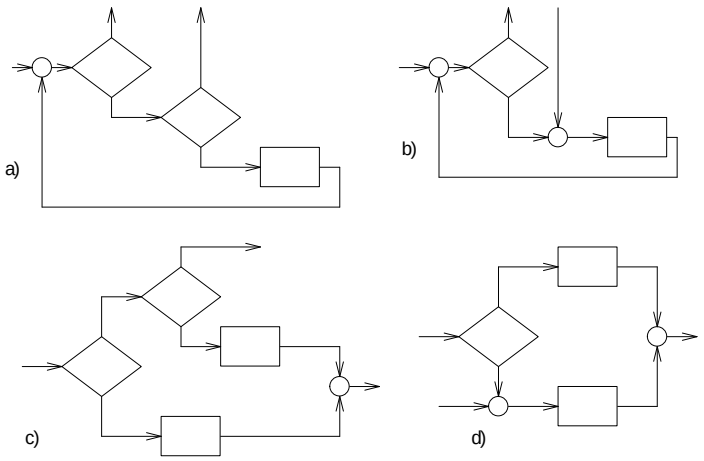


23. ábra

A példa programja olyan két egymásbafésült elágazásból áll, amelyeknél rendellenes kilépések és belépések (A rész) vannak.

9.2 Tétel: *Egy program akkor és csak akkor nem strukturált, ha van olyan részgráfja, amely több ki-, ill. belépő éllel rendelkező ciklus, vagy elágazás.*

Tekintsük az alábbi négy nemstrukturált programrészletet!

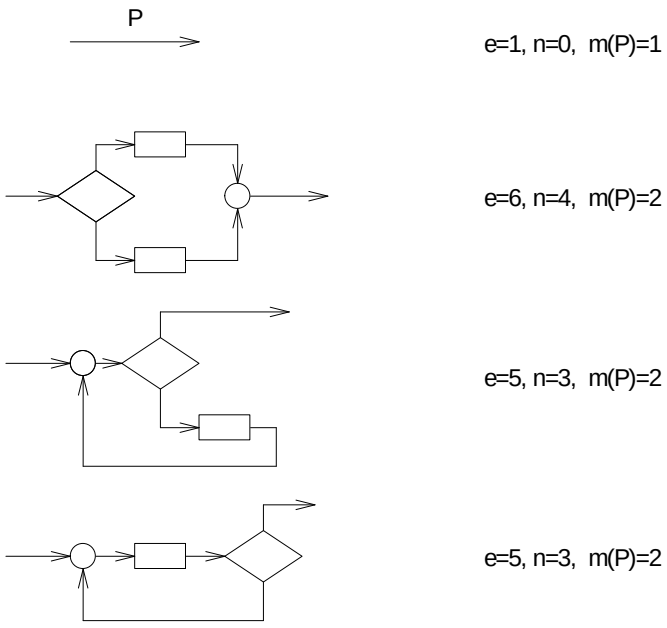


24. ábra

Az a) ábrán több kilépő éllel, a b) ábrán pedig több belépő éllel rendelkező ciklust látunk. A a c) ábrán több kilépő éllel, a d) ábrán pedig több bemenő éllel rendelkező döntést (elágazást) látunk.
9.3 Tétel: *Egy nem strukturált programnak a 24. ábra gráfjai közül legalább kettőt kell tartalmaznia.*

3.5 Programok szerkezeti bonyolultsága

Programok szerkezeti bonyolultságát többféleképpen mérjük. McCabe javasolta 1976-ban a vezérlőgráf ciklomatikus számának használatát. Egy összefüggő (G, V) gráf ciklomatikus száma: $V(G) = e - n + 1$, ahol e az élek száma, n pedig a csomópontok száma.
9.5 Definíció: *A P programgráf ciklikus bonyolultságán az $m(P) = e - n$ számot értjük.*
Példa (strukturált elemi részgráfok)



25. ábra

9.4 Tétel: *Egy P programgráf ciklikus bonyolultságára fennáll, hogy $m(P) = n_p + 1$, ahol n_p a program predikátum csomópontjainak száma.*

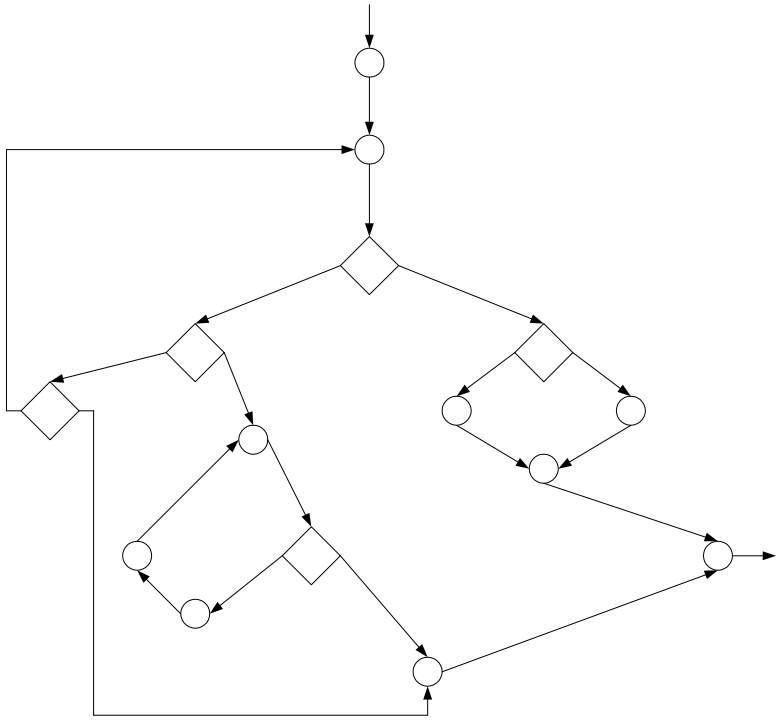
Bizonyítás: A 9.1 Lemma alapján $e = 3n_p + n_f + 1$. Másrészt $n = n_p + n_f + n_c = 2n_p + n_f$. Ezért $e - n = n_p + 1$. Q.E.D.

9.5 Tétel: *Nem strukturált program ciklikus bonyolultsága legalább 3.*

Bizonyítás: A 9.3 Tétel alapján a program legalább két szerkezetét tartalmazza a 24. ábrának. Ezért $n_p \geq 2$ és $m\,(P) \geq 3$. Q.E.D.

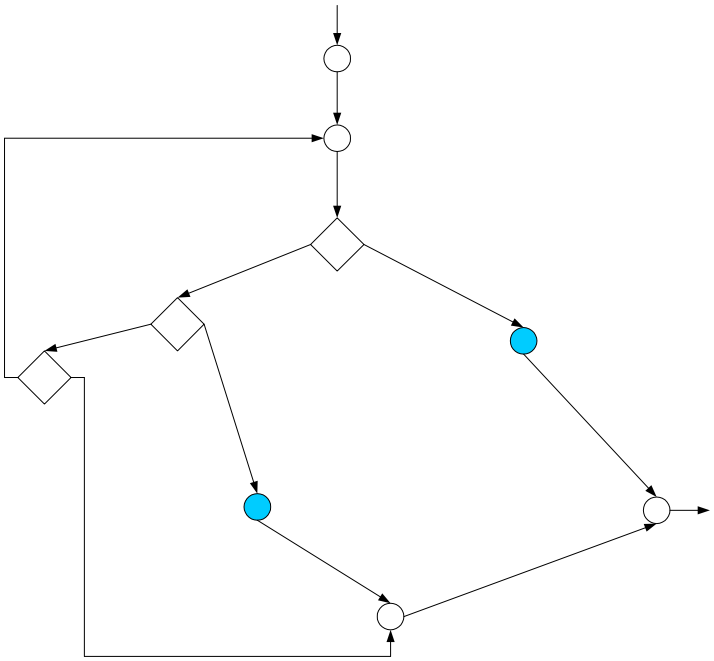
A strukturált program lebontható egy ciklikus bonyolultságú programgráfra. Nemstrukturált program bonyolultságát lebontással csak bizonyos mértékig lehet csökkenteni.

Példa:



26. ábra

Az ábra nem strukturált programjában 5 predikátum csomópont van. Ezért a program ciklikus bonyolultsági száma 6. Ha a programot lebontjuk, akkor a következő gráfot kapjuk:



27. ábra

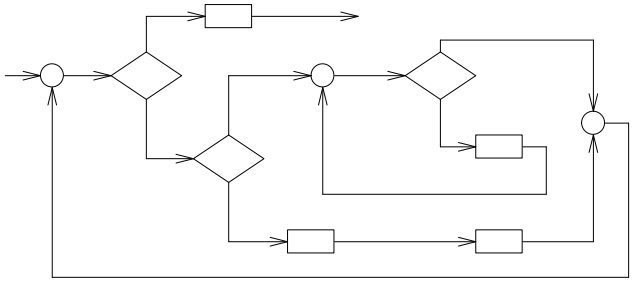
Ennek ciklikus bonyolultsága: 4.

Kézenfekvő gondolat egy program bonyolultságát a lebontással elérhető ciklikus bonyolultsági számmal mérni.

9.6 Definíció: *Jelölje k a P programgráf olyan részgráfjainak számát, amelyek az ITE, WD, DU formulák valamelyikével felírhatók. Ekkor a program lényeges bonyolultságán az $M(P) = m(P) - k$ számot értjük.*

Az előző példa esetén $M(P) = 4$, ami egyezik a lebontott gráf ciklikus bonyolultságával.

Példa (strukturált program)



28. ábra

A programgráf ciklikus bonyolultsága 4. Három strukturált részgráfja van, azaz $k = 3$. Ezért lényeges bonyolultsága $M(P) = 1$.

Nyilvánvaló a következő állítás.

9.1 Állítás: *A strukturált program lényeges bonyolultsága* 1.

Megjegyzés: *A most tárgyalt program bonyolultsági fogalom csak egy a sok gráfelméleti bázisú programminőségi mutató közül.*

4 Adattípusok és adatszerkezetek

Egy program/algoritmus adatokat alakit át:

$$\text{input adatok} \longrightarrow \text{közbülső adatok} \longrightarrow \text{output adatok.}$$

Az adatoknak bizonyos követelményeket kell kielégiteniük: csak meghatározott típusú adatok lehetnek.
Feladat: Vessük ezt össze a korábbi "memóriaállapot $\longrightarrow$ memóriaállapot" koncepcióval!
Alapfogalom:

elemi adattípus

összetett adattípus, vagy adatszerkezet

Az összetett adattípusok némi egyszerűsítéssel bizonyos típusú adatok együttesei valamilyen összefüggéssel. A két alapfogalom között nincs éles határ (van amit ide-is, oda is sorolnak).
Megjegyzések:

1. *Számos axiomatikus felépítés létezik, de nincs általánosan elfogadott.*
2. *Leginkább a program-, vagy metanyelvekhez kötődő leírásokat használjuk.*
3. *Az adattípusok, adatszerkezetek géptől és programnyelvtől függnek (implementálásfüggők).*
4. *Az adattípusokhoz hozzá kell érteni a velük végezhető műveleteket is.*
5. *Vannak alaptípusok és belőlük létrehozható bonyolultabb, un. strukturált, vagy összetett típusok. (Pascal nyelvvel kezdve). Ezek már adatszerkezeteknek tekinthetők.*

Legyen L egy programnyelv, D pedig a nyelvben definiálható adatok halmaza. Tetszőleges $d \in D$ adatnak pontosan egy meghatározott típusa van. Ezért a D halmaz felbontható

$$D = D_{t_1} \cup D_{t_2} \cup \ldots$$

alakban, ahol $t \neq r$ esetén $D_t \cap D_r = \emptyset$ és $T = \{t_1, t_2, \ldots\}$ az L nyelvben megengedett típusok azonosítóiból álló halmaz. Eszerint az L nyelv t típusú adatainak halmaza

$$D_t = \{d \mid d \in D,\ \textit{típus}\,(d) = t\}\,.$$

Legyen F az L programnyelv műveleteinek halmaza. Ezek a műveletek lehetnek aritmetikai, logikai, konverziós és egyéb műveletek. Minden $f \in F$ művelet egy

$$f : D_{t_1} \times D_{t_2} \times \ldots \times D_{t_n} \to D_{r_1} \times D_{r_2} \times \ldots \times D_{r_m}$$

alakú leképezés. Formailag a programnyelvet a (D, F) pár adja meg.

Adattipusok, adatszerkezetek megadására az alábbi közelítést választjuk:

1. Alap adattípusok.
2. Strukturált, vagy összetett adattípusok.
3. Mutató típus.

A strukturált, vagy összetett típusokat az egyszerű (és/vagy összetett) típusokból képezzük valamilyen konstrukciós elvvel.

