tanulmányok 104 / 1980

1983 ÁSR

MTA Számítástechnikai és Automatizálási Kutató Intézet Budapest

MAGYAR TUDOMÁNYOS AKADÉMIA
SZÁMITÁSTECHNIKAI ÉS AUTOMATIZÁLÁSI KUTATÓ INTÉZETE

Knuth Előd, Radó Péter, Toth Árpád

AZ SDLA ELŐZETES ISMERTETÉSE

1979. december 20. II/5

Az ebben az ismertetésben közölt koncepciók bár hosszú érlelődésen mentek keresztül, de még számos vonatkozásban nem teljesek és további mély megfontolásokat igényelnek. Ezért az Ön megjegyzéseit is örömmel és komolyan fogjuk venni.

Ezt a kutatómunkát részben a KSH OSZI támogatta.

Tanulmányok 104/1980.

A kiadásért felelős:

DR. VÁMOS TIBOR

ISBN 963 311 102 1

ISSN 0324-2951

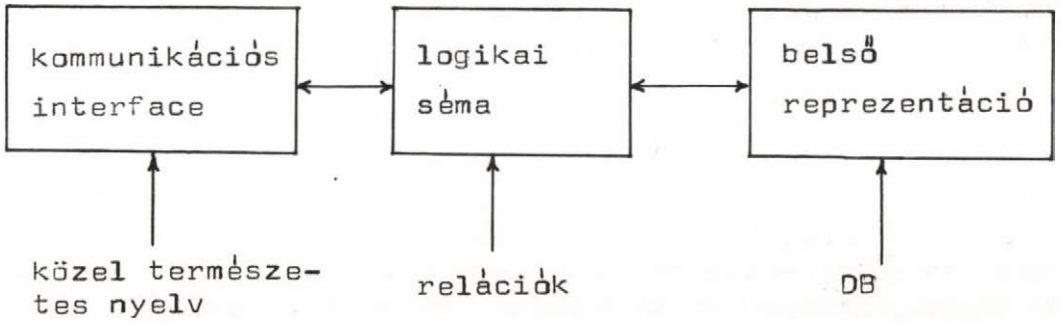
Készült a

KSH Nemzetközi Számitástechnikai Oktató és Tájékoztató Központ
Reprográfiai Üzemében 0047

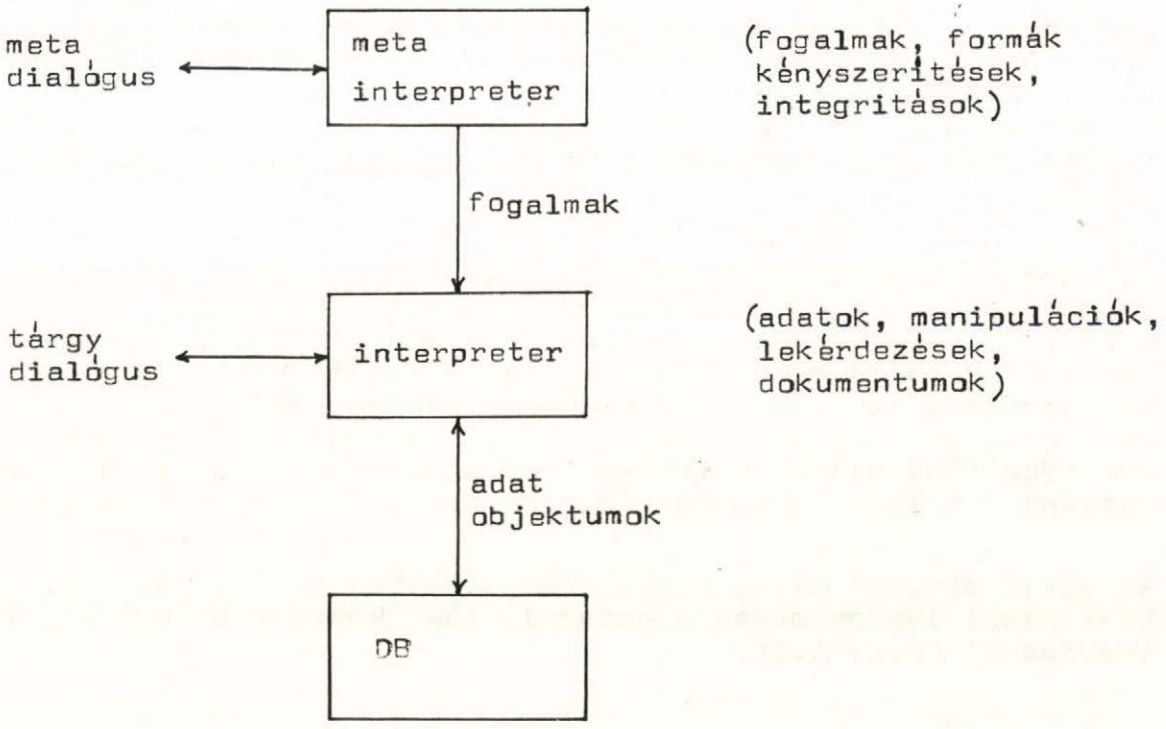
S D L A

(Structural Descriptor and
Logical Analyzer)

Horizontális szerkezet



Vertikális szerkezet



ÁTTEKINTÉS

Célja

logikailag bonyolult szerkezetű rendszerek megismerésének, kezelésének, tervezésének és előállításának segítése olyan módon, hogy a rendszerleírásokat saját adatbázisában tárolja és eszközöket nyújt annak logikai elemzésére és dokumentálására. Mint tudjuk, erre azért van szükség, mert az emberi agy önmagában képtelen a sok komponensből álló és sokféle kapcsolatot tartalmazó komplex rendszerek áttekintésére és kézben tartására.

Helye

Sok hasonló irányú célrendszerhez viszonyítva az SDLA a közös alapot kívánja megvalósítani. Azt, ami valamilyen formában minden ilyen rendszerben szükséges, és amelyre tetszőleges célorientált rendszerek építhetők.

Ma már számos sikeres rendszerleíró nyelv ismeretes, speciális területekre vonatkozók és általánosabbak egyaránt. Utóbbiak közül kiemelkedik pl. a DELTA nyelv, mely az SDLA-hoz hasonlóan "nem procedurális", azaz nem végrehajtható kód előállítását célozza. Az ilyen nyelvekhez viszonyítva a legfőbb különbség az, hogy az SDLA a leírt információt adatbázisszerűen szigorú logikai séma szerint tárolja, és az információ elemzéséhez általános matematikai logikai eszközöket is biztosít. Mindazonáltal, az SDLA leíró nyelvének meghatározásánál sokat merítettünk ezen (adatséma nélküli) nyelvek gazdag eszmevilágából.

Jól ismertek olyan megközelítések is (ISDOS, SADT, stb.), melyekhez adatbázisban való ábrázolás is tartozik (vagy rendelhető). Ezek jellemzője, hogy meghatározott fogalmakkal dolgoznak, és (érthető módon) csak előre definiált kapcsolattípusokat engednek meg. (Az SADT esetében például objektumok közötti irányított összeköttetések, stb. szerepelnek). Mi semmit sem rögzítünk előre. Ellenkezőleg: azt a mechanizmust akarjuk megadni, amely a fogalomdefiníciót teszi a rendszer alapjává.

Az elmondottakon kívül számos más lényeges eltérés is van. Ezek közül legfontosabb a metarelációk (kényszerítések és finomítások) bevezetése.

Alkalmazása

a fogalomrendszer és invariánsai (kényszerítések és integritások) megadásával kezdődik. Ilyenek könyvtárszerűen is tárolhatók. A fogalmak rögzítése után egy interaktiv adatbáziskezelő rendszerhez jutunk egy magasszintű felhasználói interface közvetítésével. Ez nem kommerciális, célja nem nagytömegű, hanem bonyolult logikai összefüggéseket ábrázoló adatok adekvát kezelése tervezési szempontok figyelembevételével.

Eszközei

1. Alapfogalmak és asszociációk egységes kezelése. Az asszociáció is fogalom. (Codd féle szemlélet elvetése).

2. Hivatkozási szemlélet. Ez megszabadit a gondtól, vajon a hivatkozott objektum egyszerű vagy összetett; biztositja továbbá a rekurziv adatszerkezetek egyszerű kezelését.

3. Szemantikai kényszerítések. Ez alatt olyan összefüggések deklarálását értjük, melyek az adattipusok jelentéséből következnek, és meghatározott új állitások automatikus generálását eredményezik.

4. Szemantikai ellenőrzések (integritás). Ezek reláció tulajdonságok ill. relációk közötti viszonyok formájában deklarálhatók, és a beérkező adatok (leirások) logikai korrektségének ellenőrzésére használhatók.

5. Fogalmak finomitásának mechanizmusa (típus - altípus). Fő célja az árnyalt típusellenőrzés megvalósitása.

* * *

A szerzők köszönetet mondanak Almásy Gedeon, Bach Iván, Benczur András, Bródy Ferenc, Dömölky Bálint, Farkas Ernő, Gáspár András, H.Kangassalo, Kiss Olivér, Kovács László Béla, Márkus András, C. A.Petri, Prékopa András, Rónyai Lajos, Szeredi Péter, Szlankó János, Szokolov Makar, és Sztanó Tamásnak megjegyzéseikért.

TARTALOM

I. AZ ÁBRÁZOLT ADATOK LOGIKAI SÉMÁJA

Az ábrázolt adatok logikai sémája azt (az adatok fizikai reprezentációjától független) szemléletmódot jelenti, ahogyan a felhasználónak adatai struktúráját kell felfognia. Ez a szemlélet relációs szemlélet, mely azonban a Codd-félétől jelentősen eltér és célja is egészen más.

Az adatbázissal való kommunikáció egy magasabbrendű, az emberi gondolkodáshoz közel álló nyelven történik, melyet a második fejezetben ismertetünk. E nyelvet egy jól definiált leképzés kapcsolja össze a relációk világával. Nem reménytelen vállalkozás olyan fejlesztésre is gondolni, melyben a felhasználónak a relációk logikai világába sem kell leereszkednie. Ez a jövő zenéje.

A logikai séma alapgondolata a "zárt referencia-mezők homomorfizmusa", ld.[6]. Jelen ismertetésben nem térünk ki absztrakt részletkérdésekre, helyette praktikusabb, felhasználói vonatkozású szemléletről beszélünk. A logikai szerkezet második fő motivuma a relációk közötti alárendeltségi viszonyok (finomítás, alárendeltségi kényszerítés) bevezetése, mely a valóság finomabb ábrázolásának döntő eszköze.

1. Alapséma

Az adatbázisban objektumokat tárolunk. Minden objektum valamely absztrakt fogalom egy konkrét példánya.

Az objektumok attributumaik által kerülnek leírásra. Az absztrakt fogalom mindig egy a fogalomhoz rendelt attributum együttes, melyhez az objektum példányok attributumai számuk és típusuk szerint (a konvencionális paraméterátadáshoz hasonlóan) illeszkednek.

Egy adott fogalomhoz tartozó objektum példányok aktuális halmaza mindig felfogható relációnak (azaz az attributum tartományok Descartes-szorzata részhalmazának). Ez a szemlélet, mint ismeretes, az adatbázis műveletek formalizálása szempontjából hasznos.

