Típus-specifikációk

Szlávi Péter

Tartalom

TÍPUS-SPECIFIKÁCIÓK[1] (ÖSSZEFOGLALÁS)

0. AZ ALAPKONCEPCIÓNKRÓL ÉS A JELÖLÉSEKRŐL

A *típus* absztrakt fogalma alatt egy négyest értünk [V2]: T=(A,R,Σ,K), ahol

- *adatelemek* (nem üres) A halmaza (bázistípusok értékhalmazai), } *a típus statikus*
- az adatelemeket egy struktúrává egyesítő R *szerkezeti összefüggés*, } *meghatározói*
- a teljes struktúra egyes részeit kiválasztó ún. *szelekciós műveletek* Σ halmaza, } *asszociált műveletek, a típus*
- teljes struktúrát eredményező ún. *konstrukciós műveletek* K nem üres halmaza. } *dinamikus meghatározói*

Megjegyezzük, hogy

1. az A halmazról feltételezzük, hogy *konstruktív*, azaz minden eleme előállítható az asszociált műveletek megfelelő sorrendben történő –véges vagy megszámlálható sokszori– egymásután alkalmazásával.

2. a műveletekre vonatkozó kategórikus szétválasztás nem végezhető el mindig teljes egyértelműséggel [V2], hiszen adódhatnak olyan műveletek is, amelyek karakterisztikus céljuk ugyan a struktúra valamely részének „tesztelése" (szelekciós funkció), mellékesen azonban a teljes struktúrát módosítják (konstrukciós funkció). Ilyen példa a klasszikus `Veremből` művelet, ami legfontosabb célja a legfelsőbb elem „bemutatása", eközben azonban a veremből azt el is távolítja, s így a verem egészét átalakítja. Persze kikerülhető ez az összefonódás úgy, hogy külön művelettel teszteljük (`Tető`) és külön leválasztjuk (`Levesz`).

3. az absztrakt tárgyalás esetén nincs szerepe az R szerkezeti összefüggésnek –ez majd az implementációnál válik fontossá–, s ezért meg lehet feledkezni róla, továbbá a műveletek fenti két halmazát is „egyként" (F) adhatjuk meg, ha külön feltételként megfogalmazzuk, hogy létezzen legalább egy konstrukciós művelet: T=(A,F) és ∃f∈F: f konstrukciós művelet. [V1]

4. a típus-definíciót a homogén, heterogén és típus-algebrák fogalmából kiindulva vezeti be Guttag és Horning nevezetes cikkében [GH]; ill. az iniciális algebra eszköztárához nyúl Goguen, Thatcher és Wagner [GTW]. Hogy miért kell komoly matematikai eszközöket fölvonultatni a típus definiálásához arra még később visszatérünk.

Valójában nem típusokról szólunk az alábbiakban, hanem típusképző mintákról, sablonokról, vagyis *típus-konstrukciós eszköz*ökről. Azt az engedményt tesszük, hogy az adatelemek A halmaza esetleg ismeretlen, pontosabban csak a típuskonstrukciós eszköz *paraméter*eként van formálisan megadva. (Ez tükröződik Varga László [V1] cikke definíciójában, ahol az A halmaznak csak egyik komponense az objektumok halmaza, s további komponenshalmazai éppen a fent említett paramétereket formalizálják. Hasonló értelemben gondolkodnak a hivatkozott irodalamakban: [V1, GTW, Latal, Sz1,Sz2,SzZs1,PSzZs1,PSzZs2, K2, 2MC].) Vizsgálatunk tárgya olyan típuskonstrukciós eszközök, amelyek *egyetlen* (H) *bázistípus*sal generálhatók. Végülis tehát ún. *sorozat*okról lesz szó ($s \in H^*$), amelyekben az elemek sorszámuk ({1,2,...}) segítségével *fölsorolható*k. Figyelem: a fölsorolhatóságnak nincs köze az adott sorozatféléhez rendelt operációkhoz, most kizárólag az A-ról beszéltünk!

Anélkül, hogy leszűkíteni igyekeznénk mondanivalónkat, a rövidségre törekvés miatt egyszerűen típust fogunk mondani, de tudatában leszünk annak, hogy típusok egy –paraméter(ek) által meghatározott– családjára vonatkoznak állításaink.

A típusok specifikációjának két formáját alkalmazzuk: az *algebrai*t és az *algoritmikus*t. Mindkettőben a típus legfontosabb jellemzőit ragadjuk meg, s ameddig lehet, igyekszünk elkerülni a konkretizálást. Az axiomatikus

[1] Megkockáztatjuk azt az előzetes megjegyzést, hogy valójában objektumosztályok specifikálásáról lesz szó, legalábbis az objektum orientáltság egyik népszerű értelmezése szerint. Erre utal egy későbbi fejezet, amelyben a tárgyalt osztályok öröklési hierarchiáját vázoljuk.

(értsd: algebrai) specifikálásnál ezt a célt el lehet érni[2], az algoritmikusnál –mint látni fogjuk– csak részben. Ui. amíg „csak” a típussal kapcsolatos „kívülről” is érzékelendő tudnivalókat összegezzük, addig pontosan a lényeget érintjük, amint azonban a megvalósítás mikéntjére térünk rá, kénytelenek vagyunk *reprezentációs* (azaz a struktúra memóriábaképzésére vonatkozó) és *implementációs* (az ábrázolást figyelembe vevő algoritmikus)[3] döntéseket hozni.

Algebrai specifikáció

Az algebrai specifikáció célja a típusnak –azaz azonos szerkezetű és viselkedésű[4]– adatok halmazának, mint *egész*nek a működését formálisan leírni. Ebben az értelemben a hagyományos programspecifikáció párja. Amíg azonban a programspecifikációt megadhatjuk egy *elő-utófeltétel* párral, addig a típusnál nincs erre mód. Oka a program és a típus eltérő céljában keresendő: míg a program esetében egyértelműen megfogalmazható az *elérendő állapot* (épp ezt rögzítjük az utófeltételben), addig a típus csak bizonyos adatféleségre vonatkozó programozási *eszközök* egy „*készlete*”, ezért nem beszélhetünk elérendő állapotról. Legfeljebb valamely végállapotfelé tartó utak „útjelzőit” verhetjük le a programkészítés rögös ösvényei mellé, amit figyelembe kell vennie a típus felkínálta eszközkészletet fölhasználó programozónak. Ezt célozzák meg az *axiómá*k.

A típust tehát nem egy kívánatos utófeltétellel specifikáljuk, hanem az eszközök, azaz az eljárások, függvények, operátorok egymásra hatásának axiómáival[5]. Megadván, hogy az egyes *operáció*knak (értsük ez alatt az eljárásokat, függvényeket és operátorokat) *értelmezési tartomány*át és *értékkészlet*ét (szintaxisát)[6] és a viselkedésüket formalizáló *axiómá*kat (szemantikát). Az persze igaz, hogy ha majd a típus valamely műveletét önmagában vizsgáljuk, akkor már van értelme, sőt kell is, megadni *elő-* és *utófeltétel*t.

A könnyebb formalizálhatóság kedvéért minden operációt az algebrai leírásban egy absztrakt *függvény*nek tekintünk, ahogy ezt számos irodalomban teszik [GH,GTW,Sz1,Sz2,SzZs1], függetlenül attól, hogy majd az implementáció során neki eljárás, függvény vagy operátor felel-e meg.[7] A nem korrekt használat érzékelhetővé tételére célszerű bevezetni egy *típusfüggetlen konstans*t [Sz1,Sz2,SzZs1], a *NemDef* értéket, amelyet az egyes operációk értékül adnak, ha képtelenek a kívánt transzformációt elvégezni.[8] Az „absztraktság” egyben azt is jelenti, hogy a függvények *mellékhatásnélküli*ek (azaz paramétereik értékeit nem módosítják a kiszámolás során).

A műveletek szignatúráját az alábbi szintaxissal adjuk meg:[9]

Operáció(*Értelmezési tartomány*): *Értékkészlet*

ami ugyanazt jelenti, mint a matematikában szokásos: *Operáció* : *Értelmezési tartomány* → *Értékkészlet* .

Pl. a verem két klasszikus műveletének szignatúrája [GHM1,Sz1,Sz2,SzZs1]:

---

[2] Bár ezen eszköz is alkalmazható a konkrét megvalósítás szintjén.

[3] Az irodalomban gyakran *implementáció* alatt az ábrázolást és az operációk algoritmusainak elkészítését együtt értik.

[4] Viselkedés alatt azok kezelését és „együttműködésük” mikéntjét értjük.

[5] Ez nem zárja ki persze annak lehetőségét, hogy alkalmasint ne mondjunk ki olyan állítást, amelynek igaz voltában nem kételkedhetünk. Ha ilyent kijelentünk azt *típusinvariáns*nak (esetleg *struktúrainvariáns*nak) nevezzük. Gyakorta fogalmazunk meg ilyen állítást, amikor egy létező típusból származtatunk az adathalmaz korlátozásával új típust. Ennek számos példájával találkozhatunk a [PSzZs2]-ben.

[6] Ezt a megadást szokás *szignatúra*ként emlegetni.

[7] Az [P1,PP1]-ben példát láthatunk arra, hogy miként lehet már a specifikációban is használni a programozási nyelvekben szokásos, „szabad szintaktikájú” operátorokat.

[8] Ez az elképzelés eltér az [2MC] irodalomban alkalmazott szokástól, ahol a sikeres vagy sikertelen voltát adják vissza az egyes függvények. Ezzel nem tudunk egyetérteni, mivel –véleményünk szerint– a hibázás visszatükrözése kevésbé fontos, mint a tulajdonképpeni kifejezendő leképezés.

Megjegyezzük, tipikusabb az a koncepció, mely szerint az axiómákat oly módon kell megfogalmazni, hogy hibás működés eleve ne fordulhasson elő. Ezt úgy lehet elérni, hogy az egyébként nem túl „logikus” szituációban is valamilyen „semmit mondó”, de legális eredménnyel érjen véget az alkalmazott operáció „számítása”. Hogy ez mennyi veszélyt hordoz, arra [GTW]-ben jó példákat olvashatuk.

Egyes szerzők [GTW] továbblépnek ezen a filozófián a preczitás növelése érdekében, és *típus*os *Error* értéket vezetnel be.

[9] Sok irodalomban az általunk használttól eltérő a szignatúramegadás szintaxisa, megegyezik a matematikában szokásosnak mondhatóval [V1,V3,GH,PP1]. Mi a későbbi algoritmikus jelölésektől való kisebb eltérés miatt javasoljuk az itteni formát.

Verembe(Verem,Elem): Verem ∪ {NemDef}

Veremből(Verem): (Verem x Elem) ∪ {NemDef}

Egyes függvények értékkészletét több „komponensből” célszerű összeállítani. (Bár el kell ismerni ez szokatlan az algebrai specifikációk publikációinál, mivel algebrai „bonyodalmakat” hoz be a formális kezelésbe. Sőt egyenesen kétségessé teszi, hogy ezen vizsgálat alapjaként emlegetett *típusalgebra* algebrai volta megmarad-e egyáltalán.) Például a Veremből függvény esetében –mint láttuk– az értékkészlet: (Verem**x**Elem) ∪ {NemDef}, ami azt fejezi ki, hogy az operáció elvégzése hatással van a verem egészére (azaz mintegy új *verem*állapotot „produkál”) és egy *elem*értéket is előállít.

Mellesleg bizonyos körülmények között „abnormális” eredményt szolgáltat, amit a *NemDef* érték jelez. Föltételezzük, hogy ha egy operáció *NemDef* eredménnyel zárult, akkor a további operációk művelete sem lehet ettől különböző, bár ez nem tükröződik az értelmezési tartományon, amely precízebben írva kellene, tartalmazza a {NemDef}-t is.

Pl. Veremből(Verem ∪ {NemDef}): (Verem x Elem) ∪ {NemDef}

Ezt az „implicit”, többnyire nem posztullált axiómát hívja az irodalom *implicit error-axióma*nak [GH]. A jó értelemben vett ellenpéldaként már emlegetett [GTW] munkában hiba-terjesztési *–error propagation– axiómá*knak nevezik, és a többi „normális” axiómával teljesen egyenrangúakként kezelik. Az elkövetkezőkben az említett axiómát kimondatlanul feltételezzük:

$$\forall Operáció: Operáció(NemDef)=NemDef$$

Több komponensből álló értékkészlet esetén, ha az axiómában szükséges az értékkészlet valamely komponensére hivatkozni, akkor az algoritmikus nyelveknél szokásos jelöléssel élünk: ***.****komponens*.[10] A *.komponens* azonosítója a szignatúrában szereplő név. Ez a tény kikényszeríti a szereplő halmazok néven nevezését (ami a későbbi programozási lépéseknél is kifejezetten praktikus). Az előbbi példánál maradva: a *v* verem új állapotát jelöli *Veremből(v).verem*, illetőleg a *Veremből(v).elem* annak kivett elemét. (Mindez természetesen csak akkor értelmes, ha nem *NemDef* az értéke, ellenkező esetben ezek értéke is *NemDef* az implicit error-axióma értelmében.)

Találkozunk majd az `Új` (vagy hasonló célú, de eltérő nevű, pl. `Létrehoz`) *konstrukciós művelet*tel, amelynek célja létrehozni az adott típusú objektumot annak iniciális állapotában (azaz adott esetben az egyes komponenseihez megfelelő kezdőállapotot rendelni). Feltételezzük (rekurzívan), hogy az adott komponenseknek is megvan az Új (vagy efféle) konsrtukciós művelete, amelyre az axiómákan az $Új_e$ operációval fogunk hivatkozni. (Az indexben levő *e* utalás arra, hogy *elem*típushoz tartozó műveletről van szó.) Ha ilyen nem lenne, akkor ez a NemDef értékkel azonos eredményt jelent.

A formulákban szereplő *egyenlőség* fogalmat a szokásos értelemben használjuk[11], amelyben visszavezetjük (rekurzívan) az összetett struktúrát fölépítő elemibb típusokon értelmezett *egyenlőség* fogalmára, végsőfokon az apriori definiáltnak tekintett *elemi* (*Egész, PozEgész, Logikai*, *Valós, NemNegValós*, *Karakter*, ...) típusokéra.[12] Nem különböztetjük meg jelölésben a különböző típusú objektumokra alkalmazott egyenlőség operátort. Természetesen az őt körülvevő operandusok típusa mindig egyértelműen eldönti, hogy melyikről van szó.[13]

Az axiómák megfogalmazásánál alkalmazni fogjuk a *predikátumkalkulus* szokásos fogalmait, műveleteit ($\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$, $\forall$, $\exists$). A rövidebb leírás érdekében ott, ahol nem okoz féreértést, a kvantorokat nem írjuk ki. [L. 1. ábrát!]

---

[10] Megtehetnénk, hogy definiálunk egy *projekció*t erre a célra a szokásos matematikai leírási móddal (nevéül kölcsönvéve a célhalmaz nevét). Pl.:

$$Verem(\text{Verem}\times\text{Elem})\text{: Verem},\ Verem(v,e):=v \text{ és } Elem(\text{Verem}\times\text{Elem})\text{: Elem},\ Elem(v,e):=e.$$

[11] Azaz elvárjuk tőle: *i.* x=x (reflexív) *ii.* x=y ⇒ y=x (szimmetrikus) *iii.* x=y és y=z ⇒ x=z (tranzitív), továbbá *iv.* ∀o∈F operátor esetén x=y ⇒ o(x)=o(y). [GH]

[12] Megjegyezzük, az elemi típusok axiomáit a [Latal] cikk jól összefoglalja (bár nézete eltér a mienkétől az összetett struktúrák vonatkozásában).

[13] Ez csak az első előbukkanása a(z operátor) *polimorfizmus* jelenségének, amely a programozásmódszertanban a típusok szabad definiálásával egyre komolyabb jelentőségre tett szert, s az *objektum orientált programozás* (OOP) során vált közismertté.

Típus Logikai:

Asszociált műveletek:

Igaz: Logikai
Hamis: Logikai
És(Logikai,Logikai): Logikai
Vagy(Logikai,Logikai): Logikai
Nem(Logikai): Logikai

Axiómák:

Jelölések:
l, l_1, l_2: Logikai *(azaz Logikai típusú objektumok)*

1° $És(l_1,l_2) = És(l_2,l_1)$

2° $És(Igaz,l) = l$

...

