SZLÁVI PÉTER

PROGRAMOZÁSI TÉTELEK EGYMÁSRA ÉPÍTÉSE
*
PROGRAMTRANSZFORMÁCIÓK

2000

TARTALOM

A CÉL, ALAPVETŐ DEFINÍCIÓK

Program- és kódtranszformáció

A tételek ismeretében már a feladatok zöme mechanikusan megoldható a specifikáció alapján[1], a *struktúraszerinti feldolgozás elve* segítségével és a *programozási tételek* alkalmazásával. Az egészen triviális feladatokat kivéve, a megoldás több tétel ötvözete lesz. A mechanikus tételalkalmazás –természetesen– nem adhat általában optimális megoldást. Keletkezhetnek józanésszel is javítható „algoritmikus redundanciák" (bántó kódismétlések, nyilvánvalóan elkerülhető segédváltozók stb.), amelyeket részek összevonásával, átstrukturálással korrigálni lehet.

Annak érdekében, hogy ez az optimalizáló lépés megtartsa a tételek nyújtotta „garantált" helyességet, de elérje a kívánt hatékonysági célt is, kidolgozunk néhány –az esetek többségében alkalmazható– átalakítási szabályt, ún. ***programtranszformáció***t.

A programok transzformálásának lehet egy másik célja is. Amikor egy adott programozási nyelvre kódolásnál tartunk, sokszor az algoritmus „nagystílűsége" miatt kényszerülünk bizonyos transzformációra. Itt tehát egy programozási nyelv „szószátyársága" erőszakolja ki a változtatást, nem az algoritmusban eleve meglévő lehetőségek. A programnak e két céllal történő átalakítását megkülönböztetjük egymástól, az utóbbira a ***kódtranszformáció*** kifejezést fogjuk alkalmazni. Most csak az előbbi célú átalakítással foglalkozunk.

Függvény és implementációja – a függvényeljárás kiszámítása

A könnyebb megfogalmazás miatt bevezetünk néhány fogalmat, ill. egy-két értelmező megjegyzést bocsátunk előre.

Az alábbiakban a függvény alatt mindig valamely f (*kis*betűkkel jelölt) *„absztrakt" függvény*t kiszámító F (*nagy*betűkkel jelölt) *függvényeljárás*t értünk, amely kiszámítási sémájára vonatkozólag feltételezéseket teszünk. Maguk az „absztrakt" függvények *egyértékű*ek, és a hozzájuk tartozó függvényeljárások *explicite* tartalmazzák azokat a paramétereket, amelyektől függnek, vagyis nincsenek „latens" paraméterei (pl. valamely globálisan deklarált adat).

Elvárásunk látszólag magától értetődő: y←f(x) „absztrakt" utasításnak minden körülmények között megfeleljen az algoritmusbeli `y:=F(x).`

`f: X→Y` ⇒

```
Függvény F([2]x:TX):TY
  Változó y:TY
  y:=f(x)     [f(x) kiértékelése, amely
  …            közben x is módosulhat]
  F:=y
Függvény vége.
```

---

[1] … amint olvashatjuk a „Programozási tételek szerepe a gyakorlati programozásban" c. anyagban…

[2] Szándékosan nincs jelölve a paraméter *hozzáférési jog*a, ugyanis megengedjük –az egyébként nem túl „tisztességes"– a paraméter megváltozását.

A lényeg: először az f(x) kiértékelése történik meg, majd a legvégén az értékvisszaadás. Így ha volna *mellékhatás*[3], a függvény értéke a ténylegesen kiszámított érték lesz minden zavaró körülmény ellenére. Azaz nem *interferál* a függvényérték és a paraméter.

Ezzel az értelmezéssel kikerülhető az a bizonytalanság, ami egyébként lenne az alábbi utasításnál (ha a mellékhatás megengedett): `x:=F(x)` Milyen értékkel rendelkezik az értékadás után az x?

Például:

```
x:=2
x:=F(x)
{x=?}
```

```
Függvény F(Változó x:Egész):Egész
  Változó y:Egész
  y:=2*x;  x:+1
  F:=y
Függvény vége.
```

Definíció: (*x-invariancia*)

> Egy f: X×Y→Z, f(x,y) leképezést megvalósító `F(x,y)` függvényeljárást ***x-invariáns***nak (első változó szerint invariánsnak) nevezünk, ha ∀(x,y)∈$\mathcal{D}om_f$: x'=x.
>
> Ahol $\mathcal{D}om_f$ az f függvény értelmezési tartományát jelöli, az x' pedig az x (`F(x,.)`) kiszámítás utáni értékét.

Világos, hogy egy x-invariáns f függvény `F`-beli x paraméterének nyugodtan adható *konstans* hozzáférési jog, hiszen *bemeneti* a funkciója.

E fogalom birtokában már megfogalmazhatjuk a két- vagy többváltozós függvények kiszámítására vonatkozó elvárásainkat:

```
f: X×Y→Z,
  ∃g,h,i:X×Y→Z :
  g,h,i x- és y-invariánsok
  ∧ f(x,y)=i(x,y)
  ∧ x'=g(x,y) ∧ y'=h(x,y)
```

⇒

```
Függvény F(Változó x:TX, y:TY):TZ
  Változó  xx:TX, yy:TY, z:TZ
  xx:=G(x,y);  yy:=H(x,y)
  z:=I(x,y)
  x:=xx;  y:=yy
  F:=z
Függvény vége.
```

A lényeg: először az esetlegesen módosuló paraméterek értékeit számítjuk ki, majd az eredeti bemenő értékekkel a „lényegi” számítást, a függvény értékét. (Innen már sejthető, hogy több paraméteres függvények kiszámításával szemben milyen elvárásaink lehetnek.)

Definíció: (*x-függetlenség*)

> Egy f: X×Y→Z, f(x,y) leképezést megvalósító `F(x,y)` függvényeljárást ***x-független***nek (első változójától függetlennek) nevezünk, ha *x-invariáns* és ∀x,xx∈X, ∀y∈Y: (x,y),(xx,y)∈$\mathcal{D}om_f$: f(x,y)=f(xx,y).

Nyilvánvaló, hogy ha egy `F(x,y)` *x-független*, akkor alkalmazható az `F(y)` jelölés. Ezért indokolni kell e fogalom bevezetésének értelmét! Íme: időnként szükség lesz bizonyos segédváltozók bevezetésére, de ezt a többi programrésszel való zavartalan együttműködés garantálásával (*interferenciamentesség* megtartásával) szabad csak végrehajtani. Ebben a szituációban a megfelelő segédváltozó néven nevezéséhez kell fölhasználni.

---

3 azaz valamely paraméter a függvény kiértékelése során megváltozna …

Szimmetrikus és aszimmetrikus logikai műveletek

A későbbiek során fontos lesz az, hogy egy ’`és`’, ill. egy ’`vagy`’ logikai művelet *szimmetrikus*, vagy „értelmesen” *aszimmetrikus* működésű-e, ezért ezeket a leírásban meg fogjuk különböztetni. Gondoljunk csak például az ’Eldöntés tétel’-beli ciklusfeltételére, amit „egészében” ’`i=N+1`’ esetén aggályos lenne végrehajtani, hacsak nem ’aszimmetrikus `és`’-t feltételezünk. Mindazon által az is világos, hogy egy olyan logikai feltételben, amelynek –mondjuk– második tényezője egy *mellékhatás*sal rendelkező[4] függvényt tartalmaz, akkor ennek végrehajtására mindenképpen szükség van. Ezzel indokoljuk, hogy mindkettőre szükség van, de célszerű világos megkülönböztetésükre is.