1.1 Fogalmak kijelölése

Egy fogalom megadásával adat objektumok egy osztályának előzetes kijelölését végezzük. A fogalmak megadása a meta- vagy definiciós szinten történik. A fogalomdefinicióra az alábbi formális jelölést fogjuk alkalmazni:

(1) concept fogalomnév(attributumnév1:tipus1,
attributumnév2:tipus2, ... stb.);

Ennek megfelelően, egy fogalom definiciója a következőket jelenti:

a) A fogalom nevének megadását;

b) Az attributumok számának megadását (nemnegativ egész);

c) Minden egyes attributum nevének (szelektor), és tipusának megadását.

1.1.1 Attributum tipusok

A definicióban szereplő tipusmegjelöléseknek két fajtája van:

a) hivatkozási tipus,

b) érték tipus.

A hivatkozási tipus tetszőleges (szintén definiált) fogalomnév. (Ismételjük: tetszőleges. Minden további nélkül lehet akár önmaga is. Nincsen különbség. Nem rekurzió, ugyanis hivatkozásról van szó.)

Érték tipusként a következő megjelöléseket lehet alkalmazni:

(2) integer ,
real ,
text .

1.1.2 Definiciós egység

A definiciós egység azoknak a fogalom definicióknak sorozata, melyeket a definiciós szinten megadunk. A definiciós egység zárt (vagy: "önmagában korrekt"), ha minden benne előforduló attributum tipus az egységen belül definiálva van.

Bár az elmondottakból következnek, felhivjuk a figyelmet a következőkre:

a) A definiciók sorrendjének (természetesen) nincsen jelentősége. Egy definiciósorozat (definiciós egység) korrektségének a zártság szükséges és elégséges feltétele.

b) Az attributumok száma nemnegativ. Lehet tehát egy is, nulla is. Ennek megértése nehézséget okoz, ha abban a

gondolatkörben mozgunk, mely szerint attributumok közötti asszociációkkal van dolgunk, más szóval legalább két dolog kell, hogy valamik közötti kapcsolatról beszélhessünk.

Valójában itt általánosabb értelemben vett fogalmakról van szó. Többek közt, az asszociációt (kapcsolatot) is fogalomnak tekintve. A fogalmakat szemünkben egyszerüen attributumaik reprezentálják, mégpedig annyi, amennyi az adott környezeti összefüggésben, a gondolkodás adott absztrakciós szintjén releváns. A nulla attributumszám ilyenformán egyszerű osztálykijelölés, melynek azonban mint objektumokra alkalmazható minősitésnek fontos szerepe van.

1.1.3 Példák

Az alábbi példák mindegyike (önmagában korrekt) zárt definiciós egység.

(3)
```
defunit
    concept tárgy;
endunit
```

Minthogy egyetlen attributumot sem adtunk meg, valóban minden megadott attributum tipus a definiált fogalmak közé tartozik. Kijelöltünk egyetlen osztályt, melyet (mondjuk általánossága miatt, egyelőre) nem kivánunk atrributumokkal jellemezni.

(4)
```
defunit
    concept lánc elem(következő:lánc elem);
endunit
```

(5)
```
defunit
    concept binfa elem(bal:binfa elem,jobb:binfa elem);
endunit
```

Mindkét fenti példa zárt, önmagában korrekt definiciós egység. A (4)-ben definiált "lánc elem" lehetővé teszi majd pl. egyirányú lista szerkesztését (persze kör, fa, stb. is készithető ilyen elemekből). Hasonlóan, az (5)-ben szereplő fogalom segitségével (többek között) bináris fákat készithetünk.

(6)
```
defunit
    concept férfi;
    concept nő;
    concept házasság(férj:férfi,feleség:nő);
endunit
```

(7)
```
defunit
    concept férfi(feleség:nő);
    concept nő(férj:férfi);
endunit
```

Fenti példák heterogén párkapcsolatok ábrázolásának két lehetséges változatát mutatják. A feladat dönti el, melyik ábrázolás előnyösebb. Mindkét definiciós egység láthatóan önmagában zárt. (7)-ben példát láthatunk arra, hogy egy attributum tipus (első sorban: "nő") csak később kerül definiálásra. Mint már mondottuk, ez érdektelen.

(8)
```
defunit
    concept elem;
    concept rendezés(megelőző:elem,megelőzött:elem);
endunit
```

E példa fogalmai lehetőséget adnak (véges karakterű) teljes és parciális rendezés, háló, stb. leirására. Mint látjuk, egyelőre nincsen deklarált információnk a fogalom szemantikai tartalmáról (pl. antiszimmetria). Erről később lesz szó.

1.2 Adatobjektumok megadása

Tegyük fel, hogy előzetesen megadtunk egy zárt definiciós egységet. Ezzel módot nyitottunk adatobjektumok leirására. A következő jelölést fogjuk alkalmazni:

(9) fogalomnév objektumnév(attr1,attr2,...stb.);

és

(10) fogalomnév (attr1,attr2,...stb.);

Az aláhúzott "fogalomnév" a definiciós részben definiált fogalom neve kell legyen. Ezt a létrehozott adatobjektum minősitésének (tipusának) fogjuk nevezni.

A (9)-ben szereplő "objektumnév" a megadott objektum névszerinti azonositását teszi lehetővé. Amennyiben a (10) formát használjuk, a leirt objektum nem lesz elérhető neve alapján, de ugyanúgy létező (és pl. globális kereső eljárásokkal visszanyerhető) adattételt eredményez.

1.2.1 Tipus illeszkedés

A (9) és (10)-ben zárójelben megadott attributumoknak ("attr1", "attr2", stb.) számuk és tipusuk szerint meg kell felelniük a definicióban megadott attributum specifikációknak. Ez pontosabban a következőket jelenti:

a) Ha az attributum érték-tipusú, akkor az objektum leirásban ennek megfelelő értéket kell megadni.

b) Ha pedig hivatkozási tipusú, akkor olyan objektum nevét kell megadni, melynek minősitése megegyezik a a szóbanforgó hivatkozási (fogalmi) tipussal.

c) Mindkét esetben megengedjük az attributum helyének üresen hagyását (a "meg nem adást") ha értékét (még) nem ismerjük.

1.2.2 Adatleirási egység

Az adatleirási egység objektum leirások ((9) vagy (10)) sorozata. Az adatleirási egység zárt, ha hivatkozási értékként nem tartalmaz olyan aktuális paramétert (attributumként megadott objektumot), mely az egységen belül nincsen maga is leirva.

Ez a felfogás lehetővé teszi, hogy az adatleirási szinten is megszabadulhassunk a szekvenciális értelmezhetőség terhes igájától, (mely egyébként procedurális nyelvek hasonló konstrukcióiban persze nem megvalósitható). Ilyenformán, a definiciós egységhez hasonlóan, a sorrend adatleirások esetén is érdektelen. Az igy nyert kellem rögtön szembetünik, mihelyt megkisérelj ük pl. a (15) példát mondjuk SIMULA-ba átirni.

1.2.3 Példák

Az alábbi példa (4) felhasználásával egy (egyelőre persze meglehetősen öncélú) listát ad meg.

(11)
```
defunit
   concept lánc elem(következő:lánc elem);
endunit;

dataunit
   lánc elem A(B);
   lánc elem B(C);
   lánc elem C(D);
   lánc elem D( );
endunit
```

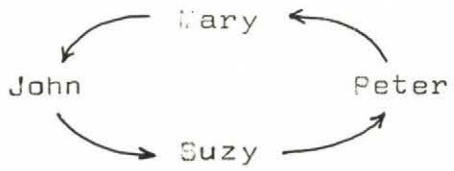
Hasonlóan, (5)-re hivatkozva egy bináris fa megadása pl. az alábbi lehet:

(12)
```
defunit
   concept binfa elem(bal:binfa elem,jobb:binfa elem);
endunit;

dataunit
   binfa elem gyökér(A,B);
   binfa elem A(C,D);
   binfa elem B(E,);
   binfa elem C(,F);
   binfa elem D(,);
   binfa elem E(,);
   binfa elem F(,);
endunit
```

Észrevehetjük, hogy a (11) és (12) példák utolsó sorai meglehetősen semmitmondóak, csak az egység zártsága érdekében szükségesek. Megjegyezzük, hogy még mindig a logikai sémáról beszélünk, és a (II. fejezetben részletezendő) felhasználói nyelvben ilyenek megadása nem szükséges.

(13)
```
defunit
   concept férfi;
   concept nő;
   concept házasság(férj:férfi,feleség:nő);
endunit;

dataunit
   férfi John;
   férfi Peter;
   házasság (John,Mary);
   házasság (Peter,Mary);
   nő Mary;
endunit
```

Mint látjuk, (6) felhasználásával a fenti példában bigámiát ábrázoltunk. Ezt (7) használata kizárja (egyértelműsége miatt), ellenben lehetőséget ad pl. egy "reménytelen szerelmi ciklus" leírására:



(14)
```
defunit
    concept férfi(sweetheart:nő);
    concept nő(sweetheart:férfi);
endunit;

dataunit
    férfi John(Suzy);
    nő Suzy(Peter);
    férfi Peter(Mary);
    nő Mary(John);
endunit
```

Fenti "egyoldalúságot" viszont (6) nem engedte volna meg. Mindez azt mutatja, hogy a fogalmak megfelelő megválasztasa létfontosságú az adatleirás kényelme, biztonsága, adekvátsága szempontjából. Ez azonban még más eszközöket is igényel, ld. később.

(15)
```
dataunit
    férfi John(Suzy);
    nő Suzy(John);
    férfi Peter(Mary);
    nő Mary(Peter);
endunit
```

E példában (még mindig (7) ill. (14)-re hivatkozva) a párkapcsolatokat kölcsönössé tettük. A megadás (a procedurális nyelvekkel ellentétben) egyszerü és világos, ami annak következménye, hogy nem ragaszkodtunk a szekvenciális értelmezéshez.

1.3 Egyenlőség

Az alapséma döntően fontos kérdése az, mikor tekintünk két fogalmat, ill. két adatobjektumot azonosnak. Nos nemes egyszerűséggel: semmikor. Ehhez azonban némi kommentárt fűzünk.

1.3.1 Fogalmak egyenlősége

Két fogalom mindig különböző. Ennek elfogadása csak akkor okozhat némi fejtörést, ha történetesen összes attributumaik tipusában páronként rendre megegyeznek. Vegyük azonban tudomásul, hogy elsősorban osztálykijelölésről (egy minőség definíciójáról) van szó, melyben a konkrét szerkezet (vagyis a relevánsnak tekintett attributumok együttese) másodlagos.

Legyen például

(16) concept man;
concept woman;

E két tipus összes attributumában rendre megegyezik, minthogy nincsenek nekik. Azonosnak nyilvánitásuk mégsem éppen gyümölcsöző álláspont. (Más kérdés, hogy általánosságban "ember"-ről is célszerű lehet beszélni, mellyel a "férfi" és "nő" ha nem is egyenlő, de valamilyen "meta"-értelemben része. Erről később lesz szó.)