1. ábra
Példa az axiomatizálásra: a Logikai típus

Megjegyzés: az axiómákat bevezető „Jelölések ...” rész fölsorolja azokat az *individuumváltozókat*, amelyek a későbbi formulákban *univerzális kvantor*ral bevezetve fordulnának elő. Itt rendelünk hozzájuk valamilyen típust, amely alaphalmazából vehetnek föl tetszőleges értékeket. Tehát az axiómák ténylegesen így értendők:

1° $\forall l_1,l_2 \in Logikai: És(l_1,l_2) = És(l_2,l_1)$

2° $\forall l \in Logikai: És(Igaz,l) = l$

Az algebrai formulák és az elő-utófeltételek leírásánál szükség lesz a *sorozat* egyes elemeire hivatkozni. Ez azonban nem jelenti azt, hogy az adott típusú sorozatra megkövetelnénk az egyes elemek közvetlen elérését biztosító (az ún. *indexfüggvény*) művelet létét, csupán a formulákban való szerepeltethetőségét kívánjuk lehetővé tenni. Ekkor alkalmazzuk a matematikában szokásos –alsó indexes– jelölést. Pl. az s sorozat i. elemét s_i-vel fogjuk jelölni; a *teljes sorozat* –mint elemegyüttes– $s=(s_1,s_2,...,s_n)$, az $s=()$ az *üres sorozat*. A sorozat hosszára a *Hossz* függvénnyel fogunk hivatkozni, az alábbi „magától értetődő” szemantikával”:

$Hossz(()):=0$

$Hossz((s_1, s_2, ..., s_n)):=n \quad n>0$

Egy halmaz számosságát a *Számosság* függvénnyel adhatjuk meg. Amikor az argumentumába egy típusnevet írunk, akkor természetesen annak bázishalmazának számosságára gondolunk.[14] Azaz

$Számosság(H)=Card(H)$, ha H tetszőleges halmaz

$Számosság(T)=Számosság(A)$, ha $T=(A,F)$ tetszőleges típus

Speciális sorozatokként használjuk a szabványos inputról érkező *InputSorozat*ot és a szabványos outputra kerülő *OutputSorozat*ot. Az utófeltételben szükség lehet valamely objektum bemeneti és kimeneti értékére is, ekkor a *kimeneti érték*ét a neve mellétett aposztroffal fogjuk ellátni.[15] Az állításokban az egyes operációk mint *függvényhívás*ok szerepelnek, ha muszáj, segédobjektumok bevezetésével csökkentjük a formuláink bonyolultságát.[16]

[14] Az algebrai leírásnál nem különböztetjük meg a típust a bázishalmazától.

[15] Ebben is a [2MC]-ben alkalmazottól eltérünk, abban éppen az aposztroffal ellátott a korábbi érték. Jelölésünk védelmében: egy specifikációban minden objektumot minden állapotában ugyanazon jellel jelöljük (akár az elő-, akár az utófeltételben).

[16] További eltérés [2MC]-től, ui. abban definiálnak külön *kimeneti változó*t, amivel rövidíteni lehet a formula leírását. A kétségtelen előny ellenére mi a *kifejezőbb, világosabb* –bár hosszabb– teljes *függvényhívásos* jelölés mellett maradunk.

Az axiómáknál egy rövid informális szöveggel kezdjük, amivel a mondanivaló legfontosabb állítását igyekszünk megfogalmazni, azonban ez sokszor lényegesen „soványabb” a tényleges mondanivalónál. Ennek célja pusztán „képbe helyezés”.

Praktikus kérdésként fogalmazható meg, hogy egy tervezés alatt álló típust hány és milyen jellegű axiómával kell leírnunk a *teljesség* reményében, illetve hogy nem tartalmaz-e *ellentmondás*t [GH]. A darabszámra könnyen mondható alsó érték. Nyilvánvaló ugyanis, hogy ha k db konstrukciós és s db szelekciós műveletünk van, akkor k*s db állítás kell kell ahhoz, hogy az egyes konstrukciók hatását az egyes szelekciós műveletekkel leírhassuk. Megjegyezzük, hogy az axiómák összeállításánál csoportosításokat, összevonásokat fogunk tenni, így pusztán az axiómaszám semmit nem mond a teljességről. Ami a teljességet –elvileg– illeti: sajnálatosan általánosan nem eldönthető probléma [GH], így minden esetben külön kell vizsgálni.

Algoritmikus specifikáció

A specifikáció másik fajtája közelebb lép formalizmusban a programozási nyelvhez, hiszen célja épp az, hogy tisztázza, miként lehet az egyes operációkat alkalmazni az adott programban. Eközben arra törekszünk, hogy ne adjuk föl a típuskonstrukció általánosságát. Itt további két dologról kell döntést hoznia a típust definiálónak: 1) az exportált fogalmak (operációk és –esetleg– egyebek) „minéműsége”, szintaktikája, és 2) a megvalósítás sok-sok „hogyanja”. Az elsőt fogalmazzuk meg az ún. *ExportModul*ban, az utóbbit a reprezentációs-implemetációs *Modul*ban.[17,18] Az ExportModul a szintaxis mellett az axiómákba foglalt szemantikát *elő- és utófeltételpár*ok segítségével viszi tovább. Ezen állítások célszerűen már csak műveletekként az általuk képviselt leképezést „önmagukban”, tehát a többi operáció hatásától mentesen, tartalmazzák. A reprezentációs-implementációs Modult tekinti az irodalom *procedurális specifikáció*nak[19].

A modulok feje egyben *szintaktikai mintá*ul is szolgál. Általában a modulfejet így írhatjuk le:

```
Modul TípusNév(formPar1 elválasztóJel1 formPar2 ...)
```

A *formális paraméter*eknél[20] meg kell adni a „fajtát” (Konstans, Típus, Függvény, Operátor stb.) és a formális nevet. Az természetes, hogy a neveknek és az elválasztójeleknek világosan megkülönböztethetőknek kell lenniük! A zárójeles rész elmaradhat, ekkor egy *konkrét típus*t definiál a modul.[21] (L. a 2. ábrát!) Ha van paramétere, akkor helyesebb *típuskonstrukció*nak [Sz1,Sz2,SzZs1,PSzZs1,PSzZs2] vagy *paraméteres típus*nak [V1, GTW, Latal], *generic modul*-nak [K2], *template*-nek, azaz sablonnak [2MC] nevezni.

A fenti mintának az alábbi felhasználás felel meg:

```
Típus KonkrétTípus = TípusNév(aktpar1 elválasztóJel aktpar2 ...)
```

[17] E kettő szétválasztása két előnnyel is jár. Egyrészt lehetővé válik, hogy a fordítóprogram anélkül is elvégezhesse a program lefordítását, hogy a típust megvalósító kód jelen lenne. (Azt persze nem jelentheti, hogy kész kód is generálható nélküle.) Másrészt több változat is adható egy-egy típusmegvalósításhoz, s mindez anélkül, hogy a ráépülő programokat újra kellene fordítani (vagy pláne: írni). (Az igaz, hogy más típusmegvalósításra áttéréskor a programot újra kell linkelni, azaz újra össze kell szerkeszteni.)

[18] Az Eiffel-ben érdekes módon elállnak a két dolog különválasztásától, s a *class*-ban külön *export*-rész van a kívülről látható fogalmak felsorolására. [ME] Az Euclid-ban sincs meg a különválasztás, de a nyelv „korai” volta (1976-77) miatt ez nem különösebben meglepő. [Latal]

[19] A specifikáció szó itt már valóban elvesztette azt az absztraktságot, amellyel eddig rendelkezett. Mentségünkre legyen két tény: 1) a programfejlesztésben jólismert VDM specifikációs eszköz is használ ún. *explicit specifikáció*t, ami nem más mint saját nyelvezetén megfogalmazott algoritmus [SzW1]; 2) a modul fogalmunk is egy *absztrakció*, amelyről –ma még– át kell térni kódolással valamely konkrét programozási nyelvre.

[20] Ezt szokták *formális generic paraméter*nek nevezni.

[21] A Modula-2 nyelv modul fogalma ezt célozza meg, s csak ügyeskedések árán tehető univerzálissá. [K1]

```
ExportModul Logikai:
  Konstans Igaz, Hamis: Logikai
           Max'Logikai,Min'Logikai: Logikai
           Számosság'Logikai: Egész      [lásd SzZs2]
  Infix Operátor És(Konstans e1,e2:Logikai): Logikai
    Ef: -
    Uf: ( e1 És e2 ) = e1 ∧ e2
  Infix Operátor Vagy(Konstans e1,e2:Logikai): Logikai
    Ef: -
    Uf: ( e1 Vagy e2 ) = e1 ∨ e2
  Operátor Nem(Konstans e:Logikai): Logikai
    Ef: -
    Uf: ( Nem e ) = ¬e
  ...
Modul vége.
```

2. ábra
A `Logikai` *típus ExportModulja.*

A *NemDef* érték a megvalósításkor *Hiba*állapotként jelenik meg. A típus reprezentációjában a hibásság/hibátlanság állapotát egy további attribútum (hagyományosabb szóval élve: mező) rögzíti. Ezt az algoritmikus specifikációink szerint a *Hibás?* függvénnyel lehet tesztelni (és egyben alapállapotba hozni). E függvény minden egyes típusbeli objektumhoz *egyedileg* van rendelve.

Az irodalomban nem egységes az elképzelés az *Új konstrukciós művelet* előfeltételét illetően. Vannak, akik nem korlátozzák az efajta művelet alkalmazását eddig még létre *nem* hozottakra, s így lehetővé teszik már létező struktúra *iniciális* értékének beállítására. Mások viszont –veszélyességére hivatkozva– ezt tiltják [2MC, Sz1, Sz2], s az alkalmazás feltételéül a még nem létezést szabják. Igaz, ha ezen műveletet mint egy nullaváltozós függvényt vagy konstanst definiáljuk, akkor az előfeltételt el kell hagyjuk (hisz nincs objektum-paraméter, amire vonatkoztathatnánk). Ez utóbbi elképzelés viszont akkor válhat veszélyforrássá, ha definiált a struktúrára az *értékadás* operáció. Nos mindezek figyelembe vételével mi az előfeltétellel szigorított (parametrizált) változat mellett tesszük le voksunkat.

A megvalósítás során komolyan vetődik föl az *azonosság* és *értékadás* operációk pontosításának igénye. Ugyanis az összetett objektumok esetében kézenfekvő reprezentációs szokás a mutatókra épített *dinamikus* szerkezet alkalmazása. Ekkor azonban *értékmegosztás* történik, aminél már nem magától értetődő, mit értsünk pl. két objektum azonosságán: 1) az elemi típusok szintjén való értékazonosságot, vagy 2) a legfelsőbb szinten érzékelhető címazonosságot. További –még az axiómákkal is összefüggő– kérdőjeleket jelent az értékmódosító operációk léte: jogos-e, mikor jogos egy objektum(-rész) megváltoztatása, hisz az operáció operandusaként explicit módon nem jelzett objektumokat (amelyek a közös, a megosztott értékkel rendelkeznek) is súlythat a változás. [22]

Ezen bevezető gondolataink lezárásaként egy további felvetés: háromféleképpen is megközelítjük ugyanazt a dolgot. Logikus fölvetni a kérdést, valóban egymással összhangban írják-e le a megcélzott típust? Azaz az axiómákkal és az elő-utófeltételpárokkal megadott specifikációk megfelelnek-e egymásnak, ill. az utóbbinak megfelelő állapottranszformációt végzik-e el a procedurák? Ennek a problémának kezelésére több módszer is kínálkozik. Anélkül, hogy a részletekben elmerülnénk, háromra utalunk.

Elsőként említjük a [V1]-ben bevezetettet. Lényege, hogy az algoritmikus specifikáció elő- és utófeltételeit magukból az axiómákból közvetlenül származtassuk. Ekkor tehát az összhang bizonyítására semmi szükség nincs. A baj legfeljebb annyi, hogy az egyes operátorokhoz tartozó feltételpárban előfordulnak hivatkozások más operátorokra is, így ezek nem egymástól függetlenül vannak megragadva.

Egy másik módszer kiindulópontja, hogy a kétféle formális specifikációbeli logikai formulákat egy rendszerbe, valamilyen egységes formalizmussal szerepeltessük, majd az így előállt integrált rendszer ellentmondásmentességét és teljességét vizsgáljuk. Az világos, hogy az utóbbi elképzelés kevesebbet céloz meg a kívánatosnál, de cserében alkalmas lehet arra, hogy számítógéppel, automatikusan végezzük el.

[22] Az Eiffel ezen probblémák „tisztességes” megoldására vezeti be az *azonosság*ra/*értékmódosítás*ra kétféle értelmezésű operátort: `equals` (`deepequals`)/`copy` (`deepcopy`) [Me].

Harmadikként utalunk a legkorábbi idekapcsolódó eredményre: az algoritmikus specifikáció két része közötti összhang formálisan jól kezelhető, jól ismert, elegendő utalni Hoare munkássága nyomán kialakult, annak szellemében készült publikációkra.

Ami a matematikai szigor létjogusáltságát illeti, érdemes néhány gondolattal eljátszani ezzel kapcsolatban is. A típus T=(A,F) tulajdonképpen egy *reláció struktúra* [MV], amelyben az F-beli relációk *függvények*. Vajon egy axiómarendszerrel leírt szemantika valóban biztosítja-e az F-beli relációk függvényvoltát? Ez éppúgy alapvető kérdés, akár a teljesség, akár az ellentmondásmentesség! Miért is érdekes ez? A programozás során a programozó *több lépésben* gyártja egymásból a reláció struktúrákat. Szándéka szerint ezek *ugyanazt* a problémát közelítik meg, azaz elvárható, hogy megtartsák a fontos algebrai tulajdonságokat. Vagyis az egyes struktúra-transzformációs lépések során *homomorfizmus*okat (azaz művelettartó leképzéseket) definiál és használ. [MV]

Jelöléseket vezetünk be: $T_{ax}=(A_{ax},F_{ax})$ – axiómákkal definiált típushoz, $T_{eu}=(A_{eu},F_{eu})$ – elő-, utófeltételekkel definiált típushoz, $T_{pr}=(A_{pr},F_{pr})$ – procedurálisan definiált típushoz. A programozó az alábbi f_1, f_2 homomorfizmusokat „definiálja” kimondatlanul is:

$$T_{ax} \xleftarrow{f_1} T_{eu} \xleftarrow{f_2} T_{pr}$$

Nem pedig az alkalmazás sorrendje szerint „logikusnak” tűnőt:

$$T_{ax} \xrightarrow{f_1} T_{eu} \xrightarrow{f_2} T_{pr}$$

Ennek magyarázata: a programfejlesztés során *absztrakt* felöl a *konkrét* felé halad, vagyis az alaphalmazra többnyire valamilyen szűkítést –konkretizálást, struktúra kikényszerítette korlátozást– alkalmaz: $A_{ax} \supseteq A'_{ax} \cong A_{eu}$, ill. $A_{eu} \supset A'_{eu} \cong A_{pr}$, ahol az „$\cong$” izomorfiát jelöli.

(Sőt a helyzet a fentinél általában még bonyolultabb amiatt, hogy a programfejlesztés során különböző okok miatt kénytelen a programozó *alaphalmazbővítés*t (sőt új *operáció*k hozzávételét) alkalmazni, vagyis újabb komponensekkel kiegészíteni az alaphalmazt; illetve a specifikáláskor bizonyos rejtve maradt tulajdonságok miatt az operációkat is kissé „átértelmezni” pl. bővíteni az értékkészletet. Kétségkívül törekedni kell ilyenek elkerülésére.)

1. ALGEBRAI SPECIFIKÁCIÓK

Ebben a fejezetben összefoglaljuk a legfontosabb típuskonstrukciók algebrai leírását. E specifikációk az alábbi szintaxis szerint szerepelnek:

Szintaktikai „minta”	Magyarázat
TípusKonstrukció(paraméterek):	*A definiálandó típuskonstrukció „alkalmazási sablona” (feje).*
Asszociált műveletek:	*Operációinak szignatúrája (értelmezési tartománya, értékkészlete)*
Operáció(értelmezési tartomány): Értékkészlet …	
Axiómák: Jelölések: • *objektum: Típus(paraméterek)* = *értelmező megjegyzés* • …	*Az operációk kapcsolatát leíró axiómák, és a bennük fölhasznált objektumok „deklarációi”.*
1° … az axióma infomális szövege ... *a formalizált axióma* 2° … …	*Esetleg kibővítve fontos következményállításokkal.*

1.1. Tömb

Típus Tömb(Típus Index,Elem):

Asszociált műveletek:

Új: Tömb

ElemSzám(Tömb): Egész

ElemÉrték(Tömb,Index): Elem ∪ {NemDef}

ElemMódosítás(Tömb,Index,Elem): Tömb

Axiómák:

Jelölések:
- *t: Tömb(Index:Elem)* = tetszőleges tömbtípusú objektum
- *i,j: Index* = tetszőleges indextípusú objektum
- *e: Elem* = tetszőleges elemtípusú objektum

1° Az Új tömb minden eleme olyan értékű, amilyent a bázistípus $Új_e$ művelet létrehoz[23]. Pontosan annyi eleme lesz, amennyit az Index-típus meghatároz.

$ElemÉrték(Új,i)=Új_e \wedge ElemSzám(Új)=Számosság(Index)$

[23] Esetleg: NemDef, ha az Elem típusban nem létezik a konstrukciós $Új_e$ művelet.

2° Egy elemének módosítása nem változtatja meg a tömb hosszát.
ElemSzám(ElemMódosítás(t,i,e))=ElemSzám(t)

Következménye:
ElemSzám(t)=ElemSzám(Új)=konstans [≡Számosság(Index)]

3° A tömb elemének az az értéke, amit az utolsó rávonatkozó ElemMódosítás művelet adott neki.