Továbbiakban a logikai műveleti jeleket az alábbi értelmezéssel fogjuk használni:

Logikai művelet	**Az aszimmetrikus jelölése**	**A szimmetrikus jelölése**
∧	*és*	*és még*
∨	*vagy*	*vagy különben*

Hogy éppen az `és`-t és a `vagy`-ot választottuk aszimmetrikusnak s nem a párjaikat, annak az az oka, hogy visszamenőleg nem akartuk a tételek algoritmusait módosítani.

[4] … vagy „suttyomban” valamely globális adatot megváltoztató …

PROGRAMTRANSZFORMÁCIÓK

Programtranszformáció alatt (a programozási tételhez hasonlóan) egy *állítás*t értünk, amely két program(darab) *azonosság*át állítja egy *feltétel* teljesüléséhez kötve. Két program(darab) akkor azonos, ha minden bemenetre ekvivalens működésűek.

Az alábbiakban fölsoroljuk a legfontosabb programtranszformációkat (PT*x*), amelyekhez közöljük – természetesen–, hogy milyen feltételek (efPT) mellett alkalmazhatók. (A feltételek nem feltétlenül lesznek a „legbővebb bemenetet" megengedők.)

PT1. Függvénykompozíció

efPT: G(y) y-invariáns ∧ P(x,z,y) y-független

```
y:=F(x);   z:=G(y)                ⇔  z:=G(F(x))
w:=P(x,z,y)                          w:=P(x,z,y)
```

Megjegyzés:

Vegyük észre, hogy 1) az első feltétel (`G(y)` *y-invariáns*) „pusztán" formális okok miatt kell, nevezetesen ahhoz, hogy a jobboldali transzformációban az `F` függvény a `G`-be *behelyettesít*hető legyen. 2) a második (`P(x,z,y)` *y-független*) pedig a („folytatólagos") környezettel való *interferenciamentesség*et jelenti; egyszerűen szólva: az y zavarmentes *beilleszthetőség*ét (fordított irányban: *elhagyhatóság*át).

PT2. Értékadások cseréje

efPT: F(x), G(x) x-invariáns

```
y:=F(x);   z:=G(x)                ⇔  z:=G(x);   y:=F(x)
```

Megjegyzés:

Vegyük észre, hogy e szerint az `'A(x,y); B(x,z)'` *eljárások* cseréjét is elvégezhetjük, ha ők *x-invariáns*ok. Ui. ez épp azt jelenti, hogy paraméterei szétválaszthatók *csak bemenet*ire, ill. „másra".

PT3. Párhuzamos értékadás I.

efPT: F(x,y) y-invariáns ∧ G(x,y) x-invariáns ∧ P(x,y,sx,sy) sx-, sy-független

```
x,y:=F(x,y),G(x,y)                ⇔  sx:=x;   sy:=y
w:=P(x,y,sx,sy)                      x:=F(x,y);   y:=G(sx,sy)
                                     w:=P(x,y,sx,sy)
```

Megjegyzés:

A végrehajtás *párhuzamosság*a (a baloldali programdarabban) azt jelenti, hogy az utasítás után teljesül az alábbi állítás:

$$x' = F(x,y) \quad \wedge \quad y' = G(x,y)$$

Figyelem: az `F(x,y)` *y-invarianciá*ja[5] és a `G(x,y)` *x-invarianciá*ja a párhuzamos végrehajtásnak, s nem a programtranszformáció alkalmazhatóságának a feltétele!

---

[5] Emlékeztetőül az *y-invarianciá*ról: egy f: X×Y→Z, f(x,y) leképezést megvalósító `F(x,y)` függvényeljárást ***y-invariáns***nak (második változó szerint invariánsnak) nevezünk, ha $\forall(x,y)\in \mathcal{D}om_f$: y'=y.

PT4. Párhuzamos értékadás II.

efPT: F(x,y) y-invariáns [∧ G(x,y) x-invariáns[6]] ∧ P(x,y,sx) sx-független

```
x,y:=F(x,y),G(x,y)                ⇔  sx:=x
w:=P(x,y,sx)                         x:=F(x,y);  y:=G(sx,y)
                                     w:=P(x,y,sx)
```

PT5. Feltételcsere logikai kifejezésben

efPT: F(x), G(x) x-invariáns

```
l:=F(x) és G(x)                   ⇔  l:=G(x) és F(x)
```

Megjegyzés:

Hasonló programtranszformáció fogalmazható meg az 'és még' tényezőinek fölcserélhetőségére is.

PT6. Az 'és' és az 'és még' ekvivalenciája

efPT: G(x) x-invariáns

```
l:=F(x) és G(x)                   ⇔  l:=F(x) és még G(x)
```

Megjegyzés:

Hasonló programtranszformáció tartozik a 'vagy' és a 'vagy különben' azonosságához is.

Fontos következmény: ha tehát a tényezők mellékhatás-nélküli kifejezések, akkor nincs jelentősége annak, hogy szimmetrikus ('és még'/'vagy különben') vagy aszimmetrikus ('és' /'vagy') kapcsolja őket össze! Sőt azt is látni kell, hogy a logikai kifejezés bárhol (pl. elágazás, vagy ciklus feltételében) szerepelhet. (Lényegében csak a „nyomaték kedvéért" fogalmazzuk meg ezt a PT7-ben, PT8-ban.)

PT7. 'és'-feltételszétbontás az elágazásban I.

efPT: G(x) x-invariáns

```
Ha F(x) és G(x) akkor   z:=H(x)      ⇔  Ha F(x) és még G(x) akkor   z:=H(x)
               különben z:=I(x)                              különben  z:=I(x)
```

Megjegyzés:

A cím arra utal, hogy (a baloldali algoritmusban) az 'és' művelet megvalósítható a feltételek *szétbontás*ával így is:

```
Ha F(x) akkor
  Ha  G(x)  akkor   ...IGAZ-ÁG (igaz-F(X)-G(X))...
            különben ...HAMIS-ÁG (igaz-F(X),hamis-G(X))...
különben ...HAMIS-ÁG (hamis-F(X))...
```

PT8. 'vagy'-feltételszétbontás az elágazásban II.

efPT: G(x) x-invariáns

```
Ha F(x) vagy G(x) akkor   z:=H(x)    ⇔  Ha F(x) vagy különben G(x)
                különben  z:=I(x)                 akkor   z:=H(x)
                                                  különben z:=I(x)
```

[6] Gondolja meg: mi történik, ha G(x,y) **nem** *x-invariáns*?

Megjegyzés:

A cím itt is arra utal, hogy (a baloldali algoritmusban) a 'vagy' művelet megvalósítható a feltételek *szétbontás*ával így is:

```
Ha nem F(x) akkor
  Ha  G(x)  akkor  ...IGAZ-ÁG (hamis-F(X),igaz-G(X))...
            különben ...HAMIS-ÁG (hamis-F(X),hamis-G(X))...
különben ... IGAZ-ÁG(igaz-F(X))...
```

PT9. Feltétel kiemelése elágazásból

efPT: F(x) x-invariáns ∧ G(x,s) s-független ∧ T(x) x-invariáns

```
Ha T(F(x)) akkor y:=F(x)
z:=G(x,s)
```

⇔

```
s:=F(x)
Ha T(s) akkor y:=s
z:=G(x,s)
```

Megjegyzés:

A `G` függvény „egyesíti" magában a program elágazás utáni részének össztranszformációját. A feltételből kiolvasható, hogy az 's' változót semmire sem használja, így bátran választható segédváltozóként. A `G(x,s)` *s-függetlensége* a baloldali algoritmus alkalmazhatóságának a feltétele.

