tanulmányok

164/1984

MTA Számitástechnikai és Automatizálási Kutató Intézet Budapest

MAGYAR TUDOMÁNYOS AKADÉMIA
SZÁMITÁSTECHNIKAI ÉS AUTOMATIZÁLÁSI KUTATÓ INTÉZETE

SPECIFIKÁCIÓS ADATBÁZIS MODELLEK

Irta:

Knuth Előd

Tanulmányos 164/1984

A kiadásért felelős:

DR VÁMOS TIBOR

igazgató

Főosztályvezető:

DEMETROVICS JÁNOS

ISBN 963 311 183 8
ISSN 0324-2951

SZÁMALK REPRO VGM 84/315

TARTALOMJEGYZÉK

1. BEVEZETÉS

1.1 A tanulmány tárgya

A tanulmányban olyan adatkezelési technikákat ill. adatbáziskezelési modelleket ismertetünk, melyek egy sajátos célt: rendszerleírások, rendszer specifikációk, illetve általánosságban struktúrált leírások kezelését (tárolását, napra készen tartását, elemzését, konzisztenciájának ellenőrzését, riportok és dokumentációk generálását) szolgálják.

Célkitűzését tekintve a tanulmány tárgya a szoftver technológia területére tartozik. Az ilyen típusú vizsgálatok a szoftver technológia területén mintegy tíz éves múltra tekintenek vissza [65], [10], [49], amikor is egyfelől nyilvánvalóvá vált, hogy manuális kezelés útján a számítástechnikai rendszerek tervezése és kivitelezése során felhalmozódó dokumentum mennyiséggel, illetve a bennük foglalt információk összefüggéseivel egyre kevésbé lehet boldogulni; másfelől, ekkorra létrejöttek azok a számítástechnikai alapeszközök, melyek ezen információk kezelésének gépi támogatását már siker reményében megközelíthetővé tették.

Vegyük most először szemügyre azokat a sajátosságokat, melyek kezelendő információinkra (meglévő vagy tervezés alatt álló különféle rendszerek modelljei, specifikációi, leírásai) jellemzőek.

a) Az adatok (leírások) formátuma (változó hosszúságú, esetenként nagy méretű alfanumerikus részegységek) eléggé tág határok között kötetlen.

b) Alapkövetelmény, az adatobjektumok közötti sokrétű kapcsolatok ábrázolása, melyet a kapcsolatok tipizálása (kapcsolatok közötti kapcsolatok) útján célszerű rendszerbe foglalni.

c) A konvencionális adatbázisoknál megszokott procedurális szükségletek nincsenek, a hangsúly a leírások adekvát, a lekérdezések és elemzések céljait szolgáló ábrázolásán van.

d) A lekérdezés (információ visszanyerés) fő szempontja a szelektív struktúrált riportok generálásának lehetősége (valamely egyszerű, világos nyelvi rendszer, és azt megalapozó kalkulus útján).

e) Végül, bizonyos alkalmazásokban szükség van típusok és kapcsolatok halmazszerű kezelésére.

Ha meglévő eszközeinket nézzük, akkor a felsorolt követelmények egyszerű szövegszerkesztő, ill. szöveges információ feldolgozó rendszerekkel való lefedését eleve kizárhatjuk. A szokványos univerzális adatbáziskezelő rendszerek alkalmazása elvben lehetséges lenne, ezek azonban éppen univerzalitásuk miatt a specifikus követelmények hatékony kiszolgálásához nem nyújtanak támogatást, legfőképpen azonban az adatábrázolás adekvátságának elsődlegessége az, ami a hálószerű ill normalizált relációk útján történő ábrázolást nehézkessé és - természetellenessége miatt - itt értelmetlenné teszi.

Ezek az okok azok, amiért világszerte a számos helyen kidolgozott, különféle rendszerleíró nyelvekhez, az ezeken a nyelveken leírt objektumok gépi tárolására saját módszereket dolgoztak ki (pl. [50], [47], [59], [22], [9]). Mindezek a megoldások azonban ad hoc tárolási technikákra alapultak, anélkül, hogy az ilyen típusú információk kezelésének általános kérdéseit vizsgálták volna.

1.2 Az alkalmazott módszer

A módszertani megközelítés tekintetében a tanulmány tárgya az u.n. absztrakt adatszerkezetek (főbb referenciákat ld. például [14], [18], [67], [56], [48]) területére esik. A hagyományos kutatások tárgyától azonban abban térünk el, hogy míg az absztrakt adatszerkezetek vizsgálatának területe az idézett munkákban döntően a programozási nyelvekben való procedurális alkalmazásokra esett, mi ezzel szemben az absztrakt adatstruktúrák adatbázis kezelési (tehát tömeges adatokra, és nem egyes adatobjektumokra) vonatkozó törvényszerűségeit vizsgáljuk.

(Ez a kutatási irány az absztrakt adatstruktúrák területén viszonylag új [55], [63].)

A tanulmányban a matematika eszközeire, módszereire és formalizmusára építünk (logika, absztrakt algebra, ill. formális nyelvek elmélete - mint ahogy ma minden számítástudományi absztrakciónak éppen ezek a legfőbb bázisai). Maga a tanulmány azonban nem matematikai tárgyú. Nem tételeket mondunk ki (kivéve a 3.2 fejezetet), hanem logikai konstrukciókat adunk meg, melyek a számítástechnikai gyakorlatban végzett tevékenységek, módszerek, szerkezetek absztrakciói. E konstrukciók (modellek) adekvátsága matematikai eszközökkel nem bizonyítható, értéküket a gyakorlati alkalmazhatósággal lehet mérni. Ez az oka annak is, hogy az alkalmazásoknak (meta nyelvi rendszerre alapuló konstrukciók) egy teljes fejezetet (nyelvi modellek) szentelünk.

1.3 Az eredmények helye

A tanulmány négy fő fejezetre (2.-5.fejezetek) tagolódik. Mindegyikben modelleket (logikai konstrukciókat) adunk meg. E négy fejezet közül három számítástudományi, egy pedig (a 3.fejezet) matematikai konstrukciókat tárgyal.

A 2.fejezetben koncepciókat (koncepcionális modelleket) írunk le (hasonló jelleggel, mint ahogy a számítástudomány "klasszikus" koncepciói, a "szemafor", a "monitor", a "class", a "cluster", a "funel" stb. annak idején bevezetésre kerültek). Az itt megadott koncepciók által kifejezett absztrakciók gyökere kettős: egyfelől az absztrakt adatszerkezetek procedurális alkalmazásainak eredményei [14], [18], másfelől a szoftver engineering specifikációs nyelvei és módszerei, ld. például [50], [47].

A 3.fejezetben koncepcionális modelljeink matematikai hátterét adjuk meg. Rámutatunk arra, hogy a modern matematikai eredmények mennyire nélkülözhetetlenek a számítástechnikai problémák hibamentes programozói megol-

dásainak szempontjából. A fejezet első részében vizsgált problémakör kettős általánosítás: egyfelől a klasszikus adatbázis relációk [13], [62], másfelől a procedurális adat absztrakciók [14], [48] kereteibe illeszkedik. A fejezet második fele tisztán matematikai jellegű. Egy "hidat", egy közös magot kisérel megadni a Petri hálók, a "path kifejezések" és a "trace nyelvek" három iskolája között az adatbázis operációk közötti u.n. "liveness problem" vonatkozásában. (Az irányzatok reprezentánsaiként ld. [58], [54] ill. [45].)

A 4.fejezetben rendszerleíró nyelveket ill. rendszerábrázolási modelleket és módszereket adunk meg. Ilyenek persze nagy számban készültek már világszerte. Az általunk megadott modellek és nyelvek azonban mind egyetlen keretbe, a 2.fejezetben megadott meta nyelvi rendszerbe illeszkednek, és így a nyelvi objektumok adatbázis objektumokra való leképezését is tartalmazzák. Levezetünk ismert nyelveket is (PSL [65], Jackson módszer [21]), és megadunk újakat is.

Az 5.fejezetben a szoftver engineering problémakörét egy másik oldalról, a szoftver rendszerekben kialakítható logikai architektúrák oldaláról vizsgáljuk. Ilyen típusú modellek kialakítására irányuló kutatások csak a legutóbbi időben indultak (OSI [20] referencia modell, [60] folyamatirányítási szoftver modell) és a szoftver technológia jelentős mértékű javításához éppen ezek igérik ma a legtöbb lehetőséget. Mi ebben a fejezetben fogalmakat és elemeket vezetünk be, melyek általánosságban használhatók szoftver rendszerek logikai szerkezetének megadására, ill. a fent említett értelemben vett referencia modellek építésére. Ezek alkalmazásaként egyúttal a korábbi fejezetekben ismertetett modellek számítógépen való kivitelezésének fő vonalait is bemutatjuk.

Köszönetnyilvánítás

A legnagyobb köszönettel közvetlen munkatársaimnak: Bodó Zalánnak, Demetrovics Jánosnak, Halász Ferencnek, Kis Olivérnek, Radó Péternek, Rónyai Lajosnak és Szilléry Andrásnak tartozom. A gondolatok állandó szembesítése és az értékes észrevételek mellett nekik köszönhető az is, hogy az értekezésbe foglalt modellek a gyakorlatban is megvalósultak, működő számítástechnikai rendszerek formájában (SDLA és MDB különböző változatai).

Igen sok segítséget jelentettek azok az észrevételek, melyeket az évek során e témakör problémáit vitatva Andréka Hajnal, Benczur András, Bernus Péter, D. Bjørner, Dömölky Bálint, Fidrich Ilona, Hatvany József, Havass Miklós, C.A.R.Hoare, V. Kotov, P. Lauer, Németi István, E.J.Neuhold, Peák István, C.A. Petri, Reino Kurki Suonio, Szeredi Péter, A. Tarski, Tóth Attila, Tóth Árpád, D. Teichroew-tól kaptam.

2. KONCEPCIONÁLIS MODELLEK

Ebben a fejezetben koncepciókat, pontosabban számítástechnikai adatkezelési koncepciókat kifejező modelleket adunk meg. Ezek a modellek a bázisukon megvalósítandó számítástechnikai rendszereknek (az ISO OSI [20] filozófiája értelmében vett) felsőbb szintjeit határozzák meg (az alapjukul szolgáló megvalósítási mechanizmusokat ezen a szinten - részletkérdésként - nyitva hagyva. Ezekről az alapul szolgáló más modellekről a 3.fejezetben szólunk.) Ezen felső szintű modellek a megvalósítandó számítástechnikai rendszereket azok célja felől közelítik meg úgy, hogy a megadott koncepciók e rendszerek későbbi részleteinek tervezését már determinálják. (Nem tévesztendők össze tehát az itt megadott koncepcionális szintű modellek az ANSI SPARC [5] terminológia szerinti, u.n. konceptuális sémákkal.)

2.1 Elemi konstrukciók

Ebben az alfejezetben egy igen egyszerű szerkezetű, de egy u.n. öndefiniáló sémával rendelkező (automatizált sémekezelésű) adatbázis modellt mutatunk be, mely egyfelől alapját képezi a későbbi fejezetekben szereplő gazdagabb szemantikájú modelleknek, másrészt, mint a gyakorlatban is realizált rendszer, a specifikációs adatkezelési feladatok egy nagyobb területén önmagában is alkalmazhatónak bizonyult.

2.1.1 Információ szerkezet

Mint minden számítástechnikai rendszer koncepciójának kialakításakor, az adatbázis kezelő eszközök tervezésénél is az igények oldaláról kell kiindulnunk:

2.1.1.1 Információ elemek

Számítógépben tárolandó és manipulálandó leírások (rendszerleírások, specifikációk, dokumentációk) készítésekor az esetek nagy részében egyszerű tőmondatok formájában szokás (sőt

célszerű) tagolni a leírandó információt. Valamely rendszer-részlet specifikációjában például tipikusan efféle mondatok szerepelhetnek:

"Hibajelző modul.
Hívja a hibaelemző modul.
Használja a hibakód táblázatot.
Jelzéseit a 2-es konzolon adja."
Stb.

Mint általánosságban és a számítástechnika más területein is ismert: a rendszerezés legegyszerűbb, jól bevált módja a tipizálás (típusok mint osztályok bevezetése). Fenti kijelentéseket például az alábbi módon tipizálhatjuk:

modul: hibajelző
hívó: hibaelemző
használt adat: hibakód táblázat
kijelzés: 2-es konzol
stb.

ahol az aláhúzott szavak jelenthetik az osztály (típus) neveket.

Matematikai jelölésekkel a fentiek értelmében célszerű tehát beszélni u.n. "típusok" egy T halmazáról, melynek $t \in T$ elemei u.n. "értékekből" ($e \in E$) álló halmazok: $t \in E$. Nevezzük "információ elemnek", "atomnak" (vagy tipizált mondatnak) a (t,e) rendezett párokat, ahol $e \in t$ (azaz egy értéket a típusával együtt). Mármost:

(A1) Az $i_1 = (t_1, e_1)$ és $i_2 = (t_2, e_2)$ információ elemeket akkor és csak akkor tekintjük azonosnak, ha $t_1 = t_2$ és $e_1 = e_2$.

(Megjegyzés: Választható természetesen (A1) helyett az annál erősebb, az osztályozást kimondó axioma is, mely kizárná, hogy azonos értékek különböző típusok mellett előforduljanak. A választás helyességét végső soron itt az adatbázis felhasználójának kényelme dönti el. (A1) választása mellett a gyakorlati tapasztalatok szóltak.)

2.1.1.2 Információ szakaszok

Leírások (rendszerleírások, specifikációk) készítésekor a mondatokat célszerű csoportokba rendezni. Egy ilyen csoport (szakasz) célszerűen valamilyen összefoglaló címet visel (vagy megfordítva fogalmazva: mondatai a címet jelentő "fejmondatra" vonatkoznak). A 2.1.1.1-beli példa is tekinthető ilyen "szakasznak", nézzünk most mégis egy másik tipikus példát (ezúttal angolul):

```
File address table.
     Accessible by super-users.
     Maintained by file management modul.
     Parts are: directory area, label area,
                                overflow area.
     Storage mechanism is B-tree.
     Recovery is based on duplicate storage.
     Etc.
```

Tipizált mondatok (információ elemek) formájába rendezve, a fenti információ szakasz például így nézhet ki:

```
data: file address table
   access limitation: super-users only
   maintenance: file management modul
   parts: directory area,
          label area,
          overflow area
   storage mechanism: B-tree
   recovery mechanism: duplicate storage
```

Most vegyük szemügyre először heurisztikusan azokat a szempontokat, melyekkel kívánatos, hogy az információ szakaszok rendelkezzenek:

(i) "kötetlenség": egy információ szakasz megadásakor abba tetszés szerinti információ elemek beírhatók;

(ii) "változtathatóság": egy meglévő szakasz a későbbiek során tetszés szerinti új információ elemekkel bővíthető kell legyen, ill. bármely hozzá csatolt elem bármikor törölhető kell legyen;

(iii) "multiplikáció": egy típusból, egyazon szakaszban, tetszés szerinti számú információ elem fordulhat elő.

E szempontokból már látjuk, hogy a fenti értelemben vett információ szakaszokat ésszerűtlen lenne akár hagyományos rekordokként, akár (változó hosszúságú) relációkként kezelni. Ehelyett, a jól ismert u.n. "Entity Relationship Attribute (ERA)" [12] adatmodell filozófiájának megfelelően (részleteiben azonban a későbbiekben majd e modelltől némileg eltérően) az információ szakaszokat a fejmondat és minden egyes belső mondata között létrehozott bináris relációk formájában fogjuk adatmodell szinten ábrázolni. Pontosabban:

Valamely i_1, i_2 információ elemekből alkotott (i_1,i_2) rendezetlen párt ezen információ elemek "kapcsolatának" fogjuk nevezni.
Ennélfogva:

(A2) Az (i_1,i_2) és (i_3,i_4) kapcsolatokat akkor és csak akkor tekintjük azonosnak, ha vagy $i_1=i_3$, $i_2=i_4$, vagy $i_1=i_4$, $i_2=i_3$, ahol az egyenlőségjelek az (A1) axióma értelmében vett azonosságot jelentik.

(Megjegyzés: Lehet rendezett párokra alapuló adatmodellt is építeni. Ez természetesen mind az általa implikált tárolási módokban, mind a felhasználói dialógus szerkezetekben gyökeresen eltérne az általunk választott rendezetlen párokra épülő modelltől. A mi választásunk végső oka a felhasználói kényelem – az invertált lekérdezések egyszerű módjának biztosítása, ld. később.)

2.1.2 Elsődleges operációk

Mielőtt egy információ feldolgozási dialógust felvázolnánk, szükségünk van az eddig megadott elemi információ struktúrákon végezhető legegyszerűbb adatbázis műveletek (felvitel, törlés, azonosítás stb.) pontos definicióira. (Ezek a műveletek a programozók szemében triviálisaknak szoktak tűnni, épp ezért, az információs rendszerek tervezésekor a tapasztalatok szerint kevés gondot fordítanak az elemi operációk szemantikájának pontos specifikációjára, amely

azután rendszerint a későbbiekben válik súlyos és "megmagyarázhatatlan" hibák forrásává.)

Először jelöléseket vezetünk be.Jelölje

I_D az adatbázis által az adott pillanatban éppen tartalmazott információ elemek halmazát. (Feltételezzük, hogy kezdetben,az adatbázis inicializálásakor az I_D halmaz üres.) Jelölje

T_D az I_D-be tartozó információ elemek típusainak halmazát: $T_D = \{t : \exists\, i = (t,e);\ i \in I_D\}$. Jelölje

C_D az adatbázis által az adott pillanatban éppen tartalmazott információ elemek között $\{(i,j)\}$ kapcsolatok halmazát az (A2) azonossági axiómának megfelelő értelemben. (Kezdetben a C_D halmaz üres.) Jelölje

R_D azon (t_1,t_2) rendezetlen tipus-párok halmazát, melyekhez létezik az adatbázisban ilyen típusú elemek közötti kapcsolat: $R_D = \{(t_1, t_2) : : \exists\, i_1,i_2;\ i_1 = (t_1, e_1) \in I_D,\ i_2 = (t_2,e_2) \in I_D,\ (i_1,i_2) \in C_D\}$.

2.1.2.1 Elem-műveletek

Valamely (t,e) információ elemre vonatkozóan a felhasználó háromféle műveletet kezdeményezhet:

1) Az információ elemet fel akarja vinni az adatbázisba. Jelölje ezt az operációt:

input (t,e).

2) A nevezett elemet meg kívánja keresni. (Most mindegy, miért, akár lekérdezés eredményéhez, akár valamely más összetett operáción belül operandusként.) Jelölje ezt az operációt:

atom (t,e).

3) Az információ elemet törölni akarja. Jelölje ezt az operációt:

delete (t,e).

Definiciók:

input (t,e)

a) Ha (t,e) $\in I_D$, akkor nincs operáció. Az eredmény "neutrális". Különben:

b) Ha t $\in T_D$, akkor az elem felvitelre kerül. (Az I_D halmaz, és ezen belül az adott t-tipusú elemek halmaza kibővül.) Az eredmény: "bővítés". Különben:

c) ((t,e) $\notin I_D$, sőt t $\notin T_D$.) Az új elem felvitelre kerül. (Az I_D halmazon kívül maga T_D is bővül az új t típussal, melyhez most egyetlen elem tartozik.) Az eredmény: "séma bővítés".

atom (t,e)

a) Ha (t,e) $\in I_D$, akkor az eredmény "pozitív", (és az elem, vagy a címe a megfelelő helyen regisztrálásra kerül). Különben:

b) Ha t $\in T_D$, akkor az eredmény "negatív". Különben:

c) (Nincs ilyen típus sem.) Ekkor az operáció "hibás hívásáról" beszélünk.

delete (t,e)

1.lépés: atom (t,e).

2.lépés:

a) Ha az eredmény "pozitív" volt, a (t,e) elem minden egyes kapcsolatára vonatkozóan végrehajtjuk a "disconnect" operációt (ld.2.1.2.2), majd a (t,e) elemet az I_D halmazból töröljük. Ezután:

a1) Ha a t típusú elemek halmaza üressé vált, e típust a T_D halmazból töröljük. Az eredmény: "séma szűkítés". Különben:

a2) Az eredmény: "szűkítés".

b) Ha az eredmény "negatív" volt, akkor nincs operáció, és most az eredmény: "neutrális". Végül:

c) Ha az eredmény "hibás hívás" volt, úgy most is az.

2.1.2.2 Kapcsolat-műveletek

Valamely (i_1,i_2), $i_1=(t_1,e_1)$, $i_2=(t_2,e_2)$ kapcsolatra vonatkozóan a felhasználó szintén háromféle műveletet kezdeményezhet:

1) Kapcsolat létrehozása. Jelölje ezt:

connect (i_1,i_2).

2) Adatbázisban tárolt kapcsolat megkeresése. Jelölje ezt:

connection (i_1,i_2).

3) Kapcsolat megszűntetése. Jelölje ezt:

disconnect (i_1,i_2).

Definiciók

connect (i_1,i_2)

a) Ha $(i_1,i_2) \in C_D$, akkor nincs operáció. Az eredmény "neutrális". Különben:

b) Ha $(t_1,t_2) \in R_D$, akkor:

b1) Ha $i_1 \in I_D$, $i_2 \in I_D$, akkor az új kapcsolat regisztrálásra kerül. Az eredmény: "bővítés". Különben:

b2) (Az összekapcsolandó elemek valamelyike nincsen benne az adatbázisban.) Eredmény: "hibás hívás".

Különben:

c) Ha $t_1 \in T_D$, $t_2 \in T_D$, akkor az új kapcsolat regisztrálásra kerül. Az eredmény: "séma bővítés". Különben:

d) (Valamelyik típus nem létezik.) Eredmény: "hibás hívás".

connection (i_1,i_2)

a) Ha $(i_1,i_2) \in C_D$, akkor az eredmény "pozitív" (és a kapcsolat, vagy címe megjegyzésre kerül). Különben:

b) Ha $i_1 \in I_D$, $i_2 \in I_D$, akkor az eredmény "negatív". Különben:

c) (Valamelyik elem nem létezik.) Az eredmény: "hibás hívás".

disconnect (i_1,i_2)

1. lépés: connection (i_1,i_2).

2. lépés:

a) Ha az eredmény "pozitív" volt, a kapcsolatot a C_D halmazból töröljük. Ezután:

a1) Ha a t_1, t_2 tipusú elemek közötti kapcsolatok halmaza üressé vált, a (t_1,t_2) párt az R_D halmazból töröljük. Eredmény: "séma szűkítés". Különben:

a2) Az eredmény "szűkítés".

b) Ha az eredmény "negatív" volt, akkor nincs operáció és most az eredmény "neutrális". Végül:

c) Ha az eredmény "hibás hívás" volt, úgy most is az.

2.1.3 Dialógus szerkezet

Információ szakaszokat leírni és számítógépbe feldolgozásra bejuttatni természetesen a régebben elterjedt, un. "kötegelt feldolgozási módokon" is lehet (off-line adathordozókon előkészítve), sőt, néha erre ma is szükség van. Mégis: a meghatározó üzemmód ma már az on-line párbeszédes feldolgozás, ezért az adatbázis legfőbb interfészeinek kialakítását is ez kell meghatározza. A dialóg feldolgozás (általában) egy képernyő kép támogatásával történik, melyen egyaránt jelenhetnek meg a felhasználó által írt és a rendszer által írt információk. Esetünkben pl. maga egy a képernyőn látható "információ szakasz" is létrejöhet olyan módon, hogy egyes részei a felhasználótól, más részei pedig a számítógépes rendszertől származnak.

(Megjegyzés: Az alábbiakban a dialógus konkrét szintaxisának részleteire - pl., hova kell a típusnevet írni, hány szóból állhat stb. - nem térünk ki. Ezeket az olvasó [31] ill. [37] -ben megtalálhatja.)

2.1.3.1 Kezdeti információ felvitel

Tegyük fel most, hogy az adatbázis üres, és az adatbázis kezelő rendszer olyan állapotban van, hogy adatok fogadására alkalmas. Példaképpen ekkor - fő vonalaiban - egy dialógus célszerűen az alábbi módon építhető fel:

A felhasználó a képernyőre ír egy bevitelre szánt információ elemet. Például:

ACTION: SEARCH NEXT DATA.

A rendszer ezek után végrehajtja e paraméterekkel az "input" operációt (ld. 2.1.2.1). (Az input operáció háromféle eredményétől függően a rendszer különféleképpen válaszolhat ill. intézkedhet. Például, ha a rendszer által még nem ismert típusról van szó - mint ez esetben is - a rendszer kérhet u.n. "megerősítést", vajon nem csupán elírásról van-e szó. Az ilyen típusú részletek azonban - noha az összes felhasználói igényeket, ergonómiai és egyéb szempontokat maradéktalanul kielégítő adekvát dialógus megtervezése igen nehéz feladat - ezen értekezésnek nem képezik tárgyát, e részleteket illetően

[43]-ra hivatkozunk.)

Egészítsük most ki a képernyőn látható információ elemet további elemek hozzáadásával egy információ szakasszá pl. az alábbi módon:

```
ACTION: SEARCH NEXT DATA.
   INPUT: PREVIOUS DATA,
          SEARCH CONDITION.
   OUTPUT: DATA FOUND.
   MECHANISM: VSAM TREE.
   CONTROLLED BY: DATA ACCESS UNIT.
```

A rendszer most sorra végrehajtja az

```
input (INPUT, PREVIOUS DATA)
input (INPUT, SEARCH CONDITION)
input (OUTPUT, DATA FOUND)
stb.,
```

majd mindezekre az elemekre és a fej-sorra vonatkoztatott "connect" operációkat. Ezzel létrejön az adatbázisban a fenti információ szakasz egyfajta reprezentációja (melyből a fenti szakasz alkalmas algoritmussal visszanyerhető).

Az adatbázis jelenlegi állapotában tehát:

$$T_D = \{ \text{ACTION}, \text{INPUT}, \text{OUTPUT}, \text{MECHANISM}, \text{CONTROLLED BY} \}.$$

és

$$R_D = \{(\text{ACTION}, \text{INPUT}) (\text{ACTION}, \text{OUTPUT}), (\text{ACTION}, \text{MECHANISM}), (\text{ACTION}, \text{CONTROLLED BY})\}.$$

2.1.3.2 Ismételt felvitel

Megjegyzés: látszólagos ellentondásnak tűnhet az eddigiek alapján, hogy míg az adatbázis "homogén": nincsenek benne sem kitüntetett típusok, sem kitüntetett információ elemek, a relációknak nincs iránya stb., eközben az információ szakaszban egy kiemelt elem (a fej) van. Nos, természetesen: bármely információ elem elővehető szakasz-fejként is (a hozzá tartozó szakasz elemekként mindazokat az elemeket tekintve, melyekkel kapcsolata van).