1.3.2 Adatok egyenlősége

Adatok egyenlőségéről természetesen csak azonos tipusok esetén lehetne szó. Két adatot azonban (még ekkor is) minden esetben különbözőnek tekintünk. Megintcsak az okozhat problémát, ha a két objektum attributumainak értékei rendre megegyeznek. Itt a név szerepére hivjuk fel a figyelmet. Az adatobjektumok egyedisége az adatokat azonositó nevek alkalmazásának következménye. Például:

(17) concept dolgozó(hely:osztály,szülév:integer);

eseten az alábbi két adatobjektum

(18) dolgozó Pintér(munkaügy,1950);
dolgozó Sintér(munkaügy,1950);

minden józan meggondolás szerint különböző (noha tartalmuk azonos).

Mindazonáltal megjegyezzük, hogy az itt alkalmazott egységes szemlélet nem az egyedüli lehetőség. Olyankor, amikor az adatobjektum jelentésében az általa reprezentált kapcsolat szemantikai tartalma dominál (és az esetleges név másodlagos) kényelmesebb lehetne a nevek közönséges (text tipusú) attributumként való kezelése. Ez az egyenlőségi problémát egészen más megvilágitásba helyezné. (E problémát azonban másképp fogjuk megközeliteni, ld. később "függvénykapcsolatok"-nál).

1.4 Konkluzió

Kétféle egységről, nevezetesen definiciós- és adategységről beszéltünk. Mindkettőtől a zártságot követeltük meg. A zártság teljesen formális tulajdonság, mely a korrektség szükséges és elégséges feltétele, és amely

szüksegtelennè teszi hogy tartományok meghatározásának kérdéseivel kelljen foglakoznunk.

E formális tulajdonság és a referencia szemlélet révén érdektelennè válik a definiciók/objektumok sorrendje, továbbá a látszólagos rekurzió.

A két egységet a tipusilleszkedés szabálya kapcsolja össze.

2. Relációk

2.1 A rélációs szemlélet

2.1.1 Fogalmakhoz rendelt táblázatok

Fogalmainkat úgy fogjuk tekinteni, hogy egyúttal egy relációt definiálnak, melyet pl. egy kitöltetlen táblázattal szemléltethetünk, ld.1.ábra. A fogalom neve egyúttal a reláció (táblázat) azonositója. A szelektorok (attributumnevek) a táblázat oszlopait azonositják. Legyen például:

(19) concept felhasználói útmutató(tárgy:program, környezet:alrendszer,katalógusszám:integer);

ahol feltételezzük, hogy a "program" és az "alrendszer" szintén definiált (esetleg szintén összetett - ez a referencia szemlélet miatt érdektelen!) fogalmak. Ekkor a megfelelő táblázat:

FELHASZNÁLÓI ÚTMUTATÓK TÁBLÁZATA:

1.ábra

A táblázat sorai az adatleirások hatására kerülnek kitöltésre. Az egyes oszlopokba csakis olyan elemek kerülhetnek, melyek ti-

pusa a definicióval összhangban van. Esetünkben:

a) Az első oszlopba mindig program-objektumra való hivatkozás kerül;

b) A második oszlopba mindig alrendszer-objektumra való hivatkozás kerül;

c) A harmadik oszlopba egész érték kerül.

Például:

(20) felhasználói útmutató (bérlista,pénzügy,1713);

hatására:

tárgy	környezet	katalógusszám
hivatkozás a "bérlista" -programra	hivatkozás a "pénzügy" -alrendszerre	1713

2.ábra

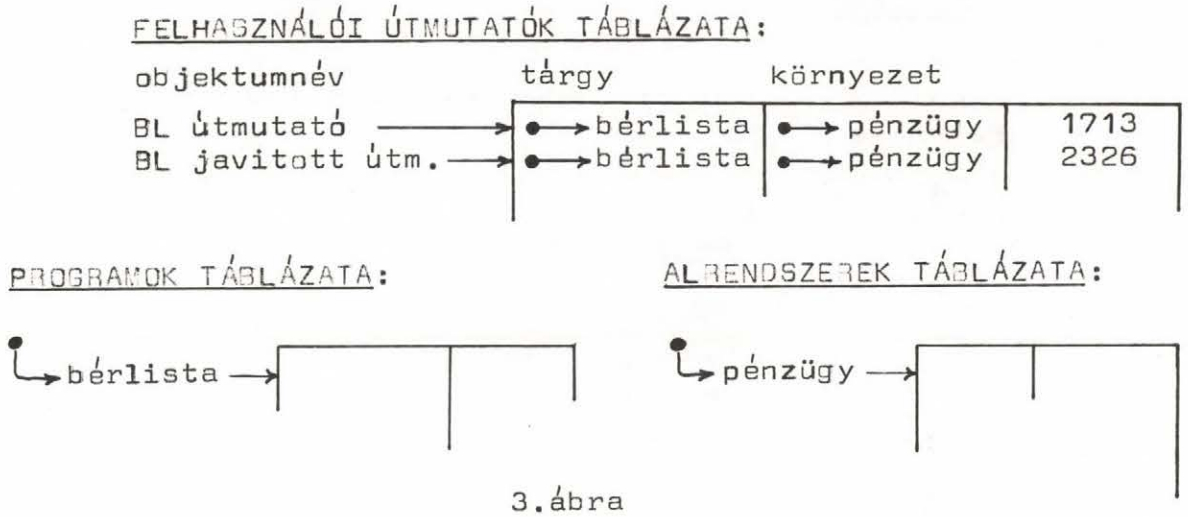
2.1.2 Egyedi nevek

Mint láttuk, a (9) forma alkalmazása esetén az adattételek egyedi nevekkel lesznek ellátva, mely alapján közvetlenül hivatkozhatunk rájuk. Ezt egy a relációkhoz rendelt névszerinti elérési mechanizmus biztositja. Például:

(21) felhasználói útmutató BL útmutató(bérlista, pénzügy,1713);

felhasználói útmutató BL javitott útmutató(bérlista, pénzügy,2326);

esetén:



3.ábra

Fontos tudni azonban, hogy a hivatkozási értékek kivánt beállitása érdekében az egyedi nevek alkalmazása nem elengedhetetlen. Ennek kényelmes módját a felhasználói nyelvben fogjuk látni. Másfelől, mint mondottuk, az egyedi névvel nem rendelkező tételek is egyenrangúak, és közvetett módon szintén elérhetők.

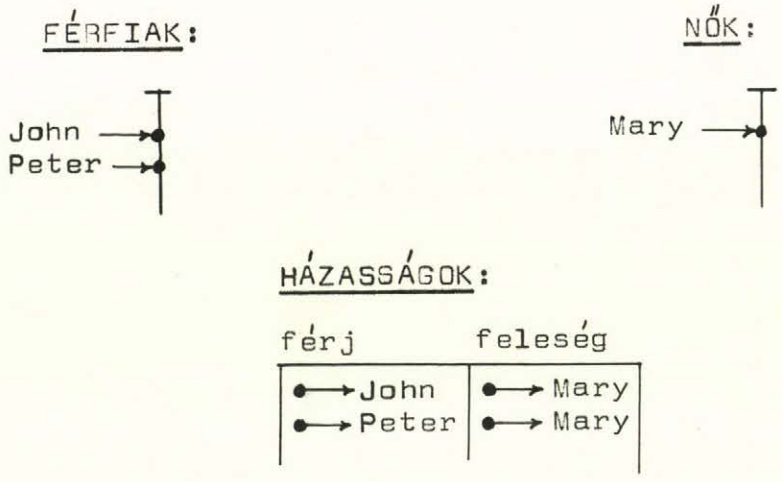
2.1.3 Elfajult relációk

Minthogy a fogalmak attributumszáma lehet akár nulla vagy egy, a megfelelő táblázatok oszlopszáma ugyanúgy lehet nulla vagy egy. Ez azonban csak első pillanatra különös, valójában semmi különbséget nem jelent a (kettőnél kevesebb változós vagy a legalább két változós) relációk kezelése.

Ha az attributumszám nulla, akkor az objektum ugyan nem tárol semmilyen információt, de mint (absztrakt) egyed mégis létezik, és

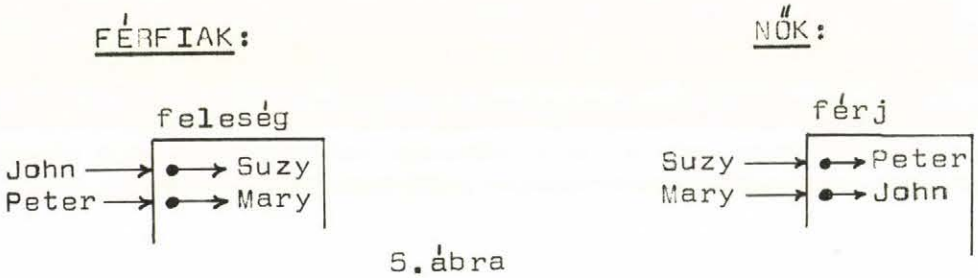
a) hivatkozni lehet rá,
b) esetleg neve is van.

Példaképpen bemutatjuk a (13) példában látott bigámia relációs szemléletét:



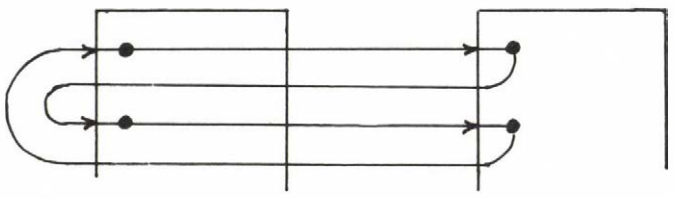
4.ábra

Hasonlóan, a (14) példa esetén az alábbi ábrázolást kapjuk:



5.ábra

ami a hivatkozások tényleges berajzolása esetén az alábbi formát mutatja:



6.ábra

Erről az ábráról az látható, hogy példánkban a referencia-séma a valóságos viszonyok pontos modelljét adja (mely maga is hivatkozásokból áll), míg a szokványos relációs modell esetén a hivatkozási viszonyok csupán szövegekbe kódoltan ábrázolhatók (és igy addicionális, attributum-szöveg alapján visszakereső mechanizmust igényelve).

2.2 Reláció műveletek

Mint mondottuk, a relációs szemléletnek kulcsszerepe van a visszakeresésben, a leirt információ elemzésében. Mielőtt erről szólnánk, áttekintjük a szóbajövő főbb műveleteket. A Codd-féle modell itt megfelelő kiindulópont számunkra, az alábbiak azonban szemléleti eltérést jelentenek:

a) A hivatkozási tipusú oszlopok révén relációk relációkon is lehetnek értelmezve. (Itt többről van szó mint a Codd-féle 1. normálalak elvetéséről, mivel mi az atomokra való visszavezethetőséget sem feltételezzük, másfelől érték tipusok esetleg egyáltalán nincsenek is egyes modell-leirásokban).

b) A szemantikai kényszeritések (ld. később) bizonyos projekciók megkülönböztetett kezelésához vezetnek.