PT10. Ciklusok összevonása

efPT: L(x), M(x), G(x,y), H(x,z) x-invariáns

```
x:=k;  y:=L(x)
Ciklus amíg Cf(x)
  y:=G(x,y);  x:=I(x)
Ciklus vége
x:=k;  z:=M(x)
Ciklus amíg Cf(x)
  z:=H(x,z);  x:=I(x)
Ciklus vége
```

⇔

```
x:=k;  y:=L(x);  z:=M(x)
Ciklus amíg Cf(x)
  y:=G(x,y);  z:=H(x,z);  x:=I(x)
Ciklus vége
```

Megjegyzés:

Mint látható: az `x` tölti be a *ciklusváltozó* szerepét, az `L` és `M` függvények a *kezdőértékek* beállítását végzik, az `I` –közös– függvény pedig a *ciklusok szervezését*, a `G` és `H` függvények „lényegi" (az `y`-ra, ill. a `z`-re vonatkozó) *transzformációk*at.

PT11. Függvény kiemelése a ciklusfeltétel „belsejéből"

efPT: G(x,y,fx) fx-független ∧ T(x) x-invariáns

```
Ciklus amíg  T(F(x))
  y:=Cm(x,y)
Ciklus vége
z:=G(x,y,fx)
```

⇔

```
fx:=F(x)
Ciklus amíg  T(fx)
  y:=Cm(x,y);  fx:=F(x)
Ciklus vége
z:=G(x,y,fx)
```

Megjegyzés:

A `G` függvény a program „utóéletét" szimbolizálja, amely során az `fx` változót nem, viszont akár `x`, akár `y` értékét fölhasználhatja. A `T(x)` *x-invarianciá*ja a baloldali transzformáció formális feltétele.

Vegyük észre, hogy a ciklus vezérlését (azaz az `x` módosítását) két függvény együttese végzi: az `F(x)` és a `Cm(x,y)`.

PT12. Ciklusfeltétel részbeni kiemelése – indextúllépés elkerülésére I.

efPT: e≤v ∧ ∃Előző(e) ∧ [∃Következő(v)] ∧ T(x) [, Következő(x)] x-invariáns

```
x:=e
Ciklus amíg x≤v és nem T(x)
  x:=Következő(x)
Ciklus vége
l:=x≤v
```

⇔

```
x:=Előző(e);  l:=Hamis
Ciklus amíg x<v és nem l
  x:=Következő(x);  l:=T(x)
Ciklus vége
```

Megjegyzés:

A programdarabokban szereplő `x` (ciklust vezérlő) változó típusa *rendezett*, amelyen értelmezve van az `Előző` és a `Következő` függvény (nevükből kihámozható szemantikával)[7].

A `Következő(v)` létezése nem a programtranszformáció feltétele, hanem a baloldali algoritmus működésének.

Ha `e≤v` feltételt elhagyjuk az efPT-ből, akkor (`e>v` esetben) a két algoritmus `l` szempontjából ekvivalens ugyan, de `x` szempontjából nem.

A következő egy ritka transzformáció, de nagyon közel áll az előbbihez, ezért önálló sorszámot nem is kap, csak egy aposztrófot, amivel utalunk arra, hogy az előbbi egy változatáról van szó. A kevés különbséget színnel emeltük ki.

PT12'. Ciklusfeltétel részbeni kiemelése – indextúllépés elkerülésére I'.

efPT: e≤v ∧ ∃Előző(e) ∧ [∃Következő(v)] ∧ T(x,y) [, Következő(x)] x-invariáns ∧ T(x,y) y-invariáns

∧ G(x,y) x,y-invariáns

```
y:=y0;  x:=e
Ciklus amíg x≤v és nem T(x,y)
  y:=G(x,y)
  x:=Következő(x)
Ciklus vége
l:=x≤v
```

⇔

```
y:=y0;  x:=Előző(e);  l:=Hamis
Ciklus amíg x<v és nem l
  x:=Következő(x)
  y:=G(x,y)
  l:=T(x,y)
Ciklus vége
```

PT13. Ciklusfeltétel részbeni kiemelése – indextúllépés elkerülésére II.

efPT: e≤v ∧ T(x) [, Következő(x)] x-invariáns [∧ ∃Következő(v)]

```
x:=e
Ciklus amíg x≤v és nem T(x)
  x:=Következő(x)
Ciklus vége
l:=x≤v
```

⇔

```
x:=e;  l:=T(x)
Ciklus amíg x<v és nem l
  x:=Következő(x);  l:=T(x)
Ciklus vége
```

Megjegyzés:

A `Következő(v)` létezése nem a programtranszformáció feltétele, hanem a baloldali algoritmus működésének.

---

[7] Nyilvánvaló tulajdonságokkal:
`∃Előző(x) ⇒ Előző(x)<x ∧ Következő(Előző(x))=x`
`∃Következő(x) ⇒ x<Következő(x) ∧ Előző(Következő(x))=x`

PT14. Utolsó elem vizsgálatának kiemelése a ciklusfeltételből

efPT: T(x) [, Következő(x)] x-invariáns [∧ ∃Következő(v)]

```
x:=e                                        ⇔  x:=e
Ciklus amíg x≤v és még nem T(x)                Ciklus amíg x<v és még nem T(x)
  x:=Következő(x)                                x:=Következő(x)
Ciklus vége                                    Ciklus vége
l:=x≤v                                         l:=T(x)
```

Megjegyzés:

Az előző megjegyzés itt is érvényes.

PT15. Elágazások transzformálása I.

efPT: ∀i∈[1..N]: $F_i(x)$, $P_i(x,y)$ x-invariáns ∧ ∀x: ∀i≠j: ¬($F_i(x)$∧$F_j(x)$)

```
Ha F1(x) akkor P1(x,y)                      ⇔  Elágazás
Ha F2(x) akkor P2(x,y)                            F1(x) esetén P1(x,y)
…                                                 F2(x) esetén P2(x,y)
Ha FN(x) akkor PN(x,y)                            …
                                                  FN(x) esetén PN(x,y)
                                                  Elágazás vége
```

Megjegyzés:

Az előfeltétel utolsó részében azt fogalmaztuk meg, hogy a feltételek páronként egymást kizáróak, azaz bármely x-re legfeljebb egy feltétel teljesülhet.

PT16. Elágazások transzformálása II.

efPT: F(x), P(x,y), Q(x,y) x-invariáns

```
Ha F(x)  akkor  P(x,y)                      ⇔  Ha F(x)  akkor  P(x,y); R(x,y)
       különben Q(x,y)                               különben Q(x,y); S(x,y)
Ha F(x)  akkor  R(x,y)
       különben S(x,y)
```

Megjegyzés:

Az előfeltétel utolsó részében azt fogalmaztuk meg, hogy a feltételek páronként egymást kizáróak, azaz bármely x-re legfeljebb egy feltétel teljesülhet.

PT17. Azonos utasítások kiemelése elágazás mindkét ágából

efPT: F(x) x-invariáns

```
Ha F(x) akkor   P(x,y); Q(x,y); R(x,y)      ⇔  P(x,y)
       különben P(x,y); S(x,y); R(x,y)           Ha F(x) akkor   Q(x,y)
                                                        különben S(x,y)
                                                 R(x,y)
```

Megjegyzés:

Az előfeltételbeli x-invariancia kizárólag a P előre emeléséhez kell.

TÉTELEK EGYMÁSRA ÉPÍTÉSE

Az összeépítésben szereplő tételek

Az alábbiakban csak azokat a tételeket adjuk meg, amelyek fontos szerepet kapnak a fő témánk, a tételek összeépítése szempontjából. Mindegyik kap egy azonosítót, amellyel a későbbiek során rövidíthetjük a rájuk történő hivatkozást. (Pl. ***MT***=Másolás Tétel...)