ElemÉrték(ElemMódosítás(t,i,e),i)=e

4° Az elemek értéküket módosításig megőrzik. (Ez egy szokatlan axióma, amely azt mondja meg, hogy az *ElemMódosítás* művelet mire **nincs** hatással.)

$i \neq j \Rightarrow$ *ElemÉrték(ElemMódosítás(t,i,e),j)=ElemÉrték(t,j)*

Állítás: a tömb létrehozása után az egyes elemei az első ElemMódosítás-ig kezdőértékükkel rendelkeznek.

Biz.: a 3° és 4° folyamánya.

Megjegyzés: az egyetlen kezelési anomáliáról, az indextúllépésről nem kell rendelkezni az axiómákban, hiszen ez az egyik bázistípust érintő sértés lévén már korábban –az Index-típushoz tartozó axiómák megsértése miatt– hibás, azaz nemdefiniált eredményre vezetne.

1.2. Lista

A lista az első példánya azon típuskonstrukcióknak, amelyek a használat során képesek dinamikusan változtatni méretüket. Mivel szándékunk a „realitások talaján maradni” a tervezésnél is, ezért számolunk a memória végességével, vagyis gondolunk arra is, hogy a struktúra használata közben esetleg elfogy a memória. Ennek axiomatikus kézbentartására bevezetjük a `Tele?` függvényt, amely akkor *igaz, ha már újabb elemmel nem bővíthető* a szerkezet.

A ’tele’-séggel kapcsolatos axiómákat tekinthetjük pusztán „technikai axiómáknak” is. (Az irodalom így is tesz: nem ismert olyan publikáció, amely számításba venné e szituációt.) A „teleség” állapota rendelkezik egy axiomatikus „féloldalisággal”. Nem vagyunk képesek leírni, hogy mikor következik be ez az állapot –hiszen a konkrét ábrázolástól is függ–, csak arról szólhatunk, mi teljesüljön a szerkezet állapotára, amikor bekövetkezett ez az állapot.

Az alábbiakban a legáltalánosabb értelemben használt lista fogalmat, a *kétirányú lista* fogalmát fogjuk specifikálni. Ennek köszönhetően igen bő operációkészletet definiálunk hozzá.

Típus Lista(Típus Elem):

Asszociált műveletek:

Üres: Lista	
Üres?(Lista): Logikai	
Tele?(Lista): Logikai	
ElemSzám(Lista): Egész	
ElemÉrték(Lista): Elem ∪ {NemDef}	[az aktuális elem értéke]
ElsőE?(Lista): Logikai ∪ {NemDef}	
Első(Lista): (*Lista* x Elem) ∪ {NemDef}[24]	
Elsőre(Lista): Lista ∪ {NemDef}	[Első(l)=ElemÉrték(Elsőre(l))]
Következő(Lista): (*Lista* x Elem) ∪ {NemDef}	
Következőre(Lista): Lista ∪ {NemDef}	[Következő(l)=ElemÉrték(Következőre(l))]
BeszúrMögé(Lista,Elem): Lista ∪ {NemDef}	
BeszúrElé(Lista,Elem): Lista ∪ {NemDef}	

[24] Magyarázat az első pillantásra meglepő értékkészletre: az Első/Következő/Utolsó/Előző műveletek során megváltozik a *teljes* lista állapota abban az értelemben, hogy mi lesz ettől kezdve az aktuális elem.

Kihagy(Lista): Lista ∪ {NemDef}
EgymásUtán(Lista,Lista): Lista
UtolsóE?(Lista): Logikai ∪ {NemDef}
Utolsó(Lista): (*Lista* x Elem) ∪ {NemDef}
Utolsóra(Lista): Lista ∪ {NemDef} [Utolsó(l)=ElemÉrték(Utolsóra(l))]
Előző(Lista): (*Lista* x Elem) ∪ {NemDef}
Előzőre(Lista): Lista ∪ {NemDef} [Előző(l)=ElemÉrték(Előzőre(l))]

Axiómák:

Jelölések:
- *l,l',l'': Lista(Elem)* = listatípusú objektum
- *e: Elem* = elemtípusú objektum
- *k,m: Egész*

Megállapodunk abban, hogy csak azt axiomatizáljuk, amire az adott művelet *hatással van*, külön nem emeljük ki azon állapotkomponenseket, amit változatlanul kell hagynia. Továbbá most sem vizsgáljuk az axiómarendszerünk minimálisságát.

1° Az Üres lista üres.

$l=Üres \Rightarrow Üres?(l)$

Megjegyzés: az implikáció arra utal, hogy olyan állapotba, amelyre az *Üres?* igaz értéket ad, nem csak az *Üres* operáció hatására kerülhet.

2° Üres listának nincs eleme.

$Üres?(l) \Leftrightarrow ElemSzám(l)=0$

3° Az üres listának nincs aktuális, első, ... utolsó eleme.

$ElemÉrték(Üres)=NemDef \wedge$
$ElsőE?(Üres)=NemDef \wedge Első(Üres)=NemDef \wedge Következő(Üres)=NemDef$
$Előző(Üres)=NemDef \wedge Utolsó(Üres)=NemDef \wedge UtolsóE?(Üres)=NemDef$

4° Nincs elsőt megelőző és nincs utolsót követő eleme a listának.

$Előző(Elsőre(l))=NemDef \wedge Következő(Utolsóra(l))=NemDef$

5° A lista Utolsóra művelettel kiválasztott eleme az ElemSzám-adik (az elsővel kezdve a számozást), az első elemét az Elsőre művelet választja ki.

$UtolsóE?(Utolsóra(l)) \wedge ElsőE?(Elsőre(l)) \wedge_h$
$h=ElemSzám(l)-1 \Rightarrow Következőre_h(Elsőre(l))=Utolsóra(l) \wedge$
$h \in (0,ElemSzám(l)-1) \Rightarrow \neg UtolsóE?(Következőre_h(Elsőre(l))) \wedge$
$\neg ElsőE?(Következőre_h(Elsőre(l)))$

1. Állítás: A listának ElemSzám db eleme van.

Biz.: $5° \Rightarrow$ $\exists$ *legalább ElemSzám(l) db eleme, hisz az utolsó az annyiadik:*
$\forall k \in [0,ElemSzám(l)): \exists ElemÉrték(Következőre_k(Elsőre(l)))$
$4° \Rightarrow$ $\exists$ *legfeljebb ElemSzám(l) db eleme, hisz minden elem az utolsó előtt van (egy sincs mögötte);*
$Következőre_{ElemSzám(l)}(Elsőre(l))=NemDef.$

2. Állítás: $\forall$l listának $\exists$ olyan állapota, amelyben ElemÉrték(l)=NemDef.

Biz.: pl. $\forall l: Következőre_{ElemSzám(l)}(Elsőre(l))=NemDef$, *ami után az* $4°$ *miatt igaz az állítás.*

6° A BeszúrMögé művelet bővíti a listát az adott elemmel. Az új elem az 'aktuális' mögé kerül. Üres listába elsőként kerül be az új elem. Az új válik 'aktuálissá'. Ha nincs 'aktuális' elem, akkor a művelet eredménye nem definiált.

$Tele?(l) \Rightarrow BeszúrMögé(l,e)=NemDef$

$\neg Tele?(l) \Rightarrow$

$f=ElemÉrték(l) \Rightarrow \exists l'=BeszúrMögé(l,e):$
$ElemSzám(l')=ElemSzám(l)+1 \wedge$
$ElemÉrték(l')=e \wedge Előző(l').Elem=f \wedge$

$Üres?(l) \Rightarrow ElemSzám(BeszúrMögé(l,e))=1 \wedge ElemÉrték(BeszúrMögé(l,e))=e \wedge$

$\neg Üres?(l) \wedge ElemÉrték(l)=NemDef \Rightarrow BeszúrMögé(l,e)=NemDef$

A BeszúrElé művelet ennek analógiájára fogalmazható meg.

7° Kihagyni az 'aktuális' elemet lehet, az 'aktuális' a következő lesz. Ha nincs az 'aktuális' elem kijelölve, akkor nem lehet kihagyni sem.

$\exists ElemÉrték(l) \Rightarrow \exists l'=Kihagy(l):$
$ElemSzám(l')=ElemSzám(l)-1 \wedge$
$ElemÉrték(l')=Következő(l).Elem$

$ElemÉrték(l)=NemDef \Rightarrow Kihagy(l)=NemDef$

8° Két egymásután illesztett lista listát alkot. Elemei az eredeti kettő elemei lesznek, az eredeti sorrendben.

$Legyen\ h=ElemSzám(l),\ h'=ElemSzám(l')!$
$l''=EgymásUtán(l,l'):$
$\forall k \in [0,h): ElemÉrték(Következőre^{k}(Elsőre(l'')))=ElemÉrték(Következőre^{k}(Elsőre(l))) \wedge$
$\forall k \in [0,h'): ElemÉrték(Következőre^{k+h}(Elsőre(l')))=ElemÉrték(Következőre^{k}(Elsőre(l')))$

3. Állítás: ElemSzám(EgymásUtán(l,l'))=ElemSzám(l)+ElemSzám(l')

Biz.: az 1. állítás és a 8° következménye.

9° A következő elem az, amelyet a következőre lépés után 'aktuálisként' érzékelünk, az előző az, amelyet az előzőre lépés után aktuálisként érzékelünk, ...

$ElemÉrték(l)\neq NemDef \Rightarrow Következő(l).Elem=ElemÉrték(Következőre(l)) \wedge$
$Előző(l).Elem=ElemÉrték(Előzőre(l)) \wedge$
$Első(l).Elem=ElemÉrték(Elsőre(l)) \wedge$
$Utolsó(l).Elem=ElemÉrték(Utolsóra(l))$

1.3. Verem

Típus Verem(Típus Elem):

Asszociált műveletek:

Üres: Verem

Üres?(Verem): Logikai

Tele?(Verem): Logikai

Tető(Verem): Elem ∪ {NemDef}

Verembe(Verem,Elem): Verem ∪ {NemDef}

Veremből(Verem): (Verem x Elem) ∪ {NemDef}

Axiómák:

Jelölések:

- *v,v': Verem(Elem)* = veremtípusú objektum
- *e,e': Elem* = elemtípusú objektum

1° Az Üres verem üres.

$v=Üres \Rightarrow Üres?(v)$

2° Az a verem, amelyben legalább egy elem van, az nem Üres.

$\neg Tele?(v) \Rightarrow \neg Üres?(Verembe(v,e))$

3° Az üres veremnek nincs legfelső eleme.[25]

$Üres?(v) \Rightarrow Tető(v)=NemDef$

3'° Az üres veremből nem lehet kivenni elemet.

$Üres?(v) \Rightarrow Veremből(v)=NemDef$

3"° A tele verembe nem lehet további elemet betenni.

$Tele?(v) \Rightarrow Verembe(v,e)=NemDef$

4° A verem legfelső eleme az utoljára betett eleme.

$\neg Tele?(v) \Rightarrow Tető(Verembe(v,e))=e$

5° A veremből a legfelső elemet lehet kivenni (a többi elem nem változik).

$\neg Tele?(v) \Rightarrow Veremből(Verembe(v,e))=(v,e)$
$\neg Üres?(v) \Rightarrow Verembe(Veremből(v))=v$

Állítás: A verem LIFO (Last In First Out, vagyis az Utolsóként Betett kerül Elsőként Ki) szerkezet.

Biz.: (strukturális indukció: indukció a verem mélységére)

1. Üres veremre igaz az 5° axióma miatt:
$Veremből(Verembe(Üres,e))=(Üres,e) \wedge$
$3° \Rightarrow \neg\exists(v',e'): Veremből(Üres)=(v',e')$

2. (indukciós lépés)
Tfh.: v LIFO n-elem esetén, azaz:
$v_1=Verembe(Üres,e_1) \wedge v_2=Verembe(v_1,e_2) \wedge \ldots\ v_n=Verembe(v_{n-1},e_n) \Rightarrow$
$Veremből(v_n)=(v_{n-1},e_n) \wedge \ldots\ Veremből(v_2)=(v_1,e_2) \wedge Veremből(v_1)=(Üres,e_1)$
az n+1 eleműre az 5° axióma miatt:
$Veremből(Verembe(v_n,e_{n+1}))=(v_n,e_{n+1}) \wedge$
v_n *(indukciós feltétel miatt) n-elemű LIFO.*

Megjegyzés: a Verembe és a Veremből műveleteknél a verem többi elemét vizsgálni azért nem kell, mert az 5° axióma biztosítja a többi meg nem változását. (A Tető függvény eleve szóba se jöhet értékkészlete miatt.)

1.4. Duplaverem

A lényeges különbség az előzőhöz képest, hogy az operációknál azt is rögzíteniük kell, hogy melyik a kettő közül, amelyikre vonatkozik. A kettő közüli választást a {Bal,Jobb} halmazon „nyugvó” *Melyik* típus bevezetésével oldjuk meg.

Ennek bevezetése mögött az a ki nem mondott szándék húzódik, hogy a megvalósításnál olyan folytonos ábrázolást választunk, amelynél a „párosság” ténye hatékonyan fölhasználható.

Típus DuplaVerem(Típus Elem):

Asszociált műveletek:

Üres: DuplaVerem

[25] Az állítás megfordítása is igaz. Azaz az implikáció helyettesíthető ekvivalenciával is.

Üres?(Melyik,DuplaVerem): Logikai
Tele?(DuplaVerem): Logikai

Tető(Melyik,DuplaVerem): Elem ∪ {NemDef}

Verembe(Melyik,DuplaVerem,Elem): DuplaVerem ∪ {NemDef}

Veremből(Melyik,DuplaVerem): (DuplaVerem x Elem) ∪ {NemDef}

Axiómák:

Jelölések:

- *v,v': DuplaVerem(Elem)* = duplaverem-típusú objektum
- *e,e': Elem* = elemtípusú objektum
- *m: Melyik=(Bal,Jobb)* = absztrakt felsorolástípusú objektum

1° Az Üres duplaverem üres.

$v=Üres \Rightarrow Üres?(m,v)$

2° Az a duplaverem, amelyben legalább egy elem van, az nem Üres.

$\neg Tele?(v) \Rightarrow \neg Üres?(m,Verembe(m,v,e))$

3° Az üres duplaveremnek nincs legfelső eleme.

$Üres?(v) \Rightarrow Tető(m,v)=NemDef$

3'° Az üres duplaveremből nem lehet kivenni elemet.

$Üres?(v) \Rightarrow Veremből(m,v)=NemDef$

3''° A tele duplaverembe nem lehet további elemet betenni.

$Tele?(v) \Rightarrow Verembe(m,v,e)=NemDef$

4° A duplaverem legfelső eleme az utoljára betett eleme.

$\neg Tele?(v) \Rightarrow Tető(m,Verembe(m,v,e))=e$

5° A duplaveremből a legfelső elemet lehet kivenni (a többi elem nem változik).

$\neg Tele?(v) \Rightarrow Veremből(m,Verembe(m,v,e))=(v,e)$
$\neg Üres?(m,v) \Rightarrow Verembe(m,Veremből(m,v))=v$

Megjegyzés: a Verembe és a Veremből műveleteknél a verem többi elemét vizsgálni azért nem kell, mert az 5° axióma biztosítja a többi meg nem változását. (A Tető függvény eleve szóba se jöhet értékkészlete miatt.)

1.5. Sor

Típus Sor(Típus Elem):

Asszociált műveletek:

Üres: Sor

Üres?(Sor): Logikai

Tele?(Sor): Logikai

Első(Sor): Elem ∪ {NemDef}

Sorba(Sor,Elem): Sor ∪ {NemDef}

Sorból(Sor): (Sor x Elem) ∪ {NemDef}

Axiómák:

Jelölések:

- *s,s': Sor(Elem)* = sortípusú objektum
- *e,e': Elem* = elemtípusú objektum

1° Az Üres sor üres.

$s=Üres \Rightarrow Üres?(s)$

2° Az a sor, amelyben legalább egy elem van, az nem üres.

$\neg Tele?(s) \Rightarrow \neg Üres?(Sorba(s,e))$

3° Az üres sornak nincs első eleme.

$Üres?(s) \Rightarrow Első(s)=NemDef$

3'° Az üres sorból nem lehet kivenni elemet.

$Üres?(s) \Rightarrow Sorból(s)=NemDef$

3"° A tele sorba nem lehet további elemet betenni.

$Tele?(s) \Rightarrow Sorba(s,e)=NemDef$

4° A sor első eleme a legrégebben betett elem.

$Üres?(s) \wedge \neg Tele?(s)$[26] $\Rightarrow Első(Sorba(s,e))=e \wedge$
$\neg Üres?(s) \wedge \neg Tele?(s) \Rightarrow Első(Sorba(s,e))=Első(s)$

5° A sorból az első elemet lehet kivenni, s a többi nem változik.

$Üres?(s) \wedge \neg Tele?(s) \Rightarrow Sorból(Sorba(s,e))=(s,e)$
$\neg Üres?(s) \wedge \neg Tele?(s) \Rightarrow Sorból(s)=(s',e) \wedge e=Első(s) \wedge$
$Sorba(s',e')=Sorból(Sorba(s,e')).$Sor

Állítás: A sor FIFO (First In First Out, vagyis az Elsőként Betett kerül Elsőként Ki) szerkezet.