Alapelvünk a következő. Az előző részben minden fogalomhoz egyértelmüen hozzárendeltünk egy relációt (táblázatot). Ezeket a rendszer alapreláciőinak, a hozzájuk tartozó fogalmakat pedig a relációk tipusainak fogjuk nevezni. A reláció műveletek segitségével az alaprelációkból új relációkat származtathatunk. A származtatott relációk közül azonban csak bizonyosakhoz rendelünk tipust. A tipussal nem rendelkező relációk felhasználását ugyanakkor korlátozzuk.

2.2.1 Nagyitás ("Zoom")

Ezen első és alapvető művelet csak referencia reláció univerzumban értelmezhető, a klasszikus reláció kalkulusban megfelelője természetesen nincsen (igy Codd-nál sem).

Formája a következő:

(22) reláció.szelektor

vagy általánosabban:

(23) relációkifejezés.oszlopkiválasztás

ahol az oszlopkiválasztás vagy sorszám vagy szelektornév (utóbbi csak alapreláció esetén), és ahol feltételezzük, hogy a kiválasztott oszlop hivatkozási tipusú. (Vigyázat: nem tévesztendő össze e jelölés a később, az objektumoknál alkalmazandó hasonló jelöléssel - azaz a szokványos "távoli eléréssel").

Jelentése a következő. A megadott relációból elhagyunk minden oszlopot, kivéve a kiválasztottat, majd ezen belül elhagyjuk a duplikációkat. Ennek elemei most mind olyan tipusúak, mely defi-

nició szerint az oszlophoz van rendelve. Legyen ez T. Ezután minden elemet helyettesitünk annak T-beli tartalmával. Az igy nyert új reláció a T relációnak halmazelméletileg része lesz. Az új relációhoz minősitésként a T-tipust fogjuk rendelni.

Tegyük fel pl. hogy a következő táblázatokkal rendelkezünk:

HASZNÁLAT:

folyamat	adat
P1	A1
P7	A3

ADAT:

	tulajdonos	méret
A1	T1	25
A2	T2	43
A3	T3	12
A4	T3	10

7.ábra

Ekkor

(24) használat.adat ill. használat.2

a következő relációt jelenti

	tulajdonos	méret
A1	T1	25
A3	T3	12

8.ábra

mely az "adat" reláció részrelációja, és tipusa: "adat". A következő példában ugyanezt látszólagos rekuzióval együtt fogjuk tekinteni. Tegyük fel, hogy a következő relációval rendelkezünk:

FÉRFI:

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

9.ábra

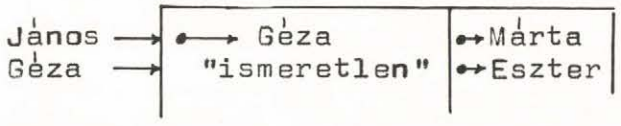
Ekkor a

(25) férfi.apja

kifejezés hatására az első oszlop tartalmának kibontása után (az alábbi sorrendben) a következőket nyerjük:

10. ábra

ami az azonos nevű sorok elhagyása után valójában a következőket jelenti:



11. ábra

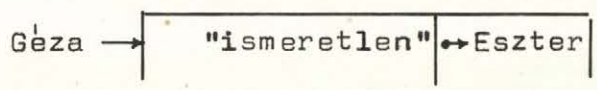
Ez tehát azoknak a férfiaknak halmaza, akik az eredeti táblázatban egyben apák is. Minthogy az igy nyert reláció, azaz a

férfi.apja

relációkifejezés szabályaink szerint maga is "férfi" tipusú, érvényes a

(26) férfi.apja.apja

relációkifejezés is, és könnyen láthatóan esetünkben a következő:



12. ábra

2.2.2 Kiválasztás (projekció+permutáció)

Majdnem a megszokott dologról van szó. Egy relációból bizonyos oszlopokat elhagyunk és a megmaradt oszlopokat előírt sorrendbe rendezzük. (Megjegyezzük, hogy a Codd-féle modell esetén itt még a redundáns sorokat is el szokás hagyni. Mi ezt nem tesszük, mert az azonos sorok a mi esetünkben nem redundánsak, ld. még 1.3.2 és 3.3.2 -t. Jegyezzük meg ugyanakkor azt is, hogy a kiválasztással ellentétben a "nagyitás"-nál a redundáns sorok elhagyása szükségszerű volt.)

Tekintsük most pl. az alábbi fogalomhoz rendelt relációt:

(27) concept update(by:process,updated:data,using:data);

Ha most szükségünk van egy táblázatra, melyben elől az "update" -elt adatok állnak, utána pedig mondjuk csupán az "update" -elő folyamat áll, akkor e kiválasztást a következő módon jelölhetjük:

(28) (updated,by) update

vagy

(29) (2,1) update

Mindkét forma alkalmazását megengedjük, (28)-at azonban csak alaprelációk esetén. A kiválasztással nyert relációhoz nem rendelünk tipust.

Megjegyezhetjük, hogy ha pl. egyetlen oszlopra vetitünk, akkor ezt 2.2.1 alapján "kinagyithatjuk" és igy már tipussal is rendelkező relációhoz jutunk. (Noha erre nincsen különösebben szükség, mivel "(1)reláció.1" nyilván azonos magával "reláció.1" -el.)

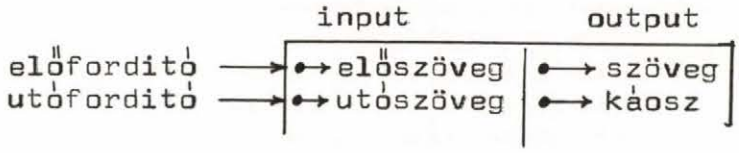
2.2.3 Kicsinyités

E művelet a nagyitás forditottja és elsősorban csupán technikai szükséget elégit ki. Legyen T egy tetszőleges reláció. Ekkor

(30) [T]

a T reláció "kicsinyitése" olyan egyoszlopos relációt jelent, melynek egyetlen oszlopa T tipusú, és elemei T soraira mutatnak. Például:

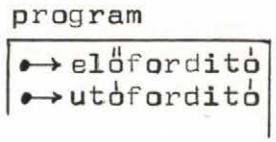
PROGRAM:



13. ábra

esetén [program] hatására a következő relációt nyerjük:

[PROGRAM]:



14. ábra

Végezetül megjegyezzük még, hogy [T].1 nyilvánvalóan minden T -re magával T -vel azonos.

2.2.4 Illesztés (Join)

Legyenek S és T legalább egyoszlopos relációk, és tételezzük fel, hogy S utolsó oszlopa azonos tipusú T első oszlopával. Ekkor

(31) S * T

a két reláció "join"-ját fogja jelenteni az adatbázisoknál megszokott értelemben. Legyen például:

S:

V	U
C	A
D	A
C	B

T:

	U	Z	T
H	A	E	H
I	B	F	J
J	A	F	J

15. ábra

Ekkor S*T eredménye, mint ismeretes:

V	U	Z	T
→ C	→ A	→ E	→ H
→ C	→ A	→ F	→ J
→ D	→ A	→ E	→ H
→ D	→ A	→ F	→ J
→ C	→ B	→ F	→ J

16.ábra

Az illesztéssel nyert relációhoz általánosságban nem rendelünk tipust, kivéve ha valamelyik komponense egyoszlopos, mikoris a tipus a másik komponens tipusa lesz (és az eredmény nyilvánvalóan annak halmazelméletileg része is lesz). Az olvasóra bizzuk annak megfontolását, mi következik abból, ha mindkét komponens egyoszlopos.

Tételezzük fel most, hogy az alábbi fogalmakkal rendelkezünk:

(32) concept része(ami:elem,aminek:csoport);
concept használ(ami:eljárás,amit:elem);

Legyen most C egy csoport tipusú reláció. Ekkor a

(33) része*[C]

kifejezés "része" tipusú reláció, és azokat a sorokat tartalmazza melyekben a C-beli csoportelemek szerepelnek,

(34) (része*[C]).1

pedig már tisztán a C-beli csoportok elemeinek halmaza, mely már "elem" tipusú reláció. Végül, könnyen láthatóan

(35) (használ*[(része*[C]).1]).1

a C részeit használó eljárások halmaza. (Itt jegyezzük meg, hogy a felhasználói nyelvben e müveletek más, kényelmesebb és szemléletesebb formában szerepelnek.)

Tekintsük még a 2.2.1 -beli példát is. Mint láttuk

(36) férfi.apja

azoknak a férfiaknak apjaiból álló (férfi tipusú) reláció, akiknek apja szerepelt az eredeti táblázatban (azaz János és Géza). Most

(37) ([férfi]*férfi.apja).1

viszont maguknak azon férfiaknak soraiból álló reláció, akiknek apja is szerepelt (vagyis akiknak apja János vagy Géza volt) azaz: István, János, Péter, Ferenc.

2.2.5 Halmazelméleti műveletek

közül az uniót, metszetet és kivonást engedjük meg, az alábbi szabályok szerint:

a) A két operandus oszlopainak tipusai rendre egymással meg kell egyezzenek;

b) Ha még az operandusok tipusa is azonos, ez egyben az eredményezett reláció tipusa is lesz, ellenkező esetben pedig nem rendelünk hozzá tipust.

2.3 Leirások elemzése

A leirt információk elemzése a lekérdező nyelv segitségével történik. Ez két részre osztható, az általános és a standardizált kérdésfeltevésekre. Előbbiek eszközei a reláció műveletek, utóbbiaké pedig speciális parancsok.

2.3.1 Általános kérdésfeltevés

Kérhetjük tetszőleges reláció tartalmának táblázatos kiirását az az alábbi formában:

(38) list "reláció-kifejezés";

ahol a "reláció-kefejezés" vagy egy alapreláció neve, vagy a megengedett műveletekkel képzett származtatott reláció.

A kifejezések képzésekor itt még egy további lehetőséget megengedünk, nevezetesen a megszoritást (behelyettesitést) egy konkrét objektummal (mely természetesen tipusában illeszkedik). Ilyen módon szintén relációhoz jutunk, de mert e művelet operandusai már nem tisztán relációk (az "objektum" ugyanis az adat-szintre tartozik), ezért ezt általánosságban (a 2.2 részben) nem tárgyalhattuk. (Pontosan fogalmazva itt tulajdonképpen arról a műveletről van szó, mely egy objektumból az egyedül őbelőle álló relációt képezi.)

Az eredmény kiirásakor megkapjuk a reláció nevét (ill. a szóbanforgó reláció kifejezést), az egyes oszlopok tipusait, valamint a reláció teljes tartalmát az alábbi módon:

a) érték-tipusnál az értéket;

b) egyedi névvel rendelkező hivatkozási-tipusnál a nevet;

c) névtelen objektumnál egy belső azonositót, (mely a hivatkozás követését a nyert dokumentumokban lehetővé teszi);

végül minden sorban kiirásra kerül a sor (mint objektum) belső vagy valódi neve.