A $t_1, \ldots, t_n$ típusokból képezett $t \in T$ típust a

$$C : D_{t_1} \times D_{t_2} \times \ldots \times D_{t_n} \to D_t$$

konstrukciós függvény (konstruktor) és az

$$S_i : D_t \to D_{t_i} \quad (i = 1, \ldots, n)$$

szelekciós függvény jellemzi. A $t \in T$ típusú adatokat a C leképezés értékkészlete adja, azaz

$$D_t = \{C\,(d_1, \ldots, d_n) \mid d_1 \in D_{t_1}, \ldots, d_n \in D_{t_n}\}\,.$$

Ha $d \in D_t$, akkor $S_i(d)$ adja meg a t-típusú adat t_i-típusú komponensét (összetevőjét). A C és S_i függvényeknek ki kell elégíteniük a következő szabályokat:

$$S_i(C(d_1, \ldots, d_n)) = d_i \quad (d_1 \in D_{t_1}, \ldots, d_n \in D_{t_n})$$

és

$$C(S_1(d), \ldots, S_n(d)) = d \quad (d \in D_t).$$

Az alábbiakban felsoroljuk a főbb adattípusokat.

1. Elemi adattípusok:

Egyszerű (v. skalár, v. megszámozható) adattípus

Szabványos elemi adattípusok

- logikai (Pascal nyelvben: Boolean)
- karakter (Pascal nyelvben: char)
- egész (Pascal nyelvben: integer)
- valós (Pascal nyelvben: real)

2. Struktúrált adattípusok:

tömb
rekord
egyesítés
halmaz
sorozat
rekurzív típus (nem tárgyaljuk)

A Pascal nyelvben az egyszerű adattípusok között szerepel a részintervallum típus, amelyet megszámozható adattípus esetén lehet definiálni.

Az elemi adattipusokat nem tárgyaljuk.

4.1 A tömb típus

A tömb típus a lineáris algebrából ismert vektor és mátrix fogalom megfelelője. Ennek megfelelően a tömb típus definíciója (konstrukciója) igen egyszerű.

Legyen T_0 egy már definiált típus. A

$$T_0^n = \underbrace{T_0 \times T_0 \times \ldots \times T_0}_{n}$$

direkt szorzat elemeit (rendezett elem n-eseket) tekintjük n (hosszúságú) T_0 tipusú tömböknek (vektoroknak). A T_0 típusú tömbök (adatok) száma: $|T_0|^n$.

A konstrukciós függény:

$$x_1, x_2, \ldots, x_n \in T_0 \to (x_1, \ldots, x_n) \in T_0^n.$$

A szelektor:

$$(x_1, \ldots, x_n) \in T_0^n \overset{x[i]}{\to} x_i \in T_0.$$

A T_0 alaptípus maga is lehet tömb. Pl. kétdimenziós T_0 tipusú tömböt ($m \times n$ mátrixot) a T_0^n tömb m-szer önmagával vett direkt szorzatával képezhetünk:

$$T_0^{m \times n} = \underbrace{T_0^n \times T_0^n \times \ldots \times T_0^n}_{m}.$$

Ennél a definíciónál némileg bonyolultabb a következő - Pascal nyelvhez kötődő - definíció.

Legyen I a valós, vagy egész típust kivéve egy megszámozható, vagy részintervallum típus, T_0 pedig tetszőleges alaptípus. Az I indexhalmazú T_0 típusú tömb Pascal definíciója:

$$\textbf{type } T = \textbf{array } I\textbf{of } T_0$$

Itt a T típusú A tömböt az

$$A : D_I \to D_{T_0}$$

leképezéssel, pontosabban ennek értékkészletével azonosítjuk. Tehát a T típus az $D_I \to D_{T_0}$ leképezések halmazaként is felfogható. Legyen $I = \{i_1, i_2, \ldots, i_n\}$. Ekkor az A leképezés értékkészlete

$$\{A[i_1], A[i_2], \ldots, A[i_n]\},$$

ahol $A[i]$ az A értéket jelöli az $i \in I$ elemen. Az A tömböt az

$A[i_1]$	$A[i_2]$	$\ldots$	$A[i_n]$
i_1	i_2		i_n

alakban is ábrázolhatjuk.

Tekintve, hogy I rendezett és megszámlálható, a T tömb típust azonosíthatjuk az $T_0^{|I|}$ halmazzal is, ami éppen a tömb első meghatározásának felel meg.

Példák:

type A =**array** [1..20] **of** integer

type $alkalmazott = (Tanaka, Nakamura, Kato, Yamamoto)$;
$jovedelem$ =**array** [$alkalmazott$] **of** integer

Legyen $a_k = A[i_k]$ $(k = 1, \ldots, n)$. Ekkor a T típusú A tömböt az

$$A = T(a_1, \ldots, a_n) = T(A[i_1], \ldots, A[i_n]) = (A[i_1], \ldots, A[i_n])$$

formában írhatjuk fel. A $T : D_{T_0^n} \to D_T$ függvényt *tömbkonstruktornak* nevezzük. A konstruktor inverz művelete a *szelektor*, amely a tömb egyes komponenseit választja ki:

$$A[i]$$

az $i \in I$ komponensnek megfelelő érték, $[i] : D_T \to D_{T_0}$ indexfüggvény,

$$T(a_1, \ldots, a_n)[i] = a_i.$$

Az i index kifejezés kiértékelésével is megkapható. Adott i index esetén az i-edik komponens értékét megváltoztathatjuk:

$$A[i] := a$$

ahol a egy T_0 típusú adat.

Példa: A $jovedelem$ típus esetén a $Kato$ indexű elem/komponens: $A[Kato]$.

4.1.1 Többdimenziós tömbök (mátrixok)

Tömbtípusok összetevői maguk is lehetnek strukturáltak. Egy tömbváltozót, amelynek komponensei szintén tömbök, *mátrixnak* nevezünk. Legyen $I_1, I_2, \ldots, I_n$ a valós, vagy egész típust kivéve egy megszámozható, vagy részintervallum típus, T_0 pedig tetszőleges alaptípus. Az n dimenziós tömb típust a

$$\textbf{type } T = \textbf{array } [I_1] \textbf{ of array } [I_2] \textbf{ of } \ldots \textbf{ of array } [I_n] \textbf{ of } T_0$$

vagy az ekvivalens

$$\textbf{type } T = \textbf{array } [I_1, I_2, \ldots, I_n] \textbf{ of } T_0$$

előírás definiálja.

Ha A egy T típusú tömb, akkor az

$$A : D_{I_1} \times D_{I_2} \times \ldots \times D_{I_n} \to D_{T_0}$$

leképezés értékkészletével azonosítjuk.

Az A tömb egy elemét $A[i_1, i_2, \ldots, i_n]$ formában adjuk meg, ahol $i_1 \in I_1, i_2 \in I_2, \ldots, i_n \in I_n$. Ez ugyanaz mint a szelektorok egymásután fűzésével kapott alak: $A[i_1][i_2]\ldots[i_n]$.

Példák:

$T =$**array** [1..10] **of array** [1..5] **of** real

$T =$**array** [1..10, 1..5] **of** real

Az elnevezés eredete: $A = [a_{ij}]_{i,j=1}^{m,n}$ mátrix szokásos alakja:

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix},$$

ahol a_{ij} az (i, j) indexű elem. Az $A =$**array** $[1..m, 1..n]$ **of** T_0 tömb ezzel analóg.

4.2 A rekord típus

A rekord típus definíciója (konstrukciója) is egyszerű. Rekordoknak a

$$T = \underbrace{T_1 \times T_2 \times \ldots \times T_n}_{n}$$

direkt szorzat elemeit (rendezett elem n-eseket) tekintjük. A T_i típushalmazok között lehetnek azonosak is, de az egyes komponenseket (mezőket) különbözőnek tekintjük.

A konstrukciós függény:

$$x_1 \in T_1, x_2 \in T_2, \ldots, x_n \in T_n \rightarrow (x_1, \ldots, x_n) \in T.$$

A selector:

$$(x_1, \ldots, x_n) \in T \stackrel{x[i]}{\rightarrow} x_i \in T_i.$$

A tömbhöz hasonlóan, képezhetjük a rekordok rekordjait is, és így tovább.

Tekintsük most a rekord Pascal nyelvhez kötődő definícióját is! Legyen $T_1, T_2, \ldots, T_n$ típusazonosító, $s_1, s_2, \ldots, s_n$ pedig névazonosítók. A $T_1, \ldots, T_n$ típusokból álló rekord típus definíciója:

type T = **record** $s_1 : T_1$;
$s_2 : T_2$;
...
$s_n : T_n$
end

Az $s_1, \ldots, s_n$ azonosítók a rekord komponensek (mezők) nevei. A T típusú rekordok halmaza

$$D_T = D_{T_1} \times D_{T_2} \times \ldots \times D_{T_n}.$$

Minden $d \in D_T$ felírható $d = (d_1, \ldots, d_n)$ alakban, ahol $d_i \in D_{T_i}$. Ezt a tömbhöz hasonlóan ábrázolhatjuk:

d_1	d_2	...	d_n
s_1	s_2		s_n

Példák:

type *ido* =**record** *ora* : 1..24
perc : 1..60
sec : 1..60
end

Megjegyzés: *T_i lehet összetett típusú is.*

Legyen $d_i \in D_{T_i}$ $(i = 1, \ldots, n)$. Ekkor a megfelelő T típusú d rekordot a

$$d = T(d_1, d_2, \ldots, d_n) = (d_1, d_2, \ldots, d_n)$$

konstruktor függvény adja meg. A rekord egyes komponenseit (mezőit) az

$$.s_i : D_T \to D_{T_i} \quad (i = 1, \ldots, n)$$

szelektorfüggvények adják meg:

$$d.s_i = d_i$$

Teljesülnek a következő feltételek:

$$T(d_1, d_2, \ldots, d_n).s_i = d_i, \quad T(d.s_1, d.s_2, \ldots, d.s_n) = d.$$

Az s_i mező értékadása:

$$d.s_i := d_i$$

Példák:

$x.ora = 11;$
$x.perc := 20;$
$x.\sec := 35$

Megjegyzés: *A tömb és rekord közötti azonosságok és különbségek:*

1. Tömb esetén $T_1 = T_2 = \cdots = T_n = T_0$

2. (Főkülönbség) A tömb indexei rendezett tipusúak és így adott sorrendben feldolgozhatók. A rekord komponensei rendezetlenek és nem formálnak egy adott típust.

3. A tömb és a rekord asszociatív adatszerkezetek.

4.3 Egyesítés típus

Legyen A és B két nemüres halmaz. Az A és B halmazok diszjunkt egyesítése

$$A \oplus B = \{(s, 1) \mid s \in A\} \cup \{(s, 2) \mid s \in B\}.$$

Legyen T_1 és T_2 két adattípus, t_1 és t_2 a típusok azonosítói. A két típus T egyesítése a $t_1 : d_1$ és $t_2 : d_2$ $(d_1 \in D_{T_1},\ d_2 \in D_{T_2})$ párokból áll. Formális deklarációja:

$$\begin{array}{lll} \textbf{type } T & = \textbf{union} & t_1 : T_1; \\ & & t_2 : T_2 \\ & \textbf{end} & \end{array}$$

Tehát

$$D_T = \{t_1 : d_1 \mid d_1 \in D_{T_1}\} \cup \{t_2 : d_2 \mid d_2 \in D_{T_2}\}$$

és a T egyesített típus egy elemét a következőképpen ábrázolhatjuk:

t_i	d_i

Példa:

$$\begin{array}{lll} \textbf{type } \mathit{coordinate1} & = \textbf{record} & \\ & & x, t : \text{real} \\ & \textbf{end} & \end{array}$$

$$\begin{array}{lll} \textbf{type } \mathit{coordinate2} & = \textbf{record} & \\ & & r : \text{real}; \\ & & \theta : \mathit{angle} \\ & \textbf{end} & \end{array}$$

$$\begin{aligned}\textbf{type } coordinate \quad &= \textbf{union} \\ &\quad Cartesian : coordinate1; \\ &\quad Polar : coordinate2 \\ &\textbf{end}\end{aligned}$$

Az egyesítés egy lehetséges Pascal megvalósítása:

$$\begin{aligned}\textbf{type} \qquad ckind \quad &= (Cartesian, Polar) \\ coordinate \quad &= \textbf{record } ckind\textbf{of} \\ &\quad Cartesian : (x, y : \text{real}) ; \\ &\quad Polar : (r : \text{real}; \theta : angle) \\ &\textbf{end}\end{aligned}$$

Megjegyzés: *Az egyesítés kettőnél több komponensre is kiterjeszthető. Az union szerkezet nem Pascal utasítás. A Pascalban a szerkezetet váltakozó rekordnak hívják (case utasítással valósítják meg).*

Legyen $d \in D_T$. Két eset lehetséges: d vagy $t_1 : d_1$ ($d_1 \in D_{T_1}$), vagy $t_2 : d_2$ ($d_2 \in D_{T_2}$). Ennek megfelelően a T konstrukciós függvényt két részletben adhatjuk meg: $T = t_1$, vagy $T = t_2$, ahol $t_i : D_{T_i} \to D_T$ $(i = 1, 2)$. A $d_i \in D_{T_i}$ értékhez a $t_i : d_i \in D_T$ pár kerül hozzárendelésre. A

$$.t_i : D_T \to D_{T_i} \quad (i = 1, 2)$$

szelektor függvényeket a következőképpen adhatjuk meg:

$$d.t_1 = \begin{cases} d_1, & \text{ha } d = t_1 : d_1 \\ \text{definiálatlan egyébként} \end{cases} \qquad d.t_2 = \begin{cases} \text{definiálatlan egyébként} \\ d_2, & \text{ha } d = t_2 : d_2 \end{cases}$$

A fenti definíciókkal teljesülnek a

$$d.t_i = (t_i : d_i)\, .t_i = d_i \quad (d_i \in D_{T_i})$$

$$t_i : (d.t_i) = t_i : d_i = d \quad (d \in D_T,\ d.t_i \text{ definiált})$$

feltételek.