Legelőször is a tételt, mint leképezést írjuk le az *értelmezési tartomány*ával és az *értékkészlet*ével.[8] Azután e megadást pontosítjuk (specifikáljuk) a *bemenet*, a *kimenet*, az *elő*- és *utófeltétel*ével. A specifikációt kielégítő (egy lehetséges) megoldás *algoritmus*ával zárjuk a tételt. Az algoritmust, mint eljárást definiáljuk, amelynek föltüntetjük a *bemeneti* és a *kimeneti* adatait. (De itt más az esetleges „kívülről örökölt", feladat megfogalmazta függvényparamétereket nem.) A tétel helyességének bizonyításától ebben a részben eltekintünk.

MT: Másolás($\mathbb{N}$,H^*,$H\to G$):G^*

Be: $N\in\mathbb{N}$, $X\in H^*$, $f:H\to G$

Ki: $Y\in G^*$

Ef: $N=Hossz(X)$

Uf: $\forall i(1\le i\le N): y_i=f(x_i)$ $[\Rightarrow Hossz(Y)=N]$

Az algoritmus:

```
Eljárás Másolás(Be:[9] N:PozEgész,X:Tömb(1..N:H_elemTíp)[10],
                Ki: Y:Tömb(1..N:G_elemTíp)):

  Ciklus I=1-től N-ig
    Y(I):=f(X(I))
  Ciklus vége
Eljárás vége.
```

ST: Sorozatszámítás($\mathbb{N}$,H^*,G,GxH→G):G

Be: $N\in\mathbb{N}$, $X\in H^*$, $F:H^*\to G$

Ki: $S\in G$

Ef: $N=Hossz(X) \ \wedge \ \exists F_0\in G \ \wedge \ \exists f:GxH\to G \ \wedge$

$F(x_1,...,x_N)=f(F(x_1,...,x_{N-1}),x_N) \ \wedge \ F()=F_0$

Uf: $S=F(x_1,...,x_N)$

---

8 Itt most a korábbiaknál „precízebb" ez a szignatúra: még a sorozathosszakat is tartalmazza.

9 Talán szokatlan módon a paraméterlistán a bemeneti célra szolgáló paramétert nem a '**Konstans**' minősítővel vezetjük be, hanem a '**Be:**'-vel, a kimeneti célra szolgálót nem **Változó**-ként, hanem '**Ki:**'-ként aposztrofáljuk; így mód van az információt ki és bevivőre a '**BeKi:**'-vel ezt a kétirányúságot is kifejezni.

10 A továbbiakban is közvetlen (anonim) típusdeklarációt alkalmazunk, amely slampos ugyan, de most céljainknak megfelelő.

Az algoritmus:

```
Eljárás Sorozatszámítás(Be: N:PozEgész,X:Tömb(1..N:H_elemTíp),
                        Ki: S:G_elemTíp):
  S:=F0
  Ciklus I=1-től N-ig
    S:=f(S,X(I))
  Ciklus vége
Eljárás vége.
```

KT: Keresés(N,H,H→L):(L,N)*

Be: N∈**N**, X∈H*, T:H→**L**

Ki: VAN∈**L**, SORSZ∈**N**

Ef: N=Hossz(X)

Uf: VAN ≡ (∃i (1≤i≤N): T(x_i)) ∧
VAN ⇒ 1≤SORSZ≤N ∧ T(x_{SORSZ})

Az algoritmus:

```
Eljárás Keresés(Be: N:PozEgész,X:Tömb(1..N:H_elemTip),
                Ki: VAN:Logikai,SORSZ:PozEgész):
  I:=1
  Ciklus amíg I≤N és nem T(X(I))
    I:+1
  Ciklus vége
  VAN:=(I≤N)
  Ha VAN akkor SORSZ:=I
Eljárás vége.
```

MszT: Megszámolás(N,H,H→L):N*

Be: N∈**N**, X∈H*, T:H→**L**

Ki: DB∈**N**

Ef: N=Hossz(X)

Uf: $DB = \sum_{i=1}^{N} \chi(T(x_i))$

Def[11]: χ:**L**→{0,1}, $\chi(x) := \begin{cases} 1 & ha\ x \\ 0 & ha \neg x \end{cases}$

Az algoritmus:

```
Eljárás Megszámolás(Be: N:PozEgész,X: Tömb(1..N:H_elemTip),
                    Ki: DB:PozEgész):
```

[11] Ezt a χ függvényt a későbbiekben is fölhasználjuk, de már definiálás nélkül.

```
  DB:=0
  Ciklus I=1-től N-ig
    Ha T(X(I)) akkor DB:+1
  Ciklus vége
Eljárás vége.
```

MxT: Maximumkiválasztás(ℕ,H,HxH→ℒ):ℕ*

Be: N∈**N**, X∈H*, H rendezett halmaz (⇒∃<,≤ relációk)

Ki: MAX∈**N**

Ef: N=Hossz(X) ∧ N≥1

Uf: 1≤MAX≤N ∧ ∀i(1≤i≤N): $x_{MAX} \geq x_i$

Az algoritmus:

```
Eljárás Maximumkiválasztás(Be: N:PozEgész,X:Tömb(1..N:H_elemTip),
                           Ki: MAX:PozEgész):
  MAX:=1
  Ciklus I=2-től N-ig
    Ha X(MAX)<X(I) akkor MAX:=I
  Ciklus vége
Eljárás vége.
```

KvT: Kiválogatás(ℕ,H,H→ℒ):(ℕ,ℕ*)*

Be: N∈**N**, X∈H*, T:H→**L**

Ki: DB∈**N**, Y∈**N***

Ef: N=Hossz(X)

Uf: $DB = \sum_{i=1}^{N} \chi(T(x_i))$ ∧ Y∈[1..N]^{DB} ∧ Halmazfölsorolás(Y)

∧ ∀i∈[1..DB]: T(x_{y_i})

Az algoritmus:

```
Eljárás Kiválogatás(Be: N:PozEgész,X:Tömb(1..N:H_elemTip),
                    Ki: M:PozEgész,Y:Tömb(1..M:NemNegEgész)):
  M:=0
  Ciklus I=1-től N-ig
    Ha T(X(I)) akkor M:+1; Y(M):=I
  Ciklus vége
Eljárás vége.
```

Az összeépítés lépései általános esetben

1. *A feladat specifikációjának elkészítése (bemenet, kimenet, elő- és utófeltétel).*
2. *Az utófeltétel felbontása programozási tételek utófeltételeire (esetleg segédváltozók, -függvények bevezetésével).*
3. *A felbontás helyességének bizonyítása.*
4. *A tételek „mechanikus” alkalmazása a részspecifikációk alapján.*

5. Az algoritmus rövidítése (segédváltozók kiküszöbölése, segédfüggvény törzsének direkt beillesztése) programtranszformációk alkalmazásával.

A következőkben néhány gyakori esetet tárgyalunk példa gyanánt. Ezen minták alapján bárki elkészíthet jó néhány további tételkombinációt. Kiindulópontként érdemes a tételspecifikációk fejsoraként alkalmazott szignatúrákat figyelembe venni: hogy a formális szempontból egyáltalán elképzelhető kombinációkat földerítsük.

1. Másolással összeépítés

*Másolás*sal bármelyik programozási tétel egybeépíthető, hiszen csupán annyi a teendő, hogy a programozási tételben szereplő x_i *bemenő* adatra hivatkozást kicseréljük valamilyen g-transzformált x_i-re ($g(x_i)$).