2.3.2 Példa

Tegyük fel, hogy az alábbi leirással van dolgunk:

(39)
```
defunit
   concept tevékenység;
   concept vezérlés(vezérlő:tevékenység,
                    vezérelt:tevékenység);
   concept cél(tárgya:vezérlés);
endunit;

dataunit
   tevékenység vizsgálat;
   tevékenység utóhengerlés;
   tevékenység vágás;
   tevékenység szűrés;

   vezerlés felülvágás(szűrés,vágás);
   vezérlés X1(vizsgálat,utóhengerlés);

   cél csökkentés(felülvágás);
   cél simitás(X1);
endunit
```

Itt például az X1-el reprezentált hivatkozási viszony a felhasználói nyelvben létrehozható anélkül is, hogy explicit nevet használnánk. Tekintsük ezért most X1-et belső névnek (névtelen objektum hivatkozását ábrázoló jelölésnek). Ekkor

```
list cél;
```

hatására az alábbi táblázatot nyerjük:

CÉL:

név	tárgya
csökkentés simitás	felülvágás X1

17.ábra

majd

list vezérlés;

hatására:

VEZÉRLÉS:

név	vezérlő	vezérelt
felülvágás X1	szűrés vizsgálat	vágás utóhengerlés

18.ábra

2.3.3 Standardizált kérdésfeltevés

Bizonyos speciális relaciótipusok esetén (bináris relációk azonos tartományokkal, fák, hálók, stb.) speciális dokumentumok generálására van lehetőség (mátrix formák, grafikus formák, stb.). Továbbmenve, szükség van a teljes adatbázis tartalom különféle mélységű kiirására, dokumentálására, és a főbb jellemzőket tartalmazó report-ok előállitására.

Ezeknek a kérdésfeltevéseknek pontos listáját később fogjuk kidolgozni.

3. Meta relációk

Az eddigi 1. és 2. pontokban a logikai séma alapjait ismertük meg, mely, mint láttuk, elsősorban a referencia-szemléletben tért el a szokványos relációs modelltől. Bonyolult absztrakt rendszerek valóban adekvát leírásához azonban ennél többre, finomabb eszközökre is szükség van. Ilyen eszközökről szólunk ebben a részben. Ezek az új eszközök azok, amelyek az SDLA arculatát már döntően határozzák meg.

3.1 Kényszerités

3.1.1 Példa

Tételezzük fel, hogy pl. a következő fogalmakkal kívánunk dolgozni:

(40) concept felhasználás(tárgya:adat,használó:folyamat);
concept generálás(eredmény:adat,generátor:folyamat);
concept használat generálás céljából(használt:adat, eredmény:adat,végrehajtó:folyamat);

Ha most ezekután valamilyen D1, D2 adatok és P folyamat között fennáll mondjuk a

(41) használat generálás céljából (D1,D2,P);

reláció, akkor relációink szemantikai tartalma alapján joggal várhatjuk el, hogy a

(42) felhasználás (D1,P);
generálás (D2,P);

relációk is fennálljanak. Ez úgy biztositható, ha minden (41) formáju objektum megadásakor létrehozzuk a megfelelő (42)-beli objektumokat is. A kényszerités ezt valósitja meg.

3.1.2 Egyszerű kényszerités

A (40) példára hivatkozva az alábbi formáju kijelentések deklarálását fogjuk megengedni:

(43) constraint:
használat generálás céljából(1,2,3)⇒felhasználás(1,3);
constraint:
használat generálás céljából(1,2,3)⇒generálás(2,3);

ahol a zárójelben azonositóként szereplő számok az attributumok egymásnak való megfeleltetését határozzák meg.

Az igy deklarált kényszeritéssel egyszer s mindenkorra biztositjuk, hogy minden egyes baloldali tipusú objektum megadásakor a rendszer automatikusan előállit egy a jobboldal által meghatározott objektumot is, és azt az adatbázisban (név nélkül) elhelyezi.

3.1.3 Általános eset

A fenti tipusú kényszeritések szüksége majd minden alkalmazási területen felmerül. Lehetséges azonban általánosabb megközelités is, mely tetszőleges

(44) constraint: "reláció kifejezés" ⇒ "reláció";

formát megenged. Például

(45)
```
concept consume(used:data,user:process);
concept produce(by:process,result:data);
concept source(used:data,produced:data);
```

esetén előirható lehet a

(46) constraint:(consume * produce)(1,2,3)⇒ source(1,3);

kényszerités.

A kényszeritések ilyen általános formájának megengedése azonban (elsősorban számitásigényessége miatt) vitatható. Bevezetéséről ezért (a gyakorlati szükségletek alapján) később döntünk.

A kényszeritésnek alapcélja mellett egy másodlagos haszna is van. Bizonyos projekciókat (egy megfelelő fogalom és a rá vonatkozó kényszerités bevezetésével) kiemelhetünk, (hatékony elérésüket biztositva ezáltal).

3.2 Finomitás

A "finomitas" vagy "altipus" bevezetésének gondolata a SIMULA 67 -ben merült fel először és a jól rendszerezett áttekinthető rendszerleirások készitésének azóta nélkülözhetetlen eszközévé vált. A

"tipus" - "altipus"

viszony antiszimmetrikus (meta-) reláció. Jelentése:

(47) a) Az altipusba tartozó objektum szükségképpen beletartozik a (fő-) tipusba is.

b) Az altipusba tartozó objektum rendelkezik a (fő-) tipus minden attributumával, és esetleg további (már csak az altipusra jellemző attributumokkal) is.

3.2.1 Az altipus dilemma

Divatossá vált vitatkozni arról, hogy a tipus-altipus metarelációra vonatkozóan az antiszimmetrián (és a tranzitiv lezárás antiszimmetriáján, vagyis a körmentességen) kivül, még milyen további megszoritást ésszerű tenni. Minthogy e kérdésben irásunk szerzői között sincsen összhang, egyelőre megelégszünk a lehetőségek ismertetésével.

a) Hierarchikus (fa-) struktúra. Ez egyszerű és világos rendszerszerkezethez vezet, kezelése biztonságos és egyértelmű. Ugyanakkor, léteznek olyan jelenségek, melyek leirása a b) esetben könnyebb.

b) Hálószerkezet (szokásos algebrai értelemben). Leiró ereje közismerten rendkivül nagy. Hátránya az eredményezett struktura bonyolultsága és ennek kezelési, biztonsági, egyértelmüségi következményei.

Mint köztudott, a SIMULA szemlélete hierarchikus. Itt jelentős felhasználói tapasztalatokról is lehet beszélni. Másfelől a szokványos adatbázis szemlélet (érthető módon) hálóstruktúra. Noha utobbi nem a tipusok viszonyára vonatkozóan.

3.2.2 Hierarchikus eset

-ben az alábbi jelölést alkalmazhatjuk:

(48) concept altipus is tipus(attributumok);

Ez a következőket fogja jelenteni:

(49) a) Az igy definiált "altipus" olyan fogalom mely a "tipus" attributumaival (első helyen), és a fent megadott további attributumokkal (feltéve persze hogy számuk pozitiv) is rendelkezik.

b) Az altipusba tartozó objektumok, melyek természetesen elemei az altipushoz tartozó relációnak, egyuttal elemei lesznek a fölérendelt "tipus"-hoz tartozó relációnak is (megfelelő vetületükben persze).

c) Az altipusú objektumokra az "általánositott tipusilleszkedési szabály" válik érvényessé, ld.3.2.5.

Megjegyezzük, hogy az altipus nem speciális kényszerités. Itt ugyanis nincsen szó új adattétel generálásáról, csupán egy meglévő adat minőségének általánosabb értelmezéséről.

3.2.3 Példák

Vezessük be pl. az alábbi fogalmakat:

(50) defunit
    concept file(hely:tároló eszköz,blokkméret:integer);
    concept printfile is file(lapméret:integer);
endunit

Ekkor a "printfile" fogalom három attributummal rendelkezik, melyeket az alábbi sorrendben kell megadni: "hely", "blokkméret", "lapméret". Pl.:

(51) printfile system output(printer-1,1280,136);

Tekintsük most pl. a következő fogalmakat:

(52) defunit
    concept ember;
    concept férfi is ember;
    concept nő is ember;

E fogalmak lehetőséget adnak egyfelől olyan további fogalmak bevezetésére, melyekben az általános "ember" fogalom szerepel, pl.

(53) concept adóbevallás(bevalló:ember,hazugság:integer);

ahol a "nem" (tegyük fel) nem szignifikáns. Másfelől, ott ahol az lényeges, használhatjuk az altipusokat, pl.

(54) concept férjhezmenetel(menő:nő);

Mindennek fő jelentősége abban áll, hogy árnyalt tipusellenőrzést tesz lehetővé, (ld. később). Mindazonáltal igen fontos tudni a következőket:

> Az altipusok nem halmazelméleti osztályozások. Az "általános" maga is (önmagában is) létező fogalom. És tartozhatnak objektumok egy "általánosabb" kategoriába anélkül hogy annak bármely speciálisabb alfajába beletartoznának. Nem azért mert esetleg "nem tudjuk még" melyikbe tartoznak. (Nem is okvetlenül azért mert esetleg nincsenek is

altipusai). Hanem azért, mert mindig létezik a gondolkodásnak olyan ("általános") szintje is, melyen a speciálisról nincs értelme beszélni.

3.2.4 Ideális tipusok

Hierarchikus tipusszerkezet esetén a fogalmak fákba vannak rendezve. E fákat célszerű egy ideális tipus bevezetése útján egyetlen fává egyesiteni. Legyen ez a tipus

(55) concept universal;

melynek nincsen attributuma, (és melyet mint "standard tipust" minden definiciós egységhez hallgatólagosan hozzárendelünk). Ezzel egyidőben minden olyan fogalmat, mely eddig nem volt altipusa semminek, pl.

(56) concept fogalom(attributumok);

hallgatólagosan

(57) concept fogalom is universal (attributumok);

alakúnak fogunk tekinteni. Ezzel lehetőséget adtunk pl. ilyen specifikációk alkalmazására is:

(58) concept megnevezés(objektum:universal,név:text);

melyben 3.2.5 értelmében az "objektum" attributum értéke bármilyen (hivatkozási) tipus lehet.

Hálószerkezetű tipussema esetén ezen kivül még hasonlóan kézenfekvő lehet az ellentétes ideális tipus, mondjuk a

(59) concept "absolute special";

(avagy "empty", "void") bevezetése is, melynek egyetlen altipusa sem lehet, maga viszonyt minden tipusnak altipusa. Az "universal" -al ellentétben az "empty" attributum-specifikációként való alkalmazasa persze haszontalan, viszont egy "empty" objektum olyan jocker, melyet 3.2.5 értelmében aktuális paraméterként bárhol elfogadunk. Minthogy az "absolute special" mindennek altipusa, ezért, legalábbis elvileg, a rendszerben előforduló összes attributummal rendelkeznie kellene. Ezeket azonban meghatározatlannak tekintjük, és igy az egyszerüség kedvéért az attributumoktól el is tekintünk.

3.2.5 Az általános tipusilleszkedés

törvénye a következő:

(60) Ha egy hivatkozási attributum (definiciós szinten megadott) tipusa T, akkor az adatleirási szinten e helyen elfogadunk T-tipusu elemet, T-nek bármely altipusát is (tranzitiv értelemben) és csak ezeket.

Ez összhangban van azzal a ténnyel, hogy T-nek minden altipusa egyben T-tipusu is, (forditva azonban nincs kompatibilitás, azaz T-tipusu elemet T semmilyen altipusa helyett nem fogadunk el). A határesetekre a következőket mondhatjuk:

(61) a) "Univerzálisnak" specifikált attributum helyén mindent elfogadunk. (Univerzális minősitésű adatobjektumot viszont nincs sok teteje előállitani).

b) ("Abszolút speciálisnak" minősitett attributumot nincs értelme specifikálni). Egy abszolut speciális objektum viszont minden tipushoz illeszkedik.