Biz.: *(Strukturális indukció a sor hosszára)*

1. Üres sorra az 5° első része miatt teljesül az állítás.

2. (indukciós lépés)

Tfh.: s FIFO n elem esetén, azaz:
$s_1=Sorba(Üres,e_1) \wedge s_2=Sorba(s_1,e_2) \wedge \ldots\ s_n=Sorba(s_{n-1},e_n) \Rightarrow$
$Sorból(s_n)=(s'_{n-1},e_1) \wedge Sorból(s'_{n-1})=(s'_{n-2},e_2) \wedge \ldots Sorból(s'_1)=(Üres,e_n)$
az n+1 eleműre az 5° és a 4° axiómák második része miatt:
$Sorból(Sorba(s_n,e_{n+1}))=(s'_n,e_1) \wedge$
s_n *(indukciós feltétel miatt) n-elemű FIFO.*

Megjegyzés: a Sorba és a Sorból műveleteknél a sor többi elemét vizsgálni azért nem kell, mert az 5° axióma biztosítja a többi meg nem változását. (Az Első függvény eleve szóba se jöhet értékkészlete miatt.)

1.6. Kétvégű sor (Deck)

Ezt most nem specifikáljuk az előzővel való nagyfokú hasonlatossága miatt.

1.7. Prioritási sor

A prioritási sor „újdonsága” a prioritás fogalmában található. *Prioritás* nem más, mint egy sajátos „rendezése” a szerkezetet alkotó elemeknek, vagyis

Infix Operátor ≤(Prioritás,Prioritás): Logikai

[26] Ez a furcsaság azt fejezi ki, hogy a memória akkor is elfogyhat, amikor az adott sor első elemét igyekeznénk beletenni.

A prioritási sornak nemcsak a neve árulkodik a sorral való rokonságról, hanem majdhogynem tényleges „leszármazottja” is. Ez derül ki az alábbi –sorra való– visszavezetésből:

PrElem = Elem x Prioritás

PrSor[27] = Sor(PrElem)

A visszavezetés ily leegyszerűsítése sajnos nem teljesen igaz, a prioritási sor működését csak részben írja le jól, ezért kényszerülünk önállóan is axiomatizálni.[28] Annyi mindenestre látszik, hogy műveleteinek neve azonos a soréval. (Polimorfia)

Típus PrSor(Típus Elem,Prioritás):

Asszociált műveletek:

Üres: PrSor

Üres?(PrSor): Logikai

Tele?(PrSor): Logikai

Első(PrSor): (Elem x Prioritás) ∪ {NemDef}

Sorba(PrSor,Elem,Prioritás): PrSor ∪ {NemDef}

Sorból(PrSor): (PrSor x Elem x Prioritás) ∪ {NemDef}

Axiómák:

Jelölések:

- *s,s': Pr(Elem)* = prioritási sortípusú objektum
- *e,e': Elem* = elemtípusú objektum
- *p,p': Prioritás* = prioritás (rendezett típus)

1° Az Üres prioritási sor üres.

$s=Üres \Rightarrow Üres?(s)$

2° Az a prioritási sor, amelyben legalább egy elem van, az nem Üres.

$\neg\, Tele?(s) \Rightarrow \neg\, Üres?(Sorba(s,e,p))$

3° Az üres prioritási sornak nincs első eleme.

$Üres?(s) \Rightarrow Első(s)=NemDef$

3'° Az üres prioritási sorból nem lehet kivenni elemet.

$Üres?(s) \Rightarrow Sorból(s)=NemDef$

3''° A tele prioritási sorba nem lehet további elemet betenni.

$Tele?(s) \Rightarrow Sorba(s,e,p)=NemDef$

4° A prioritási sor első eleme a legnagyobb prioritásúak közül legrégebben betett elem.

$Üres?(s) \wedge \neg\, Tele?(s) \Rightarrow Első(Sorba(s,e,p))=(e,p) \wedge$

$\neg\, Üres?(s) \wedge \neg\, Tele?(s) \Rightarrow Első(Sorba(s,e,p)) = \begin{cases} Első(s) & Első(s).prioritás \geq p \\ (e,p) & Első(s).prioritás < p \end{cases}$

5° A prioritási sorból az első elemet lehet kivenni, s a többi nem változik.

[27] A rövidítés kedvéért a PrioritásiSor helyett PrSor-t fogunk írni.

[28] Elgondolkodtató, hogy miként lehetne az axiómákban is fölhasználni egy már axiomatizált „rokon” (ős) axiómáit. A sor–prioritási sor esetében alighanem a fordított út járható: a prioritási sor az ős (általánosabb volta miatt), a sor a leszármazott. Mondhatjuk, a sor azonos prioritású elemek prioritási sora.

Üres?(s) ∧ ¬ *Tele?(s)* ⇒ *Sorból(Sorba(s,e,p))=(s,e,p)*
¬ *Üres?(s)* ∧ ¬ *Tele?(s)* ⇒ *Sorból(s)=(s',e',p')* ∧ *(e',p')=Első(s)* ∧
Első(s).prioritás<p ⇒ *s=Sorból(Sorba(s,e,p)).PrSor* ∧
Első(s).prioritás≥p ⇒ *Sorba(s',e,p)=Sorból(Sorba(s,e,p)).PrSor*

Megjegyzés: a Sorba és a Sorból műveleteknél a sor többi elemét vizsgálni azért nem kell, mert az 5° axióma biztosítja a többi meg nem változását. (Az Első függvény eleve szóba se jöhet értékkészlete miatt.)

1.8. Táblázat

A táblázat a tömb egy általánosítása, amely definiálásánál célunk a tömb indextípusára rótt kemény feltételek (diszkrétség[29]) enyhítése, vagyis, hogy lehetővé tegyük *tetszőleges típus* ilyen célú használatát. Ehhez természetesen többlet információra van szükség a típusmegvalósításhoz. Nevezetesen: kell egy *kulcs-függvény* (*KulcsFv*), amely minden elemhez hozzárendeli azt az értéket, ami az ő egyedi indexeként használható. A táblázatot kiegészítjük olyan műveletekkel is, amelyek „csak" egy tipikus feladathoz, a struktúra bejáráshoz szükséges, helyesebben: „praktikusan" nélkülözhetetlen. Ezen műveletegyüttes egy új, a táblázat koncepciójától talán idegen fogalmat, az *aktuális elem* fogalmát igényli.

Típus Táblázat(Konstans Méret:NemNegEgész, Függvény KulcsFv(Elem):Kulcs, Típus Elem,Kulcs):

Asszociált műveletek:

Üres: Táblázat

Tele?(Táblázat): Logikai

ElemSzám(Táblázat): NemNegEgész

Beilleszt(Táblázat,Elem): Táblázat ∪ {NemDef}

Töröl(Táblázat,Kulcs): Táblázat ∪ {NemDef}

Keres(Táblázat,Kulcs): Elem ∪ {NemDef}

Első(Táblázat): (Táblázat x Elem) ∪ {NemDef} [a bejárás feladathoz]

Utolsó(Táblázat): (Táblázat x Elem) ∪ {NemDef} [a bejárás feladathoz]

Következő(Táblázat): (Táblázat x Elem) ∪ {NemDef} [a bejárás feladathoz]

Előző(Táblázat): (Táblázat x Elem) ∪ {NemDef} [a bejárás feladathoz]

Axiómák:

Jelölések:
- *t,t': Táblázat(Elem)* = táblázattípusú objektum
- *e,e': Elem* = elemtípusú objektum
- *k,k': Kulcs* = kulcstípusú objektum
- *KulcsFv: Elem → Kulcs* = kulcsfüggvény

1° Az üres táblázat minden eleme nemdefiniált értékű, és elemszáma 0; aktuális eleme nincs.

t=Üres ⇒ *ElemSzám(t)=0* ∧ *Keres(t,k)=NemDef* ∧ *Töröl(t,k)=NemDef* ∧
Első(t)=NemDef ∧ *Utolsó(t)=NemDef* ∧ ...

Megjegyzés: Nem képzelhető el a nemdefiniált érték kikerülése az elemtípusra vonatkozó *Új* művelettel létrehozott opcionális kezdőértékkel, mivel –mint a következő axióma kimondja– több azonos érték a táblázatban nem megengedett.

[29] Diszkrét, azaz *véges* és létezik egy *egyértelmű rákövetkezési reláció*.

2° Ha van még hely és még nincs ilyen kulcsú elem, akkor beszúrható a táblázatba, és a méret eggyel nő. A beszúrt követő lesz az aktuális – ha van ilyen.

$Kulcs(e)=Kulcs(e') \Rightarrow Beilleszt(Beilleszt(t,e),e')=NemDef$

$ElemSzám(t)<Méret \wedge \exists Beilleszt(t,e)=t' \Rightarrow ElemSzám(t')=ElemSzám(t)+1 \wedge$
$Előző(t')\neq NemDef \Rightarrow Előző(t').elem=e$

3° Létező (kulcsú) elem törölhető, és az elemszám eggyel csökken. Az aktuális a töröltet követő lesz – ha van ilyen; nem létező törlése nemdefiniált.

$Keres(t,k)=e \Rightarrow Kulcs(e)=k \wedge Töröl(t,k)=t' \wedge ElemSzám(t')=ElemSzám(t)-1 \wedge$
$Előző(t')\neq NemDef \Rightarrow Előző(Keres(t,k).táblázat).elem=Előző(t').elem \wedge$
$Töröl(t',k)=NemDef$

$Keres(t,k)=NemDef \Rightarrow Töröl(t,k)=NemDef$

4° Ha van keresett kulcsú, akkor az visszakereshető; a megtaláltat követő lesz az aktuális – ha van ilyen.

$\exists Beilleszt(t,e)=t' \Rightarrow Keres(t',Kulcs(e))=e \wedge$
$Előző(t')\neq NemDef \Rightarrow Előző(t').elem=e$

5° A táblázat elemei fölsorolhatók.

$N=ElemSzám(t) \wedge t=(t_1,t_2,...,t_N) \Rightarrow Első(t).elem=t_1 \wedge Utolsó(t).elem=t_N \wedge$
$\forall i \in [1,N): Következő^{(i)}(...Következő^{(1)}(Első(t).táblázat).táblázat...).elem=t_{i+1} \wedge$
$Előző^{(i)}(...Előző^{(1)}(Utolsó(t).táblázat).táblázat...).elem=t_{N-i}$

Állítás: A Következő és az Előző műveletek egymás inverzei, az első elemet nem előz meg elem, az utolsót nem követ.

Bizonyítás: *... az 5°-ből következik ...*

Megjegyzések: 1) Meggondolandó, hogy a pusztán bejárás céljára definiált Következő ... műveletek kiküszöbölhetők, ha megengedjük a *tömbbé* (és vissza-) *konvertálást.*

2) Egy másik táblázat elképzelés is megfogalmazható, amelynél a méret nem deklaratíve rögzítendő, hanem az Üres eljárás paramétereként adandó meg. Hogy nem ezt a változatot tekintettük elsődlegesnek, annak az az oka, hogy a tömb indexének általánosításából a deklaratív megadás tűnik kézenfekvőnek. A tömbbel ellentétben az erősen típusosság sem akadályozza meg az index szerepébe lépő Méret és KulcsFv kettős legalább részbeni dinamikus megadását.

1.9. Input Szekvenciális File

Az Input Szekvenciális File csak az első a **file**-típuskonstrukciók közül. Ide kívánkozik néhány bevezető gondolatot.

- Az egyes file-kategóriák halmazai (*InpSzekvFile*, *OutSzekvFile, DirektFile*...) a már nyitott, s az adott kategóriába tartozó file-okat egyesítik.
- A *File* tetszőleges (adathordozón létrehozott vagy még nem létező, a vizsgált kategóriába tartozó) file-ok halmaza. A *LétFile* az *adathordozón* már létező, a *NemLétFile* a még nem létező file-okat gyűjtik össze. Nyilvánvaló elvárásunk, hogy: File=LétFile∪NemLétFile, LétFile∩NemLétFile=∅.

  Ezen fogalmak bevezetését az indokolja, hogy a file-típuskonstrukcióknál lényegesen hangsúlyosabb szerep jut a nyitás és zárás tavékenységeknek. A nyit hatására kerül át a mindenféle file-t magába foglalóból (LétFile vagy NemLétFile) az adott típusúba, a zárás során fordítva.
- $f:LétFile \Rightarrow f=() \vee f=(f_0,...,f_n) \wedge n\geq 0$
- A *NemDef* most is egy előredefiniált, típusfüggetlen konstans.
- Az *Index* halmaz most a pozitív egészek halmaza lesz.
- Az **értelmezési tartomány fontos**, ui. azt nem axiomatizáljuk, hogy mi van akkor, ha nem az értelmezési tartományba tartozó file-ra alkalmazzák az adott műveletet, ekkor természetesen NemDef az eredmény.

Típus InpSzekvFile(Típus Elem):

Asszociált műveletek:

Nyit(File): InpSzekvFile ∪ {NemDef}

Zár(InpSzekvFile): File \ InpSzekvFile

Olvas(InpSzekvFile): (InpSzekvFile x Elem) ∪ {NemDef}

Vége?(InpSzekvFile): Logikai

Axiómák:

Jelölések:
- *f,g: InpSzekvFile(Elem)* = input szekvenciális file-típusú objektum
- *e: Elem* = elemtípusú objektum

1° Nyitni csak korábban létrehozott és nem nyitott file-t lehet.

$f \in LétFile \wedge f \notin InpSzekvFile \Rightarrow \exists Nyit(f)$ [30]
$f \notin LétFile \vee f \in InpSzekvFile \Rightarrow Nyit(f)=NemDef$

2° Megnyitás után az első elem lesz az aktuális (amire a legközelebbi olvasás művelettel hivatkozhatunk), vagy a file végénél vagyunk (nem tartalmaz egyetlen elemet sem).

$g=Nyit(f) \Rightarrow \neg Vége?(g) \wedge Olvas(g)=(g,g_0) \vee Vége?(g) \wedge g=()$

3° A vége után nem lehet olvasni.

$Vége?(f) \Rightarrow Olvas(f)=NemDef$

4° Olvasás után az aktuális elem a következő lesz, vagy a végére értünk.

$Olvas(f)=(g,f_i) \wedge g=(f_0,f_1,...,f_i,...,f_N) \Rightarrow f_i \neq NemDef \wedge$
$\neg Vége?(g) \wedge Olvas(g)=(g,f_{i+1}) \vee Vége?(g) \wedge g=(f_0,f_1,...,f_i)$

1.10. Output Szekvenciális File

Típus OutSzekvFile(Típus Elem):

Asszociált műveletek:

Nyit(File): OutSzekvFile ∪ {NemDef}

Zár(OutSzekvFile): File \ OutSzekvFile

Ír(OutSzekvFile): (OutSzekvFile x Elem)

Axiómák:

Jelölések:
- *f,g: OutSzekvFile(Elem)* = Output szekvenciális file-típusú objektum
- *e: Elem* = elemtípusú objektum

1° Nyitni csak még nem létező file-t lehet.

$f \notin LétFile \Rightarrow \exists Nyit(f)$ [31]

[30] azaz $Nyit(f) \in InpSzekvFile$

2° A file csak a lezárással válik (az adathordozón) létezővé.
$f \in OutSzekvFile \Rightarrow f \notin LétFile \wedge Lezár(f) \in LétFile$

3° Megnyitás után az első elem lesz aktuális.

$g=Nyit(f) \Rightarrow Ír(g,e)=(e)$

4° Írás után az aktuális elem a következő lesz.

$f \in OutSzekvFile \wedge f=(f_0,f_1,...,f_N) \wedge Ír(f)=(g,e)=(f_0,f_1,...,f_N,e)$

Állítás: ugyanaz a file többször is megnyitható. A korábbi állapota elvész!

Biz.: 1°, 2° és 3° következménye.

1.11. Direkt File

Típus DirektFile(Típus Elem):

Asszociált műveletek:

Nyit(File): DirektFile ∪ {NemDef}

Létrehoz(DirektFile): DirektFile ∪ {NemDef}

Zár(DirektFile): File \ DirektFile

Ír(DirektFile,Index,Elem): DirektFile ∪ {NemDef}

Olvas(DirektFile,Index): Elem ∪ {NemDef}

Hossz(DirektFile): NemNegEgész

Szűkít(DirektFile): DirektFile

Bővít(DirektFile,Elem): DirektFile

Axiómák:

Jelölések:
- *f,g: DirektFile(Elem)* = direkt file-típusú objektum
- *e: Elem* = elemtípusú objektum

1° Nyitni csak létező file-t lehet. Nyitás után a file hossza az elemek számával egyezik meg.

$f \in LétFile \Rightarrow g=Nyit(f) \wedge g=(f_0,f_1,...,f_N) \wedge Hossz(g)=N+1$
$f \notin LétFile \Rightarrow Nyit(f)=NemDef$

2° Létrehozni csak még nem létező file-t lehet. Létrehozás után a file hossza 0.

$f \notin LétFile \Rightarrow g=Létrehoz(f) \wedge g=() \wedge Hossz(g)=0$
$f \in LétFile \Rightarrow Létrehoz(f)=NemDef$

3° Nyitás után bármely (hozzátartozó) eleme elérhető, s értéke az, amit a legutóbbi módosítás során kapott, vagy amivel a nyitáskor rendelkezett. (Kizáró vagy.)