Ha tehát most pl. egy újabb "ACTION"-t kívánunk leírni, mondjuk:

ACTION: NATURAL JOIN.

akkor a rendszer most már képes önmaga megjeleníteni a képernyőn eddigi kapcsolatait:

```
ACTION: NATURAL JOIN.
   INPUT:
   OUTPUT:
   CONTROLLED BY:
   MECHANISM:
```

melyeket ezután csak ki kell tölteni, akár egy űrlapot, a különbség azonban az, hogy esetünkben:

a) A szakasz szerkezete továbbra sem válik rögzítetté. Bármikor, tetszés szerint, további információkat írhatunk hozzá.

b) Már meglévő típusú szakasz elemeket is szabadon írhatunk bele (felsorolás, felsorolás bővítés), ill. törölhetünk.

Ugyanakkor végezhetünk felvitelt bármely szempont szerint "invertált" módon is: ha például szakaszfejként mondjuk az

INPUT: JOIN PARAMETERS.

sorral kezdünk, akkor annak eddig ismert egyetlen kapcsolata:

INPUT: JOIN PARAMETERS
ACTION:

jelenik meg (persze lehetne más is, mely ez esetben szintén megjelenne).

2.1.3.3 Módosító operációk

Az adatbázis "naprakész" állapotban tartásához szükséges felhasználói operációkat célszerű a már ismert fogalmak bázisán, szintén dialóg interaktív üzemhez tervezve kialakítani. Ennek egy célszerű módja az alábbi.

A módosító operációkat mindíg a képernyőn megjelenített információ szakaszokon, u.n. "képernyő-szerkesztési" (screen editor) manipulációk útján végezzük el úgy, hogy az u.n. "kurzor"-ral a megváltoztatandó (módosítandó, törlendő,beszúrandó stb.) pontra mutatunk, majd oda bejegyzést teszünk (új információt írunk, törlési kódot írunk be stb.). A szerkesztés pontos szintaxisa itt részletkérdés, ezért erre e szakasz végén csak egy példával utalunk. Az operációk megválasztása és szemantikája ezzel szemben igen lényeges. A gyakorlat (több kisérlet után végül is) az alábbi készlet helyességét igazolta:

1. add new item (t,e) (új sor hozzáírása a szakaszhoz)

Adatbázis operáció: input (t,e);
connect ("fej",t,e);

Képernyő operáció:

a) Ha (t,e) már eddig is a szakaszban volt, nincs operáció. Különben:

b) Ha t-tipusú elem már szerepelt a szakaszban, az új elem a régiek mellett felsorolásra kerül. Különben:

c) Az új elem a szakaszt kiegészítő, önálló, új elemmé válik.

2. delete item (t,e) (szakaszbeli sor törlése)

Adatbázis operáció: disconnect ("fej", t,e);
delete (t,e);

Képernyő operáció:

A tétel törlődik és a szakasz értelemszerűen átrendeződik. (Jegyezzük meg, hogy az operáció előtt mind atom (t,e), mind connection ("fej", t,e) szükségképpen "pozitívak" voltak, mivel létező szakaszban, létező elemre mutattunk rá.)

3. delete section (t,e) (szakasz megszűntetése)

Adatbázis operáció: efface (t,e)

(Ezen operációt ld. a 2.1.4.2 pontban. Lényege most számunkra a következő:) Hatására az adatbázis olyan állapotba kerül, hogy

a) a (t,e) elem újbóli felírása után az előhívott szakasz üres lesz;

b) A korábban a szakaszhoz tartozott elemek közül mindazok az adatbázisból is törlődnek, melyek semmilyen más szakaszban nem szerepelnek.

(Megjegyzés: a "szakasz törlés" operációnak megadható lenne a fentinél gyengébb és erősebb változata is. A fenti választás a gyakorlati tapasztalatokra alapuló mérlegelés eredménye.)

Képernyő operáció:

A képernyő kiürül.

(Megjegyzés: A törlések körében maradva bevezethetnénk még pusztán kapcsolat törlést is, minthogy az az adatmodell logikájából kézenfekvően adódik. A tapasztalat mégis azt mutatja, hogy az editor operációk körében ez felesleges.)

4. replace item (t,e_1,t,e_2) (a szakaszban kijelölt információ elem cseréje)

((t,e_1) a kijelölt elem, e_2 az új érték)

Adatbázis operáció: disconnect ("fej",t,e_1);
delete (t,e_1);
input (t,e_2);
connect ("fej",t,e_2);

Képernyő operáció:

A megadott sorban az érték rész helyére az új érték kerül.

5. rename item (t,e_1,t,e_2) (érték rész megváltoztatása)

Adatbázis operáció: rename (t,e_1,t,e_2);

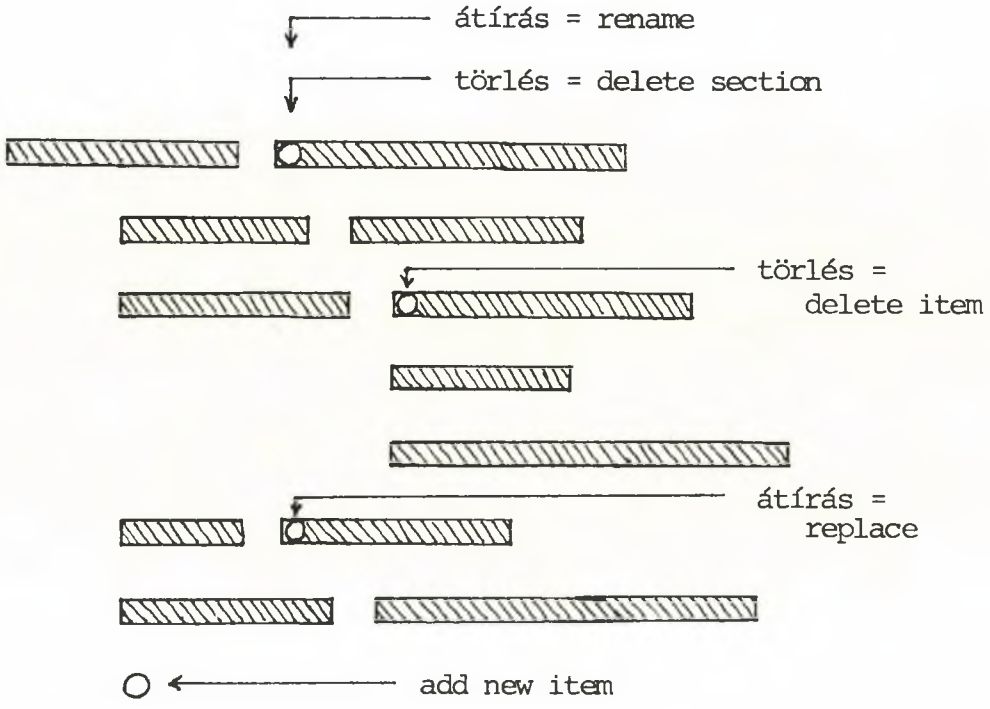
(Ld. még 2.1.4.3.) Lényege: Az elem összes kapcsolatai az új érték dacára változatlanok maradnak. (Vegyük észre, hogy az előző operációban a delete (t,e_1) művelet (ld.2.1.2.1) az összes kapcsolatot is felbontotta.)

Képernyő operáció:

(Mint előbb.)

(Megjegyzés: Vegyük észre, hogy az adatbázis szemantika szempontjából nem mindegy, hogy pl. valakinek a lakáscíme azért változik meg, mert elköltözik, vagy azért, mert átnevezték az utcát, amelyben lakik. Utóbbi esetben olyan operációt észszerű használni, mely az utcanév változást egyetlen tranzakcióval az összes érintett ott lakókra vonatkozóan elvégzi. Ez a különbség az utolsó két fenti operáció között.)

Végezetül a következő ábrán egy bevált sémát vázolunk a felsorolt operációk editor manipulációkra való leképezésére.



2/1. ábra

Editor akciók és adatbázis operációk egy lehetséges megfeleltetése

2.1.3.4 Keresési, megjelenítési követelmények

Rendezés

Ha az adatbázisban tárolt különféle adatok között nincs semmiféle rendezési reláció, a szokásos fizikai szintű adatbázis kezelési technikák arra vezethetnek, hogy az adatbázisból generált különféle riportok egyenrangú tételeinek sorrendje véletlenszerű lesz abban az értelemben, hogy az adatbázis tartalmán végrehajtott igen kis változások a lekért (esetleg igen hosszú) listák tételei sorrendjének gyökeres átrendeződéséhez vezetnek. Ez a gyakorlati használat számára elfogadhatatlan, hiszen egy hosszabb dokumentumban valamely részt mindíg ott fogunk keresni, ahol azt már megszoktuk. Szükséges tehát valamilyen ésszerű rendezési elvet követni, mely a kisebb változásokra kis mértékben érzékeny.

Szélsőséges esetben a következő rendezési relációkra lenne szükség:

a) Magának a T_D tipushalmaznak a rendezése.

b) T_D minden t eleméhez egy τ_t rendezési reláció I_D t-típusú elemeinek halmazára.

c) T_D minden t eleméhez egy R_t rendezési reláció T_D azon részhalmaza felett, melynek elemei R_D-ben t-vel relációban vannak.

d) Végül c)-hez hasonlóan C_D minden egyes projekciójának rendezése.

Ezen kívül, amennyiben az információ szakaszok képernyőn való megjelenítése szempontjából nem elégszünk meg a c) (esetleg az a)) által indukált rendezéssel, magukhoz a szakaszok képernyő képeihez is egy-egy rendezést rendelhetünk. (Egy ilyen addició azonban már megbontaná modellünk "harmóniáját" abban az értelemben, hogy az adatséma önmagában, addicionális információ nélkül, nem határozná meg egyértelműen az abból generálható riportok szerkezetét. (A gyakorlat egyébként ezt a fajta kiegészítést nem is teszi szükségessé.)).

A gyakorlati szempontokat megfelelően kielégíti az, ha csupán a τ_t rendezési relációkat kezeljük specifikusan, az összes többi rendezési reláció esetében pedig a számítógép karakterkészletéből következő természetes rendezést (a teljes, u.n. "collating sequence"-re kiterjedő lexikografikus rendezést választjuk). τ_t esetében a következő megkülönböztetéseket célszerű tenni: lexikografikus, numerikus érték szerinti, kronológikus.

Végső soron: a modell szempontjából mindegy, hogy a rendezési relációkat miként választjuk meg. A továbbiak szempontjából annyi lényeges, hogy az a)-d) pontban szereplő halmazok mindegyike rendezett.

Visszakeresés

Specifikációs adatbázisok esetén a keresés, az eredmény előállítás szempontjai, a megszokott kommerciális adatbázis alkalmazásokétól (pl. pénzügyi, gazdálkodási, raktá-

rozási stb. rendszerek) némileg eltérő. Általában nem kérdésekre keresünk választ, hanem dokumentumokat generálunk. A dokumentumok tartalma kiválasztásának és összeszerkesztésének kifejezésére elsősorban nem a logikai kalkulus formalizmusát, hanem magát az adatleíró nyelvet célszerű felhasználni. Az alábbiakban csupán egyszerű példákon keresztül érzékeltetjük azt, hogyan lehet egy ilyen lekérdező nyelvet kialakítani (egy teljes nyelvi rendszer leírása a [37] dokumentumban található meg).

1. Séma lekérdezése:

Minthogy a séma automatikus kezelésű, és a felhasználás során dinamikusan változik, kell legyen eszközünk az aktuális séma dokumentálására. Ilyenek lehetnek a következő felhasználói operációk:

?? TYPE — Az éppen létező típusok rendezett listájának előállítása.

?? SECTION — Az összes konstruálható szakaszok szerkezetének megadása.

?? <típusnév> — Adott típushoz rendelhető szakasz szerkezetének megadása.

2. Adatokból konstruált dokumentumok:

? <típusnév> — Adott típusú elemek rendezett listájának előállítása.

? <típusnév-1> <típusnév-2>

Az első típus minden egyes eleméhez az összes meglévő második típusú elemekkel való kapcsolatainak megadása.

Példa:

```
? INPUT
    ACTION
```

Eredmény: "INPUT" adatokként csoportosítva, mindazon "AKCIÓ"-k kigyűjtése, melynek a szóbanforgó adat bemenő adata.

```
? <típusnév-1>          (Természetes "join" operáció.)
     <típusnév-2>
          <típusnév-3>
```

Az előzőhöz hasonló hierarchikus kifejtés egy mélységgel tovább. (Ezt egyébként természetesen akármeddig lehet folytatni.)

Lehet magukat a kérdéseket is hierarchikusan felépíteni. Egy példán mutatjuk meg:

```
? KONFERENCIA
     RÉSZVEVŐ
        ELŐADÁSCÍM
     SZÁLLÁS
        SZOBASZÁM
        TELEFON
     stb.
```

Mindeddig a kérdéseket csupán egyszerű típusnevekből építettük fel, melyek itt egy-egy halmazt (az ilyen típusú összes elemek halmazát) reprezentáltak, a kérdés szerkezete által megjelölt kapcsolatok létezésével megszorítva. E halmazok helyett lehet eleve szűkítéseikből is kiindulni:

a) <típus><érték-1>::<érték-2>

   Intervallum kijelölése az adott típushoz tartozó rendezési reláció szerint.

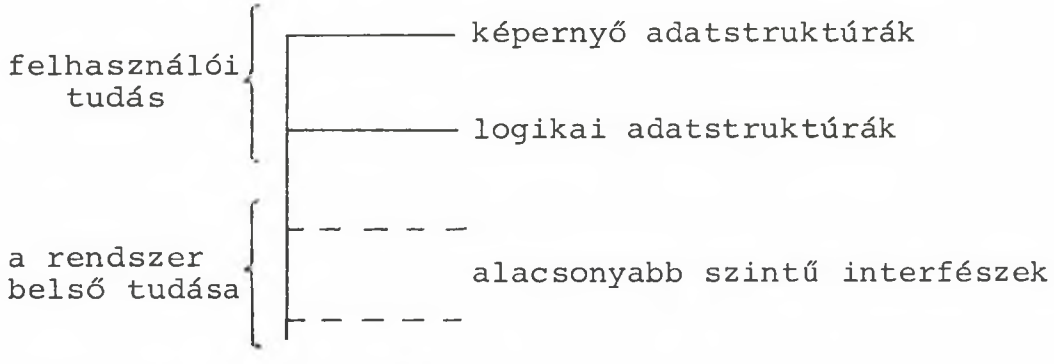
b) <típus><érték>

   Egyelemű halmaz.

A kérdezési lehetőségek további részleteitől és szemantikájának pontos leírásától itt most eltekintünk (ld.: [64]). Célunk ugyanis az adatmodellel szembeni felhasználói követelmények érzékeltetése volt. Ezzel elérkeztünk oda, hogy az u.n. logikai adatmodellt most már megalapozottan lerögzítsük:

2.1.4 Logikai adatbázis modell

A logikai adatmodell (adatbázis modell) az adatbázis sémájának, adatszerkezetének, operációinak absztrakt, lényegi részét foglalja össze. A logikai adatmodell a képernyő adatszerkezetektől eltérően nem az, amit a felhasználó a használat közben közvetlenül lát, hanem a logikai szintek közül lefelé haladva az utolsó, amelyet még ismerni kell (szabad) ahhoz, hogy az adatbázis működését megértsük. A logikai adatmodell nem tartalmazza az adattárolás fizikai megvalósításának módját, a tárolási, indexelési, keresési stb. mechanizmusokat. Ezeket már az alsóbb szintű modellek tárgyalják.



2/2. ábra

Adatszerkezetek logikai szintjei.

2.1.4.1 Séma

Az adatbázis S sémájának az $S=(T_D,R_D)$ párt nevezzük, ahol T_D a típusok, R_D pedig a relációk rendezett halmaza (korábbi definicióinknak megfelelően, ld.: típus: 2.1.1.1; reláció: 2.1.1.2; T_D,R_D: 2.1.2).

Esetünkben (definicó szerint) az adatbázis adattartalma a pillanatnyi sémát egyértelműen meghatározza. Az adatbázis inicializálásakor mind az adatbázis, mind a séma (pontosabban a T_D,R_D halmazok) üresek.

Megjegyzés: Definiciónk értelmében sem olyan típusa, sem olyan relációja a sémának nem lehet, melyhez az adatbázisban egyetlen adat ill. kapcsolat sem tartozik. Ez egy olyan invariáns tulajdonság (adatbázis integritás), melynek mindenkori teljesülését operáció definicóink garantálják.

A sémával kapcsolatos operációkat a 2.1.4.3 szakasz 2/3 ábráján szemléltetjük.

2.1.4.2 Adatbázis

Adatbázisnak a $D=(I_D,C_D)$ párt nevezzük, ahol I_D az információ elemek (atomok), C_D pedig a kapcsolatok halmaza a 2.1.1.1, 2.1.1.2, 2.1.2-beli definiciók értelmében. T_D egy osztályozást indukál I_D felett, R_D pedig C_D felett, melyekben minden osztály egy-egy rendezett halmaz.

Ez utóbbi fogalmazható úgy is, hogy a séma ill. az adatbázis egy-egy univerzum, melyeket homomorfizmus kapcsol össze (a séma ill. adatbázis operációra vonatkoztatva - bővebben ld. a 3.fejezetben).

2.1.4.3 Operációk

Adatmodellünk manipulására 28 alapvető operációra van szükségünk (melyek segítségével további más, összetettebb operációkat is megvalósíthatunk). Az alap-operációkat a 2/3 ábra foglalja össze.

		SÉMA		ADATBÁZIS	
		típus	reláció	atom	kapcsolat
STÁTUSZ	létezik: boolean	1.		15."atom"	22."connection"
	számosság: integer				
UPDATE	felvitel			17."input"	24."connect"
	törlés			18."delete"	25."disconnect"
	átnevezés			19."rename"	
VISSZA-KERESÉS	első elem				
	következő elem				28.

2/3.ábra

Alap-operációk táblázata

E táblázatban azokat a mezőket töltöttük ki, melyekhez tartozó operációknak a korábbiakban már nevet adtunk és szemantikájukat definiáltuk. A többi mezőnek is adhatunk persze valamely operáció nevet (ld. pl. [43]). Az üresen hagyott mezőkhöz tartozó operációk jelentése és szükséges paraméterei - úgy érezzük - nyilvánvalóak, ezért ezek részletezésébe itt nem megyünk bele (megtalálhatók pl. [43]-ban).

Gyakorlati szempontok szükségessé teszik a fenti operáció készlet kiegészítését olyan (ezekre visszavezethető) további operációkkal, melyek a logikai adatbázis interfész használatát célszerűbbé teszik. Ilyenek felsorolása [40]-ban található. Itt csak egyet emelünk ki, az információ elem u.n. erős törlését:

efface (t,e)

1.lépés: sorra veszünk minden olyan elemet, mely a (t,e) elemmel kapcsolatban van. Ezek közül a "delete" operáció útján töröljük mindazokat, melyek semmilyen más elemmel nincsenek kapcsolatban.

2.lépés: delete (t,e).

2.2 Típus modellek

A 2.1 fejezetben bemutatott adatmodell szándékosan a lehető legegyszerűbb volt, amely az adatbázis kezelő rendszerek e kategóriájában megadható. Az egyszerűség persze sok esetben előny is, mindenekelőtt arra gondolva, hogy a számítástechnikai rendszer felhasználóinak köre egyre szélesedik, és a mélyebb, absztraktabb ismeretek a felhasználók döntő többségétől ma már nem várhatók el.

Mindazonáltal szükség van a specifikációk, leírások kezelésének területén olyan adatmodellekre is, melyek a szemantikai jellegű információk ábrázolására gazdagabb eszközöket biztosítanak. Itt nem csupán az intelligensebb felhasználókra kell gondolni, ezek az eszközök az információfeldolgozás egyre magasabb szintű automatizálásának előfeltételei.

2.2.1 Altípusok

Az első lépés az ismertetett adatmodell gazdagítására a típusok T_D halmazában egy u.n. típus-altípus reláció bevezetése. Ennek szükségességét egyszerű példákon mutatjuk be. Tekintsük a következő két információ elemet:

FIZIKUS: FRANKLIN BENJAMIN
ELNÖK: FRANKLIN BENJAMIN

A felhasználó egyelőre nem tudhatja, hogy e két személy azonos-e, vagy sem. Mindenesetre, definiciónk értelmében (ld. 2.1.1.1) a két információ elem különböző.
Vagy: Tekintsük a következő két információ szakaszt:

INTÉZMÉNY: SZÁMÍTÁSTECHNIKAI ÉS DÍJBESZEDŐ VÁLLALAT.
IGAZGATÓ: KOVÁCS GÉZA.

SZEMÉLY: KOVÁCS GÉZA.
ÉLETKOR: 45.

Mivel most is hiányzik az adatbázisból az az információ, hogy minden "IGAZGATÓ" egyúttal "SZEMÉLY" is, eddigi eszközeinkkel e két szakasz közt nem tudunk kapcsolatot teremteni (pl. join-operációt végezni).

Legyen most $\succ$ egy hierarchikus részben rendezési reláció (fa vagy erdő) T_D felett. Ha a $t, t' \in T_D$ tipusokra $t \succ t'$, akkor azt fogjuk mondani, hogy minden t' típusú információ elem egyúttal t tipusú is (annak egy speciális változata).

Valamely $t \in T_D$-re jelölje $[t]$ a maximális típust, melynek t altípusa. (Ilyen minden T típushoz egy és csak egy létezik, minthogy hierarchiáról van szó.) Ezek után az (A1) azonossági axiomát a következő módosított formában mondjuk ki:

(A3) Az $i_1 = (t_1, e_1)$ és $i_2 = (t_2, e_2)$ információ elemeket akkor és csak akkor tekintjük azonosnak, ha

$[t_1] = [t_2]$ és $e_1 = e_2$.

Megjegyzés: látszólag előnyös volna most további korlátozást nem tenni (azaz a lehetőségeket gazdagabbnak hagyni). (A3) ugyanis lehetővé teszi még pl. a következőt:

Példa:

legyen $T_D = \{$SZEMÉLY,
FIZIKUS,
ELNÖK$\}$

és FIZIKUS $\prec$ SZEMÉLY
ELNÖK $\prec$ SZEMÉLY a rendezés.

Ekkor a két információ elem:

FIZIKUS: FRANKLIN BENJAMIN.
ELNÖK: FRANKLIN BENJAMIN.

azonossága (A3) által kétségtelenül garantálva van.

A példa által illusztrált problémát azonban helyesen mégsem így kell megoldani.A (SZEMÉLY, FRANKLIN BENJAMIN) információ elem legszűkebb típusa ugyanis ezesetben nem egyértelmű. Az absztrakt adatszerkezetekkel kapcsolatos évtizedes gyakorlati tapasztalatok igazolták, hogy az u.n. "szigorú típuskezelés" (strong typing) által nyert előnyök (világosabb és konzisztensebb modell, szigorúbb ellenőrzések) a felhasználás során többet kamatoznak, mint a gazdagabb lehetőségek által megengedett "lazaságok".

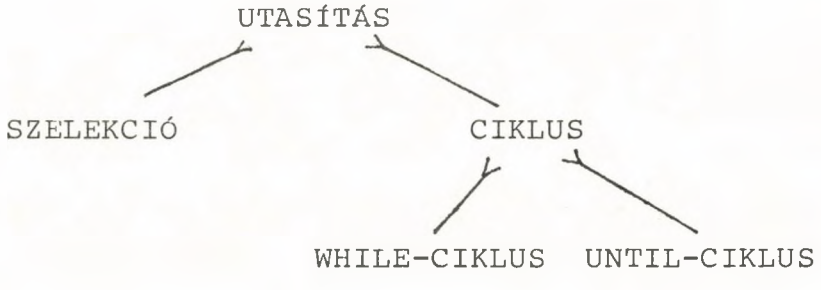
Jelölje most $t_{min}(t,e)$ a legkisebb olyan t' típust, melyre (A3) értelmében $(t,e)=t',e)$, amennyiben ilyen egyértelműen létezik. Legyen most (t,e) egy új információ elem $(t,e) \notin I_D$), és tegyük fel, hogy $([t],e) \in I_D$ és $t_{min}([t],e)$ létezik. Ekkor a (t,e) információ elemet illegálisnak nevezzük, ha t és $t_{min}([t],e)$ a $\succ-$ reláció szerint nem összehasonlíthatók.

(A4) Az adatbázis illegális elemmel nem bővíthető.

(Világos, hogy kezdeti állapotban, mikor I_D üres, t_{min} létezik I_D minden elemére. (A4) pedig garantálja azt, hogy t_{min} továbbra is létezni fog I_D minden elemére.)

Példa:

Legyen T_D a következő:



Ha most (UTASÍTÁS, U) egy információ elem, akkor ez az U érték még bármelyik altípusként is legális. Ha azonban egyszer mondjuk bekerült I_D-be a (WHILE-CIKLUS,U) információ elem is, akkor ezentúl U nem minősíthető pl. "SZELEKCIÓ"-nak, azaz a (SZELEKCIÓ,U) információ elem illegális.

A típus-altípus viszony az operációkat is érinti. Erről a kérdésről külön fejezetben szólunk (ld. 3.).

Altípusok alkalmazása esetén a felhasználói interfészen is változtatni kell. Ennek egyszerű módja az, ha minden sorban a sor t típusa után [t]-t is feltüntetjük (akkor, amikor $t \neq [t]$). Ezt először a felhasználó végezheti (ezzel definiálja a $>\!-$ relációt), majd $t \neq [t]$ esetén a továbbiakban a rendszer automatikusan kijelzi. Például:

```
INTÉZMÉNY: ENSZ
    ELNÖK: SZEMÉLY: HAMMARSKJÖLD
```

2.2.2 Reláció nevek

Az (A2) axioma értelmében egy relációt a részvevő két típus határozott meg (egyértelműen). Ennek feloldásával egy lényegesen árnyaltabb modellhez juthatunk.

Legyen N egy tetszés szerinti halmaz "a nevek halmaza", egy kitüntetett $n_\emptyset$ elemmel, melyet a "névtelennek" nevezünk. Rendeljünk hozzá minden (i_1,i_2) kapcsolathoz egy $n(i_1,i_2) \in N$ elemet. (Változatlanul: $(i_1,i_2) = (i_2,i_1)$ definiciónk szerint.) Ezek után (A2)-t a következő axiomával helyettesítjük:

(A5) Az (i_1,i_2), (i_3,i_4) kapcsolatok akkor és csak akkor azonosak, ha (A2) feltétele teljesül (azaz vagy $i_1=i_3$, $i_2=i_4$ vagy $i_1=i_4$, $i_2=i_3$) és $n(i_1,i_2) = n(i_3,i_4)$

Az ugyanolyan két típus közötti és azonos nevű kapcsolatok halmazát az ezzel a névvel ellátott relációnak nevezzük. A relációk tehát szimbolikusan

$n(t_1,t_2)$ alakban írhatók, ahol $n \in N$,

$t_1,t_2 \in T_D$.

A (t_1,t_2) párt az $n(t_1,t_2)$ reláció, és egyúttal az őt alkotó kapcsolatok "szignatúrájának" nevezzük.