Ha például egy számsorozat elemeinek négyzetösszegét kellene megadnunk, az egy ***másolás***t (számokhoz számok négyzeteinek vagy –általánosan: valamely:– g-transzformáltjainak a rendelése) és egy ***sorozatszámítás***t (pl. összegzést) tartalmaz. Nézzük meg e két tétel általános egymásra építését!

1. lépés: a specifikáció elkészítése:

Be: $N \in \mathbf{N}$, $X \in H^*$, $F: G^* \to E$, $g: H \to G$

Ki: $S \in E$

Ef: $N = Hossz(X) \quad \wedge \quad \exists F_0 \in E \quad \wedge \quad \exists f: E \times G \to E \quad \wedge$
$F(y_1, \ldots, y_N) = f(F(y_1, \ldots, y_{N-1}), y_N) \quad \wedge$
$F() = F_0$

Uf: $S = F(g(x_1), \ldots, g(x_N))$

2. lépés: az utófeltétel felbontása:

$$Uf \quad \Leftrightarrow \quad S = F(y_1, \ldots, y_N) \qquad (1)$$
$$\wedge$$
$$\forall i\,(1 \le i \le N): \; y_i = g(x_i) \qquad (2)$$

3. lépés: a felbontás helyességének belátása:

Triviálisan igaz: az (1)-be történő '$y_i \leftarrow g(x_i)$' behelyettesítéssel az *Uf*-t kapjuk.

(A behelyettesítés után:

$$S = F(g(x_1), \ldots, g(x_N)) \qquad (1)$$
$$\wedge$$
$$\forall i\,(1 \le i \le N): \; g(x_i) = g(x_i) \qquad (2)$$

(2) azonosság, így elhagyható, (1) meg maga az *Uf*.)

4. lépés: a „naiv" algoritmus generálása:

Az (1) az Y sorozatra vonatkozó ***ST*** utófeltétele. A (2) pedig az ***MT*** X-re vonatkozó utófeltétele. Mivel ez utóbbi operál a bemeneti sorozattal, ezért először azt kell alkalmazni. Ennek eredménye lesz az ***ST*** bemeneti Y sorozata. Tehát a két tétel egymásután alkalmazása

$$S := \textbf{\textit{Sorozatszámítás}}(N, \textbf{\textit{Másolás}}(N, X, g), F_0, f)$$

szolgáltatja az alábbi megoldást:

```
Eljárás SorozatosMásolás(Be: N:PozEgész,X:Tömb(1..N:H_elemTíp),
                         Ki: S:E_elemTíp):

  ...
  Ciklus I=1-től N-ig
    Y(I):=G(X(I))
  Ciklus vége
  S:=F0
  Ciklus I=1-től N-ig
    S:=f(S,Y(I))
  Ciklus vége
Eljárás vége.
```

5. lépés: az algoritmus „naivitásának" megszüntetése:

Az 'amíg-os' ciklusra átírás után ***PT10*** miatt –mivel jelen esetben teljesülnek a feltételei– átalakítható az alábbi hatékonyabb és egyszerűbb alakúvá, persze a 'számlálásos' ciklusra visszatérés után:

```
...
S:=F0
Ciklus I=1-től N-ig
  Y(I):=G(X(I))
  S:=f(S,Y(I))
Ciklus vége
...
```

A ***PT1*** miatt tovább egyszerűsíthető (értékadások egyesítése, `Y` kiküszöbölése):

```
...
S:=F0
Ciklus I=1-től N-ig
  S:=f(S,G(X(I)))
Ciklus vége
...
```

Második példaként vegyük a ***másolás*** és a ***maximum-kiválasztás*** összeépítését! Ebben a maximális elem értékét és indexét is meghatározzuk. (Az előző feladat analógiájára ilyen lehet a legnagyobb abszolútértékű szám abszolútértékének meghatározása egy számsorozatból. Általánosság kedvéért az abszolútérték függvény helyett legyen egy g, amelyet a `G` implementál az algoritmusban.)

1. lépés: a specifikáció elkészítése:

Be: $N \in \mathbf{N}$, $X \in H^*$, C rendezett halmaz ($\Leftrightarrow \exists <, \leq$ relációk),

$g: H \to C$

Ki: $MAX \in \mathbf{N}$, $MAXÉRT \in H$

Ef: $N = Hossz(X) \quad \wedge \quad N \geq 1$

Uf: $MAX \in [1..N] \quad \wedge \quad \forall i \in [1..N]: g(x_{MAX}) \geq g(x_i) \quad \wedge$

$MAXÉRT = g(x_{MAX})$

2. lépés: az utófeltétel felbontása:

$Uf \Leftrightarrow$ $MAX \in [1..N] \wedge \forall i \in [1..N]: y_{MAX} \geq y_i$ (1)
$\wedge$
$\forall i \in [1..N]: y_i = g(x_i)$ (2)
$\wedge$
$MAXÉRT = y_{MAX}$ (3)

3. lépés: a felbontás helyességének belátása:

Triviálisan igaz: az (1), (3)-ba történő '$y_i \leftarrow g(x_i)$' behelyettesítéssel az *Uf*-t kapjuk.

4. lépés: a „naív" algoritmus generálása:

Az (1) az ***MxT*** Y sorozatra vonatkozó utófeltétele. A (2) pedig az ***MT*** X-re vonatkozó utófeltétele, (3) pedig –az értékadás szemantikája alapján– egy egyszerű értékadással megvalósítható. Mivel ismét a második tétel algoritmusa operál a bemeneti sorozattal, ezért először azt kell alkalmazni. Ennek eredménye lesz az ***MxT*** bemeneti Y sorozata. Tehát a két tétel egymásután alkalmazása

MAX:=***Maximumkiválasztás***(N,***Másolás***(N,X,g),<)

szolgáltatja megoldást. A MAXÉRT meghatározása már egy egyszerű értékadással megoldható.

Az algoritmikus megoldás „első olvasatban" tehát az alábbi lesz:

```
Eljárás MaximumosMásolás(Be: N:PozEgész,X:Tömb(1..N:H_elemTíp):
                         Ki: MAX:PozEgész, MAXÉRT: H_elemTíp):

  ...
  Ciklus I=1-től N-ig
     Y(I):=G(X(I))
  Ciklus vége
  MAX:=1
  Ciklus I=2-től N-ig
     Ha Y(MAX)<Y(I) akkor MAX:=I
  Ciklus vége
  MAXÉRT:=Y(MAX)
Eljárás vége.
```

5. lépés: az algoritmus „naivitásának" megszüntetése:

A ***PT10*** miatt átalakítható az alábbi hatékonyabb és egyszerűbb alakúvá, feltéve, hogy az I=1 esetet különválasztjuk az első ciklustól:

```
  ...
  Y(1):=G(X(1)); MAX:=1
  Ciklus I=2-től N-ig
     Y(I):=G(X(I))
     Ha Y(MAX)<Y(I) akkor MAX:=I
  Ciklus vége
  ...
```

Alakítsuk át a megoldást úgy, hogy az utófeltételben szereplő $MAXÉRT = y_{MAX}$ formula *invariáns állítás* (azaz a ciklus akárhányadik lefutásakor igaz) legyen a teljes programban:

```
...
Y(1):=G(X(1)); MAX:=1; MAXÉRT:=Y(MAX)
Ciklus I=2-től N-ig
   Y(I):=G(X(I))
   Ha Y(MAX)<Y(I) akkor MAX:=I; MAXÉRT:=Y(MAX)
Ciklus vége
...
```