$g=Nyit(f) \wedge i,j \in [0,Hossz(f)) \wedge j \neq i \Rightarrow Olvas(Ír(f,i,e),j)=Olvas(f,j) \wedge Olvas(Ír(f,i,e),i)=e$
$g=Nyit(f) \wedge i \notin [0,Hossz(f)) \Rightarrow Olvas(f,i)=NemDef \wedge Ír(f,i,e)=NemDef$

[31] $Nyit(f) \in OutSzekvFile$

4° Szűkítéskor az utolsó elemmel csökken a file.

$Hossz(f)>0 \Rightarrow g=Szűkít(f) \wedge Hossz(g)=Hossz(f)-1 \wedge$
$\forall i \in [0,Hossz(g)): Olvas(f,i)=Olvas(g,i)$
$Hossz(f)=0 \Rightarrow Szűkít(f)=NemDef$

5° Bővítéskor az utolsó elemmel növekszik a file.

$Hossz(f)>0 \Rightarrow g=Bővít(f,e) \wedge Hossz(g)=Hossz(f)+1 \wedge Olvas(f,Hossz(g))=e$
$\forall i \in [1,Hossz(f)]: Olvas(f,i)=Olvas(g,i)$

6° Minden elemmódosító műveletnek rögvest a háttértárolón is „látszik a hatása”.

$f \in DirektFile \Rightarrow f \in LétFile$

Megjegyzés: A direkt file-ra többfajta elképzelés is forgalomban van. A lényegük a következő:

1. A direkt file potenciálisan végtelen számú elemből áll, kifejezetten file-bővítő művelet nincs. Tetszőleges indexű elemének érték adható. A legnagyobb olyan elemig tart a file, amely értékét már definiáltuk. Az utolsó elemet megelőző, de értéket nem kapott vagy törölt elemek értéke nem definiált.
2. A direkt file nagyobb „darabokban” bővíthető, vagy szűkíthető. Az effajta műveletek a file utolsó valahány –paraméterrel meghatározott számú– elemét érintik. A bővítéskori új elemek értéke (a) vagy nem definiált –hasonlóan az analógia alapjául szolgáló tömbhöz–, (b) vagy szintén paraméterrel meghatározott kezdő érték.

1.12. Asszociatív File és Index-Szekvenciális File

Ennek a file-fajtának a kulcs-gondolata, hogy a direkt file indextípusára rótt szigorú követelményt, hogy nemnegatív egész legyen, enyhitsük, lehetővé tegyük tetszőleges típus ilyen célú használatát. Ehhez természetesen több információra van szükség a típusmegvalósításhoz, mint a direkt file esetében. Kell egy *kulcs-függvény*, amely minden elemhez hozzárendeli azt az értéket, ami az ő egyedi indexeként használható, továbbá – ha szekvenciális hozzáférést is biztosítunk, akkor– egy *rendezési relációt* definiáló operáció is.

Típus AsszocFile(Függvény KulcsFv(Elem):Kulcs, Típus Elem,Kulcs):

Asszociált műveletek:

Nyit(File): AsszocFile ∪ {NemDef}

Létrehoz(AsszocFile,NemNegEgész): AsszocFile ∪ {NemDef} [adott méretű file létrehozása]

Zár(AsszocFile): File \ AsszocFile

ElemSzám(AsszocFile): NemNegEgész [a definiált értékű elemek száma]

Méret(AsszocFile): NemNegEgész

Olvas(AsszocFile,Kulcs): Elem ∪ {NemDef}

Ír(AsszocFile,Elem): AsszocFile ∪ {NemDef}

Töröl(AsszocFile,Kulcs): AsszocFile ∪ {NemDef}

Axiómák:

Jelölések:

- *f,g: AsszocFile(Elem)* = asszociatív file-típusú objektum
- *e: Elem* = elemtípusú objektum
- *db: NemNegEgész* = hosszúság
- *k: Kulcs* = kulcstípusú objektum

1° Nyitni csak létező file-t lehet. Nyitás után a file elemeinek száma egyenlő a definiált elemek számával.

$f \in LétFile \Rightarrow \exists g=Nyit(f) \wedge g=(f_0,f_1,...,f_N) \wedge Méret(g)=N+1 \wedge Méret(g) \geq ElemSzám(f)$
$f \notin LétFile \Leftrightarrow Nyit(f)=NemDef$

2° Létrehozni csak még nem létező file-t lehet. Létrehozás után a file mérete a méretparaméterrel egyező, elemszáma 0.

$f \notin LétFile \Rightarrow \exists g=Létrehoz(f,db) \wedge Méret(g)=db \wedge \forall i \in [0,db): g_i=NemDef \wedge ElemSzám(g)=0$
$f \in LétFile \Rightarrow Létrehoz(f,db)=NemDef$

3° (Nyitás után) bármely eleme elérhető, s értéke az, amit a legutóbbi módosítás során kapott, vagy amivel a nyitáskor rendelkezett. (Kizáró vagy.)

$Olvas(Ír(f,e),Kulcs(e))=e \wedge$
$(f=(f_0,f_1,...,f_N) \wedge k: \forall i \in [0,N]: Kulcs(f_i) \neq k \Rightarrow Olvas(f,k)=NemDef)$

4° Nem létező kulcsú elem írásakor az elemszám eggyel nő, feltéve, hogy nem telt be a file.

$f=(f_0,f_1,...,f_N) \wedge \forall i \in [0,N]: Kulcs(f_i) \neq Kulcs(e) \wedge ElemSzám(f) < Méret(f) \Rightarrow$
$ElemSzám(Ír(f,e))=ElemSzám(f)+1$
$ElemSzám(f)=Méret(f) \Rightarrow Ír(f,e)=NemDef$

5° Adott kulcsú elem módosítása más elem értékét és az elemszámot nem változtatja meg.

$Kulcs(e) \neq k \Rightarrow Olvas(Ír(f,e),k)=Olvas(f,k) \wedge ElemSzám(Ír(f,e))=ElemSzám(f)$

6° Törléskor az adott elem nem definiált értékűvé válik és az elemszám eggyel csökken, nem létező kulcsú elem törlése nem definiált.

$f=(f_0,f_1,...,f_N) \wedge Kulcs(f_i)=k \Rightarrow g=Töröl(f,k) \wedge ElemSzám(g)=ElemSzám(f)-1 \wedge Olvas(g,k)=NemDef$
$f=(f_0,f_1,...,f_N) \wedge k: \forall i \in [0,N]: Kulcs(f_i) \neq k \Rightarrow Töröl(f,k)=NemDef$

Megjegyzés: Meggondolandó, hogy megengedjük az asszociatív file konvertálását akár egy más (nem kisebb) méretű asszociatív file-lá, vagy egy azonos típusú elemekből álló direkt file-lá (és vissza).

Az alábbiakban az indexelt (szekvenciális) file-nak csak az asszociatív file-tól eltérő tulajdonságait írjuk le. A megváltozott axiómákat újraírjuk (azonos sorszámmal), a bővülteket aposztrof jellel kapcsoljuk a korábbihoz. Lényeges differencia a szekvenciálisan fölsorolhatóság, és a dinamikusan bővíthetőség. Sejthető, hogy amíg az előbbi reprezentációja praktikusan egy háttértáron tárolt táblázat lesz, addig itt egy segéd (index-) file-lal „bővített" file.

Mivel a szekvencialitás megkívánja a „hol is tarunk"-ra emlékezést, ezért a kulcsos Olvasáshoz művelet értékkészéletéhez is hozzátartozik maga a teljes struktúra is.

Típus IndexeltFile(Függvény KulcsFv(Elem):Kulcs, RendRel(Elem,Elem):Logikai, Típus Elem,Kulcs):

Asszociált műveletek:

Létrehoz(IndexeltFile): IndexeltFile $\cup$ {NemDef}

Olvas(IndexeltFile,Kulcs): (IndexeltFile x Elem) $\cup$ {NemDef}

Olvas(IndexeltFile): (IndexeltFile x Elem) $\cup$ {NemDef}

Vége?(IndexeltFile): Logikai

Axiómák:

Jelölések:

- *f,g: IndexeltFile(Elem)* = indexelt file-típusú objektum
- *e: Elem* = elemtípusú objektum
- *k: Kulcs* = kulcstípusú objektum

- *γ: Elem→PozEgész* = „rendező” függvény, amely fölsorolja az elemeket kulcs-szerinti rendezésben, vagyis az alábbi tulajdonsággal bír:

1. ha f=(f_0,f_1,...,f_N), akkor ∀i∈[0,N]: γ(f_i)∈[0,N]

2. ∀i,j∈[1,N]: f=(...,f_i,...,f_j,...) esetén Kulcs(f_i)<Kulcs(f_j) ⇔ γ(f_i)<γ(f_j)

2° Létrehozni csak még nem létező file-t lehet. Létrehozás után a file mérete 0.

f∉LétFile ⇒ g=Létrehoz(f) ∧ Méret(g)=0
f∈LétFile ⇒ Létrehoz(f)=NemDef

3'° Nyitás után szekvenciális olvasással a rendezés szerinti első elemet lehet elérni, ha van egyáltalán eleme.

g=Nyit(f) ∧ ¬Vége?(g) ∧ ∃g_i≠NemDef ⇒ Olvas(g).elem=e ∧ γ(e)=0

3"° Sikeres szekvenciális olvasás, kulcsos olvasás, ... után az érintett elemet rendezés szerinti követő elemet lehet szekvenciális olvasással elérni, vagy ha ilyen nincs, akkor az olvasás eredménye nem definiált.

∃(g,e)=Olvas(f) ⇒ ∃g_i=Olvas(g).elem ∧ γ(e)=γ(g_i)-1 ∨
∀g_i:γ(e)≥γ(g_i) ∧ Vége?(g) ∧ Olvas(g).elem=NemDef

(ehhez hasonlóan fogalmazható meg a többi művelet utáni állapot)

1.13. Rekurzió

A rekurzió mint típuskonstrukciós eszköz szokatlan az irodalomba. [PSzZs1]

A jelölések terén is be kell vezetni néhány újdonságot:

1. A definícióban szereplő típusnévre hivatkozás: *Rekurzió* névvel történik.

2. *Szelektor* mint paraméter: az implicite definiált szelektorsorozat valamely eleme. A szelektorsorozatot a fejsor definiálja a *Szelektor* kulcs-szó után írt azonosítókkal. (A szelektort lásd még később, a rekordoknál.)

Típus Rekurzió(Szelektor szelektor, Típus Elem,
Szelektor szelektor1, Típus Rekurzió,
...):

Asszociált műveletek:

Üres: Rekurzió	[üres struktúra]
Üres?(Rekurzió): Logikai	[üres-e a struktúra]
Létrehoz(Elem): Rekurzió	[egy elemű struktúra]
Gyökér(Rekurzió): Elem ∪ {NemDef}	[gyökérelem értéke]
Rész(Rekurzió,Szelektor): Rekurzió ∪ {NemDef}	[szelektorral kiválasztott rész-struktúra]
Illeszt(Rekurzió,Szelektor,Rekurzió): Rekurzió ∪ {NemDef}	[szelektorral hozzáilleszt]
Módosít(Rekurzió,Elem): Rekurzió	[gyökérelem értékét módosítja]

Axiómák:

Jelölések:
- *r,q,p: Rekurzió(Elem...)* = rekurzió-típusú objektum
- *e: Elem* = elemtípusú objektum
- *s: Szelektor* = szelektor objektum

1° Az üres rekurzió üres, nincs gyökér-értéke, nem lehet módosítani, üres bármely rész-struktúrája.

Üres=r ⇒ Üres?(r)
Üres?(r) ⇒ Gyökér(r)=NemDef ∧ Módosít(r,e)=NemDef ∧ Üres?(Rész(r,s))

2° Létre lehet hozni közvetlenül egy-elemű struktúrát, amelynek üresek a részei és gyökérértéke az adott elem.
r=Létrehoz(e) ⇒ Gyökér(r)=e ∧ Üres?(Rész(r,s))

3° Adott szelektorral kiválasztott üres rész-struktúra helyére illeszthető struktúra, nem üresre nem.

Üres?(Rész(r,s)) ⇒ p=Illeszt(r,s,q) ∧ Gyökér(r)=Gyökér(p) ∧ Rész(p,s)=q

4° Módosítás után a gyökérérték az új érték lesz.

¬ Üres?(r) ⇒ q=Módosít(r,e) ∧ Gyökér(q)=e

1.14. Absztrakt Sorozat

Ez a típuskonstrukciós eszköz a jólismert Szöveg (String) típus általánosítás. Így nem okoz semmi meglepetést a hozzá asszociált operációk halmaza. [Sz1,PSzZs2]

Típus AbSo(Típus Elem):

Asszociált műveletek:

Üres: AbSo

Hossz(AbSo): NemNegEgész

BalRész(AbSo,NemNegEgész): AbSo ∪ {NemDef}

JobbRész(AbSo,NemNegEgész): AbSo ∪ {NemDef}

ElemÉrték(AbSo,NemNegEgész): AbSo ∪ {NemDef}

Egyesít(AbSo,AbSo): AbSo

Axiómák:

Jelölések:
- *a,b,c: AbSo(Elem)* = absztrakt sorozat-típusú objektum
- *e: Elem* = elemtípusú objektum
- *i: PozEgész* = index objektum

1° Az üres absztrakt sorozat üres, hossza 0, bármely részsorozata üres, elemelérés nemdefiniált.

Üres=a ⇒ Üres?(a)
Üres?(a) ⇒ Üres?(BalRész(a,i)) ∧ Üres?(JobbRész(a,i)) ∧ ElemÉrték(a,i)=NemDef ∧ Hossz(a)=0

2° Az absztrakt sorozat rész-struktúrái az elemértékek egyesítésével fölépíthetők.

¬ Üres?(a) ∧ i∈[1,Hossz(a)] ⇒ Egyesít(BalRész(1,i-1),ElemÉrték(a,i))=BalRész(a,i) ∧
Egyesít(ElemÉrték(a,i),JobbRész(1,i+1))=JobbRész(a,i) ∧
(i>Hossz(a) ⇒ BalRész(a,i)=BalRész(a,Hossz(a)) ∧
Üres?(JobbRész(a,i)))

3° Az egyesítés során az operandusbeli absztrakt sorozatok hivatkozási sorrendben helyeződnek egymásután.

c=Egyesít(a,b) ⇒ i∈[1,Hossz(a)]: ElemÉrték(c,i)=ElemÉrték(a,i) ∧
j∈[1,Hossz(b)]: ElemÉrték(c,Hossz(a)+i)=ElemÉrték(b,i) ∧
Hossz(c)=Hossz(a)+Hossz(b)

1.15. Gráf

Kétféleképpen közelítjük meg ezt a típuskonstrukciót. Az első megközelítésben egy „korlátozott” képességekkel ellátott, párhuzamos éleket nem tartalmazó, *irányított gráf*ot definiálunk, amelyre csak bizonyos struktúramódosító műveletek lesznek alkalmazhatók. A pontok pozitív egész indexekkel címezhetők meg, élekre közvetlenül nem hivatkozhatunk (csak a két végpont megadásával). A hossz nemnegatív valós szám.

A második megközelítésben ezt általánosítjuk elhagyva a fenti korlátokat, és teljessé tesszük úgy az operációk terén, mint az őt felépítő kellékek típusa terén.

Típus StatikusGráf(Típus Elem, Konstans PontSzám: NemNegEgész):

Asszociált műveletek:

Új(Egész): Gráf	[létrehoz egy egyelőre izolált pontokból álló gráfot]
Összeköt(Gráf,Pont,Pont,Hossz): Gráf	
Elszakít(Gráf,Pont,Pont): Gráf	
KövetkezőPont(Gráf,Pont,PozEgész): Pont	
KövetkezőPontokSzáma(Gráf,Pont): NemNegEgész	
Érték(Gráf,Pont): Elem	
PontMódosít(Gráf,Pont,Elem): Gráf	
ÉlHossz(Gráf,Pont,Pont): Hossz	

Axiómák:

Jelölések:

- *g,gg: Gráf(Elem,PontSzám)* – gráf-típusú objektum
- *p,q:Pont=[1..PontSzám]⊂PozEgész* – pont
- r*:PozEgész* – pontindex
- *t:Hossz=NemNegValós* – hosszúság
- *e: Elem* – elemtípusú objektum

1° Az új gráfnak PontSzám db izolált (+∞-távolságú) pontja van, amelynek $Új_e$ elem-típusú kezdőértéke van.

$g=Új(Elem,N) \Rightarrow KövetkezőPontokSzáma(g,p)=0 \wedge Érték(g,p)=Új_e \wedge ÉlHossz(g,p,q)=+\infty$

2° A gráf értéke az, amit a PontMódosít ad neki.

$Érték(PontMódosít(g,p,e),p)=e$

3° A gráf bármely 2 pontja összeköthető irányított éllel. Ha nincs él-kapcsolat valamely két pont között, akkor távolságuk +∞.