Az ilyen módon gazdagított adatmodell a felhasználói nyelvben például a következő módon alkalmazható. Engedjük meg, hogy az információ szakaszok szakasz-belső soraiba reláció neveket lehessen írni, mondjuk így:

```
VÁLLALAT: ALFÖLDI KŐBÁNYA.
    (IGAZGATÓ)  SZEMÉLY: FÖLDES FERENC.
    (PÉNZTÁROS) SZEMÉLY: KÖVES GÉZA.
    (ELLENŐR)   SZEMÉLY: KÖVES GÉZA.
    CÍM: 1056 PF. 63.
```

(A névvel el nem látott kapcsolatoknak a "névtelen" $n_0 \in N$ elemet feleltetve meg.) Ilyen módon, többi halmazainkhoz hasonlóan N is a felhasználói dialógusból építhető fel automatikusan, és a rendszer által megjelenített szakasz-képekben a nevek szintén automatikusan jeleníthetők meg.

Megjegyzés: A figyelmes olvasó nyilván végiggondolta már azt, hogy ugyanezt a problémát egy másik módon is meg lehetett volna közelíteni. Beszélhettünk volna információ elemek szakaszokban betöltött "szerepeiről" (mint azt pl. a [24] javaslat teszi). A különbség annyi, hogy a szerepeknek "irányuk" van (tehát ez esetben inverz szerepekre is szükség van, pl.

```
INSTITUTE:                          PERSON:
   (DIRECTOR) PERSON:     ill.        (DIRECTOR-OF) INSTITUTE:
```

Ezt a megkülönböztetést mi nem érezzük szükségesnek, mivel a szimmetria fenntartása által nyújtott gyakorlati előnyök nagyobbak.

2.2.3 Altípusú egyedek

Az altípusok használatára vonatkozóan most egy finomabb megkülönböztetést is kell tennünk. Nem mindegy, hogy egy adott szakaszban (tehát relációban) egy szakaszon belüli poziciót

a) eleve végérvényesen és kötelezően valamely altípusba tartozónak deklarálunk,

b) vagy esetenként kívánunk ilyen vagy olyan (a rendezés szerint esetleg egymással össze sem hasonlítható) altípusokba tartozó elemeket szerepeltetni (egy főtípuson belül).

Eddigi formalizmusainkkal ez akkor jelent problémát, ha felvitelkor az elem szűkebb típusát meg kívánjuk adni, azt azonban nem kívánjuk, hogy az adott helyen ez az altípus kötelező legyen. Például a

```
LÉGIJÁRAT: MA 741.
   (PILÓTA) NŐ: MAGOS KATALIN.
```

szakasz megadása után férfiak már más járatokat sem vezethetnének.

E felviteli probléma megoldására a következő formalizmust vezethetjük be. A relációbeli szerep számára megadott típus mellett zárójelben adjuk meg az elem aktuális típusát (ha ezt szükségesnek tartjuk). Esetünkben:

```
LÉGIJÁRAT: MA 741.
   (PILÓTA) SZEMÉLY: (NŐ) MAGOS KATALIN.
```

2.2.4 Záró megjegyzések

A 2.1.4 szakaszban összefoglalt alapmodell gazdagítására a most bemutatottakon kívül számos egyéb lehetőség is nyílik (pl. általános relációk mellett függvénykapcsolatok bevezetése, bináris relációk mellett általános n-es relációk megengedése, egymásba skatulyázott

hierarchikus input szakasz szerkezetek bevezetése stb.). A józan ész azonban a fantázia fegyelmezését követeli. (Túl könnyen csúszunk át mindannyian arra a lejtőre, mely feleslegesen túlbonyolított, következetlen és karbantarthatatlan számítástechnikai rendszerek előállításához vezet.) Ezen fenti, és más további szemantikai követelmények már egy más természetű modellt igényelnek, mellyel a következő, 2.3 alfejezet foglalkozik.

Az eddig tárgyalt modell és 2.2-beli kiegészítései lezárják azt a részt, ameddig öndefiniáló sémáról (a gép által automatikusan és dinamikusan kezelt sémáról) beszélhetünk. Mint mondtuk: az eddig bemutatott modellek csupán koncepcionális modellek: a rendszer funkciója oldaláról a lényeget ragadták ki. Az ezeket a funkciókat ténylegesen megvalósító számítástechnikai rendszer (adatbázis kezelő rendszer) kivitelezéséhez még igen nagy számú további "részletkérdés" megoldása szükséges. Ezek a [38], [42], [43], [37] dokumentumokban találhatók meg. Ebben az értekezésben (a későbbi fejezetekben) ezek közül csak a leglényegesebb elemeket (pl. szoftver architektúra) fogjuk összefoglalni.

Végezetül még egy megjegyzés. Egy számítástechnikai rendszer koncepciójának átgondolásától a gyakorlati felhasználók által is kipróbáltan, minden igényt kielégítően jól működő, befejezett rendszerig vezető út nagyon hosszú (sőt, elviselhetetlenül hosszú – ez tulajdonképpen a sokat emlegetett "szoftver krízis"). Az első próbaüzemek idején mindíg számtalan olyan új szempont ("részletkérdés") merül fel, melyre a tervezés során még nem is lehetett gondolni. A ráfordítások aránya az első, az előírt funkciót hiánytalanul megvalósító, bemutatható, működő rendszer, és az idegen felhasználónak megbízhatóan átadható, valódi termékszintű szoftver között általában egy a tízhez. A köszönet munkatársaimat illeti meg, akikkel együtt az öndefiniáló sémakezelésű adatbázis koncepciót illetően (az u.n. "MDB" adatbázis kezelő rendszerek különböző változatai) sikerült idáig is eljutni.

2.3 Az u.n. "concept" modell

Ezen adatmodell előzményeinek a SIMULA 67 "class" [14], a Smalltalk "object" [23], ill. a PSL/PSA [65] és az ERA [12] adatmodelljei tekinthetők. Az első kettő gondolati hátterében a procedurális alkalmazások voltak, így adatbázis kezelési oldaluk nem volt. Utóbbi kettő pedig nélkülözte az előbbiekben lévő szemantikai kifejező erőt. A "concept" modell egy kisérlet a "class", ill. az "object"-nek megfeleltethető fogalmi rendszer megadására az adatbázis kezelési környezetben. Az így létrejött modell számos publikációban már igen részletesen közlésre került (pl.[30], [33], [15]), ezért itt legfontosabb elemeinek összefoglalására szorítkozunk.

Az u.n. "concept" modell két részből tevődik össze:

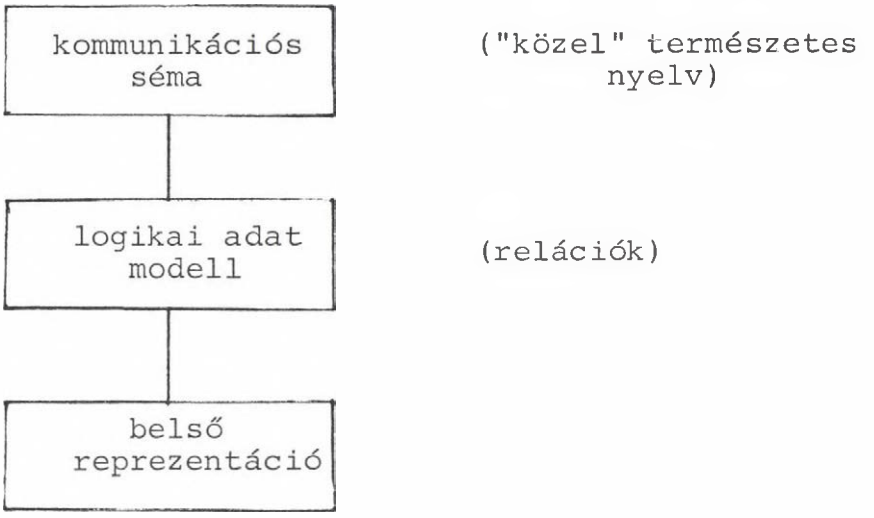
- adatmodell,
- nyelvi rendszer.

Az adatmodell egy relációs típusú modell, melynek a hagyományos modelltől való fő eltérései az alábbiak:

- típusokra alapul (mint az előző alfejezetek modelljei)
- referencia jellegű (azaz a relációk elemei maguk is relációk, melynek sajátos következményei vannak: pl. 0-oszlopos relációk, u.n. "zoom" és "ascent" referencia műveletek stb.),
- információ finomítást tesz lehetővé (típus-altípus viszonyok).

2.3.1 Felépítési séma

A szerkezeti séma a már megismertekhez hasonló itt is:



2/4. ábra

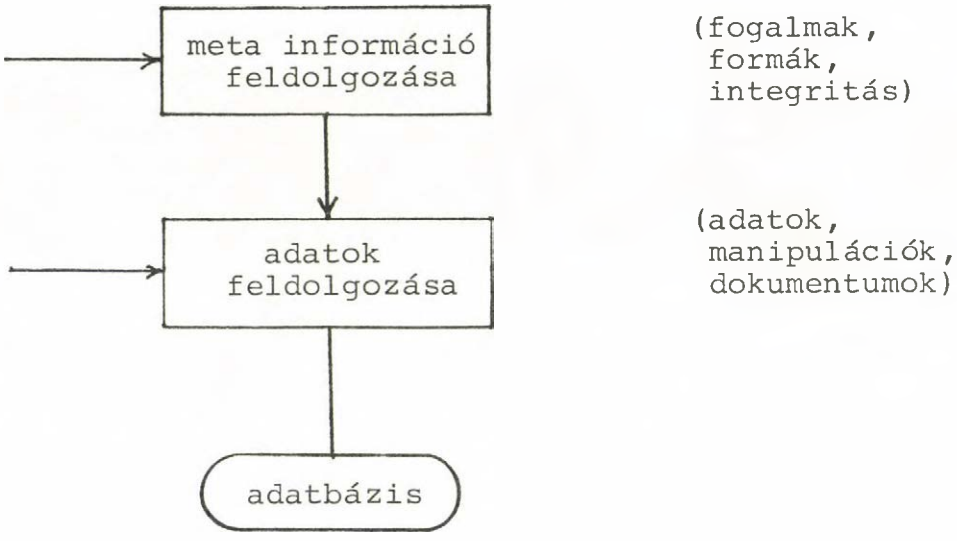
Szerkezeti és interfész séma

ezzel szemben a működési séma a már ismert modellétől eltérő, u.n. kétszintes séma. Az első az u.n. definiciós szint, mely

- fogalomdefiniciókat,
- kényszerítési és integritási szabályokat,
- nyelvdefiniciót

dolgoz fel. Ezzel a felhasználó saját maga teremt egyfajta "világot", készítendő leírásai, specifikáció, dokumentációi számára.

A második szinten az első szint fogalmaival kifejezhető, a megadott nyelven megfogalmazható leírásokat dolgozunk fel. Az első szint a másodikat "vezérli". Csupán az adatbázis kezelési oldalra koncentrálva pedig azt is mondhatjuk: Az első szint az adatbázis séma szintje, a második szint pedig a konkrét adatok szintje:



2/5. ábra

Vezérlési séma

2.3.2 Logikai adatmodell

2.3.2.1 Fogalom definició

Az első lépés, mielőtt bármiről számítógéppel feldolgozható leírást, specifikációt készíthetnénk, a leírandó jelenségkört jellemző fogalomkör azonosítása és formális megadása. A fogalmak megadását mi u.n. "attributumok" segítségével végezzük. Az attributumok

("szelektor", "típus")

párok, melyek megadása a következő formában történik:

concept fogalomnév (szelektor-1: típusnév-1,...
... szelektor-n: típusnév-n);

ahol n nemnegatív, szelektor-i tetszés szerinti azonosító, típusnév-i pedig (vagy u.n. értéktípus-kulcsszó, neve-

zetesen INTEGER, REAL, TEXT, BOOELAN – a továbbiakban azonban az értéktípusoktól, mint kevéssé érdekes esettől eltekintünk; vagy) olyan fogalomnak a neve, mely "concept"-ként szintén definiálásra került. (Ez az u.n. zártsági axioma, melyet a továbbiakban (A6)-tal jelölünk.)

Példák:

```
concept module;
concept data (part-of:data);
concept condition (data-associated:data);
concept precondition (to-activate:module,when:condition);
concept postcondition (termination-of:module,
                                    resulting-in:condition);
concept invariant (condition, associated:module);
```

(Megjegyzések:

a) Ismert technika a szelektornevek elhagyása, ahol a típusnév egyben szelektorként is szolgálhat, ld. fent.

b) Az attributumok száma természetesen lehet nulla (ez esetben a típusba tartozó adatpéldányok u.n. "nulla oszlopszámú relációt" alkotnak majd.))

(A7) Fogalmak azonossága: A fogalmakat neveik azonosítják. (Azaz két fogalom mindíg különböző.)

2.3.2.2 Altípusok

Bármely fogalomnak (concept, típus) tetszés szerinti számú altípusát lehet definiálni. (A típus-altípus viszonyt hierarchikusnak tekintjük. Ez nem szükségszerű, lehetne háló, sőt még általánosabb részben rendezés is. A típusok tekintetében azonban a gyakorlat a fánál általánosabb struktúrát nem indokol. Ez természetesen nem tévesztendő össze maguknak az adatszerkezeteknek egymáshoz való viszonyával.)

A fogalom definició általános formája ezek után a következő:

concept fogalomnév is ősfogalom (extra attributumok listája);

Értelmezés:

Az altípus örökli mindazokat az attributumokat, amelyekkel az "ős" rendelkezett (tranzitíve természetesen), és ezen kívül rendelkezik még azokkal az extra attributumokkal, melyek e speciális fajtára vonatkozóan a listán külön megadásra kerültek. (Ez az elv azonos a SIMULA 67 [14]-ben programozási nyelvekre alkalmazott típus finomítási elvvel, az alábbi eltéréssel.)

A típusok tehát egy erdőt alkotnak. Célszerűségi okokból ezt mi kiegészítjük fává, egy "ideális" elem hozzávételével: Legyen

concept universal;

az egyetlen eleve létező, u.n. "rendszer típus", melynek minden fogalom altípusa. Az "is" részt nem tartalmazó fogalom definiciókat tehát így is írhatnánk:

concept név is universal (stb.);

Példák:

(1) concept process;
concept manual-process is process;
concept computerized-process is process;

(2) concept file (opening:procedure,blocked:boolean);
concept printfile is file (page-size:integer,
line-length:integer);

(3) concept input (process data);
concept usage is input;
concept control is input;

Az altípusok bevezetésének fő célja az adatokon végrehajtható árnyaltabb típus illeszkedési ellenőrzés lehetővé tétele. Ld. később.

2.3.2.3 Adat objektumok

Tegyük fel, hogy rendelkezésünkre áll egy zárt definiciós egység: fogalmak egy (külső hivatkozást nem tartalmazó) halmaza. Ekkor a fogalmaknak "példányait", adat objektumokat" adhatunk meg, pl. a következő módon:

fogalomnév objektumnév (attributumok);

ahol "fogalomnév" egy definiált fogalom neve, "objektumnév" tetszés szerinti azonosító, az "attributumok" pedig más objektumok neveinek (vesszővel elválasztott) sorozata.

Zártsági szabály:

(A8) Az attributum értékekként megadott objektumnevek létező objektumok nevei kell legyenek. (Ez célszerűségi okokból nem vonatkozik a feldolgozás közbeni ideiglenes állapotokra, ld. [30].)

Illeszkedési szabályok:

(A9) Az attributum értékekként megadott objektumok száma azonos kell legyen a fogalom összes (beleértve az örökölt attributumokat is) attributumainak számával.

(A10) Egy attributum értékként megadott objektum típusa vagy azonos kell legyen a fogalom definicióban megjelölt attributum típussal, vagy annak valamely altípusa kell legyen.

Példa: A

concept input (process, data);
concept control is input;

fogalmakra vonatkoztatva az alábbiak korrekt adat objektumok:

process specifikáció-feldolgozás;
data meta-információ;
control feldolgozás-vezérlés (specifikáció-feldolgozás, meta-információ);

Végül adat objektumok azonosságáról szólunk:

(A11) Az adat objektumokat nevük azonosítja. (Tartalmuk nem. Később beszélni fogunk "névtelen" objektumokról is, ezeket azonban úgy tekintjük, hogy "belső nevük" van, így szabályunkat nem sértik.)

2.3.2.4 Relációk

Minden fogalomhoz rendelhetünk egy relációt, melyet az ilyen típusú objektumok halmaza alkot. A reláció oszlopait az attributum értékek (pontosabban az értékként megadott objektumokra vonatkozó hivatkozások alkotják). Mivel azonban modellünkben altípusok is szerepelnek, fenti definiciónkat pontosítani kell:

Referencia_reláció

Egy adott típushoz (fogalomhoz) tartozó reláció alatt az ilyen típusú, és az e típus altípusaiba tartozó összes objektumok halmazát értjük a következő értelemben. A relációnak annyi oszlopa van, ahány attributuma az adott típusnak. Az altípusokba tartozó objektumok esetén ezen objektumoknak csak a fenti oszlopokra vonatkozó

projekcióit tekintjük. (Ennél még pontosabb definiciót a 3.fejezetben fogunk adni.)

Megjegyzés: Sok olyan hagyományos számítástechnikai alkalmazási terület van (pl. mesterséges intelligencia szakértői rendszerei, szimulációs lista kezelés, logikai programozás stb.), melyek régóta (már az adatbázis kezelő rendszerek fogalmának létrejötte előtt is) kezeltek igen bonyolult szerkezetű és logikai összefüggésű (absztrakt) adatszerkezeteket. Ezt (a programozási nyelvek automatizmusai által a fizikai objektumokra végzett leképezést) ma egyfajta "inherens", "naív" adatbázis kezelésnek kell tekintenünk, mely addig működik, míg az adatok mennyisége nem lép át egy kritikus határt. Ahhoz, hogy a szó valódi értelmében adatbázis kezelésről beszélhessünk, arra van szükség, hogy nemcsak az egyedi, hanem a tömeges adatokra vonatkozóan is (reláció, set stb.) modellünk és operációink legyenek. A referencia reláció fogalma ebben az értelemben tekinthető pl. a SIMULA [14] objektumok tömeges adatokra vonatkozó kiterjesztésének is.

Referencia relációban értelmezhetők a szokásos reláció algebrai műveletek. Noha ez a triviális úton nem végezhető el (szükség van egy, a Herbrand univerzumhoz, vagy az u.n. "term-algebra"-hoz hasonló konstrukcióra, ld. 3.fejezet), most nem ezt, hanem a "szokatlanabb", a referencia jellegű műveleteket mutatjuk be. (Megjegyezzük, hogy ezek a műveletek elsődlegesen nem felhasználói célokat szolgálnak, hanem azért szükségesek, mert nélkülük bizonyos fontos logikai kifejezés-típusoknak nem volnának reláció kifejezésekben megfelelői.)

Referencia műveletek

1. Nagyítás ("zoom")

Formája:

reláció.szelektor

vagy általánosságban:

relációkifejezés.oszlopkijelölés

Jelentése:
....

Az operandusként megadott reláció megjelölt oszlopát kiválasztjuk (a többi oszlopot elejtjük), majd a megadott oszlop elemeit hivatkozási tartalmuk szerint (egy lépésben) "kifejtjük" (azaz a hivatkozott objektumokkal helyettesítjük).

Példa:
.....

Legyen a "FÉRFI" reláció az alábbi:

FÉRFI

	apja	felesége
István	→ János	→ Teréz
János	→ Géza	→ Márta
Péter	→ János	→ Judit
Géza	→ "ismeretlen"	→ Eszter
Ferenc	→ Géza	→ Mária

ekkor a "férfi.apja" művelet eredménye:

János	→ Géza	→ Márta
Géza	→ "ismeretlen"	→ Eszter

és a "férfi.apja.apja" művelet eredménye pedig:

Géza	→ "ismeretlen"	→ Eszter

2. Kicsinyítés

Jelölése: szögletes zárójel:

[reláció]

Jelentése:

Az R reláció "kicsinyítése" olyan egyoszlopos relációt jelent, melynek egyetlen oszlopa R-típusú, és elemei rende R soraira mutatnak.

(Világos,hogy [R].1 = R .)

2.3.2.5 Adatbázis integritás

A "concept" modell által megvalósított adatbázisok elsődleges integritási tulajdonságai ("primitív invariánsok"), azok a tulajdonságok, amelyekről a korábbiakban már szóltunk: zártság (2.3.2.1 és 2.3.2.3), típusilleszkedés (2.3.2.2). Ezeket az invariánsokat maguk az adatbázis operációk garantálják (ld. 3.fejezet). Most további eszközöket mutatunk be.

Automatikus implikációk

Mint tudjuk, egy adatbázis objektumai (ill. az objektumokból alkotott kifejezések) mindíg interpretálhatók úgy is, mint valamilyen logika itéletei ill. predikátumai.

Például az

output (signal S, process P)

adatbázis objektum interpretálható úgy is, hogy a "P folyamat az S üzenetet küldi" itélet igaz. Tételezzük fel most például a következő definiciókat:

concept output (signal, process);

concept alarm is output (delay-time: real);

Ez esetben, ha mondjuk az

alarm (áramkiesési-jel, stabilizátor, 0)

objektum az adatbázisban van, vagyis igaz, hogy "a stabilizátor riasztó jelet ad", akkor egyúttal az is igaz, hogy "a stabilizátor output-ot ad", minthogy az "alarm" típust az "output" fogalom altípusaként definiáltuk. (Vegyük még észre, hogy 2.3.2.4-beli általános reláció definiciónk is ezzel a logikai szemlélettel ban összhangban.)

Deklarált kényszerítések

Lehetőség van arra, hogy más implikációkat (melyek automatikusan nem jönnek létre) alkalmas formában a definiciós szinten deklaráljunk. Ennek formáját egy példán mutatjuk be:

```
concept use-to-maintain(used:information,
                        user:process,
                        maintained:relation);
    implies maintain (user, maintained);
    implies use (used, user);
    etc.
```

ahol feltettük, hogy "use" és "maintain" szintén definiált fogalmak.

E deklarációknak az a következménye, hogy valahányszor az adatbázisba egy új "use-to-maintain" típusú objektumot viszünk fel, az implikáció által kijelölt további objektumok (megfelelő mechanizmus útján) szintén automatikusan létrejönnek.

Függőségek

Az objektumokra vonatkozó azonossági axiománk értelmében az attributum értékek az adatobjektumokat általában nem határozzák meg (nem azonosítjuk). Előfordulhat azonban,

hogy bizonyos fogalmak esetén úgy kívánunk rendelkezni, hogy valamely megadott attributum csoport (esetleg az összes együtt) legyen azonosító az objektum számára (azaz, kulcs a relációban). Ennek megadási módja a következő (példák):

```
concept feldolgoz (adat, folyamat);
    function;
```

deklarálja azt, hogy a teljes attributum együttes az objektumot azonosítja; ill.:

```
concept fájl(cím, hossz, blokkméret, mód);
    function-of cím;
```

a "cím" attributumot kulcsként jelöli meg.

E függőségek deklarálásának következménye az, hogy amennyiben a rendszerbe olyan adat érkezik, mely valamely függőségi viszonyt sért, akkor az elutazásra kerül.

Reláció tulajdonságok

A bináris reláció tulajdonságok (pl. antiszimmetria, irreflexivitás, részben rendezés, háló, fa stb.) gyakran igen jól alkalmazhatók az adatokban rejlő szemantikai összefüggések kifejezésére. Ezért e tulajdonságok definiciós szinten való deklarálására is célszerű lehetőséget biztosítani. (Ezt nem részletezzük, ld. [30].)

2.3.3 Nyelvi rendszer

A harmadik definició csoport, amit a definiciós szinten kell megadnunk, magának a leíró nyelvnek a meghatározása, amely segítségével megadott fogalmainkat kifejezzük. Mint ismeretes, nyelvek definiálására számos elterjedt módszer létezik. Mi most mégis egy céljainknak megfelelő, de szokatlanabb módot választunk. Ennek első lépése a következő:

2.3.3.1 Formák

Mivel minden struktúrált leírás (mint már a 2.1 alfejezetben is láttuk) célszerűen blokkokból (esetleg egymásba skatulyázott blokkok hierarchiájából) épül fel, az egyes fogalmak kifejezési módjára vonatkozóan ésszerű megkülönböztetéseket tenni aszerint, hogy milyen szövegösszefüggésben hangzanak el. A szövegösszefüggéseket a fogalmak attributumai szerint különítjük el. A fogalmak kifejezésére szolgáló mondatformák deklarációjának módját e szellemben egy példán mutatjuk be:

```
concept use-to-derive    (used:data, user:process,
                                  derived:data);
    form user: uses data to-derive derived;
    form used:used-by user to-derive derived;
    form derived: derived-by user using used;
```

(A mondatformák természetesen az angol helyett bármely más, tetszés szerinti nyelven is megadhatók, függetlenül a fogalom definiált nevétől.) E formák jelentése mármost a következő:

Az első forma a "user", tehát az attributum deklaráció szerint egy "process" nézőpontjából hangozhat el. Ha tehát a leírásban valamely helyen éppen egy P folyamatról beszélve mondunk el rá vonatkozó állításokat, akkor ezek között szabad az elsőként definiált mondatot használni, pl.:

```
process P;
   uses D to-derive E;
```

Hasonlóan, ugyanezt az állítást a fordított kontextusokban a másik két forma segítségével fejezhetjük ki. (Példáinkban az első sor, a "szakaszfej", a kontextust fejezi ki.)

```
data D;
   used-by P to-derive E;
```

illetve:

```
data E;
   derived-by P using D;
```

(A logikai adatsémában mindhárom fenti állításunk a "use-to-derive (D,P,E)" objektumnak felel meg.) Ezek után a leírások feldolgozása általánosságban a következőképpen történik:

2.3.3.2 Nézőpont verem

A leírásokban ezúttal tetszés szerinti mélységű hierarchikus szerkezetet (egymásba skatulyázott blokkokat, azaz mellé- és alárendelt mondatszerkezeteket engedünk meg). Egy tipikus leírás részlet lehet pl. a következő:

```
1. module M;
2.     part-of subsystem S;
3.     generates information I1, I2;
4.         using database D;
5.             via interface F;
6. accesses file J mode read only;
```

Feldolgozás:

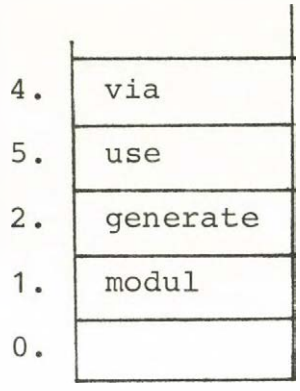
1. Az első sor elfogadásra kerül, ha "module" egy definiált fogalom neve.

2. A második mondat akkor elfogadható, ha az egy olyan deklarált forma, melynek definicióját "module" nézőpontból mondtuk ki.