3.2.6 Példa

(62)
```
defunit
    concept IO device;
    concept input device is IO device;
    concept output device is IO device;
    concept printer is output device;
    concept plotter is output device;
    concept opening(periféria:IO device);
    concept clear buffer(periféria:output device);
    concept set origo(periféria:plotter);
endunit
```

Legyen most

```
IO device negyedik csatorna;
output device rendszer output;
printer nagybetüs nyomtató;
plotter calcomp;
```

Ekkor elfogadhatók az

```
opening (negyedik csatorna);
opening (nagybetüs nyomtató);
opening (calcomp);
```

állitások, hiszen az "opening" fogalom bármely "IO device"-ra vonatkozhat. Elfogadhatók a

clear buffer (rendszer output);
clear buffer (nagybetüs nyomtató);

állitások is, mert a "clear buffer" fogalom "output device"-ra vonatkozik, mely mindkét esetben teljesül. Ezzel szemben inkorrekt a

set origo (nagybetüs nyomtató);

kijelentés, mert a "set origo" fogalom a "printer"-re nincsen értelmezve, sőt elfogadhatatlan a

set origo (rendszer output);

is, mert a rendszer output elvben ugyan lehetne akár plotter is, de definiciója alapján nem biztos hogy az. A "set origo" művelet ilyenformán nem feltétlenül értelmes rá nézve, ami elegendő indok ahhoz, hogy 3.2.5 alapján a kijelentést elutasitsuk.

3.2.7 Null objektum

Az elöbbiek szerint az "universal" tipust standard tipusnak tekintjük. Hasonlóan, az objektumok körében is bevezethetünk egy (vagy akár több) "absolut special"-tipusú standard objektumot. Legyen például

(63) absolute special nil;

Szó volt már arról, hogy az adatmegadáskor az aktuális attributumok értéke lehet üres is: "nem ismert". Most ezt úgy fogalmazzuk, hogy ekkor a nil értéket veszi fel. Mint láttuk, ez az általános tipusilleszkedés törvényével teljesen összhangban van.

3.3 Integritás

Az adatbázis integritása alatt azon (előre megadott) tulajdonságok összességét értjük, melyek az adatbázis tartalmára vonatkozoan invariánsak. Ezen invarianciák biztositása esetünkben elsősorban az input adatok szürése útján történik. Ez azt jelenti, hogy az integritásnak ellentmondó adategységeket (az adatbázis pillanatnyi tartalmától függően persze) elutasitjuk.

Relációs adatbázisok integritását, mint ismeretes, függőségek (FD és MVD) formájában szokás megadni. Nekünk azonban ennél finomabb eszközökre is szükségünk van. Gyakorlati szempontok fi-

gyelembevételével ezidőszerint az alábbiak jöttek szóba.

3.3.1 Egyszerű függvénykapcsolat

A reláció legfontosabb speciális esete az, amikor valamelyik komponens egyértelműen meghatározza a többit. A kommerciális adatbázisokkal ellentétben a mi esetünkben ennek jelentősége nem az így biztositott kulcs szerinti elérés lehetőségében, hanem a függőségnek mint integritásnak biztositásában, ellenörzésében áll. Legyen például

(64) concept fogalom(név:text,prefix:fogalom,többi:attr rész);

Ekkor a definicióhoz csatolt

(65) function of név;

kiegészités deklarálja azt, hogy két különböző fogalomnak nem lehet ugyanaz a neve, ez ugyanis sérti a kimondott integritást (és ezért a rendszer a másodikat el fogja utasitani).

3.3.2 Többváltozós függvénykapcsolat

A függvénykapcsolatot deklaráló kiegészitő specifikáció általános formája a következő:

(66) function of "szelektorok felsorolása";

ahol a felsorolt szelektorok csupán együttesen határozzák meg egyértelműen a reláció sorait. Amennyiben a felsorolással deklarált "kulcs" történetesen az összes szelektor, ezesetben egyszerüen a

(67) function;

formát alkalmazhatjuk.

1.3.2-ben rámutattunk arra, hogy egy reláció (az objektumnévtől eltekintve) mindig tartalmazhat azonos sorokat. Ezt most a function deklarációval megtilthatjuk.

3.3.3 Általános függvényforma

Általános esetben nem csupán alaprelációk, hanem származtatott relációk is lehetnek valamely kulcs szerint függvények. Az ilyen (általános értelemben vett) függvénykapcsolat deklarációjára az alábbi formát alkalmazzuk:

(68) integrity relációkifejezés function of
oszlopkijelölések;

Végezetül, a függvénykapcsolatra vonatkozóan a következőt kell megjegyeznünk: A tárgyalt függvényforma (a szokásos FD-vel ellentétben nem attributumok közötti viszonyra, hanem egy attributum együttes "kulcs"-ként való deklarálására vonatkozik. Ez erejében ekvivalens az FD formával, ha a fogalmak alkalmas megválasztási lehetőségét is figyelembe vesszük.

3.3.2 Bináris tulajdonságok

Bináris relációk egy sereg olyan speciális tulajdonsággal rendelkezhetnek, melyekkel absztrakt rendszerek karakterisztikus integritásai fejezhetők ki. Például a

(69) concept származás(szülő:ember,gyerek:ember);

kapcsolat nyilvánvalóan antiszimmetrikus kell legyen, ellenkező esetben valami galiba történt. Ezért indokolt az

(70) integrity: származás antisymmetric;

kikötés. Általanos esetben az ilyen fajta integritások deklarálása az alábbi formáju lehet:

(71) integrity:"bináris reláció kifejezés" "tulajdonság";

ahol a "bináris reláció kifejezés" olyan, a megengedett műveletekkel képzett, kifejezés mely bináris relaciót eredményez, "tulajdonság" cimen pedig az alábbi kulcsszavakat engedjük meg:

(72) antisymmetric
irreflexive
hierarchic
precedence
lattice

A hierarchic szóval fa-szerkezetet, precedence használatával olyan relációt melynek tranzitiv lezárása parciális rendezés, lattice esetén pedig algebrai hálószerkezetet deklarálhatunk. Világos pl. hogy helyénvalo az

(73) integrity: származás precedence;

kikötés, de

(74) integrity: származás hierarchic;

már csak pl. az apai ágra szoritkozva lenne korrekt. Befejezésül egy példát mutatunk bonyolultabb kifejezésre. (45)-re hivatkozva kézenfekvő az

(75) integrity:(1,3)(consume * produce) antisymmetric;

integritás, mert minek fogyassza valaki azt, amit maga termelt. Ha pedig még az egész rendszerben való körmentésseget is ki akarnánk fejezni, akkor antisymmetric helyett precedence kulcsszót is használhatunk.

3.3.5 Halmazelméleti viszonyok

Integritások kifejezéséhez szükséges lehet a relációknak mint halmazoknak egymáshoz való viszonyitása. Itt az egyenlőség és tartalmazás jön szóba. Például (40) esetén a (43) kényszerites az alábbi integritással ekvivalens:

(76) integrity:
(1,3)felhasználás generálás céljából ⊂ felhasználás;

A különbség az, hogy (43) alkalmazása esetén e szabályt kikényszeritjük, mig (76) utján csupán ellenőrzésére rendezkedünk be.

4. Dialogus

Mindeddig az ábrázolt információnak csupán szerkezetével foglalkoztunk. Most kialakitásának folyamatát tekintjük.

4.1 Fokozatos kiépites elve

Az adatbázis tartalmának kialakitása, valamint az ábrázolt információk elemzése dialogus során történik. Ez két részre, a metadialogusra és a tárgydialogusra oszlik. A két dialogus nem keverhető, utóbbi akkor kezdődik, mikor előbbi befejeződik.

4.1.1 A fogalomrendszer kiépitese

a metadialógus során történik. A fogalmak leirását végezhetjük akár egy, akár több definiciós egység egymás utáni megadásával.

A felbontás szüségességét egy példán mutatjuk be. Tegyük fel, hogy kidolgoztunk bizonyos igen általános fogalmakat, melyek mondjuk az információs rendszerek tervezésekor tipikusak. Ezeket a rendszernek átadjuk:

(77) defunit
]
endunit

inf.rsz. tervezés általános fogalmai

A fogalmak értelmezése után a rendszert akár rögtön használhatjuk is, vagy ezen a ponton felfüggesztett állapotában egy példányát félre is tehetjük. Ezután mondjuk kiderül, hogy jó néhány alkalmazást mondjuk kohászati környezetben fogunk végezni. Itt további fogalmak merülnek fel, melyek általánosságban ugyan nem, de e környezetben szintén tipikusak. Ekkor elővesszük a felfüggesztett változatot és egy további egységet illesztünk hozzá:

(78) defunit
endunit
] kohászati inf.rsz.-ek sajátos eszközei

A sort folytathatjuk. Kis Géza is hozzáteszi az egyéniségére jellemző különbejáratú fogalmakat. A definiciós egységek megadása végül befejeződik.

Az egymásra épülő definiciós egységek megadásakor a következő törvény érvényes:

(79) Az éppen megadásra kerülő egység tartalma a korábban megadott fogalmakkal együtt zárt kell legyen. (Másszóval bárhol hivatkozhatunk korábban megadott, vagy még ezen az egységen belül megadásra kerülő fogalomra, olyanra azonban természetesen nem, melyet egy későbbi egységben adnánk meg esetleg.)

4.1.2 Az adatbázis kiépitése

ugyanezen elv alapján történik. Ugyanúgy, egységekben adjuk meg az adatokat. Egy egység feldolgozása után a rendszert természetesen mindig passziválhatjuk, és a munkát bármikor folytathatjuk. Az egymásra épülő egységek megadásakor a (79) törvény megfelelője érvényes:

(80) Az éppen megadásra kerülő egység tartalma a korábban megadott adatokkal együtt zárt kell legyen. (Azaz hivatkozhatunk korábban megadott, vagy még ezen az egységen belül később megadásra kerülő adatra is, de későbbi egységére természetesen nem.)

4.2 A dialogus folyamata

4.2.1 Metadialogus

Mint az a korábbiakból látható, a metadialogus során

a) fogalmak,
b) kényszeritési szabályok,
c) integritási szabályok

kerülnek megadásra. Ezek sorrendje tetszőleges, az egyetlen korlátot (79) következetes alkalmazása jelenti. Például:

(81)
```
defunit
    concept data;
    concept group is data;
    concept element is data;
    concept consist(contained:data,in:group);
    integrity consist precedence;
    concept relation;
    concept associated data is data
                              (associated to:relation);
    integrity associated data function of (associated to);
endunit;
```

Ezekután a következő lépésben például:

```
defunit
    concept informative data is data;
    concept control data is data;
    concept process control(controlled:process,
                            by:control data);
    stb.
endunit
```

4.2.2 Tárgydialogus

A tárgydialogus során

a) adategységeket vihetünk be, melyekben

   i) új adatokat irhatunk le,
   ii) meglévőket módosithatunk;

b) kérdéseket tehetünk fel, mégpedig

   i) általános, és
   ii) standardizalt formában.