$Összeköt(g,p,q,t)=gg \Rightarrow$
$ÉlHossz(gg,p,q)=t \wedge$
$ÉlHossz(g,p,q)=+\infty \Rightarrow KövetkezőPontokSzáma(gg,p)=KövetkezőPontokSzáma(g,p)+1$

4° Az elszakított élek végpontjai végtelen távolságra kerülnek egymástól.

$Elszakít(g,p,q)=gg \Rightarrow ÉlHossz(gg,p,q)=+\infty \wedge$
$ÉlHossz(gq,p,q)\neq+\infty \Rightarrow KövetkezőPontokSzáma(gg,p)=KövetkezőPontokSzáma(g,p)-1 \wedge$
$ÉlHossz(gq,p,q)=+\infty \Rightarrow KövetkezőPontokSzáma(gg,p)=KövetkezőPontokSzáma(g,p)$

5° A gráf pontjainak szomszédait el lehet érni a KövetkezőPont függvény segítségével.
$r \in [1..KövetkezőPontokSzáma(g,p)] \Rightarrow \exists q = KövetkezőPont(g,p,r) \wedge$
$r \notin [1..KövetkezőPontokSzáma(g,p)] \Rightarrow KövetkezőPont(g,p,r) = NemDef$

A második megközelítésben ezt általánosítjuk: megengedve a pontok számának dinamikus alakulását, a *párhuzamos*an futó *élek*et. Lehetőségét adva arra, hogy a pontokat, s az éleket egyedi értékkel lássuk el. Teljesebbé tesszük újabb struktúratesztelő, -módosító operációk hozzávételével.

Típus DinamikusGráf(Típus Elem,Pont,Él):

Asszociált műveletek:

Üres: Gráf

Üres?(Gráf): Logikai

Komponens(Gráf,Pont): Gráf

KomponensSzám(Gráf): NemNegEgész

PontSzám(Gráf): NemNegEgész

ÉlSzám(Gráf): NemNegEgész

KomponensKezdőPont(Gráf,NemNegEgész): Gráf

Beilleszt(Gráf,Pont,Elem): Gráf

Töröl(Gráf,Pont): Gráf

Összeköt(Gráf,Él,Pont,Pont,NemNegValós): Gráf

Elszakít(Gráf,Pont,Pont): Gráf

Elszakít(Gráf,Él): Gráf

Éle?(Gráf,Él): Logikai

Pontja?(Gráf,Pont): Logikai

KövetkezőPont(Gráf,Pont,NemNegEgész): Pont

KövetkezőPontokSzáma(Gráf,Pont): NemNegEgész

Kiinduló ÉlekSzáma(Gráf,Pont): NemNegEgész

Kiinduló Él(Gráf,Pont,NemNegEgész): Él

VégPontok(Gráf,Él): Pont x Pont

Érték(Gráf,Pont): Elem

PontMódosít(Gráf,Pont,Elem): Gráf

ÉlHossz(Gráf,Pont,Pont): NemNegValós

ÉlHossz(Gráf,Él): NemNegValós

ÉlMódosít(Gráf,Él,NemNegValós): Gráf

Axiómák:

Jelölések:
- *g,h: Gráf(Elem,Pont,Él)* – gráf-típusú objektum
- *p,q: Pont* – ponttípusú objektum
- *é,é1: Él* – éltípusú objektum
- *e,e1: Elem* – elemtípusú objektum
- *r:NemNegEgész – pontindex*

1° .

2. ALGORITMIKUS SPECIFIKÁCIÓK

Ebben a részben a típuskonstrukciók megvalósítása felé teszünk lépéseket. Mint említettük: itt két dolgot kell tisztázni. Az egyik, hogy mik azok az *operáció*k (és más fogalmak), amelyeket láthat a programozó, amikor az egyes típusokat beépíti programjába, és hogy miként használhatja föl (*szintaxis minimális szemantikai utalás*okkal). Ezt írjuk le az adott típus export moduljában. A másiktól, a „tényleges” *megvalósítás*tól (részletes reprezentációtól és a hozzá igazodó implementációtól) most eltekintünk.

A modul szintaxisa:

ExportModul `TípusNév(InputParaméterek)`[32]

`[most következnek az exportálandó fogalmak „értelmesen” csoportosított felsorolása:]`

Típus `[ritkán van ilyenre szükség]`
`Típ1,Típ2,...`

Konstans `[Ritkán van ilyenre szükség.]`
`Kons1,Kons2,...` `[a TípusNév típus néhány kiemelendő konstansa]`

Függvény `Fv1(FormParam): FvTip1`
`Fv2(FormParam): FvTip2`
`...`

Eljárás `Elj1(FormParam)`
`Elj2(FormParam)`
`...`

Operátor `Op1(FormParam): OpTip1`
`Op2(FormParam): OpTip2`
`...`

Modul vége.

Megjegyzések:

- Természetesen a fő fogalom, a `TípusNév`, külön megadás híján is látszódó fogalom lesz.
- A csoportosításra semmilyen megkötés nincs, sem sorrendjére, sem arra, hogy hány „darabba” csoportosítjuk az egyes fajta fogalmakat. Tehát lehet több konstans, típus vagy más csoport is.
- Kerülni célszerű (bár nem tilos) a változók, konstansok exportálását. Ui. veszélyes, ha a rátámaszkodó programnak lehetősége van az adatokkal *közvetlen* kapcsolatba kerülni. Ha ilyen igény merülne föl, akkor definiálni kell külön erre a célra tevékenységeket (eljárásokat, függvényeket, operátorokat).
- Ha a *függvény*ek, *eljárás*ok stb. csoportosításnál nem a fentit modulszintaxist követjük, hanem valami más tartalmi jellegűt, akkor minden „kategóriaváltásnál” újból ki kell írni a kategórianevet (`Eljárás`, `Függvény` stb.). Persze „fölöslegesen” is kiírható.

A típusok eddigi megragadásánál nem foglalkoztunk a teljes struktúrára vonatkozó *Egyenlőség*, *Értékadás* műveletekkel, ill. a *kimenet*-*bement* műveleteivel. A gyakorlati programozásnál azonban ezekre szükség lesz, ezért most már nem hunyhatunk szemet fölöttük. További különbség adódik az axiómatikus specifikációhoz képest a korábban említett ’biztonságosság’ elvünk miatt, az `Új` műveleteinknek lesz *előfeltétele*, mégpedig, hogy *még nem létezik a létrehozandó struktúra*. Ezt a ’$\neg\exists t \in Típus$’ predikátummal fejezzük ki.

[32] Megjegyzések:

- Az InputParaméterek lehetnek típust jellemző konstansok, de –típuskonstrukció esetén– típusok is. (Szokás e paramétereket ’generic parameter’-nek nevezni.)
- A modulfej egyben a felhasználás „mintájául” is szolgál. Például: **ExportModul** Tömb(*Típus TIndex*: *Típus TElem*) ? **Típus** Vektor=Tömb(*1..9*:*Egész*)

Egy-két típusnál szokatlan lehet a *létrehozó* művelet használatának explicit felkínálása, mivel a programozási nyelvek legtöbb implementációja *automatikusan* gondoskodik a létrehozásról. (Pl. `Tömb`, `Táblázat`, `Statikus gráf`.) A leírás teljeségére való törekvés, illetve az esetleges „hagyományostól" eltérő ábrázolás lehetővé tétele mégis indokolttá tehetik ezt.

Ha felkínáljuk a létrehozás műveletetét, érdemes megtenni ugyanezt az ellenkező célú, `Lerombol` (v. eltérő nevű, de azonos célú) művelettel is. Mivel általánosan és egyöntetűen megfogalmazható a vele szembeni elvárásunk, itt előrebocsátjuk ezt. A `Típus` szó utal a konkrét struktúrára.

```
Eljárás Lerombol(Változó t: Típus)
   Ef: ∃t∈Típus
   Uf: ¬∃t∈Típus
```

2.1. Tömb

```
ExportModul Tömb(Típus IndexTip: Típus ElemTip):

   Eljárás Új(Változó t: Tömb)
      Ef: ¬∃t∈Tömb[33]
      Uf: ∃t∈Tömb ∧ ∀i∈[1,Számosság(IndexTip)]: t_i=NemDef

   Függvény ElemSzám(Konstans t: Tömb): Egész
      Ef: ∃t∈Tömb
      Uf: ElemSzám(t)=Számosság(IndexTip)

   Infix[34] Operátor :=(Változó t1: Tömb, Konstans t2: Tömb)[35]
      Ef: ∃t1,t2∈Tömb
      Uf: ∀i∈[1,Számosság(IndexTip)]: t1_i=t2_i

   Operátor Azonos?(Konstans t1,t2: Tömb): Logikai
        Másnéven[36]: t1=t2
      Ef: ∃t1,t2∈Tömb
      Uf: Azonos?(t1,t2)=(∀i∈[1,Számosság]: t1_i=t2_i)

   Operátor ElemÉrték(Konstans t: Tömb, i: IndexTip): ElemTip
        Másnéven: t(i)
      Ef: ∃t∈Tömb
      Uf: ElemÉrték(t,i)=t_i

   Operátor ElemMódosítás(Változó t: Tömb,
                          Konstans i: IndexTip, e: ElemTip)
        Másnéven: t(i):=e
      Ef: ∃t∈Tömb
      Uf: ElemMódosítás(t,i,e)=t'_i ∧  t'_i=e

   Operátor Kiírás(Konstans t:Tömb)
                                                [a teljes Tömb kiírása]
        Másnéven: Ki: t
      Ef: ∃t∈Tömb
      Uf: ∀i∈[1,Számosság(IndexTip)]: OutputSorozat_i=t_i
```

[33] Az *Ef* nem következménye az axiómáknak. A korábbiakban kimondott „biztonsági" koncepciónknak folyamánya.

[34] Az *Infix* minősítő azt jelenti, hogy a két operandusa közé irandó a műveleti jel, vagyis az operátor neve.

[35] Ennek az operátornak nincs értéktípusa, ez arra utal, hogy nem függvényként, hanem az eljárás „rokonaként" működik.

[36] Ez operátor az első példa az ún. *speciális* szintaktikájú operátorokra. A „specialitását", azaz a fölhasználás szintaktikáját a '**Másnéven:**' kulcs-szóval bevezetett sor rögzíti.

```
  Operátor Beolvasás(Változó t:Tömb)
                                                  [a teljes Tömb beolvasása]
      Másnéven: Be: t
    Ef:  ∃t∈Tömb
    Uf:  ∀i∈[1,Számosság(IndexTip)]: t_i=InputSorozat_i
Modul vége.
```

2.2. Lista

```
ExportModul Lista(Típus ElemTip):
  [Jelölések, megállapodások a specifikációhoz:
    1) Lista = sorozat x aktIndex x hiba
    2) aktIndex∈[1,Hossz(sorozat)]∪ {NemDef}
    3) l' a Lista „új” állapota, ha a korábbitól meg kell
       különböztetni ]

  Eljárás Üres(Változó l:Lista)
                                                        [listalétrehozás]
    Ef:  ¬∃l∈Lista
    Uf:  ∃l∈Lista ∧ l=((),NemDef,Hamis)

  Függvény Üres?(Konstans l:Lista): Logikai
                                                        [üres-e a lista]
    Ef:  ∃l∈Lista
    Uf:  Üres?(l)=(l.sorozat=())

  Függvény Tele?(Konstans l:Lista): Logikai
                                                        [tele-e a lista]
    Ef:  ∃l∈Lista
    Uf:  Tele?(l)=(... nincs mód további elem hozzávételére ...)

  Függvény ElemSzám(Konstans l:Lista): Egész
                                                        [listaelemszám]
    Ef:  ∃l∈Lista
    Uf:  ElemSzám(l)=Hossz(l.sorozat)

  Függvény ElemÉrték(Változó[37] l:Lista): ElemTip
                                                [az aktuális elem értéke]
    Ef:  ∃l∈Lista
    Uf:  l.aktIndex≠NemDef ⇒ ElemÉrték(l)=l.sorozat_l.aktIndex
         l.aktIndex=NemDef ⇒ l'.hiba=Igaz

  Eljárás Elsőre(Változó l:Lista)
                                    [az aktuális elem legyen az első]
    Ef:  ∃l∈Lista
    Uf:  l.sorozat≠() ⇒ l'=(l.sorozat,1,l.hiba)
         l.sorozat=() ⇒ l'.hiba=Igaz

  Eljárás Következőre(Változó l:Lista)
                                  [az aktuális elem legyen a következő]
    Ef:  ∃l∈Lista
    Uf:  l.sorozat=(l_1,...,l_n) ∧ l.aktIndex∈[1,n) ⇒
                                      l'=(l.sorozat,l.aktIndex+1,l.hiba)
         l=(l_1,...,l_n) ∧ l.aktIndex∈{n,NemDef} ⇒ l'.hiba=Igaz
```

[37] Nem tévedés: **változó**, hiszen hiba esetén az *l hiba*-mezoje igazra módosul. Ez igaz minden olyan tevékenységnél, amelyiknél hiba adódhat!

Eljárás Előzőre(**Változó** l:Lista)
[az aktuális elem legyen az előző]

Ef: ∃l∈Lista
Uf: $l=(l_1,...,l_n) \wedge l.aktIndex \in (1,n] \Rightarrow$
l'=(l.sorozat,l.aktIndex-1,l.hiba)
$l=(l_1,...,l_n) \wedge l.aktIndex \in \{1,NemDef\} \Rightarrow$ l'.hiba=Igaz

Eljárás Utolsóra(**Változó** l:Lista)
[az aktuális elem legyen az utolsó]

Ef: ∃l∈Lista
Uf: $l.sorozat=(l_1,...,l_n) \wedge n>0 \Rightarrow$ l'=(l.sorozat,n,l.hiba)
$l=() \Rightarrow$ l'.hiba=Igaz

Eljárás BeszúrMögé(**Változó** l:Lista, **Konstans** e:ElemTip)
[beszúrás az aktuális elem mögé]

Ef: ∃l∈Lista
Uf: $l.sorozat=(l_1,...,l_n) \wedge l.aktIndex \in [1,n] \Rightarrow$
$l'=((l_1,...,l_{aktIndex},e,...l_n),l.aktIndex+1,l.hiba)$
$l.sorozat=() \Rightarrow$ l'=((e),1,l.hiba)
$l=(l_1,...,l_n) \wedge n>0 \wedge l.aktIndex=NemDef \Rightarrow$ l'.hiba=Igaz

Eljárás BeszúrElé(**Változó** l:Lista, **Konstans** e:ElemTip)
[beszúrás az aktuális elem elé]

Ef: ∃l∈Lista
Uf: $l.sorozat=(l_1,...,l_n) \wedge l.aktIndex \in [1,n] \Rightarrow$
$l'=((l_1,...,e,l_{aktIndex},...l_n),l.aktIndex-1,l.hiba)$
$l.sorozat=() \Rightarrow$ l'=((e),1,l.hiba)
$l=(l_1,...,l_n) \wedge n>0 \wedge l.aktIndex=NemDef \Rightarrow$ l'.hiba=Igaz

Eljárás Kihagy(**Változó** l:Lista)
[az aktuális elem kihagyása]

Ef: ∃l∈Lista
Uf: $l.sorozat=(l_1,...,l_n) \wedge n>0 \wedge l.aktIndex \in [1,n] \Rightarrow$
$l'=((l_1,...,l_{aktIndex-1},l_{aktIndex+1},...l_n),l.aktIndex-1,l.hiba)$
$l.sorozat=(l_1,...,l_n) \wedge n>0 \wedge l.aktIndex=NemDef$
$\vee$ l.sorozat=() $\Rightarrow$ l'.hiba=Igaz

Eljárás Egymásután(**Konstans** l1,l2:Lista, **Változó** l:Lista)
[a két lista egyesítése]

Ef: ∃l∈Lista
Uf: $l1.sorozat=(l1_1,...,l1_{n1}) \wedge l2.sorozat=(l2_1,...,l2_{n1}) \Rightarrow$
$l.sorozat=(l1_1,...,l1_{n1},l2_1,...l2_{n2}) \wedge$ l.hiba=Hamis $\wedge$
(n1+n2>0 $\Rightarrow$ l.aktIndex=1)

Függvény UtolsóE?(**Változó** l:Lista): Logikai
[az aktuális elem az utolsó-e]

Ef: ∃l∈Lista
Uf: $l.sorozat=(l_1,...,l_n) \wedge n>0 \Rightarrow$ UtolsóE?(l)=(l.aktIndex=n)
$l.sorozat=() \Rightarrow$ l'.hiba=Igaz

Függvény ElsőE?(**Változó** l:Lista): Logikai
[az aktuális elem az utolsó-e]

Ef: ∃l∈Lista
Uf: $l.sorozat=(l_1,...,l_n) \wedge n>0 \Rightarrow$ ElsőE?(l)=(l.aktIndex=1)
$l.sorozat=() \Rightarrow$ l'.hiba=Igaz

Függvény Hibás?(**Változó**[38] l:Lista): Logikai
[történt-e hiba a listára hivatkozás során, „törli” a hibaváltozót]

Ef: ∃l∈Lista
Uf: Hibás?(l)=l.hiba ∧ l'.hiba=Hamis

Operátor Azonos?(**Konstans** l1,l2:Lista): Logikai
[a két lista azonos-e?]