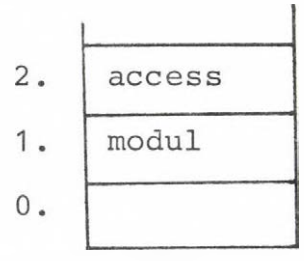
3. A harmadik mondat feldolgozásakor a rendszer először megvizsgálja, vajon az érvényes mondatformának tekinthető-e a legutóbb elhangzott (a "part-of") nézőpont felől. Ha ilyen mondatforma e nézőpontból nincsen deklarálva, akkor egy blokk-mélységgel kíjjebb lép és megvizsgálja, hogy a "module" nézőpontból elfogadható-e. Tegyük fel, hogy igen. Ez esetben a külső blokkban továbbra is a "module" nézőpont marad érvényes, míg a második szinten a "generate" nézőpontba kerülünk. Ekkor fogadjuk a negyedik mondatot:

4. Tegyük fel, hogy ez a "generate" nézőpontból kerül elfogadásra.

5. Tegyük fel, hogy a "use" nézőpontból érvényes forma. Most mindezek elfogadása után a blokkszintek pillanatnyi nézőpontjai a következők:



6. A hatodik mondat feldolgozásakor e vermet felülről sorra vizsgáljuk, mindaddig,míg olyan nézőpontot nem találunk, melyből, mint kontextusból mondatunk elfogadható (azaz deklarált mondatforma). E ciklus során a nem megfelelő nézőpontokat a veremből eltávolítjuk, majd az elfogadott nézőpontot a verem tetejére helyezzük. A hatodik mondat feldolgozása után tehát a verem állapota a következő:



stb. Mint azt példánkkal érzékeltettük, egy leírás mindíg mellé- és alárendelt mondatok sorozatából áll (ugyanúgy, mint mindennapi beszédünk), melynek feldolgozását egy verem (az u.n. "nézőpont verem") vezérli. A verem fent vázolt működése a gyakorlatban egy igen hasznos ellenőrzést, az u.n. "szövegösszefüggés ellenőrzést" teszi lehetővé. Ez a felhasználói dialógusban a következőképpen valósul meg:

Mint láttuk, a leírás blokkjainak vizuális képe hierarchikus. A felhasználónak azonban nem kell saját magának pozi

cionálnia sorait, hanem egyszerűen a sor elején kezdi el írni mindegyiket. A fent bemutatott elemzési mechanizmus során az érvényes (felülről az első érvényes) kontextus megtalálása után, maga a rendszer a soron belül a mondatot a hierarchiának megfelelő pozicióba eltolja. Ez a működési mód azért hasznos, mert "szembetűnő": a rendszer használója azonnal vizuálisan érzékeli, mihelyt valamilyen állítást nem a megfelelő kontextusban használt.

2.3.3.3 Riport generálás

A "concept" adatmodell vonatkozásában kétféle lekérdező rendszerről lehet beszélni. Közvetlenül a logikai adatmodell fogalmaira felépíthető egy relációs lekérdező rendszer, mely a reláció algebra (referencia reláció algebra) operációira épül fel. Egy ehhez tartozó felhasználói interfész kialakítása nem különösebben nehéz, ezért itt nem tárgyaljuk (ld. [30]), magukat az operáció definiciókat pedig a 3.fejezetben ismertetjük.

Egy másik lekérdező rendszer (a riport generátor rendszer) a felhasználói nyelvre építhető. Ez megvalósítható olyan módon, hogy a felhasználó a kérdéseit (tehát a riport specifikációkat) ugyanazon a nyelven fogalmazza meg, mint amelyen leírásait készítette (vagyis azon a nyelven, melyet éppen ő maga definiált). Ez a módszer nemcsak azért előnyös, mert ezt a nyelvet tudja használni a "legotthonosabban", hanem mert a generálandó dokumentumok tartalmát és formáját egyidejűleg, egy lépésben lehet megadni. Az alábbiakban ennek elemeit mutatjuk be néhány egyszerűbb példa segítségével. Az alapforma nagyon hasonló ahhoz, amit a 2.1 alfejezetben is alkalmaztunk: Mondatformák egy tetszés szerinti hierarchiáját adjuk meg, pl.

```
REPORT
<forma - 1>
     <forma - 1.1>
     <forma - 1.2>
          <forma - 1.2.1>
     <forma - 1.3>
      stb.
END
```

Maga a kérdésfeltevés akkor korrekt (elfogadható), ha a benne szereplő mondatformák a már megismert "nézőpont verem"-mechanizmus szerinti feldolgozásban mind legálisak.

Most tegyük fel először, hogy a kérdésben lévő mondatformákban értékek nem szerepelnek. (Az értékek helye üres, vagy mondjuk megállapodhatunk abban, hogy egyöntetűen az értékek helyére egy speciális szimbólumot, pl. az "any" szimbólumot írjuk. Például:)

```
REPORT
 data any;
    created by process any;
            using information any;
    accessed by process any;
END
```

A specifikált riport megszerkesztése az ismert módon történik: sorra veszünk minden egyes "data" objektumot; mindegyik esetében megadjuk a "create" kapcsolatokat, ezen belül pedig a "using" kapcsolatokat is egyenként; amikor ez utóbbiakat befejeztük, megadjuk a szóbanforgó adatra vonatkozóan az összes "access" kapcsolatokat stb.

Az "any" szimbólum tehát az univerzális kvantor szerepét játssza. Ezt általánosítva, lehetőség van szabad változók használatára is, pl. "any X", mellyel az attributum értékek között megszorító kapcsolatokat jelölhetünk ki. Végül, megadhatunk az attributum értékek helyén objektum neveket is (egyelemű halmazokat reprezentálva).

Lehet halmaz specifikációkat is definiálni ugyan ezen a nyelven, majd e halmazokat felhasználni a változók helyén. (Ennek részleteit az olvasó [44]-ben találhatja meg.)

3. FORMÁLIS MODELLEK

3.1 Hivatkozási kalkulus

A típus-altípus viszonnyal felruházott referencia relációk a reláció algebrai műveletek kiterjesztésében számos nyitott kérdést vetnek fel. (Erre igazán akkor derült fény, amikor a hagyományos kalkulus kellő ismeretében az operációk számítógépen való kivitelezéséhez az ismert programozói magabiztossággal hozzákezdtünk.) Ebben az alfejezetben azt a formális alapot (egy lehetséges változatot) adjuk meg, mely az operációk programozását már biztonsággal lehetővé teszi.

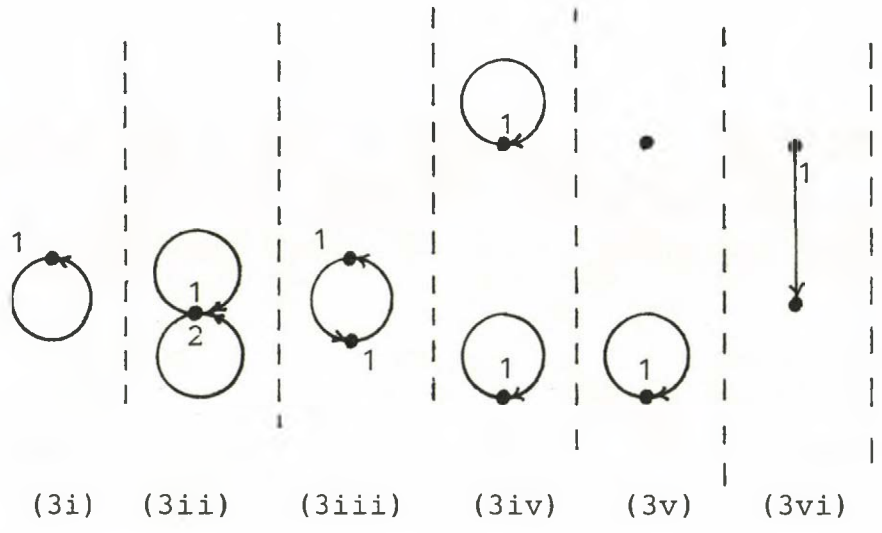
3.1.1 Hivatkozási sémák

3.1.1.1 Hivatkozási univerzum

Definició: Egy hivatkozási univerzum egy véges, cimkézett (szinezett), irányított gráf. (A tárgyalás egyszerűsítése érdekében cimkék gyanánt a továbbiakban mindíg sorszámokat használunk.)

Példák:

(1) Az üres gráf hivatkozási univerzum.
(2) Az egyetlen szögpontból álló gráf hivatkozási univerzum.
(3) A 3/1. ábrán bemutatott gráfok mindegyike egy-egy hivatkozási univerzum.



3/1. ábra

Példák hivatkozási univerzumokra

Vizuális forma

Legyen U egy hivatkozási univerzum. Az egyszerűség kedvéért jelölje U egyben a gráf szögpontjainak halmazát is. Legyen most $u \in U$ és tegyük fel, hogy az u-ból kiinduló élek rendre (az élek számozása szerint) az $u_1, \ldots, u_n$ szögpontokba vezetnek ($n \geq 0$). Ezt a tényt írásban az alábbi formában rögzítjük:

$$u(u_1,\ldots,u_n).$$

Amennyiben $n=0$, ez a forma az

$$u$$

alakra redukálódik.

Példák:

(2) írható {u} alakban.
(3i) egyetlen u(u) alakú elemből álló halmaz.
(3ii) esetén az egyetlen elem u(u,u) alakú.
(3iii) két elemű: $u_1(u_2)$,
$u_2(u_1)$.
(3iv) két elemű: $u_1(u_2)$,
$u_2(u_2)$.

Kiválasztás

Legyen $u \in U$ az U hivatkozási univerzum egy szög-pontja. Jelölje $|u|$ az u-ból kiinduló élek számát. Ekkor

$$u.i \qquad (1 \leq i \leq |u|)$$

-vel fogjuk jelölni az u-ból kiinduló i-edik él végpontját. (Például (3i) esetén: u.1=u, u.1.1=u, míg pl. u.2 nem definiált.)

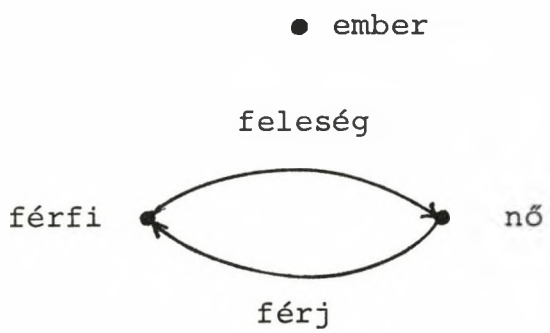
3.1.1.2 Rendezett univerzum

Definició: Egy rendezett univerzum egy olyan hivatkozási univerzum, melynek szögpontjain egy részben rendezési reláció van megadva. Ezt a relációt a továbbiakban "SUB"-bal jelöljük.

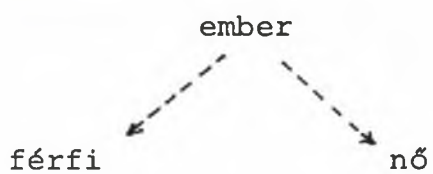
Példa: a következő univerzum

{ember, férfi(nő), nő(férfi)}

azaz



ahol a rendezés



azaz SUB = Id U {(férfi,ember)} U {(nő,ember)}
egy rendezett univerzum.

Vizuális forma

Abban a speciális esetben, mikor a részben rendezés egy fa vagy erdő, az előző szakaszbeli formát az alábbi módon terjeszthetjük ki:

$$u \ \underline{is} \ u'(u_1,\ldots,u_n)$$

kifejezve, hogy u' a SUB relációban megelőzi u-t. Fenti példánk egyszerűbben tehát így is írható:

{ember,
férfi is ember (nő),
nő is ember (férfi)}.

3.1.2 Minősített sémák

Mostantól fogva két univerzumot tekintünk, melyek egyike számunkra a "típusokat", másika pedig az "adatokat" fogja reprezentálni.

3.1.2.1 Homomorf eset

Legyenek U és V hivatkozási univerzumok, Ψ pedig egy leképezés U szögpontjairól V szögpontjaira:

$$\Psi: U \to V.$$

Azt fogjuk mondani, hogy V az U "minősítő univerzuma" a Ψ "minősítésre" nézve, amennyiben

a) $|\Psi(u)| = |u|$,

b) $\Psi(u.i) = \Psi(u).i \qquad i \leq |u|$

teljesül minden $u \in U$ szögpontra. (Azaz, Ψ egy homomorfizmus a kiválasztási operációra nézve.) A "minősítés" mint homomorfizmus tekinthető a típus illeszkedés törvényét kifejező axiomának is az altípus nélküli esetben (tehát pl. közönséges programozási nyelvek esetén az eljárás paraméterekre vonatkoztatva. Részletesebb példákat ld. [41]-ben.)

3.1.2.2 Konvex eset

Tegyük fel most, hogy V egy rendezett univerzum, és legyen ismét

$$\Psi: U \to V$$

úgy, hogy

a) $|\Psi(u)| = |u|$,

b) $(\Psi(u.i), \Psi(u).i) \in SUB \quad i \leq |u|, \forall u.$

Mint látható, fenti feltételek a típus-illeszkedés törvényének típus-altípus viszonyra általánosított formáját fejezik ki. Ez esetben a homomorfizmus nem teljesül, hanem csupán egy annál gyengébb tulajdonság: a b) feltétel értelmében Ψ u.n. konvex leképezés. (A konvex leképezések általánosításai a [3], [4]-ben bevezetett morfizmusoknak, de kezelhetők ugyanazon általános apparátusok segítségével, melyek [57], [2], [1]-ben vannak kidolgozva.)

3.1.2.3 Hiányos univerzum

Gyakorlati okok miatt célszerű olyan hivatkozási univerzumokat is megengedni, melyekben a gráf egyes élei definiálatlanok ($|u|$-ban azonban számításba vannak véve). Ez nem okoz gondot, ha az előző pontokban szereplő homomorfizmusra ill. konvexitásra vonatkozó kikötéseket csak a definiált esetben követeljük meg. Ezt precízen úgy formalizálhatjuk, ha a kiválasztási operációt parciális függvényként értelmezzük, és parciális unáris algebrák közötti homomorfizmusokra térünk át. (Ennek rutin lépéseit az olvasó [41]-ben találja.)

3.1.2.4 Reguláris univerzum

Egy fa-univerzumot (azaz, egy olyan rendezett univerzumot, melyben a SUB reláció hierarchikus) regulárisnak nevezünk, ha minden $(u,u') \in SUB$ párra a következők teljesülnek:

a) $|u| \geq |u'|$

b) $u.i = u'.i \qquad i \leq |u'|$.

Mint látjuk, reguláris univerzumban a SUB reláció által alárendelt szögpontok "öröklik" szüleik éleinek cél-szögpontjait (és rendelkezhetnek még további élekkel. Ez pontosan az a mód, ahogy pl. a SIMULA 67 nyelv [14] és a concept modell is kezeli az absztrakt adattípusokat.)

3.1.3 Alap operációk

Ahhoz, hogy a szükséges műveleteket a hivatkozási univerzumokra értelmezhessük, az univerzális algebrából ismert u.n. "term-algebrát" fogjuk használni. (Erre azért van szükség, mert operációink az eredetileg vett hivatkozási univerzumokból kivezetnek.)

Először függvény szimbólumokat vezetünk be. Legyen

a) $(i_1,\ldots,i_k)$PERM egy egyváltozós függvény szimbólum tetszés szerint $k,i_1,\ldots,i_k$ pozitív egészekre, ahol $s \neq z \rightarrow i_s \neq i_z$;

b) ASCENT egy egyváltozós függvény szimbólum;

c) PROD egy kétváltozós függvény szimbólum

d) JOIN szintén egy kétváltozós függvény szimbólum.

Legyen U egy hivatkozási univerzum. Most "term"-eket definiálunk, és mindegyikhez hozzárendeljük vagy U egy szögpontját, vagy egy új szögpontot definiálunk (olyan módon azonban, hogy az élek minden esetben U-beli szögpontokhoz fognak vezetni):

1) Ha $u \in U$, akkor u egy term, és a hozzárendelt szögpont önmaga.

2) Legyen t egy term. Ekkor t.i is term $0 < i \leq |t|$. A hozzárendelt szögpont a t-hez rendelt szögpontból kiinduló i-edik él végpontja.

3) Legyenek t' és t" termek, és a hozzájuk tartozó szögpontok rendre

$$u'(u'_1,\ldots,u'_k),$$

$$u"(u"_1,\ldots,u"_{k"}).$$

Ekkor PROD(t', t") is term és a hozzá tartozó szögpont új szögpont, melynek alakja

$$u^*(u'_1,\ldots,u'_k,\ u"_1,\ldots,u"_{k"}).$$

4) Legyenek t' és t" mint fent. Ekkor JOIN(t',t") akkor és csak akkor term, ha $k' \geq 1$, $k" \geq 1$ és $u'_{k'} = u"_1$. A hozzárendelt szögpont új szögpont, melynek alakja

$$u^*(u'_1,\ldots,u'_k,\ u"_2,\ldots,u"_{k"}).$$

5) Legyen t term, szögpontja $u(u_1,\ldots,u_n)$. Most $(i_1,\ldots,i_k)$PERM(t) akkor és csak akkor term, ha $\max(i_1,\ldots,i_k) \leq n$. A hozzárendelt szögpont új, és alakja

$$u^*(u_{i_1},\ldots,u_{i_k}).$$

6) Legyen t mint fent. ASCENT(t) akkor és csak akkor term, ha $u \in U$. A hozzárendelt szögpont új és alakja

$$u^*(u).$$

Definició: Az összes termek halmazát U lezárásának nevezzük és U*-al jelöljük. (Mint láttuk: az U* gráf minden éle U-ba vezet.)

3.1.4 Reláció kalkulus

Annak érdekében, hogy konstrukcióink könnyen áttekinthetőek legyenek, először a homomorf esetet tárgyaljuk.

3.1.4.1 Homomorf eset

Legyenek U, V hivatkozási univerzumok és $\Psi: U \to V$ homomorf leképezés (ld. 3.1.2.1).

Definició: Legyen $u(u_1,\ldots,u_n) \in U$. Ekkor a

$$(\Psi(u_1),\ldots,\Psi(u_n))$$

n-est u szignaturájának nevezzük.

Hivatkozási relációk

Legyen

$\mathbf{R} = \{ S : S \subset U^*, (x,y \in S \to x,y \text{ szignaturája azonos})\}$

$\mathbf{R}$ elemeit relációknak nevezzük (U,V,Ψ) felett. (Minden relációhoz egy egyértelműen definiált szignatura tartozik. Ennek hosszát az "oszlopszámnak" nevezzük.)

Legyen

$\mathbf{T} = \{S: S \subset U,$ S a Ker Ψ egy osztálya$\}$.

$\mathbf{T}$ elemeit minősített relációknak fogjuk nvezni (U,V,Ψ) felett. (A minősített relációk minősítése egyértelmű: elemeinek képe V-ben.)

Ker Ψ teljes kongruencia osztályait elsődleges relációknak nevezzük (U,V,Ψ) felett. Az elsődleges relációk halmazát $\mathbf{P}$-vel jelöljük.

Definicióink a szokásos reláció definiciókkal összhangban vannak, hiszen egy $R \in \mathbf{R}$ azonos hosszúságú n-esekből áll, és bármely rögzített i-re (i ≦ hosszúság) az i-edik elemek képei azonosak a Ψ minősítés szerint. Nyilvánvalóan: $\mathbf{P} \subset \mathbf{T} \subset \mathbf{R}$.

Operáció definiciók

A relációk közül minket csak azok érdekelnek, melyek reláció kifejezések útján levezethetők. Ezek mindegyikéhez egy szignaturát, és bizonyosokhoz közülük típust is (minősítést) rendelünk a következő rekurzív módon:

1) Ha $R \in \mathbf{R}$ akkor definició szerint $R = \Psi^{-1}(v)$ valamely alkalmas $v \in V$-re. Legyen $v = v(v_1,\ldots,v_n)$.

$$sign(R) \stackrel{def}{=} (v_1,\ldots,v_n),$$

$$type(R) \stackrel{def}{=} v.$$

2) Legyen R egy reláció. R.i akkor és csak akkor reláció, ha $1 \leq i \leq$ R -hossza. Definiciója a következő:

$$R.i = \{r.i \; ; \; r \in R\}.$$

Tegyük fel, hogy $sign(R) = (v_1,\ldots,v_n)$ és $v_i = v_i(v_1',\ldots,v_k')$. Ekkor

$$sign(R.i) \stackrel{def}{=} (v_1',\ldots,v_k'),$$

$$type(R.i) \stackrel{def}{=} v_i.$$

(Ez az operáció az u.n. "zoom" operáció.)

3) Legyenek R', R" relációk. Ekkor PROD(R',R") is egy reláció, és definiciója az alábbi:

$$\{PROD(r',r"); \quad r' \in R', \quad r" \in R"\} .$$

Feltéve, hogy $sign(R') = (v_1',\ldots,v_n')$,

$sign(R") = (v_1",\ldots,v_k"$

$$sign(PROD(R',R")) \stackrel{def}{=} (v_1',\ldots,v_n', v_1",\ldots,v_k").$$

4) Legyenek R', R" relációk és szignaturájuk rendre $(v_1',\ldots,v_k')$, $(v_1",\ldots,v_k")$. JOIN(R',R") - az u.n. "természetes join", akkor és csak akkor reláció,ha $k' \geq 1$, $k" \geq 1$, és $v_k',= v_1"$, és definiciója az alábbi:

$$\{JOIN(r',r"); \quad r' \in R', \; r" \in R" \quad \text{és} \; r'.k' = r".1\}.$$

És

$$sign(JOIN(R'R")) \stackrel{def}{=} (v_1',\ldots,v_k', v_2",\ldots,v_k").$$

$$\text{type}(\text{JOIN}(R',R")) \overset{\text{def}}{=} \begin{cases} \text{type}(R'), \text{ ha létezik és } k"=1, \\ \text{különben type}(R"), \text{ ha létezik,} \\ \quad \text{és } k'=1, \\ \text{egyébként nem definiált.} \end{cases}$$

(Természetesen: JOIN(R',R") lehet üres.)

5) Legyen R reláció $(v_1,\ldots,v_n)$ szignaturával. $(i_1,\ldots,i_k)\text{PERM}(R)$ akkor és csak akkor reláció, ha $k \geq 1$, $s \neq z \rightarrow i_s \neq i_z$, $\max\{i_1,\ldots,i_k\} \leq n$; és definiciója:

$$\{(i_1,\ldots,i_k)\text{PERM}(r);\ \ r \in R\}$$

$$\text{sign}((i_1,\ldots,i_k)\text{PERM}(R)) \overset{\text{def}}{=} (v_{i_1},\ldots,v_{i_k}).$$

6) Legyen T minősített reláció. Ekkor ASCENT(T) is reláció, mégpedig az alábbi:

$$\{\text{ASCENT}(t);\ \ t \in T\}.$$

$$\text{sign}(\text{ASCENT}(T)) \overset{\text{def}}{=} \text{type}(T).$$

7) Végül, megengedjük a szokásos halmazelméleti operációk alkalmazását azonos szignaturájú relációk között ($\cap, \cup, \setminus$). Ezek a szignaturát nem változtatják.

(Az olvasó könnyen verifikálhatja, hogy minden bevezetett operációnk **R** -beli relációkra alkalmazva szintén **R** -be tartozó - esetleg üres - relációt eredményez.)

Példa

Legyen V = {férfi,
nő,
házasság (férfi,nő),
válás(házasság)}.

Legyen U tetszés szerinti V minősítésű univerzum, és tegyük fel, hogy "VÁLÁS" a "válás" elsődleges relációja, és legyen például "AMAZON" egy "nő" minősítésű reláció. Ekkor a "zoom"

VÁLÁS.1

azon házasságok halmaza, melyek válással végződtek. És például a

JOIN(VÁLÁS.1, ASCENT(AMAZON)).1

kifejezés azon férfiak halmazát reprezentálja, melyek amazonoktól váltak el.

3.1.4.2 Konvex eset

Elsődleges relációk

Legyen V reguláris univerzum, és $\Psi: U \to V$ konvex leképezés. **P** definiálásánál most óvatosabban kell eljárnunk

Vezessük be a következő jelölést:

$$(v] = \{v' \in V : SUB(v',v)\}$$

Ekkor

$$\mathbf{P} \stackrel{\text{def}}{=} \{S \subset U^* : \exists v \in V \quad \text{hogy} \quad S = \Psi^{-1}((v])\} .$$

Ha tehát $R \in \mathbf{P}$, $R = \Psi^{-1}((v])$, akkor minden $u \in R$, $u = u(u_1,\ldots,u_k)$ elemre

$$|u| \geq |v|$$

teljesül. $|u| > |v|$ esetén u-nak, mint R-beli elemnek a tartalma alatt az $(u_1,\ldots,u_{|v|})$ projekciót tekintjük. (A gyakorlatban ezt úgy is fogalmazhatjuk: a reláció megjelenő tartalma függ attól a nézőponttól, ahonnan vizsgáljuk.)

Operációk

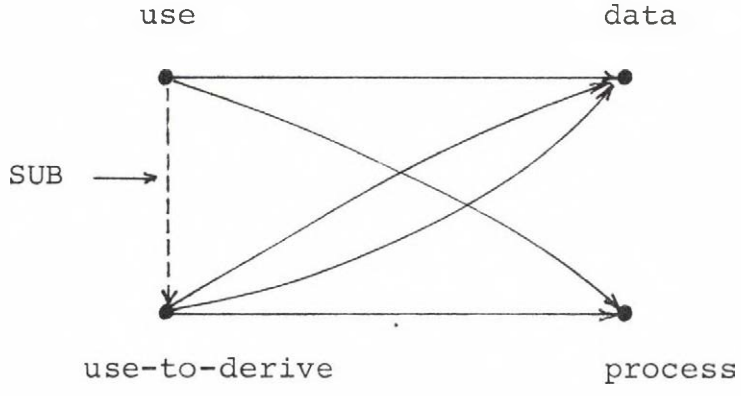
A ".", "PROD", "JOIN", "ASCENT" operációk és szignatúrájuk ugyanúgy definiálhatók, mint korábban. A halmazelméleti operációkkal azonban külön kell foglalkoznunk.

Amikor R valamely reláció művelet operandusa, a végrehajtás során a neki megfelelő adatobjektumok tartalmát az R szerinti nézőpontból kell tekintetbe venni. Ez a legfőbb eltérés az egyszerű homomorf esethez viszonyítva.

Példa: Tekintsük a következő univerzumot:

```
V = {process,
     data,
     use(data,process),
     use-to-derive is use(data)}
```

azaz



és U legyen

```
process P1,P2;
data D1,D2;
use U1(D1,P1);
use-to-derive UD(D1,P2,D2),
```

Ez esetben a "use" elsődleges reláció az alábbi:

	data	process
U1	D1	P2
UD	D1	P1

és például

JOIN(use,process).1

azon adat objektumok halmaza, melyeket a folyamatok felhasználnak.

Legyenek ezek után S, S' azonos hosszúságú relációk, és szignaturájuk: $\sigma = (v_1,\ldots,v_n)$, $\sigma' = (v_1',\ldots,v_n')$ megfelelően. A halmazelméleti operációk végrehajthatóságának <u>általános feltétele</u> az alábbi

vagy $SUB(v_i,v_i')$ vagy $SUB(v_i', v_i)$ $\forall_i$, $1 \leq i \leq n$.

E feltétel mellett operációink most már a szokásos úton definiálhatók. Szignaturáik (és típusaik) az alábbiak:

1) $sign(S \cup S') \stackrel{def}{=} (w_1,\ldots,w_n)$, ahol $w_i = max(v_i,v_i')$;

2) $sign(S \setminus S') \stackrel{def}{=} \sigma$;

3) $sign(S \cap S') \stackrel{def}{=} (w_1,\ldots,w_n)$, ahol $w_i = min(v_i,v_i')$.