4.4 Halmaztípus

Jelölje [] az üres halmazt, $[a..b]$ részintervallum típusú halmazt, $[a, b, c, d, \ldots]$ felsorolással megadott halmazt. A halmaz típus egy megadott T_0 alaptípus részhalmazainak halmazát jelenti. Formális definíciója:

$$\textbf{type } T = \textbf{set of } T_0.$$

A T_0 típus részhalmazai a

$$D_T = 2^{T_0} = \{S \mid S \subset D_{T_0}\}$$

hatványhalmazt alkotják.

Példa:

type $T =$**set of** $[1..3]$

esetén

$$D_T = \{[]\,, [1]\,, [2]\,, [3]\,, [1, 2]\,, [1, 3]\,, [2, 3]\,, [1, 2, 3]\}\,.$$

A halmaztípussal megengedett műveletek:

1. $A + B$ $(A + B = A \cup B)$
2. $A * B$ $(A * B = A \cap B)$
3. $A - B$ $(A - B = A \setminus B)$

A logikai relációk:

1. $A = B$
2. $A \neq B$

3. $A \leq B \Longleftrightarrow A \subset B$
4. $A \geq B \Longleftrightarrow B \leq A$
5. a **in** B

$$a\textbf{in } B = \begin{cases} true, & \text{ha } a \in B \\ false, & \text{ha } a \notin B \end{cases}$$

Feladat: Mi a konstruktor és mi a szelektor a halmaztípus esetén? Hány elem T típusú halmaz?

4.5 Sorozat típusok

A sorozat típust felfoghatjuk úgy, mint egy T_0 típusú adatokból álló változó hosszúságú tömbök (vektorok) halmazát:

$$\textbf{type } T = \textbf{sequence of } T_0.$$

Ennek megfelelően

$$D_T = \cup_{n=0}^{\infty} D_{T_0^n},$$

ahol $T_0^n = \underbrace{T_0 \times \ldots \times T_0}_{n}$.

Példák:

type *page* =**sequence of** *character*
type *book* =**sequence of** *page*
=**sequence of sequence of** *character*

Az $e_1, e_2, \ldots, e_n$ elemekből álló T típusú sorozatot

$$T\langle e_1, e_2, \ldots, e_n \rangle$$

formában jelöljük. Ha $n = 0$, akkor ezt $T\langle\rangle$ jelöli, és üres sorozatnak nevezzük. Az elemek kiválasztására a következő függvényeket vezetjük be. Legyen x sorozat.

a) $x[i]$ - az x sorozat i-edik eleme
b) $first(x)$ - az x sorozat első eleme
c) $last(x)$ - az x sorozat utolsó eleme

Ha $x = T\langle e_1, e_2, \ldots, e_n \rangle$, akkor

$$x[i] = e_i \quad (1 \leq i \leq n), \quad first(x) = e_1, \quad last(x) = e_n.$$

Tehát $[i], first, last : D_T \to D_{T_0}$.

Műveletek sorozatok elemeivel:

a) $tail(x)$ - az x sorozat első elemének elhagyásával nyert sorozat
b) $initial(x)$ - az x sorozat utolsó elemének elhagyásával nyert sorozat
c) $appendl(x, e)$ - az e elem x sorozat elé írásával nyert sorozat
d) $appendr(x, e)$ - az e elem x sorozat után írásával nyert sorozat.

Ha $x = T\langle e_1, e_2, \ldots, e_n \rangle$, akkor

$$\begin{aligned} tail(x) &= T\langle e_2, \ldots, e_n \rangle, \\ initial(x) &= T\langle e_1, \ldots, e_{n-1} \rangle, \\ appendl(x, e) &= T\langle e, e_1, e_2, \ldots, e_n \rangle, \\ appendr(x, e) &= T\langle e_1, e_2, \ldots, e_n, e \rangle. \end{aligned}$$

Értelemszerűen:

$$tail, initial : D_T \to D_T, \quad appendl, appendr : D_T \times D_{T_0} \to D_T.$$

Teljesülnek a következő összefüggések:

$first(appendl(x, e)) = e,$

$tail\,(appendl\,(x,e)) = x$,
$appendl\,(tail\,(x)\,, first\,(x)) = x$, ha $x \neq T\,\langle\rangle$,
$last\,(appendr\,(x,e)) = e$,
$initial\,(appendr\,(x,e)) = x$,
$appendr\,(initial\,(x)\,, last\,(x)) = x$, ha $x \neq T\,\langle\rangle$.
Bevezetjük még a következő függvényt:

$$empty\,(x) = \begin{cases} true, & \text{ha } x = T\,\langle\rangle \\ false, & \text{ha } x \neq T\,\langle\rangle \end{cases}$$

Háromféle sorozat típust vizsgálunk:

1. Szekvenciális fájl
2. Verem
3. Sor

4.5.1 Szekvenciális fájl

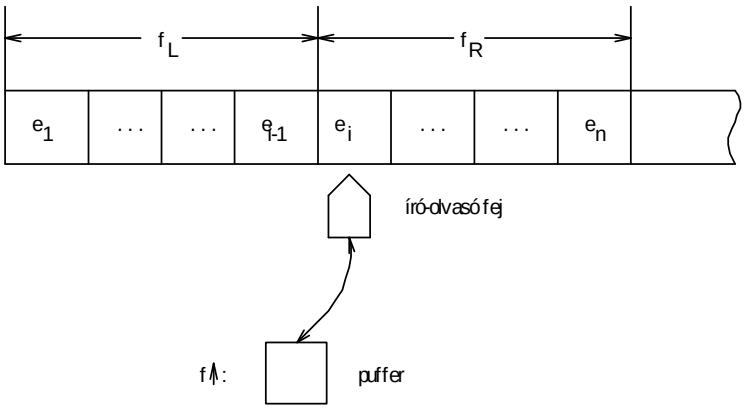
A szekvenciális fájl megadása:

type $T =$ **file of** T_0

Példa:

type *szoveg* =**file of** *char*

Az f szekvenciális fájl szerkezetet[1] legegyszerűbben egy mágnesszalagon egymásután elhelyezett adatokkal jeleníthetjük meg:



Olvasási helyzet

A fájl elemeinek olvasása és írása egy író-olvasó fejen keresztül történik. Egyszerre csak egy adatot lehet olvasni, amelyet az "író-olvasó fej" pozíciója határoz meg. Ha a pozíció az i-edik elemnél van, akkor a fájl tartalmát két halmazra bontja:

$$f_L = \langle e_1, \ldots, e_{i-1}\rangle\,, \quad f_R = \langle e_i, \ldots, e_n\rangle\,.$$

Az egész fájl tartalma ekkor konkatenációval

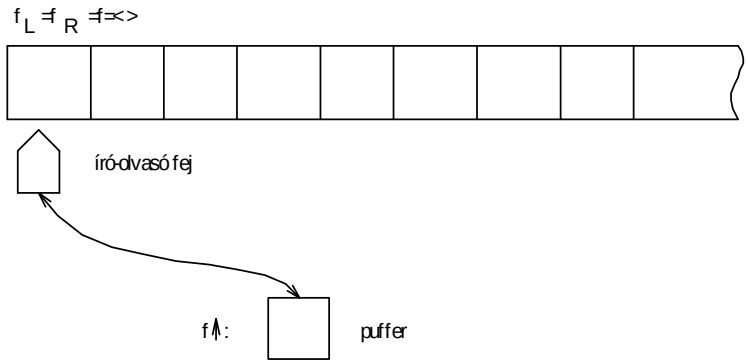
$$f = f_L f_R = \langle e_1, \ldots, e_{i-1}, e_i, \ldots, e_n\rangle\,.$$

Az írás-olvasás egy pufferen keresztül történik, amelynek tartalmát az $f\uparrow$ változó tartalmazza. Az $f\uparrow$ típusa azonos a fájl elemek típusával.

[1] A fájl tkp. a külső fizikai tárolási formákból keletkezett.

A szekvenciális fájloknál 4 műveletet (*rewrite*, *reset*, *write*, *read*) és egy logikai függvényt (eof) engedünk meg.

A szekvenciális fájlban nem lehet az egyes komponenseket felülírni. Az egyetlen lehetséges út a fájl törlése és újraírása. Ezt a *rewrite* (újraírás) utasítás végzi el.



Az újraírás eredménye

A $rewrite\,(f)$ művelet az

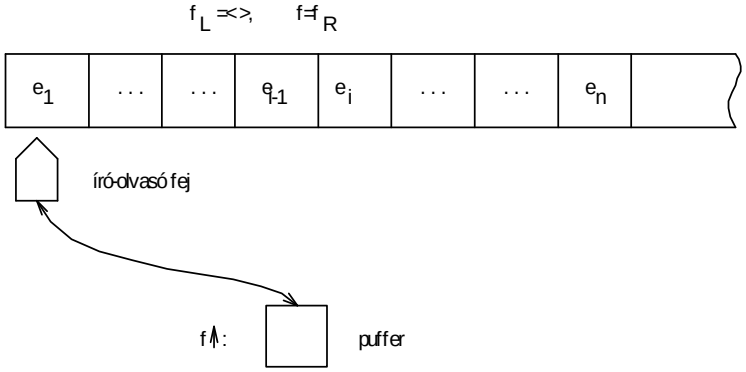
$$f := \langle\rangle$$

értékadást jelenti. Ekkor írhatjuk azt is, hogy

$$rewrite\,(f) \Longleftrightarrow f_L = \langle\rangle\,,\ f_R = \langle\rangle\,.$$

Vegyük észre, hogy a *rewrite* utasítással hozhatunk létre üres fájlt.

A *reset* utasítással az író-olvasó fej a fájl első pozíciójára helyezhető:



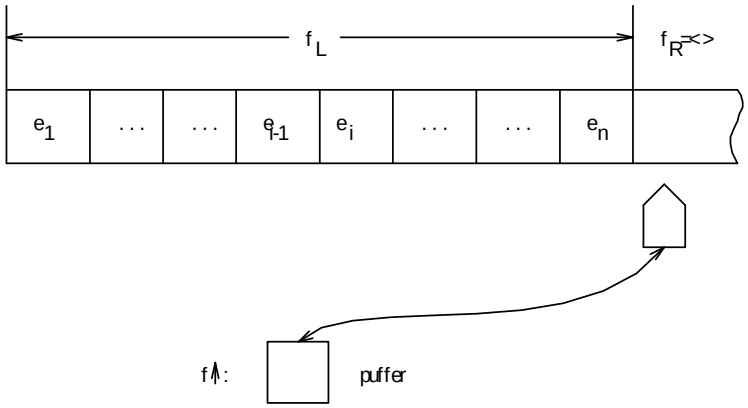
A reset utasítás eredménye

Az $eof\,(f)$ (end of file) logikai függvény a fájl végét jelzi. Definíciója:

$$eof\,(f) \equiv f_R \equiv \langle\rangle\,.$$

Tehát $eof\,(f)$=*true*, ha $f_R = \langle\rangle$, és $eof\,(f) = false$, ha $f_R \neq \langle\rangle$.

A fájlba új elemet írni csak az utolsó elem után lehetséges a *write* (Pascalban $put\,(f)$) utasítással.



írási helyzet

Az írási pozicióban $f_R = \langle\rangle$, $eof\,(f) = true$ és

$$write\,(f, e) \Longleftrightarrow f_L = appendr\,(f_L, e)\,,\ f_R = \langle\rangle\,.$$

Figyeljük meg, hogy az írás után, újra írási pozicióba kerülünk.

A fájlból olvasni csak az $eof\,(f) = hamis$ esetben lehetséges. A $read$ (Pascalban $get\,(f)$) olvasási utasítás végrehajtása után az olvasó fej egy pozícióval jobbra mozdul:

$$read\,(f, e) \Longleftrightarrow f_L = appendr\,(f_L, first\,(f_R))\,,\ f_R = tail\,(f_R)\,,\ e = first\,(f_R)\,.$$

Itt e azt az elemet jelöli, amelynél az író-olvasó fej aktuálisan áll.