Másnéven: l1=l2
Ef: ∃l1,l2∈Lista
Uf: Azonos?(l1,l2)=(l1.sorozat=($l1_1$,...,$l1_{n1}$) ∧
l2.sorozat=($l2_1$,...,$l2_{n2}$) ∧
n1=n2 ∧ ∀i∈[1,n1]: $l1_i$=$l2_i$)

Infix Operátor :=(**Változó** l1:Lista, **Konstans** l2:Lista)
[a teljes listára vonatkozó értékadás-művelet]

Ef: ∃l1,l2∈Lista
Uf: l1=(l2.sorozat,1,Hamis)

Operátor Kiírás(**Konstans** l:Lista)
[a teljes lista kiírása]

Másnéven: Ki: l
Ef: ∃l∈Lista
Uf: l.sorozat=(l_1,...,l_n) ⇒ ∀i∈[1,Hossz(l.sorozat)]:
$OutputSorozat_i$=l_i

Operátor Beolvasás(**Változó** l:Lista)
[a teljes lista beolvasása]

Másnéven: Be: l
Ef: ∃t∈Lista
Uf: InputSorozat=(x_1,...,x_n) ⇒ ∀i∈[1,Hossz(InputSorozat)]: l_i=x_i

Modul vége.

2.3. Verem

ExportModul Verem(**Típus** ElemTip):

[Jelölések a specifikációhoz:
1) Verem = sorozat ˣ hiba
2) v' a Verem „új” állapota, ha a korábbitól meg kell különböztetni]

Eljárás Üres(**Változó** v:Verem)
[veremlétrehozás a memóriában]

Ef: ¬∃v∈Verem
Uf: ∃v∈Verem ∧ v=((),Hamis)

Függvény Üres?(**Konstans** v:Verem): Logikai
[üres-e a verem]

Ef: ∃v∈Verem
Uf: Üres?(v)=(v.sorozat=())

Függvény Tele?(**Konstans** v:Verem): Logikai
[a verem megtelt-e]

Ef: ∃v∈Verem
Uf: Tele?(v)=(... a v nem bővíthető több elemmel ...)

[38] Most sem tévedés: **Változó.** Ui. e függvény másodlagos hatása: a lista hibaállapota kezdohelyzetbe jut.

Függvény Tető(**Változó** v:Verem): ElemTip
[a verem tetőelemének értéke]
Ef: ∃v∈Verem
Uf: v.sorozat=(v_1,...,v_n) ⇒ Tető(v)=v_n
v.sorozat=() ⇒ v'.hiba=Igaz

Eljárás Verembe(**Változó** v:Verem, **Konstans** e:ElemTip)
[verembetétel]
Ef: ∃v∈Verem
Uf: v.sorozat=(v_1,...,v_n) ⇒ v'=((v_1,...,v_n,e),Hamis)
Tele?(v) ⇒ v'.hiba=Igaz

Eljárás Veremből(**Változó** v:Verem, e:ElemTip)
[a tetőelem kivétele]
Ef: ∃v∈Verem
Uf: v.sorozat=(v_1,...,v_n) ⇒ v'=((v_1,...,v_{n-1}),Hamis) ∧ e=v_n
v=() ⇒ v'.hiba=Igaz

Függvény Hibás?(**Változó** v:Verem): Logikai
[történt-e hiba a veremre hivatkozás során, „törli” a hibaváltozót]
Ef: ∃v∈Verem
Uf: Hibás?(v)=v.hiba ∧ v'.hiba=Hamis

Infix Operátor Azonos?(**Konstans** v1,v2:Verem): Logikai
[a két verem azonos-e?]
Másnéven: v1=v2
Ef: ∃v∈Verem
Uf: Azonos?(v1,v2)=(v1.sorozat=($v1_1$,...,$v1_{n1}$) ∧
v2.sorozat=($v2_1$,...,$v2_{n2}$) ∧
n1=n2 ∧ ∀i∈[1,n1]: $v1_i$=$v2_i$)
v=() ⇒ v'.hiba=Igaz

Infix Operátor :=(**Változó** v1:Verem, **Konstans** v2:Verem)
[a teljes veremre vonatkozó értékadás-művelet]
Ef: ∃v∈Verem
Uf: v1=(v2.sorozat,Hamis)

Operátor Kiírás(**Konstans** v:Verem)
[a teljes verem kiírása]
Másnéven: Ki: v
Ef: ∃v∈Verem
Uf: v.sorozat=(v_1,...,v_n) ⇒ ∀i∈[1,Hossz(v.sorozat)]:
$OutputSorozat_i$=$v_{Hossz(v.sorozat)-i+1}$

Operátor Beolvasás(**Változó** v:Verem)
[a teljes verem beolvasása]
Másnéven: Be: v
Ef: ∃v∈Verem
Uf: InputSorozat=(x_1,...,x_n) ⇒ ∀i∈[1,Hossz(InputSorozat)]: v_i=x_i

Modul vége.

2.4. Duplaverem

Az előzőhöz nagyon hasonlóan fogalmazható meg az export modul így most nem részletezzük.

2.5. Sor

ExportModul Sor(**Típus** ElemTip):

[Jelölések a specifikációhoz:
1) Sor = sorozat ˣ hiba
2) s' a Sor „új” állapota, ha a korábbitól meg kell különböztetni]

Eljárás Üres(**Változó** s:Sor)
[Sorlétrehozás a memóriában]

Ef: ¬∃s∈Sor
Uf: ∃s∈Sor ∧ s=((),Hamis)

Függvény Üres?(**Konstans** s:Sor): Logikai
[üres-e a Sor]

Ef: ∃s∈Sor
Uf: Üres?(s)=(s.sorozat=())

Függvény Tele?(**Konstans** s:Sor): Logikai
[a Sor megtelt-e]

Ef: ∃s∈Sor
Uf: Tele?(s)=(... az s nem bővíthető több elemmel ...)

Függvény Első(**Változó** s:Sor): ElemTip
[a Sor első elemének értéke]

Ef: ∃s∈Sor
Uf: s.sorozat=(s_1,...,s_n) ⇒ Első(s)=s_1
s.sorozat=() ⇒ s'.hiba=Igaz

Eljárás Sorba(**Változó** s:Sor, **Konstans** e:ElemTip)
[Sorbatétel]

Ef: ∃s∈Sor
Uf: s.sorozat=(s_1,...,s_n) ⇒ s'=((s_1,...,s_n,e),Hamis)
Tele?(s) ⇒ s'.hiba=Igaz

Eljárás Sorból(**Változó** s:Sor, e:ElemTip)
[a sor első elemének kivétele]

Ef: ∃s∈Sor
Uf: s.sorozat=(s_1,...,s_n) ⇒ s'=((s_2,...,s_n),Hamis) ∧ e=s_1
s=() ⇒ s'.hiba=Igaz

Függvény Hibás?(**Változó** s:Sor): Logikai
[történt-e hiba a Sorra hivatkozás során, „törli” a hibaváltozót]

Ef: ∃s∈Sor
Uf: Hibás?(s)=s.hiba ∧ s'.hiba=Hamis

Infix Operátor Azonos?(**Konstans** s1,s2:Sor): Logikai
[a két Sor azonos-e?]

Másnéven: s1=s2
Ef: ∃s∈Sor
Uf: Azonos?(s1,s2)=(s1.sorozat=($s1_1$,...,$s1_{n1}$) ∧
v2.sorozat=($s2_1$,...,$s2_{n2}$) ∧
n1=n2 ∧ ∀i∈[1,n1]: $s1_i$=$s2_i$)
s=() ⇒ s'.hiba=Igaz

Infix Operátor :=(**Változó** s1:Sor, **Konstans** s2:Sor)
[a teljes Sorra vonatkozó értékadás-művelet]

Ef: ∃s∈Sor
Uf: s1=(s2.sorozat,Hamis)

Operátor Kiírás(**Konstans** s:Sor)
[a teljes Sor kiírása]
Másnéven: Ki: s
Ef: ∃s∈Sor
Uf: s.sorozat=(s_1,...,s_n) ⇒ ∀i∈[1,Hossz(s.sorozat)]:
$OutputSorozat_i=s_i$

Operátor Beolvasás(**Változó** s:Sor)
[a teljes Sor beolvasása]
Másnéven: Be: s
Ef: ∃s∈Sor
Uf: InputSorozat=(x_1,...,x_n) ⇒ ∀i∈[1,Hossz(InputSorozat)]: $s_i=x_i$

Modul vége.

2.6. Kétvégű sor (Deck)

Az algoritmikus specifikációtól most eltekintünk.

2.7. Prioritási sor

ExportModul PrioritásiSor(**Típus** ElemTip,PriorTip):

[Jelölések a specifikációhoz:
1) PrSor = sorozat ˣ hiba
2) sorozat = (elem ˣ pr)*, ahol az elem ElemTip típusú,
a pr PriorTip típusú mező
3) s' a PrSor „új” állapota, ha a korábbitól meg kell
különböztetni]

Eljárás Üres(**Változó** s:PrSor)
[PrSorlétrehozás a memóriában]
Ef: ¬∃s∈PrSor
Uf: ∃s∈PrSor ∧ s=((),Hamis)

Függvény Üres?(**Konstans** s:PrSor): Logikai
[üres-e a PrSor]
Ef: ∃s∈PrSor
Uf: Üres?(s)=(s.sorozat=())

Függvény Tele?(**Konstans** s:PrSor): Logikai
[a PrSor megtelt-e]
Ef: ∃s∈PrSor
Uf: Tele?(s)=(... az s nem bővíthető több elemmel ...)

Eljárás Első(**Változó** s:PrSor, e:ElemTip, p:PriorTip)
[a PrSor első elemének értéke]
Ef: ∃s∈PrSor
Uf: s.sorozat=(s_1,...,s_n) ⇒ Első(s)=s_1 ∧ s_1.elem=e ∧ s_1.pr=p
s.sorozat=() ⇒ s'.hiba=Igaz

Eljárás PrSorba(**Változó** s:PrSor, **Konstans** e:ElemTip, p:PriorTip)
[PrSorbatétel]
Ef: ∃s∈PrSor
Uf: s.sorozat=(s_1,...,s_n) ⇒ s'=((s_1,...,s_n,pe),Hamis) ∧
pe.elem=e ∧ pe.pr=p
Tele?(s) ⇒ s'.hiba=Igaz

```
Eljárás PrSorból(Változó s:PrSor, e:ElemTip, p:PriorTip)
                                          [a sor első elemének kivétele]
   Ef: ∃s∈PrSor
   Uf: s.sorozat=(s1,...,sn) ⇒ s'=((s2,...,sn),Hamis) ∧
                                s1.pr=p ∧ s1.elem=e
       s=() ⇒ s'.hiba=Igaz
Függvény Hibás?(Változó s:PrSor): Logikai
                                [történt-e hiba a PrSorra hivatkozás
                                      során, „törli” a hibaváltozót]
   Ef: ∃s∈PrSor
   Uf: Hibás?(s)=s.hiba ∧ s'.hiba=Hamis

Infix Operátor Azonos?(Konstans s1,s2:PrSor): Logikai
                                                [a két PrSor azonos-e?]
    Másnéven: s1=s2
   Ef: ∃s∈PrSor
   Uf: Azonos?(s1,s2)=(s1.sorozat=(s11,...,s1n1) ∧
                       v2.sorozat=(s21,...,s2n2) ∧
                       n1=n2 ∧ ∀i∈[1,n1]: s1i=s2i)
       s=() ⇒ s'.hiba=Igaz

Infix Operátor :=(Változó s1:PrSor, Konstans s2:PrSor)
                                        [a teljes PrSorra vonatkozó
                                                értékadás-művelet]
   Ef: ∃s∈PrSor
   Uf: s1=(s2.sorozat,Hamis)

Operátor Kiírás(Konstans s:PrSor)
                                             [a teljes PrSor kiírása]
    Másnéven: Ki: s
   Ef: ∃s∈PrSor
   Uf: s.sorozat=(s1,...,sn) ⇒ ∀i∈[1,Hossz(s.sorozat)]:
                                                 OutputSorozati=si

Operátor Beolvasás(Változó s:PrSor)
                                        [a teljes PrSor beolvasása]
    Másnéven: Be: s
   Ef: ∃s∈PrSor
   Uf: InputSorozat=(x1,...,xn) ⇒ ∀i∈[1,Hossz(InputSorozat)]: si=xi
Modul vége.
```

2.8. Táblázat

```
ExportModul Táblázat(Konstans Méret:NemNegEgész,
                     Függvény KulcsFv(Elem):Kulcs, Típus Elem,Kulcs):
  [Jelölések a specifikációhoz:
    1) Tábla = sorozat x aktIndex x hiba
    2) aktIndex∈[1,Hossz(sorozat)] ∪ {NemDef}
    3) t' a Táblázat „új” állapota, ha a korábbitól meg kell
       különböztetni ]

  Eljárás Üres(Változó t:Táblázat)
                                    [Táblázatlétrehozás a memóriában]
     Ef: ¬∃t∈Táblázat
     Uf: ∃t∈Táblázat ∧ t=((),NemDef,Hamis)

  Függvény Tele?(Konstans t:Táblázat): Logikai
                                                 [a Táblázat megtelt-e]
     Ef: ∃t∈Táblázat
     Uf: Tele?(t)=(... az t nem bővíthető több elemmel ...)
```

Függvény ElemSzám(**Konstans** t:Táblázat): NemNegEgész
[a Táblázatban található elemek száma]
Ef: $\exists t \in$ Táblázat
Uf: $t.sorozat=(t_1,...,t_n) \Rightarrow ElemSzám(t)=n$

Eljárás Beilleszt(**Változó** t:Táblázat, **Konstans** e:Elem)
[Táblázatbatétel]
Ef: $\exists t \in$ Táblázat
Uf: $\forall i \in [1,Hossz(t.sorozat)]: KulcsFv(t_i) \neq KulcsFv(e) \wedge \neg Tele?(t) \Rightarrow$
$t'=((t_1,...t_{i-1},e,...),i+1,Hamis) \vee$
$t'=((t_1,...t_{Hossz(t.sorozat)},e),NemDef,Hamis)$
$\exists i \in [1,Hossz(t.sorozat)]: KulcsFv(t_i)=KulcsFv(e) \vee Tele?(t) \Rightarrow$
$t'.hiba=Igaz$

Eljárás Töröl(**Változó** t:Táblázat, k:Kulcs)
[törli a táblázat k-kulcsú elemét]
Ef: $\exists t \in$ Táblázat
Uf: $t.sorozat=(t_1,...,t_n) \wedge k=KulcsFv(t_i) \Rightarrow$
$i<Hossz(t.sorozat) \Rightarrow t'=((t_1,...,t_{i-1},t_{i+1},...),i+1,Hamis) \wedge$
$i=Hossz(t.sorozat) \Rightarrow t'=((t_1,...,t_{i-1}),NemDef,Hamis) \wedge$
$\neg\exists i \in [1,Hossz(t.sorozat)]: KulcsFv(t_i)=k \Rightarrow t'.hiba=Igaz$

Eljárás Keres(**Változó** t:Táblázat, k:Kulcs, e:Elem)
[kikeresi a táblázat k-kulcsú elemét]
Ef: $\exists t \in$ Táblázat
Uf: $t.sorozat=(t_1,...,t_n) \wedge k=KulcsFv(t_i) \Rightarrow t_i=e \wedge$
$i<Hossz(t.sorozat) \Rightarrow t'.aktIndex=i+1 \wedge$
$i=Hossz(t.sorozat) \Rightarrow t'.aktIndex=NemDef \wedge$
$\neg\exists i \in [1,Hossz(t.sorozat)]: KulcsFv(t_i)=k \Rightarrow t'.hiba=Igaz$

Eljárás Első(**Változó** t:Táblázat): Elem
[a táblázat első eleme]
Ef: $\exists t \in$ Táblázat
Uf: $t.sorozat=(t_1,...,t_n) \wedge n>0 \Rightarrow Első(t)=t_1 \wedge$
$n>1 \Rightarrow t'.aktIndex=2$
$n=1 \Rightarrow t'.aktIndex=NemDef$
$t.sorozat=() \Rightarrow t'.hiba=Igaz$

Eljárás Utolsó(**Változó** t:Táblázat): Elem
[a táblázat utolsó eleme]
Ef: $\exists t \in$ Táblázat
Uf: $t.sorozat \neq () \Rightarrow Utolsó(t)=t_{Hossz(t.sorozat)} \wedge t'.aktIndex=NemDef$
$t.sorozat=() \Rightarrow t'.hiba=Igaz$

Eljárás Következő(**Változó** t:Táblázat): Elem
[a táblázat következő eleme]
Ef: $\exists t \in$ Táblázat
Uf: $t.sorozat \neq () \wedge t.aktIndex=i \Rightarrow$
$i<n \Rightarrow Következő(t)=t_i \wedge t'.aktIndex=i+1$
$i=n \Rightarrow Következő(t)=t_n \wedge t'.aktIndex=NemDef$
$t.aktIndex=NemDef \Rightarrow t'.hiba=Igaz$