(Fenti általános feltételünk értelmében ezen minimumok és maximumok mindíg léteznek.) Típust a halmazelméleti operációkhoz általában nem rendelünk, kivéve:

1) Ha type(S), type(S') és max(type(S),type(S')) mindegyike létezik, akkor

type $(S \cup S') \stackrel{def}{=}$ max(type(S),type(S')).

2) Ha type(S) létezik, akkor

type $(S \setminus S') \stackrel{def}{=}$ type(S).

3) Ha type(S), type(S') és min(type(S), type(S'))
mindegyike létezik, akkor

$$\text{type}(S \cap S') \overset{\text{def}}{=} \min(\text{type}(S), \text{type}(S')).$$

Relációk

Ezek után az összes relációk **R** halmazát most már definiálhatjuk rekurzív módon:

$$\mathbf{R} = \{S \subset U^* : S \text{ levezethető } \mathbf{P} \text{ elemeiből a bevezetett operációk segítségével}\}.$$

Legyen továbbá:

$$\mathbf{T} = \{S \in \mathbf{R} : S\text{-hez típus van rendelve}\}.$$

Nyilvánvalóan most is: $\mathbf{P} \subset \mathbf{T} \subset \mathbf{R}$. (Indukcióval könnyen belátható, hogy **T** valamely $v=v(v_1,\ldots,v_n)$ típusú elemének szignaturája $(v_1,\ldots,v_n)$.)

3.1.5 Faktorizáció

3.1.5.1 Hasonlóság

Gyakori eset (például, amikor adatainkat predikátumokként kívánjuk tekinteni), hogy maguk a reláció elemek, mint egyedi objektumok helyett, azoknak csupán a tartalmára kívánunk szorítkozni. Pontosabban: bizonyos esetekben nem kívánunk különbséget tenni $u = u(u_1,\ldots,u_k)$ és $u'= u'(u'_1,\ldots,u'_k,)$ között $u,u' \in R$, $R \in \mathbf{R}$ valahányszor

$$(H) \qquad k = k' \quad \text{és} \quad u_i = u'_i \qquad 1 \leq i \leq k.$$

Vizsgáljuk meg most precízen egy ilyen ekvivalencia következményeit. Jelölje $\sim$ a (H) által definiált ekvivalencia relációt. Legyen f a 3.1.3-beli 2)-5) függvény szimbólumok bármelyike, és k argumentumainak száma. Ekkor a definíciókból azonnal adódik, hogy

$$u_1,\ldots,u_k,\ v_1,\ldots,v_k \in U^* \quad \text{és} \quad u_i \sim v_i \qquad 1 \leq i \leq k$$

esetén

(F) $\quad f(u_1,\ldots,u_k) \sim f(v_1,\ldots,v_k)$

teljesül (valahányszor fentiek léteznek). Másszóval $\sim$ egy kongruencia reláció a ZOOM, PROD, JOIN, PERM parciális operációkra nézve. (Ugyanez nem igaz azonban az ASCENT operáció esetén.)

3.1.5.2 Redukció

Jelölje $\tilde{u}$ az $u \in U^*$-hez tartozó hasonlósági osztályt, és legyen

$$\tilde{R} = \{\tilde{u},\ u \in R\}, \quad R \in \mathbf{R}$$

(ahol a hasonlóságot az R szerinti nézőpontra vonatkozó tartalomra értjük). Ekkor $\tilde{R}$-t az R-reláció redukciójának nevezzük.

Az előző pontbeli (F) miatt a 3.1.3 2)-5) operációk a faktorizáción is definiálhatók:

$$U^*|_{\sim} = \{\tilde{u} \in U^*\}$$

és ekkor

$$f(\tilde{u}_1,\ldots,\tilde{u}_k) \overset{\text{def}}{=} \widetilde{f(u_1,\ldots,u_k)}.$$

Sőt, tovább menve, ha $R \in \mathbf{R}$ egy elsődleges relációkból a 2)-5) operációkkal levezetett reláció, akkor mindegy, hogy mikor térünk át a hasonlósági osztályokra, ugyanis (F) igaz marad kontextusok szerinti tartalomra is.

Megjegyzések

1) ASCENT nincs definiálva $U^*|_{\sim}$-on. Ez a gyakorlatban annyit jelent, hogy ASCENT operációk előtt nem térhetünk át hasonlósági osztályokra.

2) Halmazelméleti operációk $U^*|_{\sim}$ részhalmazaira is alkalmazhatók. Ezek szerint beszélhetünk tartalom szerinti halmazelméleti műveletekről is. Legyen $R_1, R_2 \in \mathbf{R}$, ekkor

$$R_1 \cap_c R_2 \overset{\text{def}}{=\!=} \tilde{R}_1 \cap \tilde{R}_2 ,$$

$$R_1 \cup_c R_2 \overset{\text{def}}{=\!=} \tilde{R}_1 \cup \tilde{R}_2 ,$$

$$R_1 \setminus_c R_2 \overset{\text{def}}{=\!=} \tilde{R}_1 \setminus \tilde{R}_2 .$$

3) Redukált relációk szignaturáját a természetes módon definiálhatjuk: $sign(\hat{R})$ $sign(R)$. Ezzel az operáció definiciók az eredeti szabályok szerint maradnak érvényben.

4) Ezen a ponton érdemes megemlíteni a [66] és [17]-ben kidolgozott elmélettel való kapcsolatot. $\tilde{R}$ az u.n. "quotient relations" egy speciális fajtájának tekinthető. A blokkokat indukáló attributumok éppen $sign(R)$ komponensei. Így Tompa [66] jelöléseit alkalmazva: $\tilde{R} = R_{\alpha(R)}$. (Az alkalmazott apparátus egy általános tárgyalását az olvasó [3]-ban találja meg.)

3.1.6 Hivatkozási sémák mint elméletek

Ebben a szakaszban röviden rámutatunk, hogy az eddig tárgyalt sémák hogyan reprezentálhatók a Burstall-Goguen [11] féle értelemben vett elméletek és elmélet morfizmusok útján.

0.) Nulladik szint

theory hivatkozási séma;
sorts objektum, típus, boolean;
operations
minősítés: objektum → típus,
szelekció-ob$_\beta$: objektum → objektum, $\beta < \omega$
szelekció-tip$_\beta$: típus → típus, $\beta < \omega$
altípus: típus, típus → boolean,
univerzális: → típus,
definiált$_\beta$: típus → boolean, $\beta < \omega$
< továbbá a szokásos logikai operációk >.

equations
< itélet kalkulus egyenletei >,
< az altípus hierarchikus voltát kifejező egyenlet >,
definiált$_\alpha$(u) → definiált$_\beta$(u), $0 < \beta < \alpha < \omega$
altípus(v'v) → (definiált$_\beta$(v) → (definiált$_\beta$(v') and
szelekció-tip$_\beta$(v) = szelekció-tip$_\beta$(v')))

altípus(minősítés o szelekció-ob$_\beta$,
szelekció-tip$_\beta$ o minősítés).

[illetőleg utóbbi a homomorf esetben:
minősítés o szelekció-ob$_\beta$ = szelekció-tip$_\beta$ o minősítés.]

1.) Első szint

Az alapstruktúrák bővítésére a Burstall-Goguen féle "enrich" operációt alkalmazzuk. Vázlat:

enrich theory hivatkozási séma by
operations

nil: → objektum (a nem definiált érték)
.i: objektum → objektum
prod: objektum, objektum → objektum
join: objektum, objektum → objektum
(i_1,..,i_k)perm: objektum → objektum $i_j \neq i_k, j \neq k$
ascent: objektum → objektum
term: objektum → boolean

equations

<3.1.3 2)-6) definicióinak leírása úgy, hogy a nem definiált értékek "nil" értéket kapnak>,

term(nil) = false,
imply(term(u) = false, u = nil) = true.

endenrich

(Az egyenleteink száma végtelen, hiszen U* végtelen.)

További kiterjesztésekkel bevezethetjük a relációkat és reláció kifejezéseket. (Ekkor a "sort"-ok listáját is bővítenünk kell már. Ezektől a részletektől itt eltérünk. A módszer további részleteit és általánosításait az olvasó a [44], [36]-ban találja.)

3.1.7 Kiegészítő megjegyzések

A 3.1 alfejezet eddigi részeiben tárgyalt modellek lehetővé teszik azt is, hogy most már ne csak a reláció kalkulust, mint az egyik lehetséges adatbázis elemző és lekérdező eszközt, hanem a tárgynyelven alapuló második lekérdező nyelvi rendszert is algebrai ill. logikai eszközökkel pontosan for-

malizáljuk. Erre a természetes út az, ha fogalmainkat a többszortú logika nyelvére írjuk át a következő módon: Valamely f fogalomnak, melynek attributumai $f_1,...,f_n$ megfeleltetünk egy F relációjelet, melynek szortja

$$(f_0, f_1,..., f_n), \quad \text{ahol} \quad f_0 = f.$$

A többszortú nyelvnek csak ezek, és az = lesznek relációjelei, ezen kívül csak konstansjeleket tartalmaz. Korábbi axiomáink könnyen átírhatók ezen jelölésrendszerre, ld. [41]. Így lehetőség nyílik arra, hogy megvizsgáljuk lekérdező rendszerünk kifejező erejét, és összehasonlítsuk más modellekkel. Ezt nem részletezzük, ezeket a [36] és [44] tartalmazzák.

A "concept" modellhez hasonlóan, a 2.1-2.2 alfejezetekben ismertetett öndefiniáló sémakezelésű modellek is könnyen reprezentálhatók a többszortú logika eszközeivel.Nem nehéz belátni, hogy az ott ismertetett lekérdező rendszerben a lekérdezhető formulák mindegyike ekvivalens egy olyannal, amelyben

a) minden kvantor egzisztenciális,
b) a kvantormentes rész prímformulák konjunkciója.

Ezen egyszerűbb modell esetében tehát csak u.n. "tisztán pozitív" információkat tudunk visszakeresni. E kérdéskör részletes tárgyalása a [36] dokumentumban található.

3.2 Operáció szinkronizáció

Adatbázis kezelő rendszerek felhasználásánál az u.n. konkurrens hozzáférés biztosítása napjainkra alapvető követelménnyé vált. Az operációk közötti sorrend és szinkronizáció kifejezésére számos apparátus létezik, ld. például [62], [58], [54]. Ebben az alfejezetben egy új eszközt mutatunk be, és rámutatunk az idézett megközelítésekhez való kapcsolatára. Ez az alfejezet az előzőktől független jelölésrendszert alkalmaz, és megközelítésében az eddieknél általánosabb érvényű.

3.2.1 Fogalmak

3.2.1.1 Jelölések és konvenciók

Legyen V egy véges halmaz, az u.n. ábécé, és jelölje a formális nyelvek elméletében megszokott módon V^* a V-ből képezhető szavak halmazát. Valamely $w \in V^*$ szó esetén jelölje $op(w)$ a benne előforduló szimbólumok halmazát. Például:

$$op(abacca) = \{a,b,c\}.$$

Jelölje $ops(w)$ a szóban előforduló, egymástól megkülönböztetett szimbólum-előfordulások halmazát. Például $ops(abacca)$ így írható:

$$\{a^{(1)},a^{(2)},a^{(3)},b^{(1)},c^{(1)},c^{(2)}\}.$$

Ha $W \subset V^*$ szavak egy halmaza, akkor

$$op(W) \stackrel{def}{=} \bigcup_{w \in W} op(w).$$

Valamely $w \in V^*$ szó mindíg tekinthető egy, az $ops(w)$ halmaz feletti T_w(teljes) rendezésnek.

Ha T egy rendezés valamely X halmazon, akkor annak megszorítását valamely szűkebb $Y \subset X$ halmazra $T|_Y$-al jelöljük. Ha $w_1,w_2 \in V^*$, akkor a konkatenációjukra vonatkozó $T_{w_1w_2}$ rendezés nyilván T_{w_1}, T_{w_2} rendezett összegeként reprezentálható, úgy hogy $ops(w_1)$ ill. $ops(w_2)$-n belül különböző előfordulás-jeleket használunk.

3.2.1.2 Projekció

Legyen $w \in V^*$ és $U \subset V$. Ekkor a

$$w|_U$$

projekciót úgy definiáljuk, hogy az a w-ből az U-hoz nem tartozó szimbólumok elhagyásával származtatott új szó.

Például:

$$abacca\big|_{\{a,b\}} = abaa \ .$$

A $w\big|_{\emptyset}$ projekció az üres szó: ε. Szavak egymásra való projekcióját is bevezetjük:

$$w_1\big|_{w_2} \overset{\text{def}}{=} w_1\big|_{op(w_2)} .$$

Továbbá halmazok projekcióit ennek általánosításaként: Legyenek X_1, $X_2 \subset V^*$; ekkor

$$X_1\big|_{X_2} \overset{\text{def}}{=} \{(w_1\big|_S) : w_1 \in X_1 \ , \quad S = \bigcup_{w_2 \in X_2} op(w_2)\} \ .$$

(Hasonlóan lehetne beszélni nyelvek projekciójáról is, most azonban erre nem lesz szükségünk.)

A projekció alábbi tulajdonságai kézenfekvőek:

1) A projekció az egymás utáni alkalmazára nézve <u>kommutatív</u>:

$$w\big|_{U_1}\big|_{U_2} = w\big|_{U_2}\big|_{U_1} \qquad w \in V^* \ , \ U_1, \ U_2 \subset V.$$

2) A projekció konkatenációra nézve <u>disztributív</u>:

$$w_1w_2\big|_U = w_1\big|_U \, w_2\big|_U \quad w_1, w_2 \in V^* \ , \ U \subset V.$$

3) A projekció <u>monoton</u>:

$$w\big|_{U_1}\big|_{U_2} = w\big|_{U_2}$$

valahányszor $U_2 \subset U_1 \subset V$.

3.2.1.3 Összefésülés

Legyen $w_1, w_2 \in V^*$. Az $ops(w_1)$, $ops(w_2)$ halmazokon belül válasszuk meg az előfordulási megjelöléseket úgy, hogy a két halmaz diszjunkt legyen. Jelölje P a $T_{w_1} \cup T_{w_2}$ reláció tranzitív lezárását. P nyilvánvalóan egy részben rendezés az $ops(w_1) \cup ops(w_2)$ halmazon. Ezek után a $w_1 || w_2$ "összefésülést" a következő halmazként definiáljuk:

$$w_1 || w_2 \stackrel{def}{=} \{ w : P \subset T_w \} .$$

Szemléletesen: az összefésülés szavai olyan, az $ops(w_1) \cup \cup (w_2)$ szimbólumaiból alkotott szavak, melyekben mindkét operandus szimbólumai megőrzik eredeti sorrendjüket. (Példaként ld. még a 3/2.-ábrát.)

Amennyiben w_1, w_2-ben nincsenek közös szimbólumok, akkor nyilvánvalóan

$$w \Big|_{w_1} = w_1, \qquad w \Big|_{w_2} = w_2, \qquad w \in w_1 || w_2 .$$

Definiciónk a kézenfekvő módon kiterjeszthető szavakból álló halmazok összefésülése is.

3.2.1.4 Kompatibilitás

A $w_1, w_2 \in V^*$ szavakat kompatibiliseknek mondjuk, ha

$$w_1 \Big|_{w_2} = w_2 \Big|_{w_1}$$

teljesül. Ez a feltétel nyilván így is fogalmazható: Legyen $c = c(w_1, w_2) = op(w_1) \cap op(w_2)$ a szavak közös szimbólumainak halmaza. Ekkor feltételünk:

$$w_1 \Big|_c = w_2 \Big|_c .$$

Amennyiben a két szó azonos szimbólumokból áll ($op(w_1) = op(w_2)$), akkor a kompatibilitás nyilván az azonossággal ekvivalens. Végül, diszjunkt szavak (azaz, $op(w_1) \cap op(w_2) = \emptyset$) mindíg kompatibilisek, hiszen ekkor

$$w_1|_{w_2} = \varepsilon = w_2|_{w_1} .$$

A kompatibilitás szimmetrikus, nem tranzitív reláció. (Ezek az u.n. "konkurrencia relációk" tipikus tulajdonságai, ld. például [ ] és [ ].)

3.2.1.5 Konkurrens szorzat

Legyen $w \in V^*$. Egy $u \in V^*$ szót w-re vonatkozóan projektívnek nevezünk, ha

$$u|_w = w.$$

Legyen $W \subset V^*$ szavak tetszés szerinti halmaza. Ekkor a $\otimes W$ konkurrens szorzatot úgy definiáljuk, mint azon szavak halmazát, melyek W összes elemére nézve projektívek:

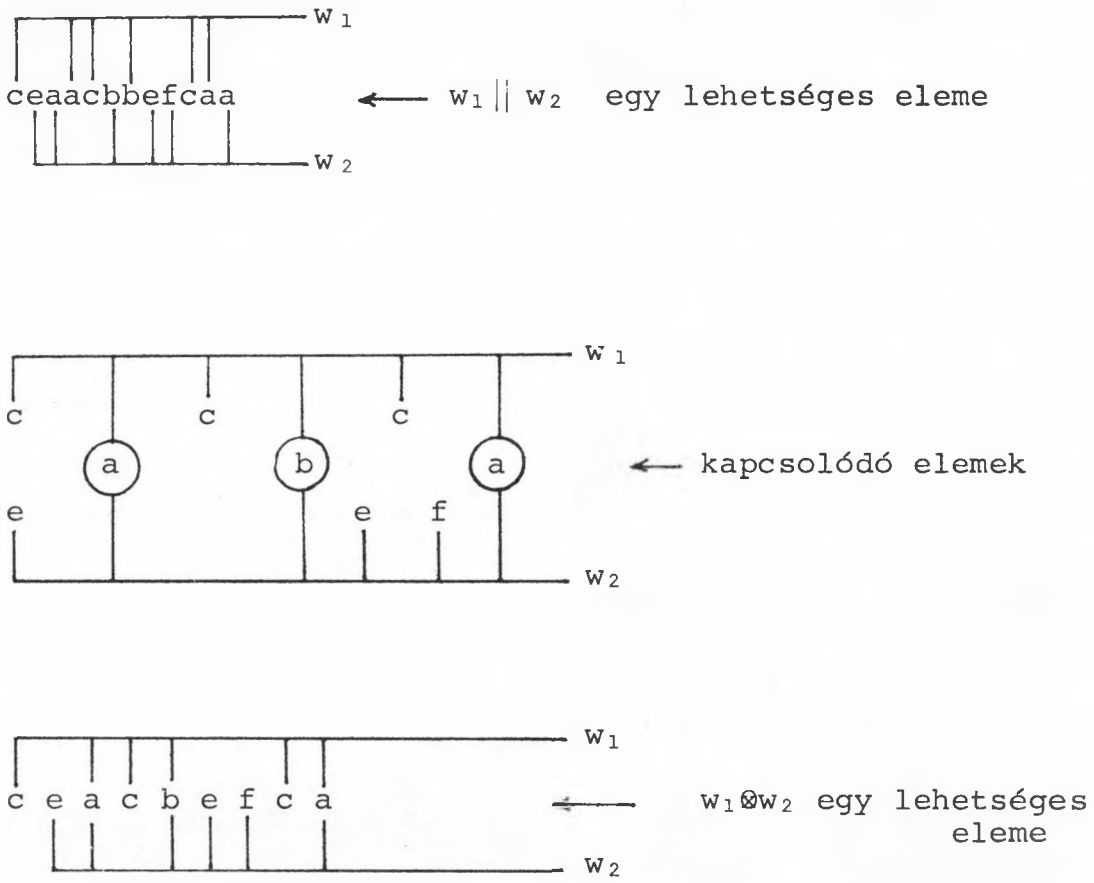
$$\otimes W \overset{\text{def}}{=} \{u : u|_w = w, \forall_w \in W\} .$$

A $\otimes W$ szorzat magjának W szimbólumaira való vetítését tekintjük és így jelöljük:

$$[\otimes W] \overset{\text{def}}{=} \otimes W|_{op(W)} .$$

Nyilván: $\otimes\emptyset = V^*$ és $[\otimes\emptyset] = \emptyset$ és $[\emptyset\{w\}] = \{w\}$ teljesülnek. Két elem esetén a szorzatot így is írjuk: $w_1 \otimes w_2$. Az alábbi 3/2. ábra $w_1 \otimes w_2$ szerkezetét hasonlítja össze $w_1 || w_2$-vel:

Legyenek például $w_1 = cacbca$, $w_2 = eabefa$.



3/2 . ábra

Szorzat és összefésülés összehasonlítása

3.2.2 Szorzatra vonatkozó alaptételek

Mint később látni fogjuk, a konkurrens szorzatok ürességének ill. befejezhetőségének problémája a szinkronizációk leírására általánosan használt Petri hálók [58] egy osztályára vonatkozóan az u.n. "liveness problémával" ekvivalens [45]. Ez indokolja e kérdések vizsgálatának fontosságát.

Megemlítjük, hogy a "liveness probléma" általános esetben nem megoldott, és most következő tételeink sem adnak eddig ismeretlen esetekre megoldást. Az eddig ismert eredményeket azonban más megvilágításba helyezik. Az általunk mutatott formalizmus alkalmazása különösen abban az esetben lehet előnyös,

amikor az adatbázis kezelési konkurrencia problémáknak azt az esetét vizsgáljuk, melyben a konzisztencia biztosítása magának az adatbázis kezelő rendszernek (és nem a felhasználónak) a feladata (bővebben ld. [62]).

3.2.2.1 Az ürességi probléma

A szorzat ürességének kérdése akkor nem triviális, ha legalább két operandusunk van. Ebben az esetben $\otimes W$ üressége ekvivalens $[\otimes W]$ ürességével.

Legyen $w_1, w_2 \in V^*$ és válasszuk meg a szimbólum előfordulási sorszámokat (1)-től kiindulva $ops(w_1)$ és $ops(w_2)$ mindkettőjében.

1.Lemma: Ha w_1 és w_2 kompatibilisek, akkor $P = T_{w_1} \cup T_{w_2}$ tranzitív lezárása egy részben rendezés $ops(w_1) \overset{w_2}{\cup} ops(w_2)$ felett.

Bizonyítás: Az egyetlen dolog, ami bizonyításra szorul, az a tranzitív lezárás antiszimmetriája. Tegyük fel, hogy $(x_1,x_2) \in P$, $(x_2,x_1) \in P$ egyidejűleg. Ha $x_1,x_2 \in w_1$ (vagy ugyanúgy, ha $x_1,x_2 \in w_2$), akkor ez csak úgy lehet, ha $x_1 = x_2$. Legyen tehát $x_1 \in w_1$, $w_2 \in w_2$. Minthogy $(x_1,x_2) \in P^*$, létezik egy olyan x' szimbólum előfordulás, melyre $(x_1,x') \in T_{w_1}$, $(x',x_2) \in T_{w_2}$. Hasonló okoskodással a másik oldalról nyerjük: $\exists x''$, hogy $(x_2,x'') \in T_{w_2}$, $(x'',x_1) \in T_{w_1}$. Minthogy $T_{w_1} \subset P^*$, $T_{w_2} \subset P^*$, és P^* tranzitív, így mindent összevetve: $x_1 = x' = x_2 = x'' = x_1$.

1.Tétel: $w_1 \otimes w_2$ akkor és csak akkor nem üres, ha w_1, w_2 kompatibilisek.

Bizonyítás:

a) Legyen $w \in w_1 \otimes w_2$. A kommutativitás folytán

$$w\big|_{w_1}\big|_{w_2} = w\big|_{w_2}\big|_{w_1} .$$

Feltételünk szerint

$$w\big|_{w_1} = w_1 , \quad w\big|_{w_2} = w_2 .$$

Így: $w_1\big|_{w_2} = w_2\big|_{w_1}$.

b) Az előző lemma értelmében $P=T_{w_1} \cup T_{w_2}$ tranzitív lezárása részben rendezés, ezért beágyazható egy T (teljes) rendezésbe, amelyet egy $[w_1 \otimes w_2]$-be tartozó szóval lehet reprezentálni.

2. Tétel: Ha $\otimes\{w_i\}_{i=1}^{n}$, $(n\geq 2)$ nem üres, akkor az w_i szavak páronként kompatibilisek.

A bizonyítást ugyanúgy végezhetjük, mint az előző a) esetben. Jegyezzük meg azonban, hogy a kompatibilitás $n>2$ esetén nem elégséges feltétel (pl. $w_1=ab$, $w_2=bc$, $w_3=ca$ szorzata üres).

Legyen most X szavak egy véges halmaza: $X = \{w_1,\ldots,w_n\}$. Azt fogjuk mondani, hogy e halmazban az

$$a_1,\ a_2,\ \ldots,\ a_k,\ \ a_{k+1}=a_1$$

szimbólum előfordulások egy k hosszúságú <u>ciklust</u> alkotnak, ha bármely két egymás utáni elemre $(a_j,\ a_{j+1}) \in T_{w_i}$; teljesül valamely alkalmas i-re. Egy ciklust <u>minimálisnak</u> nevezünk, ha semmilyen szimbólumnak sem tartalmazza egynél több előfordulását.

Az $X = \{w_1,\ldots,\ w_n\}$ halmazt <u>gyengén kompatibilisnek</u> nevezzük, ha bármely két w_i, w_j elemére

$$\forall a,((w_i\big|_a \neq \varepsilon \wedge w_j\big|_a \neq \varepsilon) \rightarrow w_i\big|_a = w_j\big|_a)$$

vagyis bármely adott szimbólumnak vagy ugyanannyi, vagy zérus előfordulását tartalmazzák.

2. Lemma: Ciklussal rendelkező gyengén kompatibilis halmaznak van minimális ciklusa.

Bizonyítás: Ha a ciklus nem minimális, van olyan szimbóluma, mely kétszer fordul elő. Rendezzük át a ciklust ciklikusan úgy, hogy a két előfordulás egyikével kezdődjön. Ejtsük el a két előfordulás közötti részt a második előfordulással együtt. A gyenge kompatibilitás miatt így egy újabb (rövidebb) ciklust nyerünk, hiszen abban a szóban, amiben a második elő-

fordulás megelőzte a rákövetkező elemet, abban (előtte) az első is elő kellett forduljon. Ezt az eljárást folytathatjuk.

3. Tétel: $\otimes X$ akkor és csak akkor nem üres, ha i) X gyengén kompatibilis és ii) nincsen ciklusa.

Bizonyítás:

a) Szükségesség. i) következik a 2.tételből. Másrészt $\otimes$ definiciója miatt $T_w \supset T_{w_i}$ minden i-re $w \in \otimes X$ esetén. T_w tranzitivitása és a ciklus létezése miatt annak bármely tagja megelőzi önmagát, amiből következik, hogy $\otimes X$ üres.

b) Elégségesség. Legyen $H = \bigcup_{w_i \in X} ops(w_i)$, és legyen R:

$$R = \{(x,y) : \exists i,\ (x,y) \in T_{w_i},\ x \neq y\} .$$

ii) miatt az R* tranzitív lezárás irreflexív, így beágyazható H egy teljes rendezésébe, melyet egy w szóval reprezentálhatunk. i) szerint $w|_{w_i} = w_i$ minden i-re teljesül, így $w \in \otimes X \neq \emptyset$.