A ***PT1*** 2-szeres alkalmazásával a ciklus előtti utasítások drasztikusan egyszerűsíthetők:

```
Y(1):=G(X(1)); MAX:=1; MAXÉRT:=Y(MAX)
⇒
MAXÉRT:=G(X(1)),
```

A ***PT1*** és a ***PT9*** „visszafelé” alkalmazásával a ciklusmag egyszerűsíthető

```
Y(I):=G(X(I))
Ha Y(MAX)<Y(I) akkor MAX:=I; MAXÉRT:=Y(MAX)
⇒
Ha Y(MAX)<G(X(I)) akkor MAX:=I; MAXÉRT:=G(X(I))
```

Természetesen ezek után az `Y(1)`-re és a ciklusban az `Y(I)`-re vonatkozó értékadások már nincsenek:

```
...
MAX:=1; MAXÉRT:=G(X(1))
Ciklus I=2-től N-ig
   Ha MAXÉRT<G(X(I)) akkor MAX:=I; MAXÉRT:=G(X(I))
Ciklus vége
...
```

Nézzünk meg egy újabb példát, amely akár alaptételként is előfordulhatna. A feladat. határozzuk meg egy X szám sorozat T-tulajdonságúak átlagát!

1. lépés: a specifikáció elkészítése:

Be: $N \in \mathbb{N}$, $X \in \mathbb{R}^*$, $T: R \to \mathbb{L}$

Ki: $átl \in \mathbb{R}$

Ef: N=Hossz(X)

Uf: $dbT = \sum_{i=1}^{N} \chi(T(x_i)) \quad \wedge$

$$dbT \neq 0 \Rightarrow össz = \sum_{\substack{i=1 \\ T(x_i)}}^{N} x_i \quad \wedge \; átl = össz/dbT \; \wedge$$

$$dbT = 0 \Rightarrow átl = 0$$

Mint látható számos segédváltozót be vezettünk az *Uf*-ben: `dbT`, `össz`.

Ami már eddig is világos, hogy az algoritmus a `dbT` meghatározása után egy elágazást tartalmaz. Ennek különben ága (`dbT=0` esetén) borzasztóan egyszerű: `átl:=0`. Vagyis a megoldás nagybani szerkezet így alakul:

```
dbT:=???
Ha dbT≠0 akkor  össz:=???; átl:=össz/dbT                    (*)
        különben átl:=0
```

A `???` részek azok, amik megfontolást igényelnek.

2. lépés: az utófeltétel felbontása:

Kétféleképpen is elkészítjük az *Uf* felbontását. Mindkettőben a tételeknél felismerhetetlen, furcsa, feltételes szumma megszüntetése a cél. Az elsőben, a már „klasszikus”, a segéd sorozat bevezetése módszert alkalmazzuk.

Uf′: $dbT=\sum_{i=1}^{N}\chi(T(x_i))$ ∧ (1′)

$\forall i\in[1..N]: ((T(x_i) \Rightarrow y_i=x_i) \wedge (\neg T(x_i) \Rightarrow y_i=0))$ (2′)

$dbT\neq 0 \Rightarrow$

$össz=\sum_{i=1}^{N} y_i$ ∧ (3′)

$átl=össz/dbT$ ∧ (4′)

$dbT=0 \Rightarrow$

$átl=0$ (5′)

A másik átiratban egy segédfüggvényt (a χ-függvényt) vezetünk be, illetve építünk be a szumma tagjaiba.

Uf″: $dbT=\sum_{i=1}^{N}\chi(T(x_i))$ ∧ (1″)

$dbT\neq 0 \Rightarrow$

$össz=\sum_{i=1}^{N}\chi(T(x_i))*x_i$ ∧ (2″)

$átl=össz/dbT$ ∧ (3″)

$dbT=0 \Rightarrow$

$átl=0$ (4″)

3. lépés: a felbontás helyességének belátása:

Ezt most elhagyjuk…

4. lépés: a „naiv” algoritmus generálása –Uf’-höz–:

Nézzük meg az elsőt! Ebben felismerhető a megszámolás tétel (1′), a másolás tétel (2′), a sorozatszámítás tétel (3′). A (4′) és az (5′) egy-egy értékadással algoritmizálható. A megoldás legfelsőbb szintje ugyanaz, amit korábban előrejeleztünk. A másolás tételbeli függvény itt könnyen belátható módon egy kétirányú elágazással valósítható meg. Vagyis:

Ha T(X(I)) **akkor** Y(I):=X(I) **különben** Y(I):=0

Figyelembe véve a korábbi algoritmusvázlatot (*), és a fentieket, kapjuk:

```
[MszT - megszámolás tétel alkalmazása:]
dbT:=0
Ciklus I=1-től N-ig
  Ha T(X(I)) akkor dbT:+1
Ciklus vége
[MT - másolás tétel alkalmazása:]
Ciklus I=1-től N-ig
  Ha T(X(I)) akkor Y(I):=X(I) különben Y(I):=0
Ciklus vége
Ha dbT≠0 akkor
  [ST - sorozatszámítás tétel alkalmazása:]
  össz:=0
  Ciklus I=1-től N-ig
    össz:=össz+Y(I)
  Ciklus vége
  átl:=össz/dbT
különben
  átl:=0
Elágazás vége
```

5. lépés: az algoritmus „naivitásának” megszüntetése –Uf’-höz–:

Az első két tétel összevonása a PT10 segítségével elvégezhető.

```
[MszT - megszámolás és MT - másolás tétel alkalmazása:]
dbT:=0
Ciklus I=1-től N-ig
  Ha T(X(I)) akkor dbT:+1
  Ha T(X(I)) akkor Y(I):=X(I) különben Y(I):=0
Ciklus vége
```

Majd a PT16-tal egyesíthető a két, azonos feltételű elágazás:

```
[MszT - megszámolás és MT - másolás tétel alkalmazása:]
dbT:=0
Ciklus I=1-től N-ig
  Ha T(X(I)) akkor  dbT:+1; Y(I):=X(I)
          különben Y(I):=0
Ciklus vége
```

Ahhoz, hogy bármi további transzformációra esélyünk legyen, ki kell emelni a sorozatszámítás tételt az elágazásból. Ez megtehető a PT17 szerint. A teljes algoritmus így adódik:

```
dbT:=0
Ciklus I=1-től N-ig
  Ha T(X(I)) akkor  dbT:+1; Y(I):=X(I)
          különben Y(I):=0
Ciklus vége
```

```
[ST - sorozatszámítás tétel alkalmazása:]
össz:=0
Ciklus I=1-től N-ig
   össz:=össz+Y(I)
Ciklus vége
Ha dbT≠0 akkor
  átl:=össz/dbT
különben
  átl:=0
Elágazás vége
```

Megint két azonos szervezésű ciklus következik egymásután. Most is összevonhatók a PT10 szerint.

```
dbT:=0; össz:=0
Ciklus I=1-től N-ig
   Ha T(X(I)) akkor  dbT:+1; Y(I):=X(I)
             különben Y(I):=0
   össz:=össz+Y(I)
Ciklus vége
Ha dbT≠0 akkor
  átl:=össz/dbT
különben
  átl:=0
Elágazás vége
```

Az első ciklusban a PT17 „fordított” alkalmazásával a ciklusmag második utasítását bevisszük az elágazás mindkét ágába:

```
   Ha T(X(I)) akkor  dbT:+1; Y(I):=X(I); össz:=össz+Y(I)
             különben Y(I):=0; össz:=össz+Y(I)
```

A PT1 már szokásos bevetésével mind az akkor-, mind a különben-ágon két-két értékadást eggyé alakítunk:

```
   Ha T(X(I)) akkor  dbT:+1; össz:=össz+X(I)
             különben össz:=össz+0
```

A különben elhagyható, hiszen valójában nem csinál semmit. Így kapjuk az alábbi teljes és végleges algoritmust:

```
dbT:=0; össz:=0
Ciklus I=1-től N-ig
   Ha T(X(I)) akkor dbT:+1; össz:=össz+X(I)
Ciklus vége
Ha dbT≠0 akkor
  átl:=össz/dbT
különben
  átl:=0
Elágazás vége
```

A másik, *Uf”* specifikációban két tételt ismerhetünk föl: a megszámolás tételt (1”) és a sorozatszámítás tételt (2”). A (3”) és (4”) itt is egy-egy értékadással elérhető. Röviden gondoljuk végig az ebből kiinduló levezetést!