Az adategységek és a kérdések nem keveredhetnek, de tetszőleges sorrendben követhetik egymást, pl:

```
dataunit
[
endunit;
list ...
dataunit
stb.
```

4.3 Adatok módosítása

Az adatokkal kapcsolatosan az előző pontban felsorolt tevékenységek közül csak a módosításról nem szóltunk még. Ehhez először az adattételek egyedi elérésének általanosabb módjat kell megismerni.

4.3.1 Objektum kifejezések

Ebben a részben kifejezéseket és hozzájuk tartozó minősítéseket vezetünk be. Elemi kifejezéseknek nevezzük a következőket:

a) a közöséges értékeket (számok, szövegek a szokványos minősítésekkel integer, real, text),

b) az adatobjektumok neveit (a megadásuknál használt típusnévvel mint minősítéssel).

Tekintsük pl. a következőket:

(82)
```
defunit
   concept data(owner:process);
   concept process(owner:system);
   concept system(size:integer);
endunit

dataunit
   system S(1000);
   process P(S);
   process R(S);
   data D(P);
   data E(R);
   stb.
```

Ezekután pl. "D" elemi kifejezés "data" minősítéssel, vagy "S" elemi kifejezés "system" minősítéssel.

(83) Általanosságban: Ha K egy kifejezés, akkor

K.szelektor

is kifejezés, feltéve hogy K minősítése hivatkozási típus, és e tipus rendelkezik a megadott szelektorral. Az egész kifejezés minősítésének pedig a "szelektor" minősítését fogjuk tekinteni.

(Vigyázat: nem tévesztendő össze e jelölés a relációk körében alkalmazott hasonló jelöléssel.)

Például, (82)-re hivatkozva

D.owner

minősitése "process" (és hivatkozási értéke P). Vagy pl.

D.owner.owner.size

minősitése integer (és értéke 1000).

4.3.2 Átutalás

Meglévő adatok módositása átutalás segitségével történhet. Ennek formája a következő

(84) K1 assign K2;

(85) ahol a K1, K2 kifejezések tipusa azonos kell legyen. Hatására a K1 kifejezés által meghatározott helyre átutalódik a K2 kifejezés értéke.

(Ez természetesen az Algol 68 elveivel összhangban történik, a referencia szemléletet azonban sehol sem tartottuk szükségesnek explicit jelöléssel hangsúlyozni). Tekintsük példaként megint (82)-t:

E.owner assign D.owner;

hatására E "owner"-attributumának helyére beiródik a D "owner" attributumának értéke (aktuálisan P).

4.4 Rendszer dinamika

4.4.1 Fogalomkör bővitése

Mint mondottuk, az adatleirások megadásakor a fogalomrendszert már rögzitettnek tekintjük. Nyilvánvaló, hogy a fogalomkörben olyan változtatásokat nem tehetünk, melyek a meglévő adatok értelmét és integritását sértenék (fogalom törlése, új integritás vagy kényszerités bevezetése), vannak azonban olyan változtatások (új fogalom bevezetése, meglevő fogalom finomitása) melyek a már bevitt adatok korrektségét nem befolyásolják.

Nincsen akadálya annak, hogy a fogalomkör ilyen jellegű módositását az adatleirási run-time részben is megengedjük. A vélemények azonban itt megoszlanak. Az alábbi érvek mindenesetre igazak:

a) Új fogalmak bevezetésének megengedése jelentősen növeli a felhasználó lehetőségeit: nem köteles már kezdetben minden fogalmat véglegesiteni.

b) Mindennemű fogalomdefinició igen felelősségteljes és messzemenő következményekkel járó tevékenység. Ilyen dinamikus eszköz segitségével a felhasználók belátha-tatlan dolgokat művelhetnek.

4.4.2 Objektumok törlése

Meglevő objektum törlésének formája a következő

(86) cancel objektum kifejezés;

utasitás. Ennek hatására a hivatkozott objektum

a) Az őt minősitő összes relációkból törlődik;

b) Neve megszünik létezni;

c) Az adatbázisban minden olyan hivatkozás, mely a törölt objektumra vonatkozott üressé válik.

A törlés másik formája a kulcs szerinti elérésen alapul, és csak akkor használható, ha a reláció "function" integritással rendelkezik. Ekkor a

(87) cancel fogalomnév by key attributum értékek;

forma használható, ahol a felsorolt attributum értékek rendre a function-ban megadott szelektoroknak felelnek meg.

II. FELHASZNÁLÓI NYELV

A rendszer használója a felhasználói nyelv segitségével érintkezik a rendszerrel, ill. a rendszeren keresztül saját adataival. A felhasználói nyelv szemantikáját a már ismert logikai sémára való leképezés útján fogjuk megadni. Elvileg a már ismert, és az I.5.-ben összefoglalt logikai sémanyelv is alkalmazható lehetne felhasználói célokra, ezt azonban senkinek sem kivánjuk. (Ez egyébként kb. annak felelne meg, mintha egy intelligensebb procedurális nyelven pl. SIMULA-ben közvetlenül akarnánk adatokat kezelni). A felhasználói nyelv célja azonban nem merül ki a már ismert sémára vonatkozóan egy kényelmesebb kommunikációs eszköz biztositásában, további lényeges funkciók ezen a szinten, ill. a két nyelv közti leképezes útján kerülnek megvalósitásra.

1. Az emberi nyelv adatszerkezetekké transzformálásáról

Kijelentő mondatokra szoritkozunk. Célunk persze nem a nyelvtudomány művelése (beszélt nyelv általában való elemzése), hanem forditott irányú: olyan nyelvet kivánunk definiálni, mely könnyen használható (nem mond ellent a beszélt nyelv szabályainak) ugyanakkor tartalma egyszerü adatszerkezetekkel (relációkkal) egyértelműen ábrázolható.

1.1 A mondatok relációs szemlélete

Olyan transzformációkra van tehát szükség, melyek mondatokat (kijelentéseket) adatszerkezetekké alakitanak:

mondatok ⟶ relációk.

Általánosságban véve erre tulságosan is sokféle lehetőség kinálkozik. Nem célunk itt az összes eshetőség taglalása. Csupán néhány tipikus esetre szoritkozunk.

1.1.1 Egyszerű minősités

Sok esetben az egyszerübb mondatok tartalmát elegendő csupán alanyának minősitésével ábrázolni. Például a

(88) "Gusztáv elenjáró."

mondat ábrázolásához bevezethetjük a

(89) concept elenjáró;

fogalmat, és a jelentést az

(90) élenjáró Gusztáv;

adattal ábrázolhatjuk. Vagy akár az

(91) "Endre munkaidőben italozik."

mondatnál is elegendő lehet bizonyos esetben a

(92) concept munkaidőben italozók;
munkaidőben italozók Endre;

ábrázolás.

1.1.2 Kapcsolat ábrázolása attributummal

Amikor egy mondatban két vagy több dolgot valamilyen kapcsolatba hozunk, felmerül kérdés, érdemes-e önálló fogalomnak tekinteni magát a kapcsolatot is, vagy lehet-e, jobb-e ezt egyszerűen attributumként kezelni. Közös recept nincsen. Sokszor mindkettő lehetséges. A tágabb környezet és a felhasználás dönti el melyik jobb.

Kapcsolatok attributumként való ábrázolása persze csak akkor jöhet szóba, ha a kapcsolat (valamelyik irányban) függvénykapcsolat. Nehány példát mutatunk:

(93) "Péter János apja."

esetén jó lehet a

(94) concept ember(apja:ember);
ember János(Péter);

ábrázolás (melyben tehát nem használtunk explicit "apaság" fogalmat) vagy pl. a fenti (91) mondatnál a

(95) concept munkatárs(fő tevékenység,napszak);
munkatárs Endre(italozás,munkaidő);

ábrázolás.

1.1.3 Kapcsolat mint fogalom

Ez az ábrázolási mód minden esetben alkalmazható. Nagyon fontos azonban a fogalmak megfelelő megválasztása. Általában ugyanis többféle lehetőség kínálkozik. Például:

(96) "Tamás udvarol Mártának, hogy barátnőjével is megismerkedhessen."

esetén jó lehet a

(97) concept udvarlás céllal(férfi,nő,cél);
udvarlás céllal (Tamás,Márta,barátnő megismerése);

ábrázolás, más esetben viszont pl. a

(98) concept udvarlás(férfi,nő);
concept udvarlási cél(udvarlás,cél);
udvarlás U1(Tamás,Márta);
udvarlási cél (U1,barátnő);

leírás célszerűbb. Hogy melyik, azt a környezet és a cél dönti el.

1.2 Az "ekvivalencia" probléma

Ugyanazokat az állításokat általában többféle nézőpontból is meg lehet fogalmazni. Az "ekvivalencia probléma" a különböző állításokban különböző formákban használt kapcsolatok közül az azonos szemantikai tartalommal rendelkezők egymáshoz rendelésében áll. Pl.

(99) "A fáradtság a munka következménye."
"A fáradtság oka a munka."

vagy

(100) "A uses B."
"B is used by A."

mondatpárokban ugyanarról a fogalomról ("okság" ill. "használat") van szó, csak más-más nézőpontból.

E problémát úgy fogjuk megoldani, hogy a fogalmakhoz lehetséges nézőpontokat rendelünk. (Ld. "relatív formák".)

1.3 A "vonatkoztatási" probléma

A mindennapi beszédben gyakoriak az olyan (egyébként értelmes) mondatok, melyekben nem látható világosan, vajon egy bizonyos határozó az alanyra, a tárgyra (stb., másra) vonatkozik-e. Pl.

(101) "Zoltán az utca közepén meglátta Matildot."

esetén nem lehetünk biztosak, hogy Zoltán, vagy Matild volt az utca közepén, vagy esetleg mindketten.

A SIMULA ismerői előtt felidézhetjük a többszörös "inspect" problémáját is, ahol a rendelkezés ugyan egyértelmű, de mert az emberi gondolkodásból e szigorú mechanizmus hiányzik, a többszörös inspect gyakran katasztrófális félreértésekhez vezet.

Világos tehát, hogy egyfelől egyértelmű vonatkoztatási rendszerre van szükség, másfelől ennek szembetünő megjelenitése is nélkülözhetetlen. Ezt célozza a mellé- és alárendelt szerkezetek javasolt kezelése, ld. később.

2. A felhasználói nyelv eszközei

2.1 Relativ formák

Formalis nyelvünk minden mondata valamilyen nézőpontból hangzik el. Összességében annyi nézőpont van ahány fogalom, és még egy. Másrészt: minden fogalom nézhető bármelyik attributumának nézőpontjából.

Tekintsük a következő példát:

(102) concept use to derive(used:data,user:process,derived:data)

Tegyük fel, hogy éppen mondjuk valamely konkrét "data" objektumról teszünk különféle kijelentéseket. Szeretnénk, ha többek között ilyeneket is mondhatnánk:

(103) used by P to derive D2;
derived by P using D3;

vagy pedig ha éppen P-rol beszélünk, akkor mondhassuk, hogy

(104) uses D1 to derive D2;

Ezeket úgy fogjuk lehetővé tenni, hogy megengedjük a fogalom definiciók kiegészitését u.n. relativ formák felsorolásával a következőképpen. Példa:

(105) concept use to derive(used:data,user:process,derived:data);
form used: used by user to derive derived;
form derived: derived by user using used;
stb.