Dekompozició

Be lehet bizonyítani, hogy bármely konkurrens szorzat kétbetűs szavak szorzataként is előállítható. Ezt nem részletezzük, a bizonyítást az olvasó [29]-ben megtalálhatja.

3.2.2.2 "Befejezhetőség"

Jelöljük $w' \prec w$ formában azt, hogy a w' szó kezdő szelete w-nek, azaz $\exists w''$, $w'w'' = w$. A $\prec$ reláció nyilvánvalóan részben rendezése V*-nak.

4. Tétel: Ha $\otimes\{w_i\}_{i=1}^{n}$ nem üres, és valamely w' szóra $w_i' = w'|_{w_i} \prec w_i$, $\forall i$ (azaz w' mindegyik projekciója a megfelelő "koordinátának" kezdőszelete), akkor w' kezdő szelete egy $w \in \otimes X$; szónak is (vagyis: ha w' "folytatható a koordináták mentén külön-külön, akkor van univezális folytatása is").

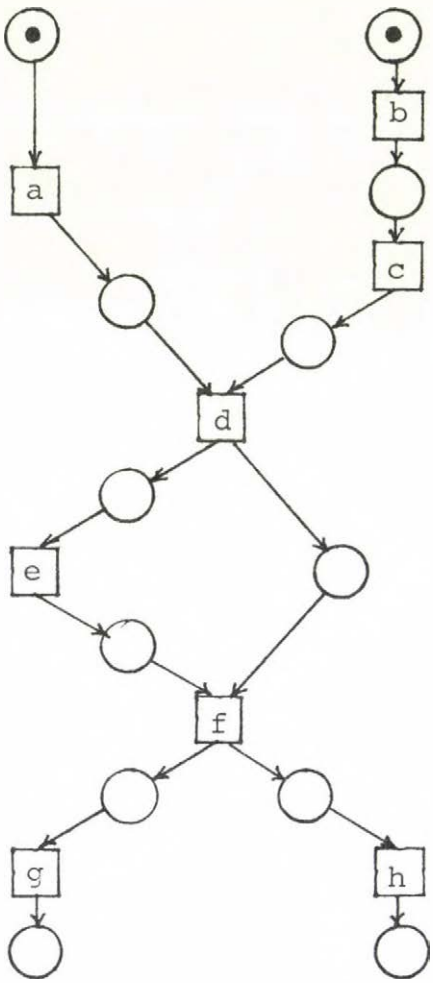
Bizonyítás: Legyenek w_i'',melyekre $w_i = w_i' w_i''$, $w_i' = w' \big|_{w_i}$. Nyilvánvalóan: $w' \in \otimes\{w_i'\}$. Ezért $\{w_i'\}$ szavai gyengén kompatibilisek, és ugyanez igaz $\{w_i''\}$-re is. Utóbbi nem tartalmaz ciklust, minthogy $\{w_i\}$ sem. Így a 3.tétel szerint létezik egy w" szó: $w'' \in [\otimes\{w_i''\}]$. Ekkor pedig a konkatenációra: $w'w'' \in \otimes\{w_i\}$, ugyanis $(w'w'')\big|_{w_i} = w_i'(w''\big|_{w_i})$ és $w_i'(w''\big|_{w_i''}) = w_i' w_i'' = w_i$ miatt már csak $op(w''\big|_{w_i}) \subseteq op(w''\big|_{w_i''})$ igazolandó, ez pedig azért teljesül, mert mindegyik w_i'' a közös szimbólumok bármelyikének ugyanannyi előfordulását tartalmazza, így ha egy szimbólum w"-ben és w_i-ben is szerepel, a w_i''-ra való megszorítás által nem kerülhetett kihagyásra.

A 4.tétel által kifejezett tulajdonság ekvivalens a Lauer-féle "path-kifejezések" u.n. adekvátságával (életképesség, ill. operáció sorozat befejezhetősége). Az alábbiakban röviden erre a kapcsolatra mutatunk rá.

Tegyük fel, hogy V elemei operációkat reprezentálnak, és tegyük fel, hogy a w_1, w_2 szavak egy-egy megengedhető operáció sorozatot jelölnek. Ekkor $w_1 \otimes w_2$ elemei pontosan azok a sorozatok, melyek a két komponens sorozat konkurren végrehajtása esetén lehetségesek. Vagy más szavakkal, a Petri hálók terminológiáját alkalmazva, $w_1 \otimes w_2$ elemei a w_1, w_2 szuperponálásával előállított Petri háló lehetséges billenés sorozatainak halmaza. Ld. 3/3 -ábra példáját.

Az adekvátság tétele itt így fogalmazható: ha $w_1 \otimes w_2$ nem üres, akkor a hálónak bármely kezdő billenés sorozata befejezhető (azaz nem jut holtpontra).

Megemlítjük még, hogy a konkurrens szorzatok és a Mazurkiewicz [54] féle u.n. trace-nyelvek között is szoros kapcsolat van. Ez nem ekvivalencia, hanem egy u.n. Galois-tipusú kapcsolat. (Ezt nem részletezzük, ld. [29].)



w_1 = adefg

w_2 = bcdfh

3/3. ábra

Szorzat mint Petri háló

4. NYELVI MODELLEK

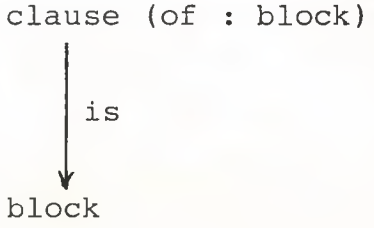
Ebben a fejezetben olyan rendszerleíró nyelveket adunk meg, melyek a 2.fejezetben ismertetett metanyelvi rendszerre alapulnak, és tipikus alkalmazási területeket fednek le.

4.1 Alapvető technikák

Ezen alfejezet a metanyelv tulajdonságaira építve néhány olyan technikát mutat be, melyek a különböző tárgynyelvekben egyaránt felhasználhatók.

4.1.1 Blokkolt struktúrák ábrázolása

Tegyük fel, hogy bizonyos típusú állításokból hierarchikus (egymásba skatulyázott blokkokból álló) szerkezeteket akarunk leírni tetszés szerinti mélységben - ezen a szinten most a fogalom konkrét jelentésétől eltekintve. Ezt az alábbi típus-pattern alkalmazásával érhetjük el:



azaz:

```
concept clause (of : block);
    form of : clause;

concept block is clause;
    form absolute : block;
    form of : block;
```

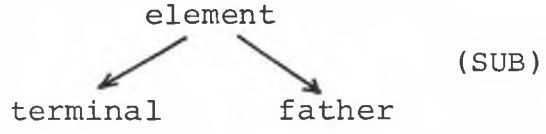
Ennek birtokában érvényes például az alábbi leírás:

```
block;
    A: clause;
    B: clause;
    C: block;
           CA: clause;
           CB: clause;
    end;
    D: clause;
end;
stb.
```

4.1.2 Rekurzív struktúrák

A következő egyszerű pattern alkalmazható valahányszor rekurzív kapcsolatokat akarunk leírni (pl. adatszerkezetekben vagy folyamatokban stb.).

origin (element, father)



azaz

```
concept origin (element, father); function;

concept element;
concept father is element;
concept terminal is element;
```

Ezen a módon szemantikailag azt ábrázoltuk, hogy minden elemnek van őse (apja), a "terminal" elemeknek azonban már nem lehet leszármazottja.

(További technikai fogások találhatók még a [34] dolgozatban.)

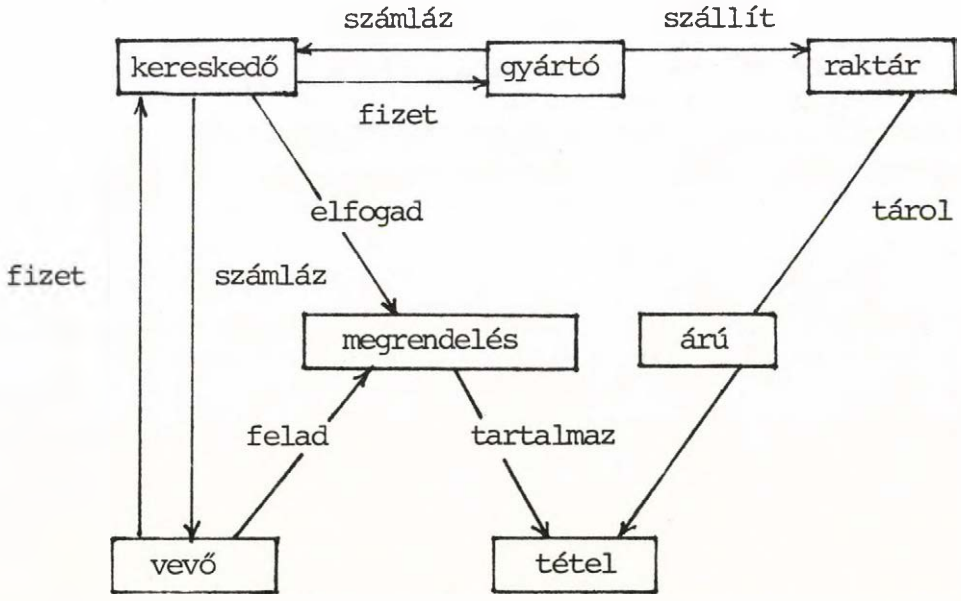
4.2 Elemi modellek

Most alkalmazásokra térünk át. Először igen egyszerű, de gyakori és általános eseteket tekintünk.

4.2.1 Irányított gráfok

legkülönbözőbb alkalmazása specifikációs és dokumentációs célokra rendkívül széles körű. Természetesen, a gráfok kifejező ereje grafikus formában a legnagyobb, azonban a gyakorlati feladatok mérete és a számítógépes analízis szükséglete indokolja adatbázisszerű kezelésüket.

Egy gyakorlati példát mutatunk, melyet a 4/1.ábra tartalmaz:



4 /1.ábra

Egy tipikus irányított gráf séma

Ilyen egyszerű séma ábrázolására az alábbi két fogalmat lehet/célszerű használni:

```
concept entity;

concept relationship(of : entity, to : entity)
   form of : relation-to to;
   form to : relation-from of;
```

Az ábrát ekkor például így írhatjuk le:

```
kereskedő : entity;
  elfogad : relation-to megrendelés;
  fizet   : relation-to gyártó;
  számláz : relation-to vevő;

vevő : entity;
   felad  : relation-to megrendelés;
   fizet  : relation-to kereskedő;
stb.
```

(Megjegyzés: a leírásban csak a "relation-to" alakot használtuk, a fordított szemléletű riportokban azonban természetesen ugyanezt a másik formával is lekérdezhetjük.)

4.2.2 Szinezett gráf modellek

Egy lépéssel bonyolultabb, amikor a gráf élei vagy szögpontjai (vagy mindkettő) szinezettek. Az elsőre példa lehet egy

4.2.2.1 CODASYL séma

leírása. Vezessük be például a következő fogalmakat:

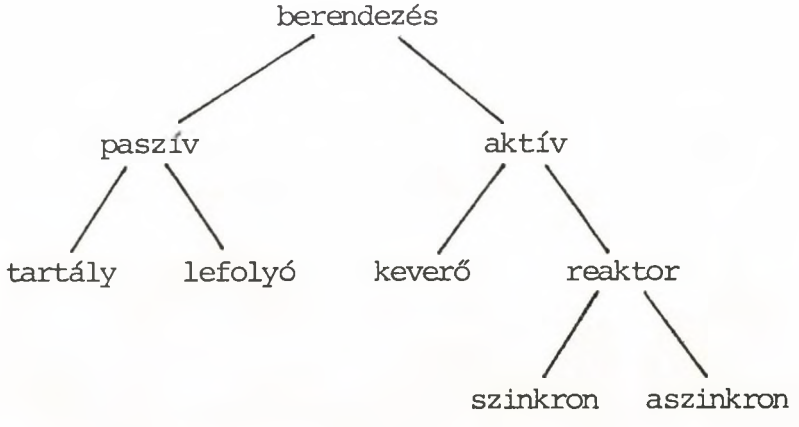
```
concept record;

concept set (owner : record, member : record);

concept chain is set;
   form member : owned-by owner linked;

concept array is set;
   form member : owned-by owner indexed;
```

E fogalmak birtokában pl. ilyen leírások készíthetők:

```
festmény : record;
    owned-by festő linked
    owned-by század linked;
    owned-by téma  indexed;

festő : record;
    owned-by iskola indexed
    év : owned-by század linked;
    hatás: owned-by festő linked;

iskola : record;
    owned-by system linked;

téma : record;
    owned-by system indexed;

stb.
```

4.2.2.2 Vegyipari üzem

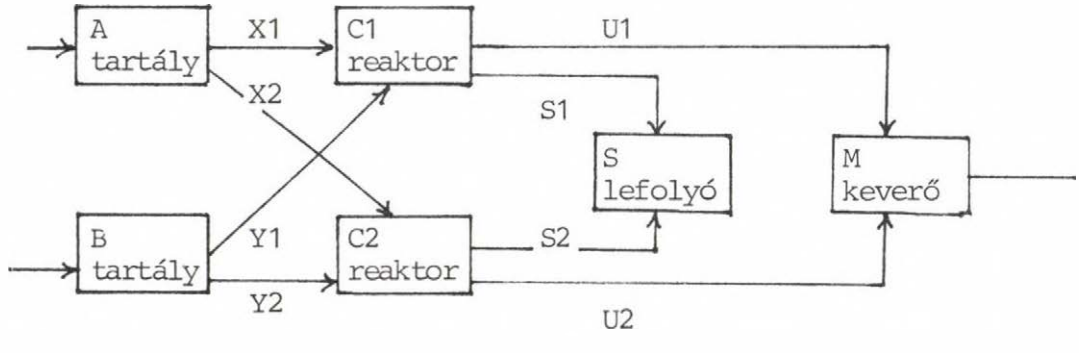
Egy érdekesebb példaként gráf szögpontok szinezésére, tekintsük a tipikus vegyipari berendezések közül néhánynak egy lehetséges osztályozását:



A megfelelő fogalmak például az alábbiak lehetnek:

```
concept berendezés:
concept passzív-berendezés is berendezés;
concept tartály is passzív-berendezés(kapacitás:real);
    form absolute : tartály méret kapacitás;

concept aktív-berendezés is berendezés(ciklusidő:real);
concept keverő is aktív-berendezés;
    form absolute : keverő periodus ciklusidő
⋮   stb., végül:

concept áram;
concept kimenet (áram,from:berendezés);
    form from: kimenő áram áram;

concept bemenet (áram,to;berendezés);
    form from:bemenő áram áram;
```

Tekintsünk ezek után egy gyártási sémát, például:



4/2.ábra

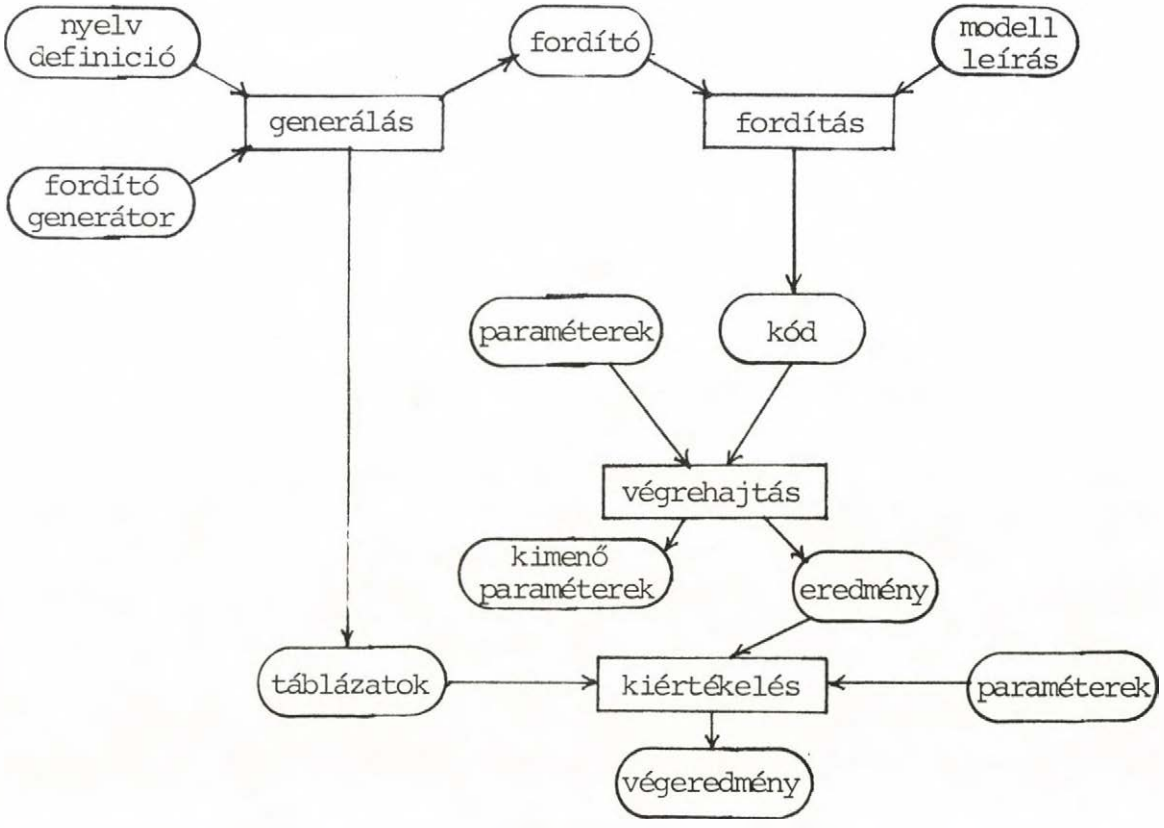
Példa szinezett gráf modell leírására

melynek leírása az alábbi formában történhet:

```
A: tartály méret 100;
     kimenő áram X1, X2;

B: tartály méret 80:
     kimenő áram Y1, Y2;

C1:reaktor periodus 20;
     bemenő áram X1, Y1;
     kimenő áram U1, S1;

stb.
```

4.2.3 Páros gráfok

Ez a csoport igen tág, felöleli az összes háló modelleket (kauzális hálók, esemény-feltétel hálók, predikátum-átmenet hálók, Petri hálók stb.). Mi most példaként a 4/3 ábrán egy u.n. kauzális hálót (ld. [58]) mutatunk be:



4/3.ábra

Példa kauzális háló alkalmazására

A leírásra a következő fogalmakat használhatjuk:

```
concept process;
concept condition;
concept use(process, condition);
    form process : uses condition;
    form condition : used-by process;

concept generation(process, condition);
    form process : generates condition;
    form condition : generated-by condition;
```

Hálónk leírása ezek után a következő:

```
generálás: process;
    generates fordító, táblázatok;
    uses nyelv definició, fordító-generátor;

fordítás: process;
    generates kód;
    uses fordító, modell-leírás;

végrehajtás: process;
    generates eredmény, kimenő-paraméterek;
    uses kód, paraméterek;
stb.
```

4.2.4 Az SADT modell

E modell alkalmazása a rendszertervezési gyakorlatban igen elterjedt, és.ott, ahol akciók és adatok hierarchikus lebontásával lehet dolgozni, igen eredményesen alkalmazható. (A modell ismertetését az olvasó pl. [61]-ban találhatja meg.) SADT gráfok ill. diagramok leírására a következő fogalmak alkalmazhatók:

```
concept diagram;
concept actigram is diagram;
concept datagram is diagram;
concept process (in : actigram, part-of : process);
   form in: process;
   form part-of: subprocess;

concept belong-p(proc: process, into: actigram);
   function;
   implies proc(in=into)

concept sub-p(proc:process, of: process);
   function;
   implies proc(part-of=of)
   form of: contains proc;

concept data (in: datagram, part-of: data);
   form in: data;
   form part-of: subdata;

concept belong-d(data, into: datagram);
   function;
   implies data (in=into);
```

```
concept sub-d(data, of: data);
   function;
   form of: contains data;

concept connection(process,data);

concept use is connection;
   form process: uses data;
   form data: used-by process;

concept control is use;
   function;
   form process: controlled-by data;
   form data: controlls process;

concept derivation is connection;
   function;
   form process: derives data;
   form data: derived-by process;
```

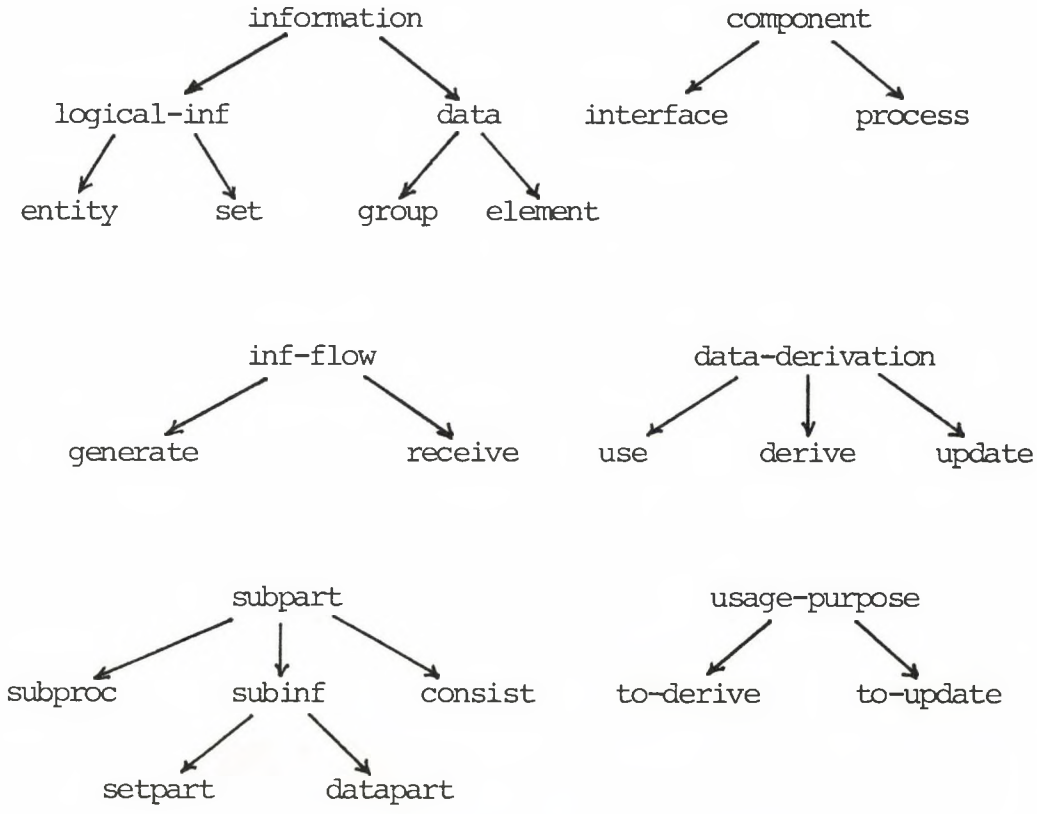
(Diagramunk leírására vonatkozó példát az olvasó [34]-ben találhat.)

4.3 Logikai tervezési modellek

Az u.n. logikai tervezési szint - az előzőekhez viszonyítva - fogalmakban már jóval gazdagabb. A tervezésnek ezen a szintjén azonban még mindíg nem szokás implementációs részletkérdésekkel foglalkozni. Ezen részletes modellekről a következő alfejezetben szólunk.

4.3.1 Információs rendszerek tervezése

E területen az egyik legsikeresebb fogalomrendszernek az ISDOS PSL/PSA [65] fogalmai tekinthetők. Ezek fő elemeit - felhasználva a típus-altípus viszony által nyújtott, az eredeti modellhez viszonyított új lehetőséget - a 4/4. ábrán látható módon foglalhatjuk új rendszerbe:



4/4.ábra

A PSL átrendezett fogalmai

A 4/4.ábrán vázolt elemek között az alábbi főbb kapcsolatokat kell létesíteni:

inf-flow(component,information)

data-derivation(process,information)

usage-purpose(used:information,process,towards:information)

to-derive ⇒ derivation(process,towards)
⇒ use(process,used)

to-update ⇒ update(process,towards)
⇒ use(process,used)

setpart(part:logical-inf,of:set);

datapart(part:data,of:group);

consist(logical-inf,of:data);

Ezen fő vonalak elfogadása után a PSL leíró nyelv (kismértékben módosított változata) az alábbi módon definiálható:

```
concept information;
form absolute: information;

concept logical-inf is information;

conept entity is logical inf;
form absolute: entity;

concept set is logical inf;
form absolute: set;

conept data is information;

concept element is data;
form absolute: element;

concept group is data;
form absolute: group;


concept component;

concept interface is component;
form absolute: interface;
```

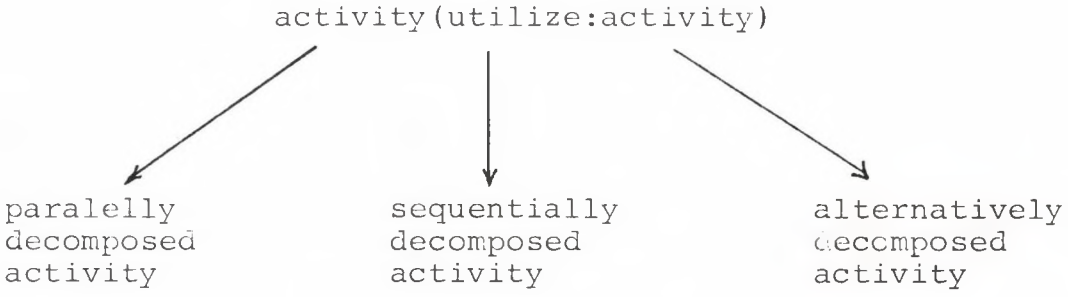
```
concept process is component;
form absolute: process;

concept inf-flow(component,information); function;

concept generate is inf-flow;
form component: generates information;
form information: generated-by component;

concept receive is inf-flow;
form component: receives information;
form information: received-by component;

concept data-derivation(process,information);function;

concept use is data derivation;
form process: uses information;
form information: used-by process;

concept derive is data-derivation;
form process: derives information;
form information: derived-by process;

concept update is data-derivation;
form process: updates information;
form information: updated-by:process;

concept subpart;

concept subpro  is subpart(process,of:process);
function;
form process: part-of of;
form of: subpar  process;

concept subin  is subpart;

concept setpart  s subinf(part:logical-inf,of:set);
function;
form part: part-of of;
form of: subpart part;

concept datapart is subinf(part:data,of:group);
function;
form part: part-of of ;
form of: subpart part;
```

```
concept consist is subpart(logical-inf,of:data);
function;
form logical-inf: consists-of of;
form of: contained-by logical-inf;

concept usage-purpose(used:information,user:process,
                      towards:information);

concept to-derive is usage-purpose; function;
implies derive(user,towards);
implies use(user,used);
form used: used-by user to-derive towards;
form user: uses used to-derive towards;
form towards: derived-by user using used;

concept to-update is usage-purpose; function;
implies update(user,towards);
implies use(user,used);
form used: used-by user to-update towards;
form user: uses used to-update towards;
form towards: updated-by user using used;


concept relation;
form absolute: relation;

concept associated(data,to:relation); function;
form to: associated-data data;
form data: associated-to to;

concept maintain(relation,process); function;
form process: maintains relation;
form relation: maintained-by process;

etc.
```

Fenti fogalmak alkalmazásával egy információs rendszer részletes specifikációjának példáját az olvasó [34]-ban találhatja meg.