Megjegyzés: *Különböző Pascal verziókban a fájl utasítások elnevezései mások is lehetnek.*

4.5.2 Verem

A verem olyan sorozattípus, ahol az írás és a törlés a sorozat ugyanazon végén történik (Last in First Out=LIFO):

var: $s:$ **stack of** T_0;
$e : T_0$

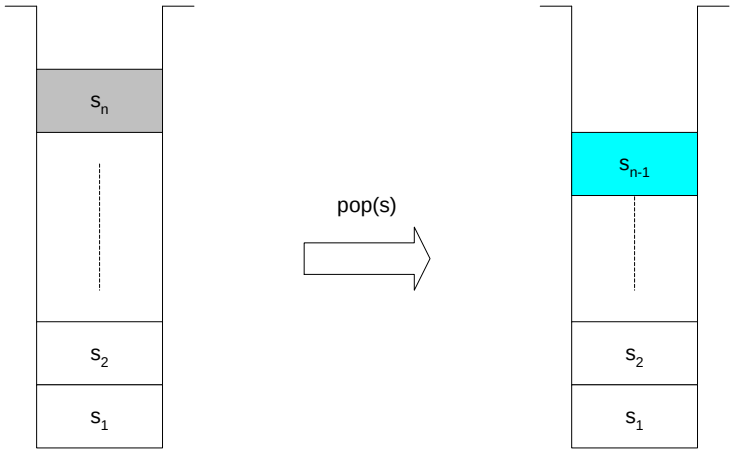
Három müvelete van:

1. *top* - megadja a sorozat utolsó elemét (verem tetejét):

$$top\,(s) = last\,(s)\,.$$

2. *pop* - törli a sorozat utolsó elemét
3. *push* - új elemet ír a sorozat végére

A törlési művelet:

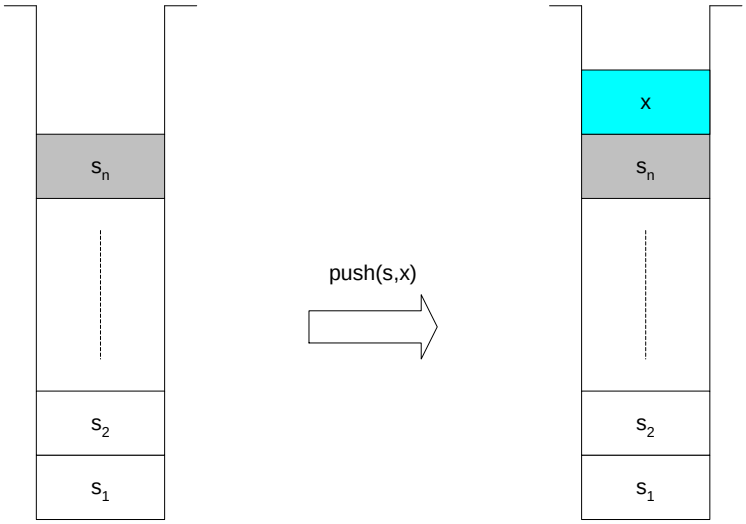


Verem felsõ elemének törlése

Képlettel kifejezve:

$$pop\,(s) \Longleftrightarrow s = initial\,(s)\,.$$

Az írási művelet:



Irás verem tetejére

Képlettel kifejezve:

$$push\,(s,x) \Longleftrightarrow appendr\,(s,x)\,.$$

4.5.3 Sor

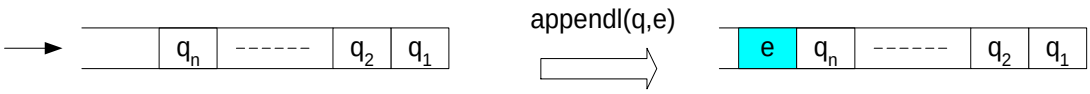
A sor olyan sorozattípus, ahol az írás a sorozat elején (*tkp. végén*), a törlés a sorozat végén (*tkp. elején*) történik (First in First Out=FIFO):

var: q : **queue of** T_0;
$e : T_0$

Két művelete van:

1. *enter* - új elem írása a sorozat végére
2. *leave* - a sorozat első elemének törlése

Az írási művelet:

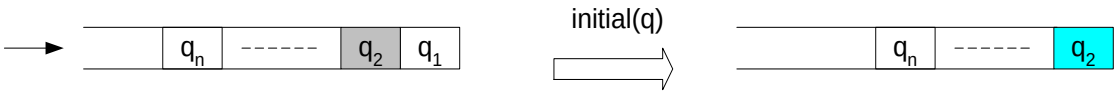


Elem hozzáadása sorhoz

Képlettel kifejezve:

$$enter\,(q,e) \Longleftrightarrow q = appendl\,(q,e)\,.$$

A törlési művelet:



Elem törlése sorból

Képlettel kifejezve:

$$leave\,(q) \Longleftrightarrow q = initial\,(q)\,.$$

5 Programozási tételek

Programozási feladatok megoldásakor a top-down (strukturált) programtervezés esetén három vezérlési szerkezetet használunk:

- szekvencia
- elágazás
- ciklus

Eddig megismertük az alábbi fogalmakat:

- állapottér ($A = A_1 \times \ldots \times A_n$)
- változó ($a_i \in A_i$)
- feladat ($F \subseteq A \times A$)
- program ($S \subseteq A \times A^{**}$)
- programfüggvény (a programfutás eredménye) ($p\,(S) \subseteq A \times A$)

Egy S program megoldja az F feladatot, ha $D_F \subseteq D_{p(S)}$ és $\forall a \in D_F : p\,(S)\,(a) \subseteq F\,(a)$. A "megoldás" ellenőrzésére és a programhelyesség részleges bizonyítására bevezettük az elő- és utófeltétel, valamint a leggyengébb előfeltétel fogalmát:

Legyen $Q, R : A \to \mathbb{L}$ két állítás, S program. A $\{Q\}\,S\,\{R\}$ szimbólummal jelöltük azt, hogy $\forall a \in [Q]$ esetén $p\,(S)\,(a) \subseteq [R]$. A $[Q]$ előfeltétel halmazt felfoghatjuk az állapottér azon részeként, amelyre meg akarjuk a feladat megoldását kapni. Az elérendő eredmény megadására szolgál az $[R]$ utófeltétel halmaz. A $\{Q\}\,S\,\{R\}$ szimbólummal jelölt implikáció igazolása jelenti a parciális programhelyesség igazolását. Az $lf\,(S,R)$ leggyengébb előfeltétel a legbővebb Q előfeltételt jelenti.

Egy feladat megadása (specifikációja) a következőképpen történhet:

- bemenő (input) adatok
- kimenő (output) adatok
- a feladatban használt fogalmak definíciója
- az eredmény kiszámítási szabálya
- a bemenő adatokra kirótt feltételeket (előfeltételek)
- a kimenő adatokra kirótt feltételek (utófeltételek).

Lényeges észrevétel: a feladatok osztályokba sorolhatók a feladat jellege szerint és a feladatosztályokhoz a teljes feladatosztály megoldó algoritmusokat tudunk készíteni. Ezeket a tipus feladatokat *programozási tételeknek* nevezzük, mert igazolható, hogy megoldásaik a feladat garantáltan helyes megoldásai.

A programozási tételek ismeretében a "legtöbb" feladat megoldása során "csak" a megfelelő programozási tételt kell alkalmazni. Így elvileg helyes (de nem mindig optimális) megoldást kapunk.

A feladatok olyanok, hogy egy, vagy több adatsokasághoz rendelünk valamilyen eredményt. Egyszerűség kedvéért feltesszük, hogy ezek sorozatok, amelyeket tömbbel ábrázolhatunk. Vizsgált feladatosztály típusok:

- egy sorozathoz egy értéket rendelő feladatok
- egy sorozathoz egy sorozatot rendelő feladatok
- egy sorozathoz több sorozatot rendelő feladatok
- több sorozathoz egy sorozatot rendelő feladatok.

5.1 Elemi programozási tételek

A feladatok általános alakja:

Input	:	$n \in \mathbb{N}_0$, $x \in H^n$, $F : H^* \to G$
Output	:	$S \in G$
Q	:	–
R	:	$S = F(x_1, \ldots, x_n)$

Egy bemenő sorozattól függő S értéket kell meghatároznunk. A feladatcsoporthoz több feladattípus tartozik.

5.1.1 Sorozatszámítás

Néhány példa:

F1. Adjuk meg n alkalmazottból álló osztály bértömegét a bérek ismeretében!

F2. Egy m elemű betűsorozat betűit füzzük össze egyetlen szövegtipusú változóba!

F3. A Balatonnál k madarász végez megfigyeléseket. Mindegyik megadta, hogy milyen madarakat látott. Készítsünk algoritmust, amely megadja a Balatonnál látott madarakat!

F4. Adjuk meg az első n természetes szám szorzatát!

A feladatokban közös: adatok sorozatához rendelünk egy értéket, amelyet, egy az egész sorozaton értelmezett függvény ad meg (n szám összege, m betű konkatenációja, k halmaz uniója, n szám szorzata).

Az algoritmus:

Változók:

n : egész	{a feldolgozandó sorozat elemszáma}
x : tömb(1..n:elemtipus)	{a feldolgozandó sorozat elemei}
F_0 : elemtipus$_1$	{a művelet nulleleme}
S : elemtipus$_2$	{az eredmény}

Input	:	$n \in \mathbb{N}_0$, $x \in H^n$, $F : H^* \to G$
Output	:	$S \in G$
Q	:	$\exists F_0 \in G$ (nullelem) és $\exists f : G \times H \to G$ és
		$F(x_1, \ldots, x_n) = f(F(x_1, \ldots, x_{n-1}), x_n)$ és $F() = F_0$
R	:	$S = F(x_1, \ldots, x_n)$

A feladat lehetséges variációi:

$$\begin{aligned} F &= \sum, \prod, \cup, \cap, \vee, \wedge, \& \text{ (konkatenáció)} \\ F_0 &= 0, 1, [], \text{alaphalmaz}, \mathit{igaz}, \mathit{hamis}, \text{""}. \end{aligned}$$

A megoldás indukció segítségével:

- $i = 0$ esetre tudjuk az eredmény: F_0,
- ha $(i-1)$-re tudjuk az F_{i-1} eredményt, akkor az i-re az eredményt $F_i = f(F_{i-1}, x_i)$ adja ($i = 1, \ldots, n$).

A feladatot megoldó algoritmus:

```
sorozatszámítás(n, x, s)
  s := F_0
  for i = 1 : n
    s := f(s, x(i))
  end
eljárás vége
```

Az F1. feladat esetén az eljárás:

```
sorozatszámítás(n, x, s)
  s := 0
  for i = 1 : n
    s := s + x(i)
  end
eljárás vége
```

Az F2. feladat esetén az eljárás:

```
sorozatszámítás(m, x, sz)
  sz := ""    {üres szöveg}
  for i = 1 : m
    sz := sz&x(i)
  end
eljárás vége
```

Az F3. feladat esetén:

```
unio(k, X, H)
  H := []
  for i = 1 : k
    H := H ∪ X(i)
  end
eljárás vége
```

Végül az F4. feladat esetén:

```
sorozatszámítás(n, x, p)
  p := 1
  for i = 1 : n
    p := p * x(i)
  end
eljárás vége
```

5.1.2 Eldöntés

A feladat annak eldöntése, hogy

a) van-e a sorozatban adott tulajdonságú elem

b) a sorozat mindegyik eleme rendelkezik-e ezzel a tulajdonsággal.

Az a) esetben a feladat alakja:

Input	:	$n \in \mathbb{N}_0$, $x \in H^n$, $T : H \to \mathbb{L}$
Output	:	$VAN \in \mathbb{L}$
Q	:	–
R	:	$VAN \equiv (\exists i \ \ (1 \leq i \leq n) : T(x_i))$

Az algoritmus:

Függvény:

T : *elemtípus* $\rightarrow \mathbb{L}$

Változók:

n : egész {a feldolgozandó sorozat elemszáma}

x : tömb(1..n:elemtipus) {a feldolgozandó sorozat elemei}

VAN : logikai {az eredmény}

A feladattípus megoldására az előző programozási tétel is alkalmas az $F = \vee$, $f = \vee$, $F_0 = hamis$ megfeleltetéssel:

eldöntés(n, X, S)
  $S := hamis$
  for $i = 1 : n$
    $S := S \vee T(x(i))$
  end
eljárás vége

Vegyük észre, hogy ha egyszer S igaz lesz, akkor igaz is marad. Tehát nem kell a ciklust végig számolni. Eszerint az igazi megoldó algoritmus:

eldöntés(n, x, VAN)
  $i := 1$
  while $(i \leq n) \wedge (\neg T(x(i)))$
    $i := i + 1$
  end
  $VAN := (i \leq n)$
eljárás vége

A b) esetben a megoldást az adja, hogy átfogalmazzuk:

$$\forall i : 1 \leq i \leq n : T(x_i) \equiv \nexists i : 1 \leq i \leq n \wedge \neg T(x_i).$$

A sorozatszámítás itt az $F = \wedge$, $f = \wedge$, $F_0 = igaz$ értékekkel lehet alkalmazni.