4. lépés: a „naiv” algoritmus generálása –Uf”-höz–:

A korábbiakat nem megismételve a „naiv” algoritmus így fest:

```
[MszT - megszámolás tétel alkalmazása:]
dbT:=0
Ciklus I=1-től N-ig
   Ha T(X(I)) akkor dbT:+1
Ciklus vége
Ha dbT≠0 akkor
   [ST - sorozatszámítás tétel alkalmazása:]
   össz:=0
   Ciklus I=1-től N-ig
      össz:=össz+χ(T(X(I)))*X(I)
   Ciklus vége
   átl:=össz/dbT
különben
   átl:=0
Elágazás vége
```

5. lépés: az algoritmus „naivitásának” megszüntetése –Uf”-höz–:

Mindenek előtt a sorozatszámítás ciklusmagjára fókuszáljunk. Ebben az alábbi módon kiszámolható függvény szerepel, egy kifejezés részeként.

```
Függvény χ(Konstans x:Logikai): Egész
  Ha x akkor  χ:=1
     különben χ:=0
Függvény vége.
```

Az értékadást a PT1 fordított alkalmazásával: (egy segédváltozó bevezetésével) két értékadásra bontjuk:

```
SS:=χ(T(X(I)))
össz:=össz+SS*X(I)
```

Világos, hogy megengedett a függvény hívása helyett magát a törzsét a hívás helyére beilleszteni –az aktuális paraméterekkel (mind a bemeneti, mind az érték paraméterre gondolva)–:

```
Ha T(X(I)) akkor  SS:=1
           különben SS:=0
össz:=össz+SS*X(I)
```

A PT17 „fordított” alkalmazásával az S-re vonatkozó értékadást bevisszük az elágazás mindkét ágába:

```
Ha T(X(I)) akkor  SS:=1; össz:=össz+SS*X(I)
           különben SS:=0; össz:=össz+SS*X(I)
```

A PT1-gyel az értékadás az `SS` segédváltozót kiküszöböljük:

```
Ha T(X(I)) akkor  össz:=össz+1*X(I)
           különben össz:=össz+0*X(I)
```

A „nem transzformáló” különben-ágat elhagyva a teljes algoritmus így áll elő:

```
[MszT - megszámolás tétel alkalmazása:]
dbT:=0
Ciklus I=1-től N-ig
   Ha T(X(I)) akkor dbT:+1
Ciklus vége
Ha dbT≠0 akkor
   [ST - sorozatszámítás tétel alkalmazása:]
   össz:=0
   Ciklus I=1-től N-ig
     Ha T(X(I)) akkor össz:=össz+X(I)
   Ciklus vége
   átl:=össz/dbT
különben
   átl:=0
Elágazás vége
```

A továbblépéshez most is a két tételt egymás közelségébe kell vinni. Ennek nincs akadálya, hiszen a dbT-től függetlenül elvégezhető az összegzés, legfeljebb „fölöslegesen” dolgozunk…

```
[MszT - megszámolás tétel alkalmazása:]
dbT:=0
Ciklus I=1-től N-ig
  Ha T(X(I)) akkor dbT:+1
Ciklus vége
[ST - sorozatszámítás tétel alkalmazása:]
össz:=0
Ciklus I=1-től N-ig
  Ha T(X(I)) akkor össz:=össz+X(I)
Ciklus vége
Ha dbT≠0 akkor
  átl:=össz/dbT
különben
  átl:=0
Elágazás vége
```

A PT10 ismételt alkalmazása után egyetlen ciklus végzi a két tétel lényegi részét. Majd a PT17 „fordított” alkalmazásával egyesítjük az elágazásokat:

```
dbT:=0; össz:=0
Ciklus I=1-től N-ig
  Ha T(X(I)) akkor dbT:+1; össz:=össz+X(I)
Ciklus vége
```

Ezzel elértük a már korábban más úton levezetett algoritmust:

```
dbT:=0; össz:=0
Ciklus I=1-től N-ig
  Ha T(X(I)) akkor dbT:+1; össz:=össz+X(I)
Ciklus vége
Ha dbT≠0 akkor
  átl:=össz/dbT
különben
  átl:=0
Elágazás vége
```

Gyakorlásként oldjuk meg a következő feladatot: számítsuk ki egy számsorozat pozitív és negatív elemeinek átlagát!

2. Megszámolással összeépítés

A *megszámolás*t három elemi programozási tétellel érdemes egybeépíteni, az *eldöntés*sel, a *kiválasztás*sal, valamint a *keresés*sel.

Itt olyan kérdéseket tehetünk fel, hogy „*van-e* egy sorozatban *legalább/legfeljebb/pontosan K db* T tulajdonságú elem”, vagy hogy „adjuk meg a sorozat *K.* T tulajdonságú elemét” stb.

Az általánossága miatt nézzük a *megszámolás* és a *keresés* egymásra építését! Az *eldöntés*nél, illetve a *kiválasztás*nál hasonlóan kellene eljárnunk, hiszen e két típusalgoritmus megoldásszövege része a *keresés* megoldásszövegének.

1. lépés: a specifikáció elkészítése:

Be: N∈**N**, X∈H*, K∈**N**, T:H→**L**

Ki: VAN∈**L**, SORSZ∈**N**

$$Ef:\ N=Hossz(X) \ \wedge\ K\in[1..N]$$

$$Uf:\ VAN=\sum_{i=1}^{N}\chi(T(X_i))\geq K \ \wedge$$

$$VAN \Rightarrow SORSZ\in[1..N] \ \wedge\ T(X_{SORSZ}) \ \wedge\ \sum_{i=1}^{SORSZ}\chi(T(X_i))=K$$

2. lépés: az utófeltétel felbontása:

$$Uf \Leftrightarrow VAN=(\exists j\in[1..N]:\ \sum_{i=1}^{j}\chi(T(X_i))=K) \ \wedge \qquad (1a)$$

$$VAN \Rightarrow SORSZ\in[1..N] \ \wedge\ T(X_{SORSZ}) \qquad (1b)$$

$$\wedge\ \sum_{i=1}^{SORSZ}\chi(T(X_i))=K \qquad (2)$$

3. lépés: a felbontás helyességének belátása:

Valójában, ami egyáltalán belátandó:

$$\sum_{i=1}^{N}\chi(T(X_i))\geq K \ \Leftrightarrow\ \exists j\in[1..N]:\ \sum_{i=1}^{j}\chi(T(X_i))=K.$$

a $K\leq N$ feltétel mellett.

($\Rightarrow$): Ha az ekvivalencia bal oldala igaz, akkor

♣ mivel $\chi(T(x_i))=0$ vagy 1, ezért a jobboldali formula $\forall j$: $j<K$-ra biztosan hamisak;

♣ továbbá a szumma monoton növekedő az elemszám növekedtével;

♣ és a legfeljebb „1-gyel növekedés” miatt a K értéket csak úgy lépheti túl, ha közben *valamilyen j-re fel is veszi azt*.