4.3.2 Folyamatirányító rendszerek

Folyamatirányítási célú specifikációs nyelvek közül az egyik legismertebb a PCSL. Ennek magja az aktivitás tipizálás, mely modellünkben így ábrázolható:



E fogalmak leírására szolgáló nyelvet az alábbi módon definiálhatjuk:

```
concept activity(alternative-of:alternative,
                 step-of:sequence,
                 co-element-of:parallel,
                 seq-no:integer);
form alternative-of: alternative;
form step-of: step seq-no activity;
form co-element-of: coactivity;

concept alternative is activity;
form alternative-of: a-alternative;
form step-of: step seq-no a-activity;
form co-element-of: a-coactivity;
form absolute: a-activity;

concept sequence is activity
form alternative-of: s-alternative;
form step-of: step seq-no s-activity;
form co-element-of: s-coactivity;
form absolute: s-activity
```

```
concept parallel is activity
form alternative-of: p-alternative;
form step-of: step seq-no p-activity;
form co-element-of: p-coactivity;
form absolute: p-activity;
```

(Leírási példára, ill. folyamatirányítási rendszerek leírásához szükséges más elemekre vonatkozóan [34]-re utalunk.)

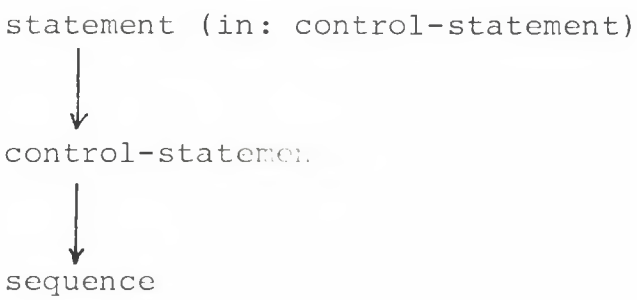
4.4 Gépközeli modellek

4.4.1 Vezérlési struktúrák

Ebben a részben a struktúrált programozás ismert jelöléseit alkalmazzuk (ld. például [21]).

4.4.1.1 Szekvencia

Szekvenciákból álló blokkok ábrázolására az alábbi fogalom-pattern-t alkalmazzuk:



Tételezzük fel tehát az alábbi deklarációkat:

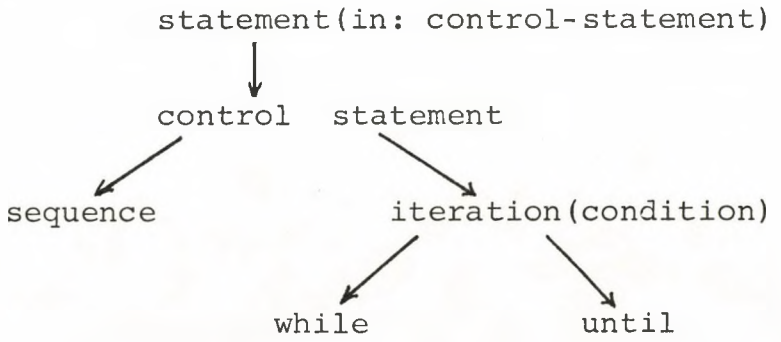
```
concept statement(in: control-statement)
   form in: statement;
concept control-statement is statement;
concept sequence is control-statement;
   form absolute: seq;
   form in: seq;
```

Ennek segítségével egyelőre szekvenciális utasításokból álló blokkszerkezeteket tudunk leírni, pl.:

```
PROG: seq;
   A: statement;
   C: seq;
         CA: statement;
         CB: statement;
   end;
   D: statement;
end;
```

4.4.1.2 Iteráció

Gazdagítsuk most típus hierarchiánkat az alábbi módon:



Legyenek definicióink a következők:

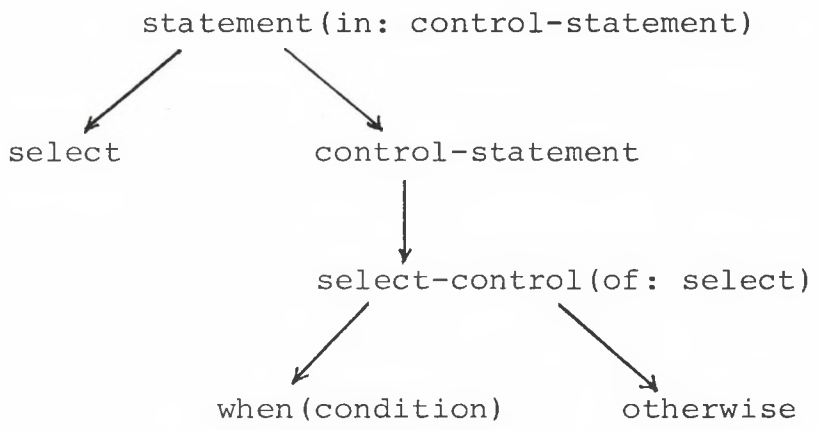
```
concept iteration is control-statement(condition:text);

concept while is iteration
form absolute: iter-while condition;
form subord-to: iter-while condition;

concept until is iteration;
form absolute: iter-until condition;
form subord-to: iter-until condition;
```

Mindezek felhasználásával lehetőségünk van az alábbi típusú szerkezetek leírására:

```
iter-until "S=limit";
   A: statement;
   B: statement;
end;

seq;
   A: statement;
   B: iter-while "key(S)≠high values";
      BA: statement;
      iter-while "0<S≤NUM-E";
          BBA: statement;
      end iter;
   end iter;
end seq;

etc.
```

4.4.1.3 Kiválasztás

Ahhoz, hogy az eset-szétválasztást (feltételes utasítást) a Jackson féle értelemben [21] megfelelően tudjuk leírni, és adatként kezelt ábrázolása is a célnak megfelelő legyen, fogalmainkat gondosan kell megválasztani. Egy, a gyakorlatban bevált séma az alábbi:



Definicióink ennek megfelelően:

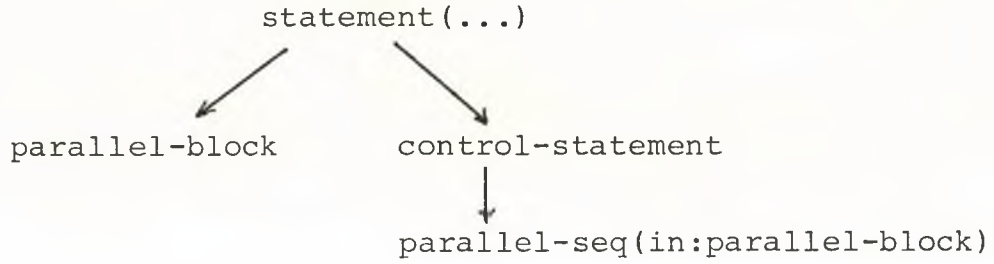
```
concept select is statement;
form absolute: select;
form subord-to: select;

concept select-control is control-statement(of:select);

concept control-statement is statement;

concept when is select-control(condition: text);
form of: when condition;

concept otherwise is select-control;
form of: otherwise;
```

Ennek használata például:

```
select;
   when "code='C'";
      CREDIT: statement;
      COUNT:  statement;
   when "code='T'";
      TRANSFER: statement
   otherwise;
      ERROR-1: statement;
end select;
```

4.4.1.4 Párhuzamosság

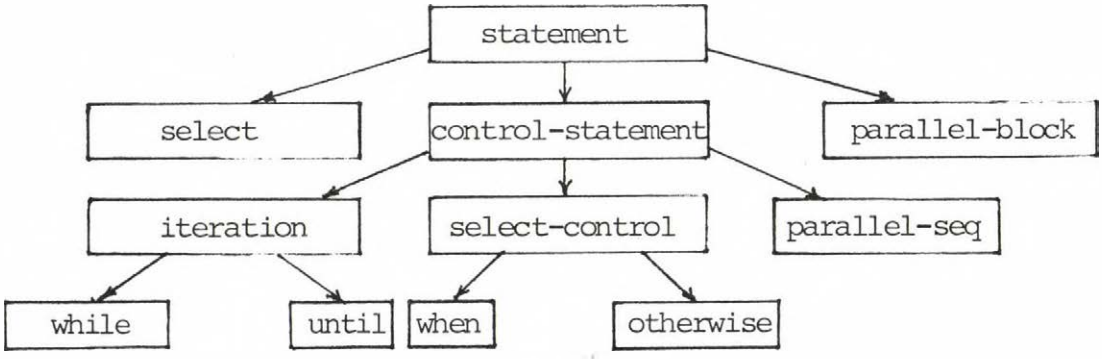
A párhuzamos végrehajtású operációk (blokkok) leírását az előzőhöz hasonló technikával végezhetjük:



azaz

```
concept statement(subord-to:control-statement);
form subord-to: statement;

concept parallel-block is statement;
form absolute: parblock;
form subord-to: parblock;

concept parseq is control-statement(in:parallel-block);
form in: co;
```

Mindezeket összegezve, a struktúrált programozás fogalmait leíró típus-fát a 4/5.ábrán láthatjuk:



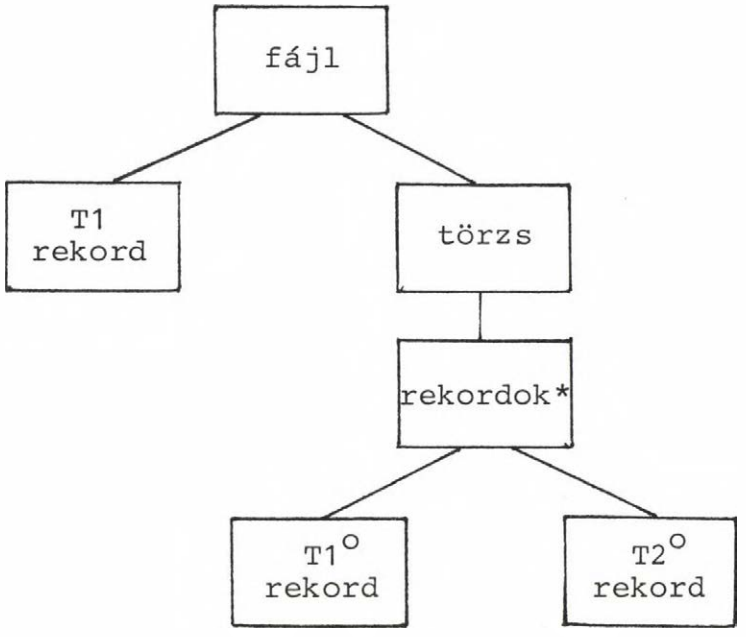
4/5.ábra

Struktúrált programozás fogalomfája

4.4.2 Adatszerkezetek leírása

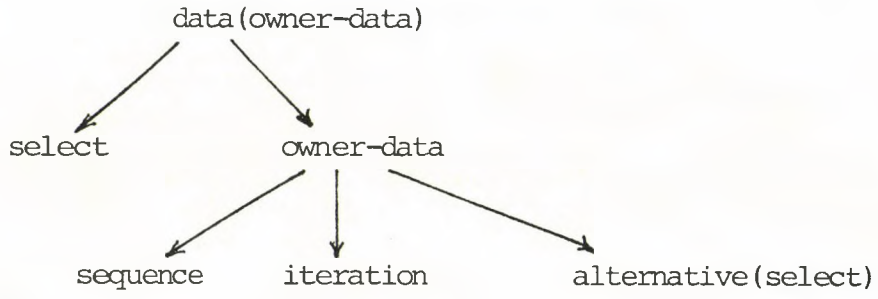
4.4.2.1 Struktúra szintaxis

Mint ahogy a struktúrált programok is a formális nyelvek elméletéből ismert reguláris kifejezésekkel voltak leírhatók, és mint tudjuk a környezet-független nyelvek szintaxisa is mindíg leírható reguláris kifejezésekkel, ugyanezt tehetjük egy szekvenciális adathalmaz szintaxisának leírásánál is. Tekintsük pl. az alábbi, a Jackson [21] féle formalizmussal ábrázolt adatstruktúrát:



(ahol Jackson reguláris kifejezésekkel ekvivalens formalizmusa szerint ez azt jelenti, hogy a "fájl" egy T1 rekordból és a "törzsből" áll, utóbbi T1 vagy T2 típusú rekordok sorozata).

A leíráshoz most a következő fogalmi sémát célszerű bevezetni:



Fogalom definicióink:

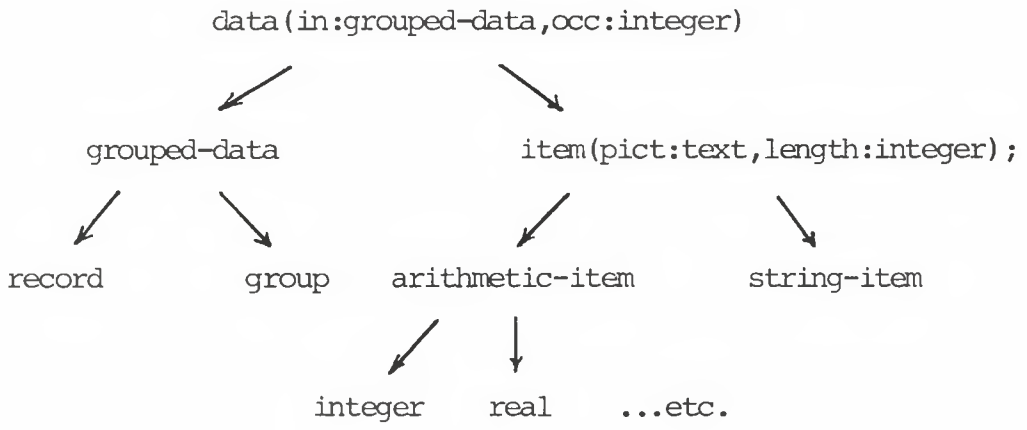
```
concept data(owner-data);
form owner-data: data;

concept select is data;
form absolute: select;
form owner-data: select;

concept owner-data is data;

concept sequence is owner-data;
form absolute: sequence;
form owner-data: sequence;

concept iter is owner-data;
form absolute: iter;
form owner-data: iter;

concept alternative is owner-data(select)
form select: either;
form select: or;
```

Példa az alkalmazásra:

```
file: sequence;
   file-body: iter;
      detail-record: select;
         either;
           T1: data;
         or;
           T2: data;
      end select;
   end iter;
   T3: data;
end sequence;
```

4.4.2.2 Típus szerkezet ábrázolás

Az ANSI X3 főbb adatszerkezeteit (melyek akár COBOL file rendszerek, akár CODASYL adatbázisok tervezésénél alkalmazhatók), az alábbi fogalom-fában rendezhetjük el:



Egy egyszerű illusztrációként:

```
R1: record;
    key D2;
    D1: real 4 pict ZD.D occ 12;
    D2: integer 3;
    D3: group occ 5;
        D31: integer 2 occ 7;
        D32: text 3 pict XXX;
end record;
```

Ezen az úton tovább lehet menni. Az eddigiekre ráépíthetők akár a fájl-kezelő rendszerek, akár az adatbázis kezelő rendszerek további specifikumai is. Ezekre vonatkozóan [ ]-re utalunk.

5. ARCHITEKTÚRÁLIS MODELLEK

Ebben a fejezetben a koncepcionális és formális szinteken már megismert adatbázis kezelési modellek realizálásával kapcsolatos általános (technológiai jellegű) vonatkozásokról szólunk.

5.1 Egy szoftver struktúrálási módszer

A világirodalom nem szűnő panaszait, újabb és újabb javaslatait az u.n. szoftver krízis (a nagy rendszerek megbízható realizálásához szükséges, egyre tűrhetetlenebb költségek és ráfordítások) kapcsán "lassan már únjuk". Pedig a helyzet lényegesen nem javul. Az automatikus programozás, a teljesen formális szemantikájú specifikációk, a kimerítő tesztelés ill. verifikálás módszerei ma még csak korlátozott méretek mellett alkalmazhatók.

Van azonban egy lényeges pont, ahol a gyorsabb előrelépés már ma is reális lehetne. Ez a szoftverek u.n. logikai architektúrájának tervezése, sőt szabályozása, abból a felismerésből kiindulva, hogy a szoftver elemek és kapcsolataik egy áttekinthető, "megfogható", adekvát rendszerbe (rendbe) foglalása még akkor is óriási haszonnal jár, ha az egyes részletek többségének szemantikai jellegű kézbentartása továbbra is az emberre marad.

Több kezdeményezés mutat ma ebbe az irányba, és minél szűkebb egy terület, annál tovább lehet haladni a területre jellemző architektúrális modell részleteinek kidolgozásában és szabályozásában. A legnagyobb horderejű ma e modellek közül az u.n. "Open Systems Interconnection" referencia-modell [20], de más területeken is kezdenek egyes modellek "felnőtté érni" (pl. a [60]-ben ismertetett modell a folyamatirányítás területén). Mi most ebben a fejezetben ilyen specifikus modellek kialakításához szükséges általános modell-elemeket vezetünk be (és mutatunk rá alkalmazásukra az adatbázis kezelő rendszerek területén).

5.1.1 Követelmények

Nem is olyan régen, a programozásról még mint művészetről volt szokás beszélni (pl. D.E.Knuth [26],[25]). Ma sem vitatható persze, hogy minden alkotás, konstrukció nélkülözhetetlen kelléke az elegancia, a szépség. Mindazonáltal, napjainkban a szoftver készítés meghatározó jellemzője az iparszerűség. Ez pedig a művészekétől alapjaiban eltérő technológiákat igényel (arról most nem is beszélve, hogy a programozók nagyon jelentős rétege művészként sem fogadható el).

Top-down szemlélet

Hagyományosan a top-down dekompozició a rendszerezés legfőbb eszköze. Ennek naív "zooming" értelmezése azonban épp elég jogos kritikát váltott már ki (pl. [51], [16]), mivel maga a fejlesztési folyamat a szintek folytonos váltogatását igényli, és nem mentes a kudarcok által indukált, meg-megújuló rekurzióktól. Mégis: valamely fejlesztési cél eléréséhez a top-down megközelítés - mint iránymutató és mint állandó támasz - nem nélkülözhető.

Lokalitás

Napjaink szoftver rendszerei méretük folytán egyetlen ember által már nem foghatók fel. Ezért életbevágó, hogy olyan nagyobb modulokra legyenek bontva, melyek önmagukban (környezetük működésének ismerete nélkül) megérthetők. E moduloknak természetesen számos kapcsolata lehet külvilágukkal, azonban csak jól definiált interfészek és protokollok útján.

Elválaszthatóság

Ma talán még erősnek tűnhet a megállapítás: szoftvernek valójában csak az olyan programot nevezhetjük, melynek készítője és tesztelője (átvevője) más-más személy (személyek). Ellenkező esetben a szoftver alkotójától soha sem válik elidegeníthetővé. Az elidegenítésnek többféle fokozatáról lehet beszélni. "Elválaszthatóság" alatt mi a modulokra vonatkozóan azt a követelményt értjük, hogy

legyen egy olyan jól definiált "felülete" (interfész), melynek tesztelése a modul teljes tesztelésével ekvivalens. (Ez enyhébb változat.)

Portabilitás

alatt az egyes (kisebb, vagy akár nagyobb) egységek környezet függetlenségét értjük, mely feltétele cserélhetőségüknek. Ennek eszközei a fent már említettek, de egy sor más is. Fontos eszközként tartozik pl. ide a "hibakezelés környezet függetlenségének" elve is, amely azt jelenti, hogy a programokban az egyes akciók hívásakor nem szabad a programkörnyezetből levezethető (de valójában implicit) asszerciókra alapítani (és ennek folytán kényelemből az akció bizonyos hibás kimeneteleinek kezelésétől eleve eltekinteni), mivel ezáltal a programokban függőségek alakulnak ki (a módosíthatóságot és karbantarthatóságot nagymértékben lerontva).

5.1.2 Dekompozició szintjei

Nagyon fontos észrevennünk azt, hogy a top-down feldolgozás során vannak u.n. "abszolut szintek", melyek mindegyike a komplexitásának megfelelő, egymástól eltérő, sajátos megközelítést igényel. (Egy csomag cigarettát pl. más módon kell megvásárolni ill. kifizetmi, mint egy autót.) A szintek abszolut rendje (vázlatosan) az alábbi sémát követi:

1. "Felső szintek", amikor olyan méretű és bonyolultságú dolgokat tekintünk, amelyek egészükben nem láthatók át, így ezekről keveset, csak a leglényegesebb dolgokat mondhatjuk el. (Pl. funkció, cél, méret, sebesség, kapcsolatok, követelmények stb.)

2. Egy kitüntetett szint, melyet "modul szintnek" nevezünk. Az egyes moduloktól a következő tulajdonságokat követeljük meg:

- Funkcionálisan specifikálhatók önmagukban úgy, hogy más modulokra nem hivatkozunk más módon, kizárólag csak interfészein keresztül.

- Minden modul egy

  - interfész és egy

  - törzs

  részből tevődik össze (lehet bármelyikük üres, ld. később).

- Modulokat más modulok csak interfészükön át használhatnak. A modulok közötti "használat" reláció nem rekurzív.

(Megjegyzés: Bizonyos alkalmazások még egy további megszorítást is sugallnak, nevezetesen azt, hogy a modul szinten további finomítások ne történhessenek, azaz modulok ne tartalmazhassanak al-modulokat. Az újabb vizsgálatok szerint azonban általánosságban érdemes ezt a lehetőséget meghagyni. Ez esetben a modul szint nem egyetlen szint, hanem szintek egy sorozata.)

Technológiai szempontból a modulok egy rendszernek azon komponensei, melyek <u>különállóan kivitelezhetők</u> (akár pl. egyidejűleg). A modulok kiválasztására vonatkozóan példát az 5.2 alfejezetekben mutatunk.

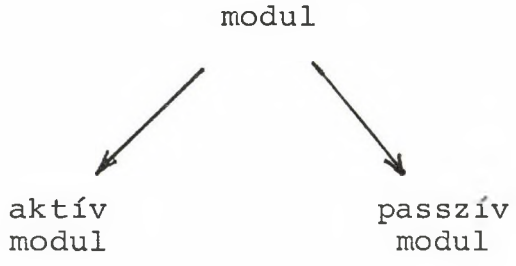
3. "Rendszerezési szintek", az egyes modulokon belüli struktúrák leírására. Ezekre azért van szükség, mert az egyes modulok (azaz a "még különállóan kezelhető részek") általában még mindíg túl nagyok ahhoz, hogy rendszerezés nélkül boldoguljunk velük. Ezeknek a finomításoknak során azonban a környezet független kezelhetőség követelménye tovább már nem tartható fenn, és általában nézve is: e lépéseket (a következő 4.csoportig) nem korlátozzuk más kötöttségekkel sem.

4. "Operációk szintje". Ezen utolsó szint az, amelyet az eljárások ill. szubrutinok végső kódja realizál. Ezen a szinten lehet jól alkalmazni a formális

specifikációs és verifikációs technikákat. Az operációk szintje erősen rekurzív, akciók finomítását és felbontását tartalmazza. Erre a szintre hárul egy fontos feladat: a hiba kezelés és feldolgozás. (Ezt azért fontos hangsúlyozni és betartani,mert egy akció valódi funkciója mellett a lehetséges hibás kimenetelek száma általában jóval nagyobb, így ha a kérdéssel a magasabb szintű specifikációkban foglalkoznánk, akkor nem volnánk képesek a lényegre koncetrálni.)

5.1.3 Altípusok

Az előző szakaszbeli értelemben vett moduloknak két fő osztályát célszerű megkülönböztetni:



Az aktív modulok nem rendelkeznek interfésszel. Ezek a rendszer vezérlő moduljai, más modulokat "használnak", őket azonban nem lehet használni (hanem csupán működésükbe "beavatkozni" – ezeket a mechanizmusokat azonban itt nem részletezzük).

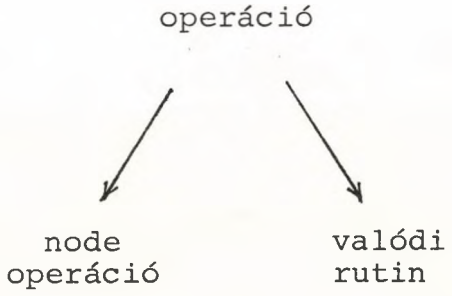
A passzív modulok szolgáltatásokat "ajánlanak", melyek interfészeiken jelennek meg. Minden passzív modul pontosan egy interfésszel rendelkezik, és tetszőleges számú más modul interfészére hivatkozhat (használhatja azt).

Itt jegyezzük meg, hogy az "Open Systems Interconnection" referencia modell [20], mint ismeretes, az interfész hivatkozásokat egy szigorúan lineáris (hét szintes)

rendszerbe kényszeríti. Ez a szigorúan lineáris rend azonban általánosságban nem tartható. E helyett mi csupán annyit követelünk meg, hogy az interfész hivatkozási reláció részben rendezés legyen.

Az operációk osztályozására vonatkozóan egy szempont az előzőekből már automatikusan adódik, nevezetesen, hogy az operáció interfész-operáció, vagy nem. Ennél azonban van egy sokkal lényegesebb osztályozási szempont:

Alapvető követelmény az, hogy egy-egy operációnak sem a specifikációja, sem a kódja nem lehet hosszú. (Ennek szükségességét most nem magyarázzuk.) E követelmény úgy teljesíthető, ha az operációk igen nagy mértékben al-operációkra vannak tagolva (ilyen hívásokból vannak felépítve), még akkor is, ha igen gyakran egy-egy rész operáció hívására a kódon belül csak egyetlen helyről kerül sor. Ezeket az operációkat, melyek kreálásának kizárólagos oka a modul transzparencia biztosítása: "node operációknak" nevezzük (arra utalva, hogy hívásuk egy hívási fával írható le). Szemben állnak ezzel a "valódi szubrutinok", azaz az általános szolgáltatásokat nyújtó operációk, melyek hívása bárhonnan történhet. (Megjegyezzük, hogy ha első pillanatra meglepő is, a jól struktúrált rendszerekben a "node operációk" aránya a túlnyomó.)



5.1.4 Hibakezelési modell

A következő induktív algoritmus alkalmazását javasoljuk:

1. lépés: Amikor a modul valamely akcióját specifikáljuk, mindíg az összes lehetséges hibás kimenetelt is áttekintjük, és egy listán "a potenciális hibák listáját" feljegyezzük. (Az összeset, tehát soha nem építünk arra, hogy bizonyos típusú hibák a speciális hívási feltételek miatt úgy sem fordulhatnak elő).

2. lépés: Amikor az akció belsejének kódját állítjuk elő (írjuk a programot), a hívott eljárások hibás kimeneteleit nem vesszük figyelembe.