A megoldó algoritmus:

eldöntés$(n, x, MIND)$
  $i := 1$
  while $(i \leq n) \wedge (T(x(i)))$
    $i := i + 1$
  end
  $MIND := (i > n)$
eljárás vége

Példa: Döntsük el, hogy egy adott sorozat monoton növekedő-e! Esetünkben $T(x_i) \equiv (x_i \leq x_{i+1})$ és ezért a ciklus csak $(n-1)$-ig mehet. A megoldás:

eldöntés$(n, x, MONOTON)$
  $i := 1$
  while $(i \leq n-1) \wedge (x(i) \leq x(i+1))$
    $i := i + 1$
  end
  $MONOTON := (i > n-1)$
eljárás vége

5.1.3 Kiválasztás

A feladat sorozat adott tulajdonságú elemének (indexének) megadása, amelynek létezését feltesszük:

Input	:	$n \in \mathbb{N}$, $x \in H^n$, $T : H \to \mathbb{L}$
Output	:	$SORSZ \in \mathbb{N}$
Q	:	$\exists i \ (1 \leq i \leq n) : T(x_i)$
R	:	$(1 \leq SORSZ \leq n) \wedge T(x_{SORSZ})$

Az algoritmus:

Függvény:

T : *elemtípus* $\to \mathbb{L}$

Változók:

n : egész {a feldolgozandó sorozat elemszáma}

x : tömb(1..n:elemtipus) {a feldolgozandó sorozat elemei}

$SORSZ$: egész {az eredmény}

A megoldás hasonlít az előző feladattípushoz:

kiválasztás$(n, x, SORSZ)$
  $i := 1$
  while $\neg T(x(i))$
    $i := i + 1$
  end
  $SORSZ := i$
eljárás vége

Feladat: A program az első T tulajdonságú elemet adja meg. Hogyan kell módosítani, hogy az utolsó ilyet adja meg?

5.1.4 Lineáris keresés

A feladat adott tulajdonságú elem megadása egy sorozatban, ha van ilyen. Ha nincs, akkor ezt a tényt kell megadni. Tehát az előző két tétel mindegyikét tartalmazza:

Input	:	$n \in \mathbb{N}$, $x \in H^n$, $T : H \to \mathbb{L}$
Output	:	$VAN \in \mathbb{L}$, $SORSZ \in \mathbb{N}$
Q	:	–
R	:	$VAN \equiv (\exists i \ \ (1 \leq i \leq n) : T(x_i)) \wedge$
		$\wedge (VAN \Longrightarrow (1 \leq SORSZ \leq n) \wedge T(x_{SORSZ}))$

Az algoritmus:

Függvény:

T : *elemtípus* $\to \mathbb{L}$

Változók:

n : egész {a feldolgozandó sorozat elemszáma}

x : tömb(1..n:elemtipus) {a feldolgozandó sorozat elemei}

VAN : logikai {az eredmény - van-e megfelelő elem}

$SORSZ$: egész {az eredmény -a megfelelő elem sorszáma}

A megoldó algoritmust az eldöntés algoritmusa adja kiegészítve a kiválasztási algoritmus megfelelő

részével:

keresés$(n, x, VAN, SORSZ)$
  $i := 1$
  while $(i \leq n) \wedge (\neg T(x(i)))$
    $i := i + 1$
  end
  $VAN := (i \leq n)$
  if VAN **then** $SORSZ := i$
eljárás vége

5.1.5 Megszámolás

A feladat annak meghatározása, hogy hány adott tulajdonságú elem van egy sorozatban:

Input	:	$n \in \mathbb{N}_0$, $x \in H^n$, $T : H \to \mathbb{L}$, $\chi : H \to \{0,1\}$
		$\chi(x) = 1$, ha $T(x)$ és $\chi(x) = 0$, ha $\neg T(x)$
Output	:	$DB \in \mathbb{N}_0$
Q	:	–
R	:	$DB = \sum_{i=1}^{n} \chi(x_i)$

Az algoritmus:

Függvény:

$T : elemtípus \to \mathbb{L}$

Változók:

n : egész {a feldolgozandó sorozat elemszáma}
x : tömb(1..n:elemtipus) {a feldolgozandó sorozat elemei}
DB : egész {az eredmény -a megfelelő elemek száma}

A feladat egy sorozatszámítás (tkp. összegzés a χ függvény értékeire). A megoldó algoritmus:

megszámolás(n, x, DB)
  $DB := 0$
  for $i = 1 : n$
    if $T(x(i))$ **then** $DB := DB + 1$
  end
eljárás vége

5.1.6 Maximumkiválasztás

A feladat az, hogy válasszuk ki egy sorozat legnagyobb (legkisebb) elemét:

Input	:	$n \in \mathbb{N}_0$, $x \in H^n$, H rendezett halmaz ($\exists <, \leq$ reláció)
Output	:	$MAX \in \mathbb{N}$
Q	:	$n \geq 1$
R	:	$1 \leq MAX \leq n \wedge \forall i\ (1 \leq i \leq n) : x_{MAX} \geq x_i$

Az algoritmus:

Változók:

n : egész {a feldolgozandó sorozat elemszáma}
x : tömb(1..n:elemtipus) {a feldolgozandó sorozat elemei}
MAX : egész {a maximális értékű elem sorszáma}

A sorozatszámításnál $F(x_1, \ldots, x_n) = \max(x_1, \ldots, x_n)$ és

$$f(x, y) = \max(x, y)$$

függvényeket használjuk és az $F_0 = x_1$ választást (Nem neutrális elem, de az indexeléssel kompenzáljuk)! A megoldó algoritmus:

maximumkiválasztás(n, x, MAX)
  $MAX := 1$
  for $i = 2 : n$
    if $x\,(MAX) < x\,(i)$ **then** $MAX := i$
  end
eljárás vége

Ha a maximális értéket is akarjuk, akkor az algoritmus:

maximumkiválasztás$(n, x, MAX, MEXERT)$
  $MAX, MAXERT := 1, x\,(1)$
  for $i = 2 : n$
    if $MAXERT < x\,(i)$ **then** $MAX, MAXERT := i, x\,(i)$
  end
eljárás vége

A legkisebb érték meghatározásához a " $<$ " relációt át kell írni a " $>$ " relációra (A változóneveket is célszerű átírni a MIN, $MINERT$ nevekre).

5.2 Összetett programozási tételek

Sorozathoz sorozatot rendelő feladatokkal foglalkozunk.

5.2.1 Másolás

A bemenő sorozatot le kell másolni, s közben az elemekre vonatkozó átalakításokat lehet végezni rajta:

Input	:	$n \in \mathbb{N}_0$, $x \in H^n$, $f : H \to G$
Output	:	$y \in G^n$
Q	:	$--$
R	:	$\forall i\ (1 \leq i \leq n) : y_i = f\,(x_i)$

Az algoritmus:

Függvény:
  $f : H - elemtípus \to G - elemtípus$
Változók:
  n : egész {a feldolgozandó sorozat elemszáma}
  x : tömb(1..n:H-elemtipus) {a feldolgozandó sorozat elemei}
  y : tömb(1..n:G-elemtípus) {a feldolgozott sorozat}
A megoldás:

másolás(n, x, y)
  for $i = 1 : n$
    $y\,(i) := f\,(x\,(i))$
  end
eljárás vége

Példa: Egy szöveg minden magánhangzóját cseréljük ki az e betűre!

5.2.2 Kiválogatás

Meg kell adni egy sorozat adott tulajdonságú elemeinek számát és indexeit:

Input	:	$n \in \mathbb{N}_0$, $x \in H^n$, $T : H \to \mathbb{L}$, $\chi : H \to \{0,1\}$
		$\chi(x) = 1$, ha $T(x)$ és $\chi(x) = 0$, ha $\neg T(x)$
Output	:	$DB \in \mathbb{N}_0$, $y \in [1..n]^n$
Q	:	–
R	:	$DB = \sum_{i=1}^{n} \chi(x_i) \wedge y \subset [1..n] \wedge$
		$\wedge\ \forall i\ (1 \leq i \leq DB) : T(x_{y_i})$

Figyeljük meg, hogy az y tömb hossza előre nem meghatározható, de legfeljebb n.

Az algoritmus:

Függvény:

T : *elemtípus* $\to \mathbb{L}$

Változók:

n : egész {a feldolgozandó sorozat elemszáma}
x : tömb(1..n:elemtipus) {a feldolgozandó sorozat elemei}
DB : egész {a megfelelő elemek száma}
y : tömb(1..n:egész) {a megfelelő elemek sorszámai}

A megoldás:

```
kiválogatás(n, x, DB, y)
  DB := 0
  for i = 1 : n
    if T(x(i)) then
      DB := DB + 1
      Y(DB) = i
    end
  end
eljárás vége
```

Példa: Adjuk meg egy osztály jeles tanulóit!

Feladat: Módosítsuk a feladatot és az algoritmust, hogy ne a sorszámokat, de magukat az elemeket gyüjtse ki.

5.2.3 Szétválogatás

A feladat egy sorozat elemeinek szétválogatása két részsorozatba, ahol az egyik sorozatban adott tulajdonságú elemek, míg a másikban az adott tulajdonsággal nem rendelkező elemek vannak:

Input	:	$n \in \mathbb{N}_0$, $x \in H^n$, $T : H \to \mathbb{L}$, $\chi : H \to \{0,1\}$
		$\chi(x) = 1$, ha $T(x)$ és $\chi(x) = 0$, ha $\neg T(x)$
Output	:	$DBY, DBZ \in \mathbb{N}_0$, $y, z \in H^n$
Q	:	–
R	:	$DBY = \sum_{i=1}^{n} \chi(x_i) \wedge y \subset x \wedge \forall i\ (1 \leq i \leq DBY) : T(y_i) \wedge$
		$\wedge\ DBZ = n - DBY \wedge z \subset x \wedge \forall i\ (1 \leq i \leq DBZ) : \neg T(z_i)$

Az algoritmus:

Függvény:

T : *elemtípus* $\to \mathbb{L}$

Változók:

n : egész {a feldolgozandó sorozat elemszáma}
x : tömb(1..n:elemtipus) {a feldolgozandó sorozat elemei}
DBY, DBZ : egész {a megfelelő sorozatok elemszámai}

y, z : tömb(1..n:elemtípus) {a megfelelő sorozatok elemei}

A megoldás:

szétválogatás(n, x, DBY, y, z, DBZ)
  $DBY, DBZ := 0, 0$
  for $i = 1 : n$
    if $T(x(i))$ **then**
      $DBY := DBY + 1$
      $Y(DBY) = x(i)$
    else
      $DBZ := DBZ + 1$
      $Z(DBZ) = x(i)$
    end
  end
eljárás vége

További változatok és idevágó tételek az irodalomban találhatók (pl. Wirth)

5.3 Rendezések

Az alapfeladat egy n elemszámú sorozat nagyság szerinti sorbarendezése. Ez feltételezi, hogy a sorozat elemeire létezik a $\leq, <$ reláció. Számos megoldó algoritmus létezik és ezekkel külön terület (tantárgy) foglalkozik.

Input	:	$n \in \mathbb{N}_0$, $x \in H^n$
Output	:	$x \in H^n$
Q	:	–
R	:	x_{ki} rendezett és $x_{ki} = permutáció(x_{be})$

Az algoritmus:

Változók:

n : egész {a feldolgozandó sorozat elemszáma}

x : tömb(1..n:elemtipus) {a feldolgozandó sorozat elemei}

5.3.1 Egyszerű cserés rendezés:

Alapötlete: Hasonlítsuk össze az első elemet a sorozat összes többi mögötte levő elemével, s ha valamelyik kisebb nála, akkor cseréljük meg őket. Ezzel elérhetjük, hogy a sorozat első helyére a legkisebb elem kerül. Folytassuk ugyanígy a sorozat második elemével, stb., utoljára pedig az utolsó előttivel.

rendezés(n, x)
  for $i = 1 : n - 1$
    for $j = i + 1 : n$
      if $x(i) > x(j)$ **then** $csere(x(i), x(j))$
    end
  end
eljárás vége

Az eljárás helyigénye: n+1. Az összehasonlítások száma: $n(n-1)/2$. A cserék száma 0 és $n(n-1)/2$ (Miért?).

5.3.2 Minimumkiválasztásos rendezés

Az előző rendezési eljárásban sok a felesleges csere. Ehelyett az aktuális első elemet a mögötte lévők közül egyedül a legkisebbel cseréljük ki. Ehhez a rendező ciklus belsejében egy minimumkeresést kell csinálni.

```
rendezés(n, x)
  for i = 1 : n − 1
    MIN := i
    for j = i + 1 : n
      if x(MIN) > x(j) then MIN := j
    end
  csere(x(i), x(MIN))
  end
eljárás vége
```

Az eljárás helyigénye: $n+1$. Az összehasonlítások száma: $n(n-1)/2$. A cserék száma: $n-1$ (Miért?).