Eljárás Előző(**Változó** t:Táblázat): Elem
[a táblázat előző eleme]
Ef: $\exists t \in$ Táblázat
Uf: $t.sorozat \neq () \wedge t.aktIndex=i \Rightarrow$
$i>1 \Rightarrow Előző(t)=t_i \wedge t'.aktIndex=i-1$
$i=1 \Rightarrow Előző(t)=t_1 \wedge t'.aktIndex=NemDef$
$t.aktIndex=NemDef \Rightarrow t'.hiba=Igaz$

```
Függvény Hibás?(Változó t:Táblázat): Logikai
                                [történt-e hiba a Táblázatra hivatkozás
                                        során, „törli” a hibaváltozót]
   Ef:  ∃t∈Táblázat
   Uf:  Hibás?(t)=t.hiba ∧ t'.hiba=Hamis
Operátor Kiírás(Konstans t:Táblázat)
                                                [a teljes Táblázat kiírása]
    Másnéven: Ki: t
   Ef:  ∃t∈Táblázat
   Uf:  t.sorozat=(t1,...,tn) ⇒ ∀i∈[1,Hossz(t.sorozat)]:
                                                        OutputSorozati=ti
Operátor Beolvasás(Változó t:Táblázat)
                                                [a teljes Táblázat beolvasása]
    Másnéven: Be: t
   Ef:  ∃t∈Táblázat
   Uf:  InputSorozat=(x1,...,xn) ⇒ ∀i∈[1,Hossz(InputSorozat)]: ti=xi
Modul vége.
```

2.9. Input Szekvenciális File

2.10. Output Szekvenciális File

2.11. Direkt File

2.12. Asszociatív File és Index-Szekvenciális File

2.13. Rekurzió

2.14. Absztrakt Sorozat

2.15. Gráf

```
ExportModul Gráf(Típus ÉlTip:Típus HosszTip,Típus PontTip:Típus ElemTip):
  [Jelölések a specifikációhoz:
    1) Gráf = sorozat x hiba
    2) g' a Gráf „új” állapota, ha a korábbitól meg kell
       különböztetni ]
  Eljárás Üres(Változó g:Gráf)
                                        [gráflétrehozás a memóriában]
    Ef:  ¬∃g∈Gráf
    Uf:  ∃g∈Gráf ∧ g=((),Hamis)
  Függvény Üres?(Konstans g:Gráf): Logikai
                                                                    []
    Ef:  ¬∃g∈Gráf
    Uf:  ∃g∈Gráf ∧ g=((),Hamis)
```

```
Függvény Komponens(Konstans g:Gráf, p:PontTip): Gráf
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)
Függvény KomponensSzám(Konstans g:Gráf): NemNegEgész
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)

Függvény PontSzám(Konstans g:Gráf): NemNegEgész
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)

Függvény ÉlSzám(Konstans g:Gráf): NemNegEgész
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)

Függvény KomponensKezdőPont(Konstans g:Gráf, Kompsorsz:NemNegEgész):
                                                                 PontTip
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)

Eljárás BeillesztPont(Változó  g:Gráf, Konstans p:PontTip, e:ElemTip)
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)

Eljárás TörölPont(Változó g:Gráf, Konstans p:PontTip)
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)

Eljárás Összeköt(Változó  g:Gráf,
                 Konstans é:ÉlTip, h:HosszTip, p,q:PontTip)
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)

Eljárás Elszakít(Változó g:Gráf, Konstans p,q:PontTip)
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)

Eljárás Elszakít(Változó g:Gráf, Konstans é:ÉlTip)
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)

Függvény Éle?(Konstans g:Gráf, é:ÉlTip): Logikai
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)

Függvény Pontja?(Konstans g:Gráf, p:PontTip): Logikai
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)
```

```
Függvény KövetkezőPont(Konstans g:Gráf,p:PontTip,Élsorsz:NemNegEgész):
                                                                PontTip
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)
Függvény KövetkezőPontokSzáma(Konstans g:Gráf, p:PontTip): NemNegEgész
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)

Függvény KiindulóÉlekSzáma(Konstans g:Gráf, p:PontTip): NemNegEgész
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)

Függvény KiindulóÉl(Konstans g:Gráf, p:PontTip, élsorsz:NemNegEgész):
                                                                 ÉlTip
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)

Eljárás VégPontok(Konstans g:Gráf,e:ÉlTípus, Változó ból,ba:PontTip)
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)

Függvény Érték(Konstans g:Gráf, p:PontTip): ElemTip
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)

Eljárás PontMódosít(Változó  g:Gráf, Konstans p:PontTip, e:ElemTip)
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)

  Függvény ÉlHossz(Konstans g:Gráf, p1,p2:PontTip): HosszTip
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)

Függvény ÉlHossz(Konstans g:Gráf, é:ÉlTip): HosszTip
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)

Eljárás ÉlMódosít(Változó  g:Gráf, Konstans é:ÉlTip, h:HosszTip)
                                                                    []
   Ef: ¬∃g∈Gráf
   Uf: ∃g∈Gráf ∧ g=((),Hamis)
```

Modul vége.

FÜGGELÉK

ALAP-TÍPUSKONSTRUKCIÓK SPECIFIKÁCIÓJA

Az alábbiakban megkísérljük összefoglalni a legfontosabb típuskonstrukciókat. Épp azokat, amelyeket a legtöbb programozási nyelv készen felkínál újabb típusok definiálásához. Különleges nehézséget okoz, hogy a nyelvek fejlődése során egyedi szintaxisuk alakult ki, amely mintázása nem megy a specifikálásban a „megszokott" módon. Az egyedi szintaxis: a kötetlen számú paraméterezés. Ahol az addig leírtak „mechanikusan" folytathatók, ott ezt a tényt a „…"-tal érzékeltetjük.[39]

A *Szelektor* kulcsszó (itt is) szelektor-azonosítók adott, speciális tulajdonságú *sorozat*ára utal. Tulajdonsága az egyediség, azaz $\forall i \neq j \in [1..Hossz(Szelektor)]: s_i, s_j \in$ Szelektor $\wedge s_i \neq s_j$.

Bevezetjük a *ETípus* jelölést a típusok azon *halmaz*ára, amelyek a résztvesznek a típuskonstrukció egyes részeinek definiálásában.

Rekord

Algebrai specifikáció:

Típus Rekord(Szelektor szelektor$_1$, Típus Elem$_1$, …,
Szelektor szelektor$_N$, Típus Elem$_N$):

Tehát Szelektor=(szelektor$_1$,…,szelektor$_N$) sorozat, és ETípus={Elem$_1$,..,Elem$_N$} halmaz.[40]

Asszociált műveletek:

Üres: Rekord	
Módosít(Rekord,Szelektor,ETipus): Rekord ∪ {NemDef}	[a rekord valamely szelektora által meghatározott mezőjének módosítása]
Komponens(Rekord,Szelektor): ETipus ∪ {NemDef}	[a rekord valamely szelektora által meghatározott mezőjének értéke]

Axiómák:

Jelölések:

- *r,p: Rekord(Szelektor$_1$,Elem$_1$...)* = rekord-típusú objektum
- *e$_i$: Elem$_i$* = elem$_i$-típusú objektum
- *s,s$_i$,...: Szelektor* = {Szelektor$_1$,...}

1° Az üres rekord minden komponensének értéke az adott típusú iniciális érték.

$Üres=r \Rightarrow \forall i \in [1..Hossz(Szelektor)]: Komponens(t,s_i)=e_i \in Elem_i \wedge e_i=Üres_{ei}$

2° Adott szelektorú komponens értéke az, amit az utolsó módosításkor kapott.

$\forall i \in [1..Hossz(Szelektor)]: Komponens(Módosít(r,s_i,e_i),s_i)=e_i$ ...

---

[39] A „…" jelölést nem tartjuk –természetesen– pontosnak és egyértelműnek, de mivel közismert fogalmak leírásánál használjuk, nem okoz zavart. Pláne amiatt, hogy nem olyan helyen kényszerülünk a pongyolaságra, ahol a mondanivalónk leglényege található.

[40] Az ETipus halmaz mivolta miatt tehettük, hogy felsoroltuk mind az N komponenstípust, bár nem szükségképpen különbözők ezek. Típusismétlődéskor természetesen a halmaz számossága nem növekszik.

3° A 'szelektor–típus' párosítás szigorú, ennek megsértése nem definiált eredményre vezet.
$\forall i \neq j \in [1..\ Hossz(Szelektor)]: e_j \notin Elem_i \Rightarrow Módosít(r,s_i,e_j)=NemDef$

Megjegyzés: ez az axióma valójában szintaktikai szinten ellenőrizhető állítást mond ki.

Algoritmikus specifikáció:

ExportModul Rekord(**Szelektor** $szel_1$: **Típus** $ETip_1$,...,
Szelektor $szel_N$: **Típus** $ETip_N$):

[Megállapodás a specifikációhoz:
1. Rekord = $ETip_1$ x … x $ETip_N$,
azaz a rekordot egy N-elemű sorozatként tekintjük
2. ETipus = $ETip_1 \cup \ldots \cup ETip_N$]

Eljárás Üres(**Változó** r:Rekord)
[rekordlétrehozás]
Ef: $\neg\exists r \in Rekord$
Uf: $\exists r \in Rekord \wedge Üres(r)=(e_1,\ldots,e_N) \wedge Üres(e_1) \ldots \wedge Üres(e_N)$

Operátor Módosít(**Változó** r:Rekord, **Szelektor** szel, **Tipus** e:ETipus)
Másnéven r.szel:=e
[rekord 'szel' szelektora által meghatározott mezőjének módosítása]
Ef: $\exists r \in Rekord \wedge szel=szel_i \Rightarrow e \in ETip_i$
Uf: $r=(e_1,\ldots,e_i,\ldots,e_N) \wedge szel=szel_i \Rightarrow$
$Módosít(r,szel,e)=(e_1,\ldots,e_{i-1},e,e_{i+1},\ldots,e_N)$

Operátor Komponens(**Konstans** r:Rekord, Szelektor szel):ETipus
Másnéven r.szel
[rekord 'szel' szelektora által meghatározott mezőjének értéka]
Ef: $\exists r \in Rekord$
Uf: $r=(e_1,\ldots,e_N) \wedge szel=szel_i \Rightarrow Komponens(r,szel)=e_i \in ETip_i$

Modul vége.

Alternatív rekord

Algebrai specifikáció:

**Típus AlterRekord(Szelektor szelektor, Típus Elem,
Konstans Logikai esetén Szelektor szelektor1, Típus Elem1, ...):**

Asszociált műveletek:

...(AlterRekord): AlterRekord ∪ {NemDef}

Axiómák:

Jelölések:
- *r,p: AlterRekord(Szelektor1,Elem1...)* = altaernatív rekord-típusú objektum
- *e: Elem* = elemtípusú objektum
- *s,s1,...: Szelektor* = {Szelektor1,...}
- *l1,...: Logikai* =

1° .

Algoritmikus specifikáció:

ExportModul `AlternatívRekord(`**Szelektor** `szel1:` **Típus** `Típ1...):`

`...`

Modul vége.

Halmaz

Algebrai specifikáció:

Halmaz(Típus Elem):

Asszociált műveletek:

Üres?(Halmaz): Halmaz ∪ {NemDef}

Üres: Halmaz

Elem?(Halmaz,Elem): Logikai

Egyesít(Halmaz,Halmaz): Halmaz

Metsz(Halmaz,Halmaz): Halmaz

Számosság(Halmaz): Halmaz

Kivon(Halmaz,Halmaz): Halmaz

Axiómák:

Jelölések:

- *f,g: Halmaz(Elem)* = halmaz-típusú objektum
- *e: Elem* = elemtípusú objektum
-
-

1°.

Algebrai specifikáció:

ExportModul `Halmaz(`**Típus** `Elem):`

`...`

Modul vége.

TÍPUSHIERARCHIA (LESZÁRMAZÁS)

Az alábbiakban összefoglaljuk az eddig ismertetett típuskonstrukciók „rokoni kapcsolatait”. Megadjuk a leszármazási fa mellett, hogy mely operációkat öröklik (filozófiájukhoz igaztva), s melyeket nem exportálják tovább.

Sorozat = `Új, BeOlvas, KiÍr, :=, =, Első, Maradék`

- Tömb `ElemSzám, ElemÉrték, ElemMódosítás,` ~~`Első`~~, ~~`Maradék`~~

- Lista ~~Új~~⇒Üres, Üres?, Tele?, ElemSzám, ElemÉrték, ElsőE?, Elsőre, Következő, Következőre, BeszúrMögé, BeszúrElé, Kihagy, EgymásUtán, UtolsóE?, Utolsó, Utolsóra, Előző, Előzőre, ~~Maradék~~

  = Verem ~~Új~~⇒Üres, Üres?, Tele?, ~~Első~~⇒Tető, Verembe, ~~Első~~⇒Veremből, ~~Maradék~~

  ≡ DuplaVerem

  = Sor ~~Új~~⇒Üres, Üres?, Tele?, Első⇒Sorból, Sorba, ~~Maradék~~

  ≡ Kétvégű Sor

  ≡ Prioritási Sor

- Táblázat

- File

  = Input Szekvenciális File

  = Output Szekvenciális File

  = Direkt File

  = Asszociatív File

  ≡ Indexelt Szekvenciális File

- Rekurzió

- Absztrakt Sorozat

- Gráf

IRODALOMJEGYZÉK

[MV] Mauer Gy.-Virág I.: Bevezetés a struktúrák elméletébe,
Dacia Könyvkiadó, 1976

[GHM1] J.Guttag - E.Horowitz - D.Musser: Abstract Data Types and Software Validation,
Communications of ACM Vol. 21 No. 12 pp. 1048-1064, 1978

[Latal] R.L.London - J.V.Guttag - J.J.Horning - B.W.Lapson - J.G.Mitchell - G.J.Popek: Proof Rules for the Programming Language Euclid,
Acta Informatica Vol. 10 Fasc. 1 pp. 1-26, 1978

[GH] J.V.Guttag - J.J.Horning: The Algebraic Specification of Abstract Data Types,
Acta Informatica Vol. 10 Fasc. 1 pp. 27-52, 1978

[GTW] J.A.Goguen - J.W.Thatcher - E.G.Wagner: An Inital Algebra Approach to the Specifcation, Correctness and Implmentation of Abstract Data Types,
???, pp. 80-149, 197?

[V1] Varga L.: Típusspecifikációk helyességének vizsgálata,
Alkalmazott Matematikai Lapok 13 pp. 57-68, 1987-88

[V2] Varga L.: Programok analízise és szintézise,
Akadémiai Kiadó, 1981

[V3] L. Varga: On the Verification of Abstract Data Types,
Acta Cybernetica Tom. 6 Fasc. 3 pp. 7-12, 1983

[Me] B.Meyer: Object-Oriented Software Construction.
Prentice Hall, 1988

[K1] Kozics S.: A Modula-2 programozási nyelv.
ELTE TTK Ált.Számtud. Tsz., 1992

[K2] Kozics S.: Az Ada programozási nyelv.
ELTE TTK Ált.Számtud. Tsz., 1992

[P1] H.A.Partsch: Formal Problem Specification on an Algebraic Basis,
Lecture Notes in Computer Science Vol. 755 pp. 183-224, Springer-Verlag, 1993

[SzZs2] Szlávi P. - Zsakó L.: Módszeres programozás - Programozási bevezető. (μLógia 18)
ELTE TTK Ált.Számtud. Tsz., 1993

[PP1] F.Parisi-Presicce - A.Pierantonio: An Algabraic Theory of Class Specification,
ACM Transaction on Software Engineering and Methodology Vol. 3 No. 2 pp. 166-199, 1994

[Sz1] Szlávi P.: Előadás a sorozattípusokról. (μLógia Szilánkok 7)
ELTE TTK Ált.Számtud. Tsz., 1994

[ChLe] Y.Cheon - G.Leavens: The Larch/Smalltalk Interface Specification Language,
ACM Transaction on Software Engineering and Methodology Vol. 3 No. 3 pp. 221-253, 1994

[2Mc] A. McMonnies - W.S. McSporran: Developing Object-oriented Data Structures Using C++,
McGrawHill Book Co., 1995

[Sz2] Szlávi P.: Előadás a file-típusokról és a táblázattípusokról. (μLógia Szilánkok 8)
ELTE TTK Ált.Számtud. Tsz., 1993,1995

[PSzZs1] Pap G.né - Szlávi P. - Zsakó L.: Módszeres programozás - Rekurzív típusok. (μLógia 27)
ELTE TTK Ált.Számtud. Tsz., 1994

[PSzZs2] Pap G.né - Szlávi P. - Zsakó L.: Módszeres programozás - Szövegfeldolgozás. (μLógia 14)
ELTE TTK Ált.Számtud. Tsz., 1994

[SzZs1] Szlávi P. - Zsakó L.: Módszeres programozás - Adatfeldolgozás. (μLógia 12)
ELTE TTK Ált.Számtud. Tsz., 1996

[SzW1] Szlávi Péter: Programozási tételek specifikálása VDM-SL-ben. http://izzo.inf.elte.hu/szlavi/public/PRT_VDM_TELJES.pdf, 1998, 2001