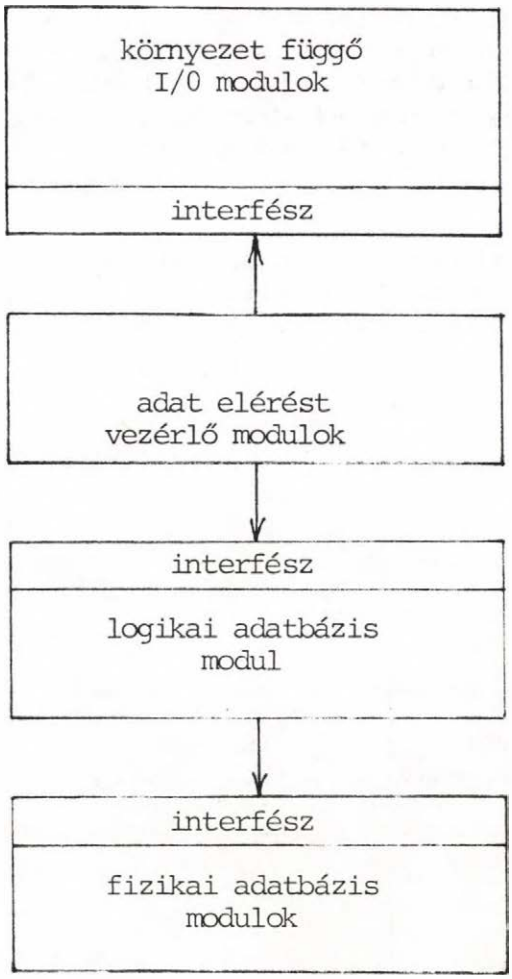
3. lépés: Amikor az első két lépéssel a modul egészére vonatkozóan elkészültünk, akkor kezdjük el szisztematikusan feldolgozni hívási helyenként a hívott lehetséges hibás kimeneteleit. (Mivel az első lépés befejezése garantálja azt, hogy a kimenő hibalistája kivétel nélkül minden akció számára elkészült, ezért a 3.lépésnél szükségtelen, hogy a hibafeldolgozást – a hívási hierarchia szerinti rendben végezzük; ehelyett az akciókat tetszés szerinti sorrendben vehetjük végig.

5.2 Specifikációs adatbázis architektúrák

Az 5.1 alfejezetben összefoglalt elveknek megfelelően (részletesebb ismertetésüket ld. [39] és [51]-ben) adatbázis modelljeink architektúrális terveit a [43] és [42] dokumentumokban leírt módon készíthetjük el. Ennek fontosabb elemeit foglaljuk össze most.

5.2.1 Modul séma

Első közelítésként azt a séma-magot vázoljuk az 5.1-beli terminológiának megfelelően, mely lényegében minden adatbázis kezelő célrendszer architektúrájára érvényesnek tekinthető. Ezt az 5/1.ábra vázolja.



5/1.ábra

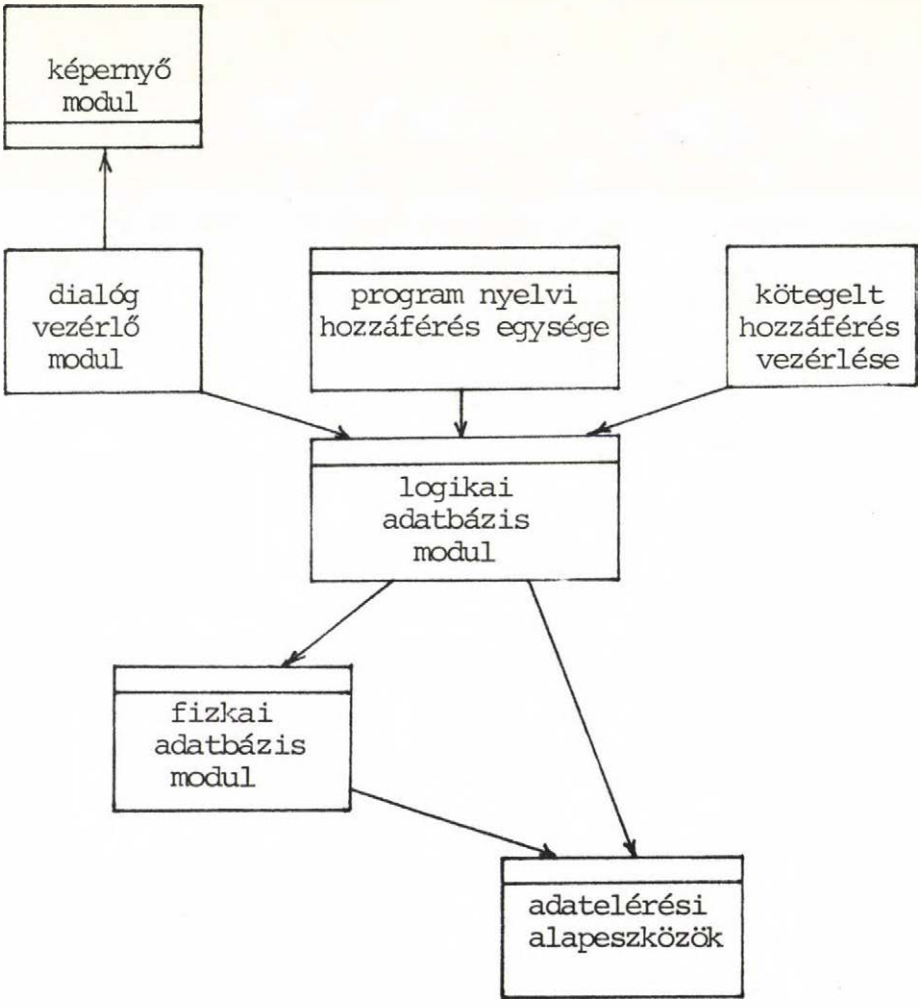
Általános adatbázis modul séma-mag

Ábránkon a nyilak az 5.1.2 értelmében vett interfész hivatkozásokat ("használat" relációt") jelentik. (5.1.3-mal összhangban a vezérlő modulok nem rendelkeznek interfész-szel.)

Az ábrán szereplő interfészeket egy adott adatbázis kezelő rendszer esetén szabványosítani és szigorúan rögzíteni kell.

Ezzel az architektúra valójában rugalmassá válik, hiszen a modulok szükség esetén cserélhetővé válnak. Ez különösen fontos a legfelső, a user-I/O modulok esetén, hiszen ez sokszor (sajnálatos módon) még azonos gépeken is különböző.

Ezen általános sémába illeszkedve, most megadjuk azt a teljes modul sémát, mely az első fejezetben leírt adatbázis kezelő rendszerek megvalósításához szükséges.



5/2. ábra

Modul architektúra

Hozzáférési modulok

Mint az ábrán is mutatjuk, minden adatbázis kezelő célrendszerhez háromféle hozzáférési módot kell biztosítani. A legfontosabb ezek közül az on-line hozzáférés, melyet a "dialóg vezérlő modul" valósít meg. Bizonyos esetekben azonban az on-line mód nem megfelelő: előfordul, hogy időnként igen nagy adatmennyiségeket kell az adatbázisba be-, vagy onnan kivinnünk. Ezt egy másik üzemmóddal, a "kötegelt hozzáférés" útján valósítjuk meg. Végül szükség van arra is, hogy az adatbázis kezelő rendszer (a hagyományos módon) programozási nyelvekből is hozzáférhető legyen. Ennek szabványos volta a "program nyelvi hozzáférés"-modulja által biztosítható. Ezen utóbbi modul (a dolog természeténél fogva) passzív modul (a vezérlés ekkor a felhasználási programokban realizálódik), míg a másik két említett üzemmód vezérlése önálló (ld. ábra).

Logikai adatbázis modul

Itt szigorúan egyetlen modulról van szó (mint azt már az általános sémán is jeleztük az 5/1.ábrán). Ennek interfésze reprezentálja kifelé azt a felületet, melyet az adatbázis használójának tudnia kell (azaz azt, ahogy ő látja az adatokat, tekintet nélkül annak fizikai reprezentációjára).

Fizikai adatbázis modul

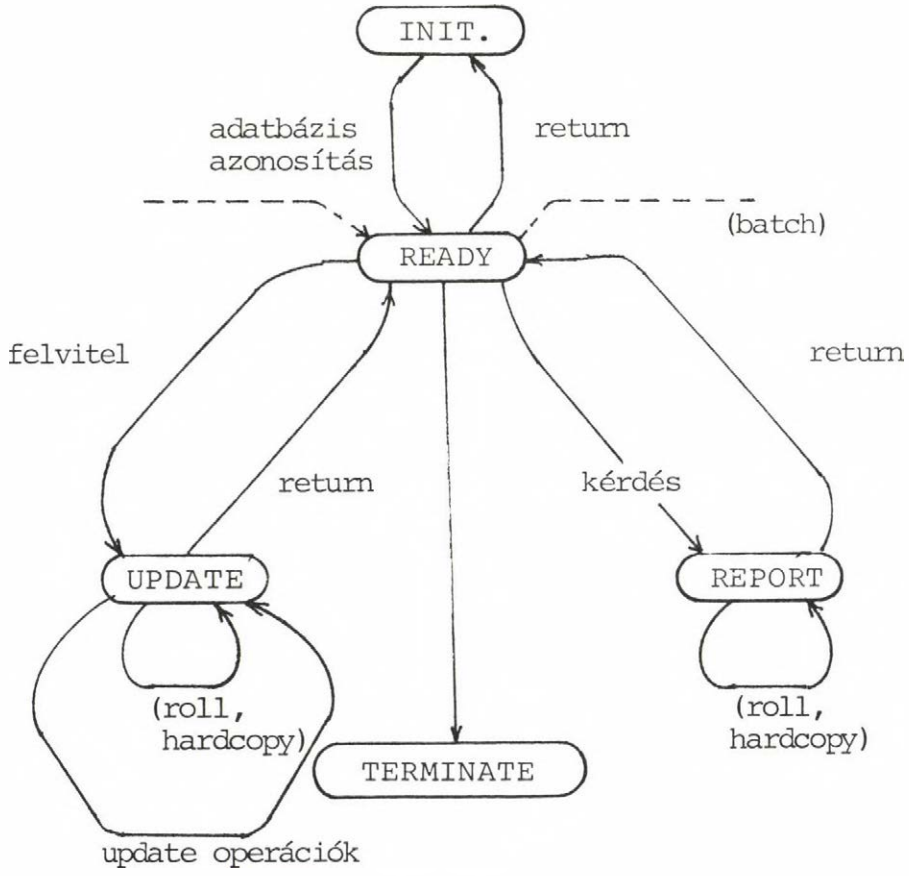
Ma már a fizikai adatábrázolásoknak is jól kialakult, gondosan kidolgozott modelljei ismeretesek (ld. például [53]). Ezek közül a legfontosabbak a VSAM [26] és a hash-technikák [52]. A fizikai adatbázis modell ezen (algoritmikusan általában igen összetett) technikákat valósítja meg olyan módon, hogy felületére a logikai adatmodell leképezhető legyen. (Ezt a leképezést valósítja meg az előző, a "logikai adatbázis modul".)

Alapeszköz modul

Végül célszerű külön modulban összefogni azokat az alsó szintű eszközöket, melyek az összes eddig mondottak kiszolgálását (a gép fájl rendszerére építve) lehetővé teszik. Ilyenek pl. a lap-kezelés (LRU algoritmus), rekord-kezelés stb. Ezen eszközöket célszerű bármely szintről hozzáférhetővé tenni (ezért nem szigorúan réteges szerkezetű az 5/2.ábrán megadott modul hozzáférési rend).

5.2.2 Operációs séma

az aktív modulok esetén tekinthető az elsődleges specifikációnak. Példaképpen ezt az 5/3.ábrán a dialógus-vezérlő modulra adjuk meg:



5/3.ábra

Állapot- és átmenet diagram

(Az ábrán látható fő állapotok mellett van természetesen egy sor kevésbé lényeges segéd állapot is.) Esetünkben az állapot/átmenet diagram az operációs séma leírásának megfelelő eszköze, általános esetben azonban más eszközök is szóba jönnek, mindenek előtt a különféle háló modellek, mint ahogy arra a 4.fejezetben példát is mutattunk.

5.2.3 Interfész szerkezet

Az 5/2.ábrán megjelölt interfészek közül adatbázis kezelési szempontból elsősorban a logikai adatbázis interfész és a fizikai adatbázis interfész érdekes. A logikai adatbázis interfész nem más, mint magához a logikai adatmodellhez tartozó adatbázis operációk összessége, úgy ahogy azt a 2. (ill. formálisan a 3.) fejezetben megadtuk.

A fizikai adatbázis interfész specifikációját példaképpen az alábbiakban rövidítve megadjuk a rendezett struktúrákkal dolgozó automatikus sémakezelésű modellre:

Az adatok tárolása lapozásra alapuló VSAM technikával történik, az [53]-ben leírt elvek szerint. Egyidejűleg több VSAM fával dolgozunk. A fa magasságát dinamikusan kezeljük. A legalsó szint kivételével a lapok index-lapok, melyek index-rekordokat tartalmaznak. (A szerkezet pontos leírása [43]-ben megtalálható.) Az operációk ill. az interfész szerkezete ezek után az alábbi (5/4.ábra).

<table>
<tr><td colspan="2" rowspan="2"></td><td colspan="2">operációk</td></tr>
<tr><td>interfész</td><td></td></tr>
<tr><td rowspan="3">fa
manipulációk</td><td>generálás</td><td>create</td><td></td></tr>
<tr><td>kezelés</td><td></td><td>split
split-page
merge</td></tr>
<tr><td>attributum
elérés</td><td>height
cardinal</td><td>find-path</td></tr>
<tr><td rowspan="2">rekord
manipulációk</td><td>elérés</td><td colspan="2">get(key)
get-first(string)
get-next(string)</td></tr>
<tr><td>update</td><td>put
delete
change</td><td></td></tr>
</table>

5/4.ábra

VSAM interfész

Mint az ábrából látható, a tulajdonképpeni, lényegi VSAM operációk (split, merge stb.) az interfésznek nem részei. Ez nem meglepő: e szervezés helyesen mint automatikus szolgáltatás kell megnyilvánuljon, és így az

interfészt használó felsőbb szintek számára már nem állhat rendelkezésre.

További részletek és az alsóbb szintű rutinok leírása a [43] dokumentumban található.

MUTATÓ

ÁBRÁK JEGYZÉKE

HIVATKOZÁSOK

1. Andréka, H., Németi, I.: Los lemma holds in every category. Studia Sci. Math. Hung.,13,361-376,1978.

2. Andréka, H., Németi, I.: Injectivity in categories to represent all first order formulas. Demonstration Mathematica 12, 717-732, 1979.

3. Andréka, H., Németi, I.: Generalization of variety and quasy-variety concept to partial algebras through category theory. Dissertationes Mathematicae (Rosprawy Math.) No.204, 1982.

4. Andréka, H., Burmeister, P., Németi, I.: Quasy equational logic of partial algebras. Bull.Section of Logic, Wroclaw, Vol.9, No.4,193-199, 1982.

5. ANSI X3H2 - Data Description Language. Jan. 1980.

6. Balzer, R. et al.: Informality in Programming Specifications. IEEE Trans.Software Engineering, Vol.SE-4, No.2, 94-103, March 1978.

7. Bekessy, A. and Demetrovics, J.: Contribution to the Theory of Database Relations. Discrete Mathematics, Vol.27, pp.1-10, 1979.

8. Bernus, P. and Hatvany, J.: Design of Integrated Manufacturing Systems. Computer in Industry, Vol.1, No.1, 1979.

9. Biewald, J. P. et al.: EPOS - A specifications and design technique for computer controlled real-time automation systems. Proc. 4th Int.Conf.Software Engineering, Munich, Sept. 1979.

10. Boehm, B.W.: Software Engineering. IEEE Trans.Computers, Vol.C-25, 1226-1241, Dec.1976.

11. Burstall, R.M., Goguen, J.A.: Putting theories together to make specifications. Proc.5th Joint Conf.on Artificial Intelligence, Cambridge, Mass.,USA, 1977.

12. Chen, P.: The Entity-Relationship Model—Toward a Unified View of Data. ACM Trans.Data Sci., Vol.1. No.1, 9-36, 1976.

13. Codd, E.F.: Relational completeness of data base sublanguages. Rustin, R.ed. Data Base Systems, Prentice-Hall.65-98, 1972.

14. Dahl, O.J., Dijkstra, E.W., Hoare, C.A.R.: Structured programming. Academic Press. 1972.

15. Demetrovics, J., Knuth, E., Radó, P.: Specification Meta Systems. IEEE Computer, Vol.15, No.5, 29-35,1982.

16. DeRemer, F. and Kron, H.H.: Programming-in-the-Large versus Programming-in-the-Small. IEEE Trans.Software Engineering, Vol.SE-2,No.2,80-86, June 1976.

17. Furtado, A.L., Kerschberg, L.: An algebra of quotient relations. Proc.of Sigmond Conf.,1-8, 1977.

18. Guttag, J.V. et al.: The design of data type specifications. 2nd Int. Conf. on Software Engineering, San Francisko, 1976.

19. Howden, W.: Contemporary Software Development Environments. ACM Software Engineering Notes, Vol.6, No.4, 6-15, 1981.

20. Information Processing Systems - Open Systems Interconnection - Basic Reference Model. Draft International Standard ISO/DIS 7498, ISO 1982. (Open Systems)

21. Jackson, M. A.: Principles of Program Design. Academic Press. 1975.

22. Kampen, G.: Experience Using Automated Software Tools in the Specification and Design of a Large Aerospace Application. Proc.Conf.Use of Computer Aids in System Development. Systems Designers Limited, U.K., 1981.

23. Kernighan, P.: Software Tools. Addison-Wesley. 1978.

24. Kiss O.: Interaktív adatbázis modell javaslat. Kézirat. MTA SZTAKI, Budapest. 1983.

25. Knuth, D.E.: Programming is an art. Comm. ACM, 1969.

26. Knuth, D.E.: The art of programming. Part I-II-III. Addison-Wesley, 1968., 1969., 1973.

27. Knuth, E.: Cycles of partial orders. Lecture Notes in Computer Science, 64, 315-325, Springer Verlag, 1978.

28. Knuth, E.: Petri nets and trace languages. 1st European Conf. on Parallel and Distributed Processing, 51-57, Toulouse, France, 1979.

29. Knuth, E., Győry, G., Rónyai, L.: Grammatical projections. MTA SZTAKI Working Paper II/3, 1-21, 1979.

30. Knuth, E., Radó, P., Tóth, Á.: Preliminary description of SDLA. MTA SZTAKI Tanulmányok 105, pp.1-80, 1980.

31. Knuth, E.: MDB-1. - Project Proposal. MTA SZTAKI Working Paper II/26,1-27, 1981.

32. Knuth, E., Győry Gy., Rónyai, L.: A study on the projection operation. In: Application and Theory of Petri Nets. - Selected paper from the First and Second European Workshop. Ed.: Girault, Reising. Lecture Notes in Computer Science, Springer Verlag, 203-208, 1982.

33. Knuth, E., Halász, F., Radó, P.: SDLA System Descriptor and Logical Analyzer. In: Information Systems Design Methodologies: A Comparative Review. Ed.: Olle, Sol and Veerijn-Stuart. North Holland, 143-177, 1982.

34. Knuth, E., Radó, P.: Principles of computer aided system description. MTA SZTAKI Tanulmányok 117, 1-48, 1981.

35. Knuth, E.: Relational operations on reference data structures. Technische Univ., Dresden, Vertr.zu Abstrakten Datentypen, Heft 52, 48-55, June 1982.

36. Knuth, E., Rónyai, L.: Leíró jellegű információk kezelésének alapkérdéseiről. MTA SZTAKI Working Paper II/48, 1-21, 1982.

37. Knuth, E., Szabó, S.: MDB-3. - Lower level dialog interface. MTA SZTAKI Working Paper II/29, 1-7, 1982.

38. Knuth, E., Radó, P., Rónyai, L.: MDB-2. - Kernel Definition. MTA SZTAKI Working Paper II/28, 1-16, 1982.

39. Knuth, E.: Software structuring: a pragmatic approach. In: Specification and Design on Software Systems. Lecture Notes in Computer Science 152, Springer Verlag, 16-27, 1983.

40. Knuth, E., Rónyai, L.: Data bases for system descriptions. MTA SZTAKI Working Paper II/58, 1-7, ú983.

41. Knuth, E., Rónyai, L.: Closed convex reference schemes. Proc.of the IFIP TC-2 Conference on System Description Methodologies, Kecskemét. North Holland, 1983 (in press).

42. Knuth, E.: MDB Data Model. MTA SZTAKI Working Paper II/59. 1-13, 1983.

43. Knuth, E., Szilléry, A., Rónyai, L.: MDB. Part 1-6. MTA SZTAKI Internal Document. 1-120. 1982.

44. Knuth E., Rónyai L.: Az SDLA/SET lekérdező nyelv alapjai. MTA SZTAKI Working Paper II/24, 1981.

45. Lauer, P.E., Campbell, R.H.: Formal semantics of a class of highlevel primitives for co-ordinating concurrent processes. Acta Informatica 5, 297-332, 1975.

46. Lautenbach, K.: Liveness in Petri nets. G.M.D.Internal report ISF-75-02-1, Bonn, July 1975.

47. Levene, A.A., Mullery, G.P.: An Investigation of Requirement Specification Languages: Theory and Practice. IEEE Computer Vol.15, No.5, 50-59, 1982.

48. Liskov, B., Zilles, S.: Specification techniques for data abstractions. IEEE Tr.on Software Engineering 4, 7-19, 1975.

49. Ludewig, J. and Streng, W.: Methods and Tools for Software Specification and Design - A Survey. EWICS TC-7, Paper No. 149. Zürich, April 1978.

50. Ludewig, J.: Process Control Software Specification in PCSL, In: V.Haase (ed.) IFAC/IFIP Workshop on Real-Time Programming, Graz. Pergamon Press, 103-108, 1980.

51. Ludewig, J.: Computer-Aided Specification of Process Control Systems. IEEE Computer, Vol.15, No.5, 12-20, 1982.

52. Marshall, C.: Advances in computers. Academic Press. 1981.

53. Martin, J.: Computer Data Base Organization. Prentice Hall. 1975.

54. Mazurkiewicz, A.: Invariants of Concurrent Programs. Proc. of the IFIP-INFOPOL Conf.on Information Processing. North Holland. 353-372, 1977.

55. McLeod, D.J.: High level domain specification in a relational data base system. Proc. AM SIGPLAN "Data: Abstraction, Definition and Structure". Salt Lake City, USA. 47-57, 1980.

56. Nakajima, R., Honda, M., Nakahara, H.: Describing and verifying programs with abstract data types. Formal Description of Programming Concepts. Ed.: Neuhold, E.J. North Holland. 527-556. 1978.

57. Németi, O., Sain, I.: Cone-implicational subcategories and some Birkhoff-type theorems. In: Universal Algebra, Colloq. Math.Soc.J.Bolyai,29, North Holland. 535-578. 1981.

58. Petri, C.A.: Concepts of Net Theory. MFCS 73 Proc., High Tatras, Math.Inst.of Slovak Acad.of Sci. 137-146. 1973.

59. Requirement Specification Language Study REP7: Specification of Requirements Using an Automated Aid. Systems Designers Limited, UK.

60. Rosene, A.F., Connolly, J.E., Bracy, K.M.: Software maintability - What it means and how to achieve it. IEEE Tr.on Reliability R-30, 3, 240-245. 1981.

61. Ross, D.T.: Structured analysis: a language for communicating ideas. IEEE SE 3, 1. 1977.

62. Sanderson, J.G.: A Relational Theory of Computing. Lecture Notes in Computer Science 82, Springer Verlag. 1-147. 1980.

63. Schmidt, J.W.: Some high-level language constructs for data of type relation. ACM TODS,V.2.No.3. 247-261. 1977.

64. Szilléry, A.: MDB User Guide. MTA SZTAKI Internal Document, 1-53, 1984.

65. Teichroew, D. and Hershey III, E.A.: PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems. IEEE Trans.Software Eng.,Vol.SE-3,No.1. 41-48. 1977.

66. Tompa, F.W.: A practical example of the specification of abstract data types. Acta Informatica 13, 205-224. 1980.

67. Wulf, W., London, R., Shaw, M.: An Introduction to the Contruction and Verification of Alphard Programs. IEEE Tr.on Software Engineering 1, 253-364. 1976.

68. Wassermann, A.I.: Characteristics of Software Development Methodologies. In: Information System Design Methodologies: A Feature Analysis (eds.: Olle, T.W., Sol, H.G. and Tully, C.J.). Elsevier Science Publishers B.V.(North Holland) 37-58. 1983.

A TANULMÁNYSOROZATBAN 1983-BAN MEGJELENTEK

140/1983 Operation Research Software Descriptions (Vol.1.) Szerkesztette: Prékopa András és Kéri Gerzson

141/1983 Ngo The Khanh: Prefix-mentes nyelvek és egyszerü determinisztikus gépek

142/1983 Pikler Gyula: Dialógussal vezérelt interaktiv gépészeti CAD rendszerek elméleti és gyakorlati megfogalmazása

143/1983 Márkusz Zsuzsanna: Modellelméleti és univerzális algebrai eszközök a természetes és formális nyelvek szemantikaelméletében

144/1983 Publikációk '81 /Szerkesztette: Petróczy Judit/

145/1983 Telcs András: Belső állapotu bolyongások

146/1983: Varga Gyula: Numerical Methods for Computation of the Generalized Inverse of Rectangular Matrices

147/1983 Proceedings of the joint Bulgarian-Hungarian workshop on "Mathematical Cybernetics and data Processing" /Szerkesztette: Uhrin Béla/

148/1983 Sebestyén Béla: Fejezetek a részecskefizikai elektronikus kisérleteinek adatgyüjtő, -feldolgozó rendszerei köréből

149/1983 L. Keviczky, J. Hethéssy: A general approach for deterministic adaptive regulators based on explicit identification

150/1983 IFIP TC.2 WORKING CONFERENCE "System Description Methodologies" May 22-27. 1983. Kecskemét. /Szerkesztette: Knuth Előd/

151/1983 Márkusz Zsuzsanna: On First Order Many-Sorted LOGIC

152/1983 Operations Research Software Descriptions /Vol.2./ Edited by A. Prékopa and G. Kéri

153/1983 T.M.R. Ellis: The automatic generation of user--adaptable application-oriented language processors based on quasi-parallel modules

1984-BEN MEGJELENTEK

155/1984 Deák, Hoffer, Mayer, Németh, Potecz, Prékopa, Straziczky: Termikus erőmüveken alapuló villamos-energiarendszerek rövidtávu, optimális, erőmüvi menetrendjének meghatározása hálózati feltételek figyelembevételével.

156/1984 Radó Péter: Relációs adatbáziskezelő rendszerek összehasonlitó vizsgálata

157/1984 Ho Ngoc Luat: A geometriai programozás fejlődései és megoldási módszerei

158/1984 PROCEEDINGS of the 3rd International Meeting of Young Computer Scientists.
Edited by J.Demetrovics, and J. Kelemen

159/1984 Bertók Péter: A system for monitoring the machining operation in automatic manufacturing systems

160/1984 Ratkó István: Válogatott számitástechnikai és matematikai módszerek orvosi alkalmazása /kandidátusi értekezés/

161/1984 Hannák László: Többértékü logikák szerkezetéről

162/1984 Kocsis J., Fetyiszov V.: Rugalmas automatizált rendszerek: megbizhatóság és irányitási problémák