5.3.3 A gyorsrendezés

A gyorsrendező (quiscksort) eljárást C.A.R. Hoare alkotta 1962-ben.
Alapötlete:
1. Felosztás: Az $\{A(p), \ldots, A(r)\}$ tömb $(p \leq r)$ felosztása két "összefüggő" (esetleg üres)

$$\{A(p), \ldots, A(q-1)\}, \quad \{A(q+1), \ldots, A(r)\}$$

résztömbre úgy, hogy
a baloldali résztömb elemei kisebb, vagy kisebb-egyenlők $A(q)$-nál,
a jobboldali résztömb elemei pedig nagyobb vagy egyenlők $A(q)$-nál.
A q index számítása a felosztó eljárás része.
2. Uralkodás: Az $\{A(p), \ldots, A(q-1)\}$ és $\{A(q+1), \ldots, A(r)\}$ résztömböket a felosztás (gyorsrendezés) rekurzív hívásával rendezzük.
3. Összevonás: Mivel a két résztömböt helyben rendeztük, az egész $\{A(p), \ldots, A(r)\}$ tömb rendezett.
Az algoritmus leírása:

```
gyorsrendezés(A, p, r)
  if p < r
    q = felosztA, p, r)
    gyorsrendezés(A, p, q − 1)
    gyorsrendezés(A, q + 1, r)
  end
eljárás vége
```

Teljes tömb rendezése: **gyorsrendezés**$(A, 1, hossz(A))$.

A tömb felosztása:

A **feloszt** függvényeljárás *helyben* átrendezi az $\{A(p), \ldots, A(r)\}$ résztömböt.

```
feloszt (A, p, r)
  x = A(r)
  i = p − 1
  for j = p : r − 1
    if A(j) ≤ x
      i = i + 1
      csere (A(i), A(j))
    end
  end
  csere (A(i + 1), A(r))
  feloszt = i + 1
eljárás vége
```

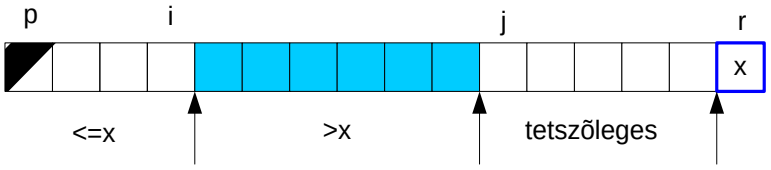
Az $x = A(r)$ elemet őrszemnek nevezzük. A **feloszt** eljárás a tömböt folyamatosan négy (esetleg üres) részhalmazra bontja.

A **for** ciklus minden iterációjának kezdetén ezek a részhalmazok (tömbök) bizonyos ciklus invariánsoknak tekinthető tulajdonságokkal rendelkeznek.

A ciklus minden iterációjának kezdetén a k tömbindexre teljesül, hogy:

1. $A(k) \leq x$,ha $p \leq k \leq i$,
2. $A(k) > x$, ha $i + 1 \leq k \leq j - 1$,
3. $A(k) = x$, ha $k = r$.

Az alábbi ábra szemlélteti ezt a szituációt.



A feloszt eljárás 4 tartománya

A j és $r - 1$ közötti indexekre a fenti feltételek nem állnak, ugyanis az itteni elemek nincsenek kapcsolatban az őrszemmel.

Az algoritmus helyességéhez vizsgáljuk meg a ciklusinvariánst! Ez kell az algoritmus helyességének bizonyításához.

1. A ciklusinvariáns teljesül az első iteráció előtt:

Ekkor $i = p - 1$ és $j = p$. Mivel nincs egyetlen elem sem p és i, sem pedig $i + 1 = p$ és $j - 1 = p - 1$ között, a ciklusinvariáns első két feltétele igaz. Az $x = A(r)$ értékadás miatt a 3. feltétel is igaz.

2. A ciklusinvariáns megmarad az iterációk alatt:

Két eset van:

a) Ha $A(j) > x$, akkor csak j értékét kell növelni 1-el ($j = j + 1$). Ekkor a 2. feltétel $A(j - 1)$-re igaz, míg a többi elem változatlan marad.

b) Ha $A(j) \leq x$, akkor i megnő ($i = i + 1$), $A(i)$ és $A(j)$ felcserélődik, majd j megnő ($j = j + 1$). A csere miatt $A(i) \leq x$ és az 1. feltétel teljesül. Az új $A(j - 1) > x$, mert az $A(j - 1)$ be került elem a ciklusinvariáns miatt nagyobb, mint x.

3. A ciklusinvariáns megmarad a befejezés után:

Befejezéskor $j = r$, ezért a tömb minden eleme a ciklusinvariánsban szereplő $\{A(i) \mid A(i) \leq x\}$, $\{A(i) \mid A(i) > x\}$ és $\{x\}$ halmaz valamelyikében van.

A **feloszt** eljárás utolsó két sora a helyére teszi az őrszemet úgy, hogy felcseréli a baloldali első, x-nél nagyobb elemmel és megadja az eljárás specifikációjában szereplő q indexet.

A **felosztó** eljárás $r - p + 1$ összehasonlítást igényel.

Ha a **gyorsrendezés** eljárást nem tudjuk rekurzívan hívni, akkor a következő módon kell eljárni:

Addig daraboljuk a tömböt a fenti felosztó eljárással, amíg az összefüggő résztömbök hossza 2 alá nem csökken. Mivel egyszerre csak egy résztömböt vizsgálhatunk, a többi résztömböt valamilyen módon nyilván kell tartani. Ehhez két verem szerkezet kell, mondjuk *verembal, veremjobb*,amelyek a részsorozatok bal és jobboldali indexhatárait tartalmazzák. Egy sorozat akkor kerül feldolgozásra, ha a bal és jobboldali indexhatárát eltávolítjuk a verem tetejéről. A felosztás után a legalább két tagot tartalmazó részsorozat bal és jobboldali indexhatárát a verembe visszahelyezzük.

Feladat: Írjunk Pascal programot erre a változatra! Mekkorára válasszuk a verem méretét, ha meg kell adni előre?

A gyorsrendezés eljárás legrosszabb esetben $n(n+1)/2$ összehasonlítást igényel. Ez akkor következik be, ha a sorozat már rendezett (Hogyan?). Az átlagos összehasonlítás szám azonban

$$\approx 1.4n \log n,$$

ami majdnem eléri az elvi, optimális értéket.

A felosztó eljárásnak számos változata van (lásd Irodalom). Az eredeti Hoare-féle felosztó eljárás a következő:

```
Hfelosztt (A, p, r)
  x = A(p)
  i = p − 1
  j = r + 1
  while i < j
    while A(j) ≤ x
      j = j − 1
    end
    while A(i) ≥ x
      i = i + 1
    end
    if i < j
      csere (A(i), A(j))
    else
      Hfelosztt = j
    end
  end
eljárás vége
```

Feladat: Elemezzük az algoritmust és irjunk programot rá!

5.4 Keresések

Fontossága miatt szintén külön terület. Két esettel foglalkozunk.

5.4.1 Lineáris keresés rendezett sorozatban

Adott egy rendezett sorozat, amelyben egy adott értékű elem sorszámát kell meghatározni, ha az benne van a sorozatban:

Input	:	$n \in \mathbb{N}$, $x \in H^n$, $y \in H$
Output	:	$VAN \in \mathbb{L}$, $SORSZ \in \mathbb{N}_0$
Q	:	$rendezett(x)$
R	:	$VAN \equiv (\exists i \ \ (1 \le i \le n) : x_i = y) \wedge$
		$\wedge VAN \Longrightarrow (1 \le SORSZ \le n) \wedge x_{SORSZ} = y$

Az algoritmus:

Változók:

n : egész	{a feldolgozandó sorozat elemszáma}
x : tömb(1..n:elemtipus)	{a feldolgozandó sorozat elemei}
y : elemtipus	{ a keresett elem}
VAN : logikai	{az eredmény - van-e megfelelő elem}
$SORSZ$: egész	{az eredmény -a megfelelő elem sorszáma}

A megoldó algoritmus:

keresés$(n, x, y, VAN, SORSZ)$
  $i := 1$
  while $(i \leq n) \wedge (x(i) < y)$
    $i := i + 1$
  end
  $VAN := (i \leq n) \wedge (x(i) = y)$
  if VAN **then** $SORSZ := i$
eljárás vége

Az algoritmus az y érték első előfordulását választja ki. Egy keresés minimum 1, maximum n, átlagosan pedig $(n+1)/2$ összehasonlítással jár.

5.4.2 Logaritmikus keresés rendezett sorozatban

A sorozat rendezettségét a következőképpen használhatjuk ki. Vizsgáljuk meg első lépésben a sorozat középső elemét. Ha ez a keresett elem, akkor készen vagyunk. Ha a keresett elem ennél kisebb, akkor csak az ezt megelőzőek között lehet, tehát a keresést a továbbiakban a sorozat első felére kell alkalmazni. Ha a keresett elem ennél nagyobb, akkor ugyanezen elv miatt a keresést a sorozat második felére kell alkalmazni.

keresés$(n, x, y, VAN, SORSZ)$
  $e, u := 1, n$
  do
    $k := entier((e+u)/2)$
    if $y < x(k)$ **then** $u := k - 1$
    if $y > x(k)$ **then** $e := k + 1$
  until $(e \leq u) \wedge (x(k) \neq y)$
  $VAN := (e \leq u)$
  if VAN **then** $SORSZ := k$
eljárás vége

Az algoritmus az y érték valamelyik előfordulását választja ki. Egy keresés minimum 1, maximum $\log_2 n + 1$, átlagosan pedig $\log_2 n$ összehasonlítással jár. Innen az eljárás elnevezése.

5.5 Programozási tételek egymásra építése

Ha programozási tételeket egymásután kell alkalmaznunk, akkor gyakran előnyös a két megoldó algoritmus egybeépítése. Néhány esetet tárgyalunk.

5.5.1 Másolással összeépítés

A másolással bármelyik programozási tétel egybeépíthető: A programozási tételben szereplő x_i hivatkozást kicseréljük $g(x_i)$-re, ahol g a másolásnál szereplő függvényt jelöli.

1. Példa: számsorozat elemeinek négyzetösszege=másolás (négyzetre emelés)+ sorozatszámítás (összegzés)

Az általános feladat a következő:

Input	:	$n \in \mathbb{N}_0$, $x \in H^n$, $F : G^* \to G$, $g : H \to G$
Output	:	$S \in G$
Q	:	$\exists F_0 \in G$ és $\exists f : G \times H \to G$ és
		$F(y_1, \ldots, y_n) = f(F(y_1, \ldots, y_{n-1}), y_n)$ és $F() = F_0$
R	:	$S = F(g(x_1), \ldots, g(x_n))$

Az algoritmus:

Függvény:

g : H-elemtípus→ G-elemtípus

Változók:

n : egész {a feldolgozandó sorozat elemszáma}

x : tömb(1..n:$H-$elemtipus) {a feldolgozandó elemek}

F_0 : G-elemtípus {a művelet nulleleme}

S : G-elemtípus {az eredmény}

A megoldás:

Másolás_sorozatszámítás(n, x, s)
  $s := F_0$
  for $i = 1 : n$
    $s := f(s, g(x(i)))$
  end
eljárás vége

2. Példa: Másolás és maximumkiválasztás egybeépítése a maximális elem értékének és indexének meghatározásával.

A feladat leírása:

Input	:	$n \in \mathbb{N}_0$, $x \in H^n$, H rendezett halmaz ($\exists <, \leq$ reláció), $g : H \to G$
Output	:	$MAX \in \mathbb{N}$
Q	:	$n \geq 1$
R	:	$1 \leq MAX \leq n \wedge \forall i\ (1 \leq i \leq n) : g(x_{MAX}) \geq g(x_i)$

Az algoritmus:

Függvény:

g : H-elemtípus→ G-elemtípus

Változók:

n : egész {a feldolgozandó sorozat elemszáma}

x : tömb(1..n:H-elemtipus) {a feldolgozandó sorozat elemei}

MAX : egész {a maximális értékű elem sorszáma}

$MAXERT$:G-elemtípus {a maximális érték}

Másolás_maximumkiválasztás$(n, x, MAX, MAXERT)$
  $MAX, MAXERT := 1, g(x(1))$
  for $i = 2 : n$
    if $MAXERT < g(x(i))$ **then** $MAX, MAXERT := i, g(x(i))$
  end
eljárás vége

5.6 Példa programhelyesség igazolására

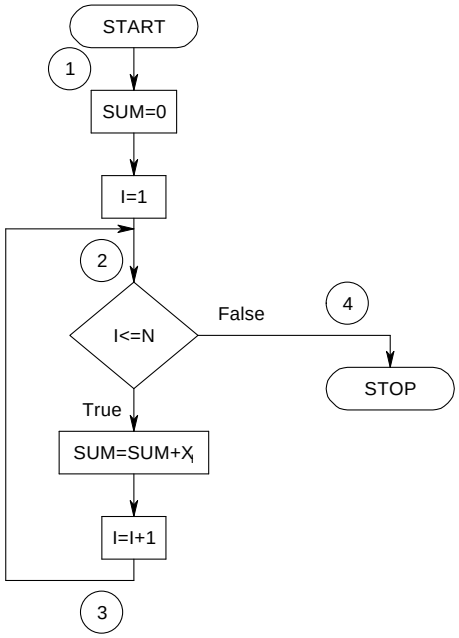
Egy un. blokkdiagram program helyességét igazoljuk teljes indukcióval Anderson könyve alapján.