($\Leftarrow$): Ha az ekvivalencia jobb oldala teljesül, akkor
mivel $\chi(T(x_i))\geq 0$, ezért $j=N$-re nyilvánvalóan $\geq K$ teljesül.

4. lépés: a „naiv” algoritmus generálása:

Az (1a-b) a ***KT*** furcsa variánsának az utófeltétele. A (2) pedig az ***MszT*** utófeltételére emlékeztet (az ottani N-t és DB-t cserélve $SORSZ$-ra és K-ra).

Érdemes figyelmünket először a ***KT*** alkalmazására koncentrálni. A megoldás keretéül a ***KT*** maga szolgál. Ennek 3. paramétere –mint korábban láttuk– egy logikai értékű függvény, ahova jelen esetben az a reláció kerül, amely egyik oldalán az ***MszT*** által $-x_i$-vel bezárólag– kiszámolt darabszám, a másikon pedig a K konstans van: ***Megszámolás***$(i,X,T)\geq K$. Az ***MszT*** a keresésben való továbblépéssel egyre bővülő sorozatra $(x_1,\ldots,x_i)$ vonatkozik:

$$(Van,SORSZ):=\textbf{\textit{Keresés}}(N,X,i\in[1..N]:\ \textbf{\textit{Megszámolás}}(i,X,T)\geq K)$$

szolgáltatja az alábbi megoldást:

```
Eljárás MegszámolvaKeres(Be: N:PozEgész,X:Tömb(1..N:H_elemTíp):
                         Ki: VAN:Logikai, SORSZ:PozEgész):
```

```
  ...
  I:=1
  Ciklus amíg I≤N és nem Megszámol(I,X)≥K
    I:+1
  Ciklus vége
  VAN:=(I≤N)
  Ha VAN akkor SORSZ:=I
Eljárás vége.
```

ahol a `Megszámol` függvényt az ***MszT*** alapján algoritmizáljuk `(2)`-höz, amely kielégíti az alábbi *utófeltétel*t:

$$\texttt{Megszámol(j,X)} = \sum_{m=1}^{j} \chi(T(x_m)) \qquad (3)$$

```
Függvény Megszámol(Be: J:NemNegEgész,X:Tömb(1..N:H_elemTíp)):
                                                        NemNegEgész
  ...
  DB:=0
  Ciklus m=1-től J-ig
    Ha T(X(m)) akkor DB:+1
  Ciklus vége
  Megszámol:=DB
Függvény vége.
```

5. lépés: az algoritmus „naivitásának” megszüntetése:

Térjünk vissza a felsőbb szintű eljáráshoz! Az egyszerűsítést nagyban akadályozó azon tényt, hogy az ***MszT*** alkalmazása a ciklus feltételében fordul elő, a ***PT11*** **programtranszformáció**val szüntethetjük meg. (Célszerűségből segédváltozóként `DB`-t választottunk. Persze tudva arról, hogy ezzel a döntésünkkel kifejezett *interferenciát* okozhatunk a már meglévő program részekkel. Mivel `DB` funkciójának megfelelő szerepet kap, ez kellő garancia arra, hogy az „áthallás” nem megy a program megbízhatóságának rovására.) Ezután kapjuk:

```
  ...
  I:=1; DB:=Megszámol(1,X)
  Ciklus amíg I≤N és nem DB≥K
    I:+1;  DB:=Megszámol(I,X)
  Ciklus vége
  VAN:=(I≤N)
  Ha VAN akkor SORSZ:=I
  ...
```

A ***PT12*** programtranszformációval megspórolhatjuk a többszörös `Megszámol` függvényhívást.

Most az

- ♣ `x:=e` helyett `I:=1; DB:=Megszámol(1,X)`[12]
  de tudva, hogy
  `előző(e)` ezekre nem más, mint `I:=0; DB=0`
- ♣ `x:=Következő(x)` helyett `I:+1; DB:=Megszámol(I,X)`
- ♣ `T(x)` helyett **`nem`** `DB≥K`
- ♣ `l` helyett `VAN`

szerepel.

[12] Mintha az absztrakt algoritmus `x`-e helyébe az `(I,DB)` pár kerülne.

```
...
I:=0; DB:=0; VAN:=Hamis
Ciklus amíg I<N és nem VAN
  I:+1; DB:=Megszámol(I,X)
  VAN:=DB≥K
Ciklus vége
Ha VAN akkor SORSZ:=I
...
```

A további ésszerűsítés már nem végezhető el egyszerűen valamilyen programtranszformáció segítségével. Mélyebb összefüggéseket kell keresnünk. E célból fogalmazzuk meg a felső szint ciklusának *invariáns állítás*át. A keresés szokásos *invariánsa*:

$$\forall j \in [1..i-1]: \neg T(x_j)$$

ami a mostani esetre újrafogalmazva:

$$\forall j \in [1..i-1]: \neg \text{Megszámol}(j,X) \geq K$$

Figyelembe véve, hogy a ciklusmag mindössze `I:+1`, így a megragadandó állításban kapcsolatot kell találni a `Megszámol(I,X)` és a `Megszámol(I+1,X)` között.

Állítás:

$$\text{Megszámol}(i,X)=L \quad \overset{I:=I+1}{\Rightarrow} \quad \text{Megszámol}(i+1,X)=L \ \vee \ \text{Megszámol}(i+1,X)=L+1$$

Bizonyítás:

Vizsgáljuk meg az állítás következményrészét:

$$\text{Megszámol}(i+1,X)=L \ \vee \ \text{Megszámol}(i+1,X)=L+1 \qquad (4)$$

A (3) és az χ függvény értelmezése miatt

$$(4) \Leftrightarrow (\text{Megszámol}(i,X)=L \ \wedge \ \neg T(x_{i+1})) \vee (\text{Megszámol}(i,X)=L+1 \ \wedge \ T(x_{i+1}))$$

Az állítás feltételét figyelembe véve ez ekvivalens az alábbival:

$$\neg T(x_{j+1}) \ \vee \ T(x_{j+1}) \ \Leftrightarrow \ \text{Igaz}.$$

(Qed.)

Az állítás fontos gyakorlati következménye, hogy az előző `Megszámol(j,X)` érték fölhasználható *újraszámolás (azaz az eljárás újrahívása) nélkül is*.

Nyilvánvaló az előbbi állítás, a (3) és az χ függvény értelmezése alapján, hogy az alábbiak ekvivalens eredményre vezetnek (nagyon is eltérő hatékonyság mellett):

`DB:=Megszámol(I,X)` , ill. **`Ha`** `T(X(I))` **`akkor`** `DB:+1`

Így jutunk a végső változathoz:

```
...
I:=0; DB:=0; VAN:=Igaz
Ciklus amíg I<N és VAN
  I:+1
  Ha T(X(I)) akkor DB:+1
  VAN:=nem DB≥K
Ciklus vége
Ha VAN akkor SORSZ:=I
...
```

3. Maximum-kiválasztással összeépítés

*Maximum-kiválasztás*sal kapcsolatban efféle kérdéseket fogalmazhatunk meg: 1) hány darab maximális értékű elem van, 2) melyek a maximális értékű elemek, 3) van-e több maximális értékű elem.

Ezeknél a *maximum-kiválasztás*t kell egybeépíteni a *megszámolás*sal, a *kiválogatás*sal, illetve az *eldöntés*sel.

Mivel a kigyűjtéses *kiválogatás* mindig tartalmaz egy *megszámolás*t, így csak a *kiválogatás*sal kell foglalkoznunk.