Feladat:

$$\sum_{j=1}^{n} x_j$$

meghatározása. A feladat a sorozatszámítás programozási tétel speciális esete.

Tekintsük az alábbi blokkdiagram sémát:



Az 1. pontban, kiinduláskor, $X_1, X_2, \ldots, X_N$ N-dimenziós valós tömb.

A 2. pontban $1 \leq I \leq N+1$, $SUM = \sum_{j=1}^{I-1} X_j$ $(\sum_{j=1}^{0} X_j = 0)$.

A 4. pontban $SUM = \sum_{j=1}^{N} X_j$.

A séma helyességét teljes indukcióval (kettős indukció) igazoljuk.

Jelölje n azt, hogy a 2. pontot hányadszorra értük el $(0 \leq n \leq N+1)$. Legyen ekkor I_n az I változó értéke, a SUM_n pedig a SUM változó értéke.

1. A 2. pont első előfordulásakor $n = 1$, $I_1 = 1$, $SUM_1 = 0$.

Minthogy $1 \leq I_1 < N+1$, $SUM = SUM_1 = \sum_{j=1}^{I_1-1} X_j = \sum_{j=1}^{0} X_j = 0$, az első előforduláskor kapott eredmény helyes.

2. Tegyük fel, hogy a 2. pontnál vagyunk, $1 \leq I_n \leq N+1$, $I_n = n$, $SUM = SUM_n = \sum_{j=1}^{I_n-1} X_j$.

Ha $I_n \leq N$, akkor folytatjuk a ciklust, $I_{n+1} = I_n + 1$ és

$$SUM_{n+1} = SUM_n + X_{I_n} = (X_1 + \ldots + X_{I_n-1}) + X_{I_n} = \sum_{j=1}^{I_n} X_j.$$

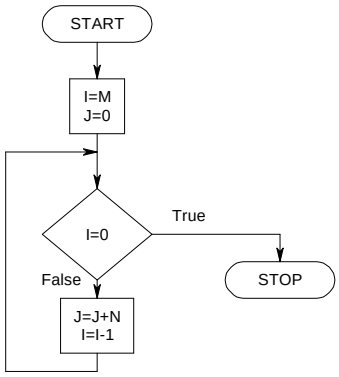
Továbbá $2 \leq I_{n+1} = I_n + 1 = n + 1 \leq N + 1$. Tehát $I = n + 1$ és

$$SUM = SUM_{n+1} = \sum_{j=1}^{n} X_j = \sum_{j=1}^{I} X_j.$$

Ha $I_n = N + 1$, akkor kilépünk a ciklusból és ekkor $I = N + 1$, $SUM = \sum_{j=1}^{N} X_j$. Tehát a blokkséma program helyességét igazoltuk.

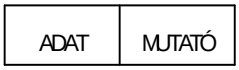
Feladat:

Mit csinál az alábbi program? Igazoljuk a helyességét!



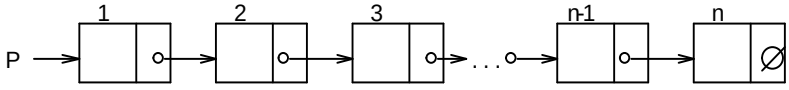
6 Listák

A láncolt lista egy olyan dinamikus adatszerkezet, amelyben az objektumok lineáris sorrendben követik egymást. A lista méretét felülről csak a rendelkezésre álló tároló helyek korlátozzák. Listák felhasználása számos helyen célszerű. Legegyszerűbb esetben egy listaelem két részből áll:



Az adat rész tartalmazza a tényleges információt, a mutató rész pedig hivatkozást a következő listaelemre (a következő listaelem címét).

A létrehozható listaszerkezet elnevezése: Egyszerűen kapcsolt, vagy lineáris lista.

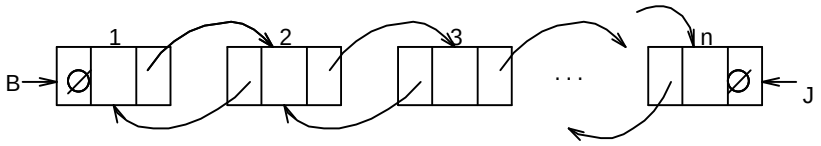


A P változó (mutató) az első listaelem címét tartalmazza. A $\emptyset$, vagy NIL jel a lista végét jelzi.

A szimmetrikus lista elemek az adatmezőn kívül két mutatót tartalmaznak:

a BM és a JM mutatót. A BM mutató az előző (baloldali) szomszéd címét, a JM mutató pedig a következő, jobboldali szomszéd címét tartalmazza.

A szimmetrikus listaelemekből álló, szimmetrikus lista általános alakja:



Itt a B mutató az első listaelem, a J mutató pedig az utolsó listaelem címét tartalmazza.

Tömbök segítségével könnyen hozhatunk létre lineáris listákat. Kell két tömb:

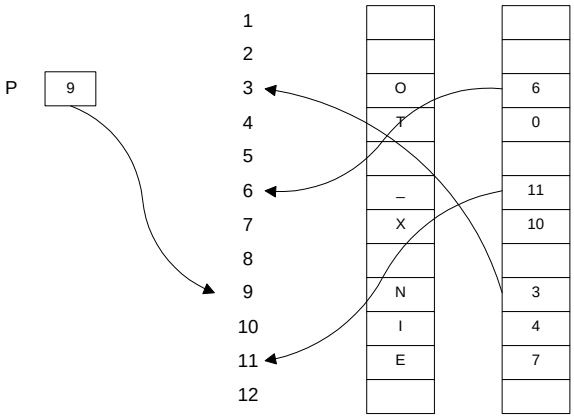
$$adat\,(N)\,, \quad mutató\,(N)\,,$$

ahol N a tárolandó adatok száma. Az

$$\{adat\,(i)\,, mutató\,(i)\}$$

páros képez egy listaelemet. A $mutató\,(i)$ tartalma a következő listaelem indexe. A lista végén $\emptyset = NIL = 0$.

Példa: Tekintsük az alábbi két tömböt



A példán látható, hogy a két tömb hézagosan van kitöltve.

Legalább két listát szokás létrehozni. A tényleges listán túlmenően létrehozzuk a rendelkezésre álló szabad tárolóhelyek listáját is. Új adat listába való elhelyezésekor a szabad tárolóhelyek listájából elveszünk, adat törlésekor pedig a szabad tárolóhelyek listájához hozzáteszünk.

A lineáris lista szekvenciális feldolgozása:

Legyen T a lista elem indexe, $\left\{ADAT\,(T)\,, MUTATÓ\,(T)\right\}$ pedig a T-ik listaelem.

$T \leftarrow P$ {Az első listaelem címe}

while $T \neq \emptyset$

  $W \leftarrow ADAT\,(T)$

  W feldolgozása

  $T \leftarrow MUTATÓ\,(T)$

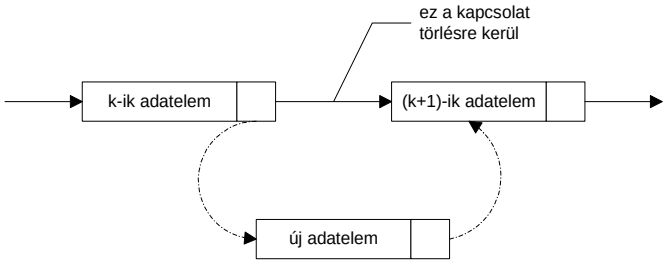
end

Megjegyzés: Ha nincs feldolgozás, akkor csak bejárásról beszélünk.

6.1 Műveletek listákban

1. Új adatelem beszúrása listába:

A művelet sematikus képe a következő: A k-ik és a $(k+1)$-ik adatelem közé teszünk be új adatelemet.



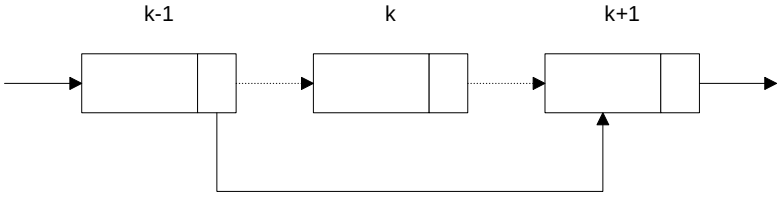
A művelet megvalósítása (programrészlet):

$MUTATÓ\,(T) \leftarrow MUTATÓ\,(k)$

$MUTATÓ\,(k) \leftarrow T$

2. Lista elem törlése:

A művelet sematikus képe a következő (a k-ik elemet töröljük):



A művelet lehetséges megvalósításai (programrészletek):

1. Változat (A $(k+1)$-ik elem előre "csúszik" a k-ik helyére):

$T \leftarrow MUTATÓ\,(k)$

if $T = \emptyset$ then hibajelzés és exit

$ADAT\,(k) \leftarrow ADAT\,(T)$

$MUTATÓ\,(k) \leftarrow MUTATÓ\,(T)$

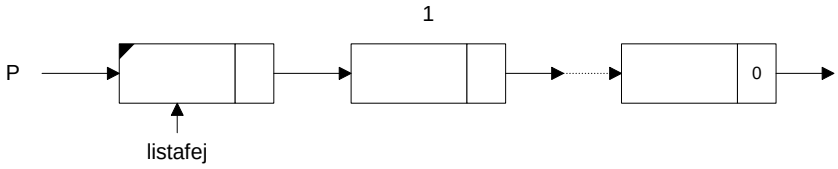
2. Változat (minden elem a helyén marad):

$MUTATÓ\,(k-1) \leftarrow MUTATÓ\,(k)$

Számos probléma merül fel az üres listák és a listák utolsó elemeinek kezelésével. A lehetséges megoldások a következők.

Listafej

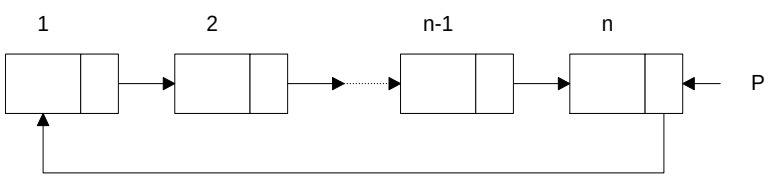
Az üres listákkal való problémák elkerülésére vezetjük be a listafejet:



Így a lista hossza soha nem lesz zérus. A listafej tartalmazhat lista információkat (pl. az elemek aktuális számát).

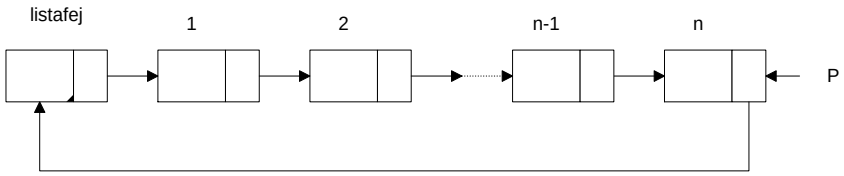
Ciklikus lista

A ciklikus, vagy cirkuláris lista az utolsó elemekkel való bajlódást kívánja csökkenteni.



A lista végén egy mutatót helyezünk el, amely az első elemre mutat. Itt a listafej mutatója P a jobb oldalon van.

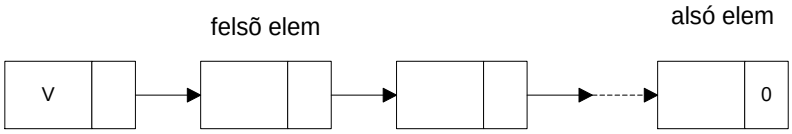
A problémát ennél a megoldásnál az jelenti, hogy nem tudjuk a lista végét. Ennek a megoldása a ciklikus lista listafejjel:



A lista szekvenciális feldolgozásának végét a listafej elérése jelzi.

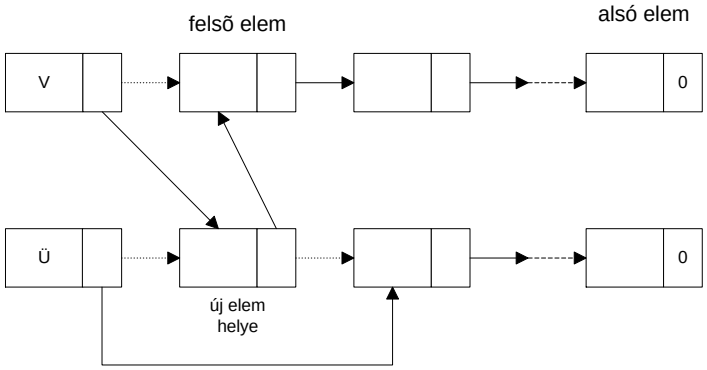
Megjegyezzük, hogy a szimmetrikus listákat is el szokás listafejjel látni.

Verem ábrázolása listával:



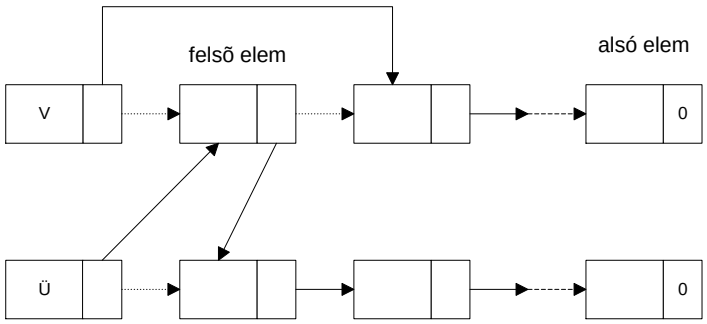
A következő lista műveleteknél feltételezzük, hogy rendelkezésünkre áll a szabad helyek listája is, amelynek fejét az *Ü* változó jelzi.

Új elem behelyezése a verembe:



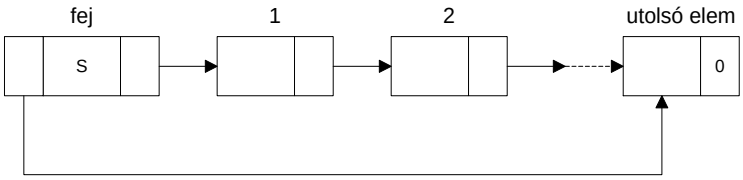
Az új elemet az üres helyek listájából elvett helyre tesszük és a mutatókat átirányítjuk mindkét listában.

Felső elem törlése veremből:

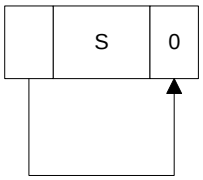


A verem felső elemét úgy töröljük, hogy a verem listafej mutatóját a következő elemre irányítjuk, a törölt felső elemet pedig az üres helyek listájának tetejére tesszük.

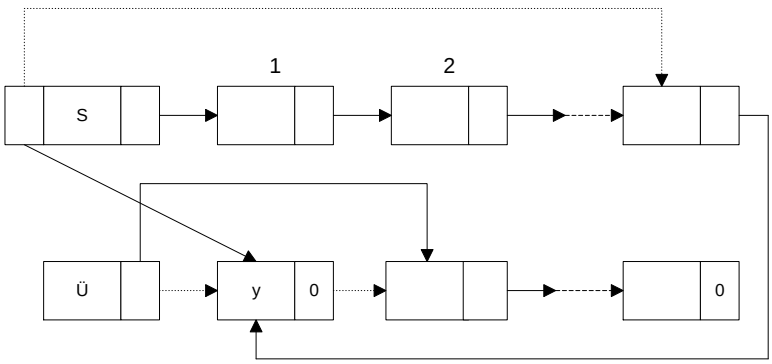
Sor ábrázolása lineáris listával



A listafej két mutatót tartalmaz. A bal mutató a sor utolsó elemére (a lista végére) mutat. Üres sor esetén a bal mutató sajátmagára (a listafejre) mutat:



A sorból történő olvasás, törlés ugyanaz mint a verem esetében.
Beírás sorba utolsó elem után:



Az új y adatelem helyét elvesszük az üres helyek listájáról és mindkét listán a megfelelő mutatókat átállítjuk. Vegyük észre, hogy az új elemhez kerül a NIL jelzés.

3. Keresés listában

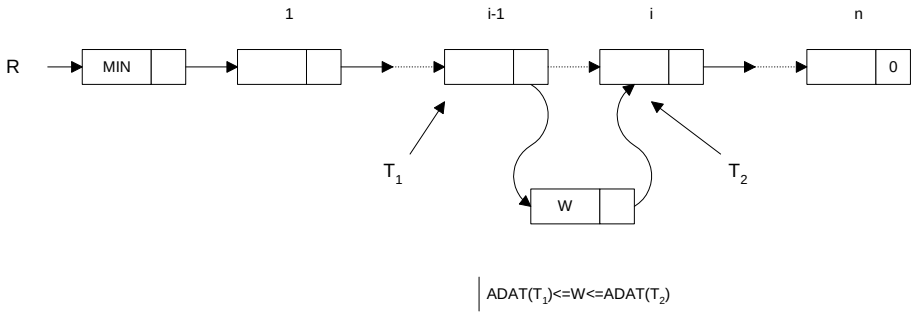
Két változatot nézünk meg.

1. Keresés rendezetlen listában:

Legyen a keresett adat (tétel) X. Az X-et tartalmazó listaelem címe -LOC- kell.

A megoldást a "lineáris lista szekvenciális feldolgozása" algoritmus adja:

$T \leftarrow P$
while $T \neq \emptyset$
    if $ADAT\,(T) = X$
        $LOC \leftarrow MUTATÓ\,(T)$
        exit
    endif
$T \leftarrow MUTATÓ\,(T)$
endwhile
Legrosszabb esetben n összehasonlítás kell. Általános esetben $n/2$ a várható összehasonlítás szám.
2. Keresés (növekvően) rendezett listában:
Itt nem alkalmazhatjuk a logaritmikus keresést, mert nem tudjuk indexelni a középső elemet.
$T \leftarrow P$
while $T \neq \emptyset$
    if $ADAT\,(T) < X$ **then**
        $T \leftarrow MUTATÓ\,(T)$
    else if $ADAT\,(T) = X$ **then**
        $LOC = MUTATÓ\,(T)$
        exit
    else
        $LOC = \emptyset$
        exit
    endif
endwhile
$LOC = \emptyset$
4. Adatok rendezése lineáris listában
A feladat: $ADAT\,(1) \leq ADAT\,(2) \leq \ldots \leq ADAT\,(N)$
A megoldás alapötletei:
1. Az adatokat úgy szúrjuk be egy listába, hogy a rendezettség megmaradjon.
2. Két mutatót használunk.
A rendezett lista feje legyen R. A beszúrás sémáját mutatja az alábbi ábra:



A MIN egy olyan adat, amelyik mindegyik rendezendő adatnál kisebb. Ezért mindig a lista első eleme marad. A W adatot a következő program részlettel helyezzük el a listában a T_1 és T_2 elemek között:
$T_1 \leftarrow R$
$T_2 \leftarrow MUTATÓ\,(R)$
while $T_2 \neq \emptyset$ and $W \nleq ADAT\,(T_2)$
    do
        $T_1 \leftarrow T_2$
        $T_2 \leftarrow MUTATÓ\,(T_2)$
    enddo
endwhile

Az általános rendezési algoritmus ezekután a következő (feltételezve, hogy az adatokat egy fájlból olvassuk be):

1. $\ddot{U}$ (üres helyek) lista létrehozása
2. Az R (rendező) lista (fejének) levágása az $\ddot{U}$ üres helyek listájáról.
3. $ADAT\,(R) \leftarrow MIN$, $MUTATÓ\,(R) \leftarrow \emptyset$
4. W beolvasása:

while $W \neq eof$
    W elhelyezése a listában
end

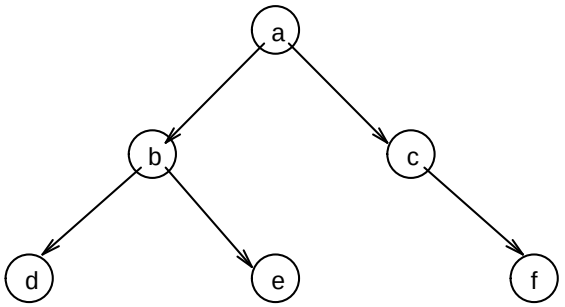
6.2 Megjegyzések

1. A listáknak más változatai is léteznek.

2. A listaelem általánosítható (multilista), amely segítségével a hierarchikus és hálós adatszerkezeteket tudjuk megvalósítani.

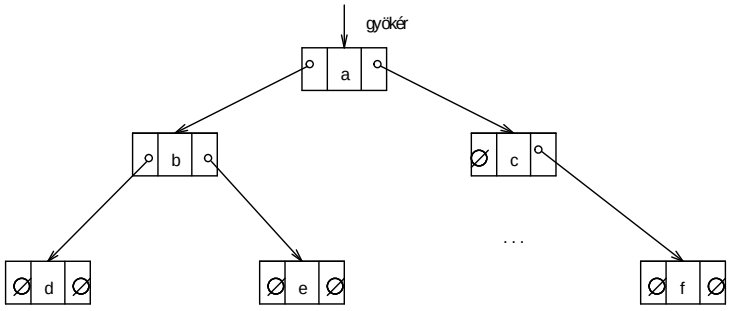
3. Speciális hierarchikus szerkezetben, a bináris fa esetén, amikor egy csomópontból legfeljebb két él indul (vagy két részfa tartozik hozzá), akkor kétszeresen láncolt listával könnyen ábrázolhatjuk, ahol a balmutató a baloldali részfa gyökerére, a jobbmutató a jobboldali részfa gyökerére mutat.

Példa: Tekintsük a következő bináris fát:



bináris fa

A megfelelő lista ábrázolás:



a bináris fa listás ábrázolása

7 Irodalom

[1] A.V. Aho-J.E. Hopcroft-J.D. Ullman: *Számítógépalgoritmusok tervezése és analízise*, Műszaki Könyvkiadó, 1982
[2] S. Alagić, M.A. Arbib: *The Design of Well-Structured and Correct Programs*, Springer, 1978
[3] R. B. Anderson: *Proving Programs Correct*, Wiley, New York, 1979
[4] Andrásfai Béla: *Gráfelmélet*, Polygon, Szeged, 1997
[5] Bárdos Attila: *A programbizonyítás alapjai*, SZÁMOK, Budapest, 1979
[6] J.L. Bentley: A *programozás gyöngyszemei*, Műszaki Könyvkiadó, 1988
[7] T.H. Cormen-C.E. Leiserson-R.L. Rivest: *Algoritmusok*, Műszaki Könyvkiadó, 1998
[8] T.H. Cormen-C.E. Leiserson-R.L. Rivest-C. Stein: *Új algoritmusok*, Scolar, 2003
[9] O.J. Dahl, E.W. Dijkstra-C.A.R. Hoare: *Strukturált programozás*, Műszaki Könyvkiadó, 1978
[10] E.W. Dijkstra: *A Discipline of Programming*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1976
[11] Fóthi Ákos: *Bevezetés a programozáshoz*, ELTE jegyzet, Tankönyvkiadó, Budapest, 1984
[12] Fóthi Ákos, Steingart Ferenc: *Programozási módszertan*, kézirat, ELTE, 1999
[13] D. Gries: *The Science of Programming*, Springer, 1981
[14] Hajnal Péter: *Gráfelmélet*, Polygon, Szeged, 1997
[15] Dr. Hetényi Pálné (szerk): *Programozási feladatok 2000-ig I-II.*, OMIKK
[16] Iványi A. (alkotó szerkesztő): *Informatikai algoritmusok I.*, ELTE Eötvös Kiadó, 2004
[17] Iványi A. (alkotó szerkesztő): *Informatikai algoritmusok II.*, ELTE Eötvös Kiadó, 2005
[18] B.W. Kernighan-B. Plauger: *A programozás magasiskolája*, Műszaki Könyvkiadó
[19] Kozma L.-Varga L.: *A szoftvertechnológia elméleti kérdései*, ELTE Eötvös Kiadó, 2003
[20] D. Knuth: *A számítógép programozás művészete I-III*, Műszaki Könyvkiadó, 1988
[21] S. Lipschutz: *Data Structures*, MacGraw-Hill, 1986 (van magyar fordítás is).
[22] Z. Manna: *Lectures on the Logic of Computer Programming*, SIAM, Philadelphia, 1980
[23] Z. Manna: *Programozáselmélet*, Műszaki Könyvkiadó, 1981
[24] J.G. Sanders: *A Relational Theory of Computing*, Lecture Notes in Computer Science, Vol. 82, Springer, 1980
[25] Szlávi Péter, Zsakó László: *Módszeres programozás: Programozási tételek, μlógia* 19, ELTE TTK, 1999
[26] Varga László: *Programok analízise és szintézise*, Akadémiai Kiadó, Budapest, 1981
[27] N. Wirth: *Algoritmusok+adatstruktúrák=programok*, Műszaki Könyvkiadó, 1982