Általánosságban ennek formáját igy vázolhatjuk fel:

(106) concept fogalomnév(attributumok);
form szelektor-i: többi szelektor szövegbe ágyazva;

Ezzel a "beágyazó szöveget" deklaráltuk, mint az i-edik szelektor nézőpontjából alkalmazható leirási formát, mely forma az adott környezetben a szóbanforgó "fogalomnév" által meghatározott fogalmat reprezentálja.

2.2 Alárendelés és mellérendelés

2.2.1 Az abszolút nézőpont

Mint láttuk, minden fogalmat nézhetünk (ha akarunk) bármely attributum felől. Ezek a relativ nézőpontok. Most ezt kiegészitjük még egy, az "abszolút" nézőponttal, melyből minden fogalom olyan, amilyen valójában (vagyis a beágyazó szöveg maga a fogalomnév az I. fejezetből megismert módon használva). Például (102) esetén az abszolút nézőpont beágyazo szövege:

(107) use to derive (, ,) .

Megjegyezhetjük azt is, hogy nulla attributumszám esetén (természetesen) csak abszolút nézőpontrbl lehet beszélni.

2.2.2 Abszolút mondat

Abszolut mondat alatt olyan kijelentést (adatmegadást) értünk, mely az abszolut forma felhasználásával történik. Szerkezetét az alábbi modon rögzitjük:

(108) fogalomnév aláhuzva, objektumnév esetleg, attributumok zárójelben;

Ha tehát

(109) concept process;

egy fogalom, akkor

(110) process;
process P;

abszolút mondatok. Vagy (102) esetén abszolút mondatok a következők:

(111) use to derive (D1,P,D2);
use to derive usage1(D1,P,D2);

(E mondatok logikai sémára való fordítása önmaga.)

2.2.3 A nézőpont-törvény

a) Minden mondat (abszolut vagy relativ) után érvényessé válik a mondat által reprezentált fogalom nézőpontja.

b) Minden olyan relativ mondat kimondható, melynek nézőpontja éppen érvényes.

(Az érvényes nézőpontok tehát egy stack-et a "nézőpont--stack" -et alkotják, melynek alján mindig az "abszolut" áll.)

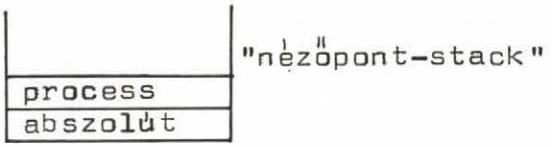
c) Egy uj mondat értelmezése mindig úgy történik, hogy megnézzük, melyik az a (legmagasabb) nézőpont melyben értelmes, és

i) a felette lévő nézőpontok érvényüket vesztik,
ii) a többi érvényes marad az új mondatéval együtt.

Példa. Legyen

(112) process P;

Kezdetben érvényes volt az "abszolút" nézőpont. Most érvényessé vált a "process" is, azaz

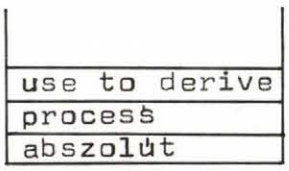


19.ábra.

Egészítsük ki (112) -t igy:

(113) process P;
uses D to derive E;

most érvényesek a

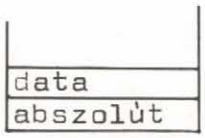


20.ábra.

nézőpontok. Mondhatunk tehát állitást e pillanatban e három nézőpont bármelyikéből. Folytathatjuk pl. ismét egy abszolút állitással, igy:

(114) process P;
uses D to derive E;
data F;

melynek hatására



21.ábra.

állapotba kerülünk, stb.

(Megjegyezzük, hogy a soron belüli pozicionálás javitja az érthetőséget, mert érzékelteti az érvényes nézőpontok stack-jének mélységét. A pozicionálást természetesen nem kivánjuk a felhasználótól, mondatait azonban automatikusan átrendezzük ilyen formára. Ez hibáinak gyors felismerését is megkönnyiti.)

2.2.4 Relativ mondat

formájának pontos megadása maradt hátra. Ezt igy rögzitjük:

(115) objektumnév esetleg kettősponttal, attributumok megadása beágyazó mondatban;

ahol az attributumokat kifejezésekkel adjuk meg 4.3.1 értelmében. Ha az attributum objektumnév, akkor megengedjük minősitésének mellékirását is (a még kifejezőbb mondatkép érdekében). Alább illusztráljuk a lehetőségeket. Tekintsük ismét a

(116) process P;
uses D to derive E;

kijelentéseket. Ezek forditása a logikai sémára a következő:

(117) process P;
use to derive (D,P,E);

Ha most e másodiknak egy külön "U" hivatkozási nevet akarunk adni, akkor a

(118) process P;
U: uses D to derive E;

formát használjuk. Végül: megengedett a "uses data D to derive data E;" forma is.

2.3 Technikai eszközök

Az ebben a részben felsorolásra kerülő további eszközök célja a felhasználó kényelmének fokozása.

2.3.1 Tipus mint szelektor

A szelektornevek kiagyalása gyakran terhes feladat, és olyankor, ha az attributumok amúgy is mind különböző tipusuak felesleges is. Szokás megállapodni abban, hogy ha nincs szeletornév, a tipusnevet egyben szelektornévnek is tekintjük. Ezt tesszük mi is. Ha tehát a felhasználói nyelvben pl. a következő definiciót használjuk:

(119) concept házasság(férfi,nő);

(ahol persze "férfi" és "nő" is definiált fogalmak), akkor a logikai sémában ez igy jelenik meg:

(120) concept házasság(férfi:férfi,nő:nő);

2.3.2 Felsorolás

Jelentősen megkönnyiti a leirások készitését az attributumok megadásakor a felsorolások megengedése. Például

```
(121)   process X;
           uses data D1;
           uses data D2;
```

helyett a

```
(122)   process X;
           uses data D1, D2;
```

formát is lehetővé tesszük. Itt azonban a következőkre kell figyelemmel lenni: (122) valójában két különböző "use" tipusú objektumot generál. Ha most ezekután valamilyen "use" nézőpontú állitást mondunk, pl.

```
(123)   process X;
           uses data D1, D2;
              to maintain relation R;
```

akkor ez utóbbi mindkettőre vonatkozik, és minden további állitás is mindaddig, amig a "use" nézőpont érvényes.

2.3.3 Makro formák

Jelen ismertetésben csupán megjegyezzük, hogy hasonló fogalomtipusok egyszerübb és strukturáltabb megadásához helyén való különféle makro formák felhasználása. Jól jöhet pl. ilyenféle:

(124) macro:
concept subpart(a-concept,a-concept);
for a-concept=process,activity,group;

Ezzel a kérdéssel azonban itt tovább nem foglalkozunk.

2.3.4 Összetett relativ forma

Olyan esetben, mikor egy fogalomra vonatkozóan érvényes egy egyszerü kényszerités, pl.:

(125) constraint:
use to derive(1,2,3) ⇒ derivation(2,3);

felmerülhet az az igény, hogy egy jobboldali objektum (itt "derivation") nézőpontjából a hiányzó attributum megadásával hozzunk létre egy baloldali (pl. "use to derive") objektumot.

Kézenfekvő itt olyan relativ forma bevezetését megengedni, mely nem az eredeti szelektorok nézőpontjából indul ki, hanem egy attributumegyütteshez rendelt fogalomból, pl.:

(126) form derivation: using 1;

Itt azonban a következőket kell megfontolni:

a) Ex- vagy implicite ismernünk kell az attributumok kölcsönös megfeleltetését;

b) Magát a kényszeritést ilyen esetben nem kell már (még-egyszer) végrehajtani.

Mindezek érdekében bevezethetjük az egyszerü kényszerités felhasználói nyelvbeli megadásának egy alternativ formáját, mely jelentésében (44) -el ekvivalens:

(127) concept use to derive(used:data,process,derived:data);
implies derivation(process,derived);

Az "imply" részt a fogalom definició fejrésze, és a formák felsorolása között kell megadni. Csakis ilyen esetben engedjük meg összetett relativ forma deklarálását. Teljes példánk tehát igy néz ki:

(128) concept derivation(process,data);

concept use to derive(used:data,process,derived:data);
implies derivation(process,derivation);
form derivation: using used;

melynek alkalmazása elveink értelmében a következő lehet:

(129) derivation (P,D);
using D2;

vagy akár

(130) data D;
derived by process P;
using data D2;

2.3.5 Abszolút beágyazás

Mint láttuk, az abszolut mondat formája némileg eltért a relativ mondatétól. Az egységes leirási forma érdekében azonban meg fogjuk engedni az abszolut mondatnak beágyazó szöveggel történő megadását is. Ennek használata a relativ formáéval megegyező. Egy példát mutatunk. Legyen

(131) concept use to derive(used:data,process,derived:data);
form absolute: process process uses used to derive
derived;

Ekkor

(132) U: process P uses D1 to derive D2;

ekvivalens a

(133) use to derive U(P,D1,D2);

abszolut mondattal (és ugyanezen logikai-sémabeli adattal).

2.3.6 Relativ forma mint művelet

Tekintsük ismét pl. a

(134) concept use to derive(data,process,derived:data);
form process: uses data to derive derived;

fogalmat. Ekkor, mint tudjuk, a

(135) uses D1 to derive D2;

relativ mondat "process"-re vonatkozóan mondható ki, és (mint minden mondat) pontosan egy (esetünkben "use to derive"-tipusú) új adattételt eredményez.

Mármost nyelvi, célszerüségi okokból meg fogjuk engedni az általános lekérdezéseknél ugyanezen mondatot reláció képző müveletként is, és azon folyamatok halmazát értjük rajta, melyek D1-et "használva" D2-t "vezetik le".

Emlékeztetünk arra, hogy eredeti müveleteinkkel ugyanezt csak kevesbé szemléletesen a

(136) use to derive(D1,,D2).2

formulával fejezhettük ki.

Általanosságban: a relativ formába mint müveletbe való behelyettesitést az objektummal való megszoritáshoz hasonlóan szabályozzuk: tipusában illeszkedő objektum vagy relációkifejezés irható bele. Tartalmát illetően a relativ formába helyettesités a behelyettesitettek kicsinyitéseivel való illesztést (join), végül a forma nézőpontjának kiválasztását jelenti. Jelentősége, hogy e meglehetősen komplex absztrakt tartalommal szemben szemléletes tartalma felhasználói szemmel is egyszerü és világos.

Tekintsük második példaként az I.2.2.3 -belit. Legyen

(137) concept use(process,data);
form process: uses data;

concept subpart(data,of:data);
form data: part of of;

Ekkor pl. egy adott D adat részeit használó folyamatok halmaza az eredeti kalkulus alkalmazásával igy állitható elő:

(138) (use*[subpart(,D).1]).1

Ezt most egyszerüen igy irhatjuk:

(139) uses (part of D)

2.4 A "nyitott leirás" alternativája

Mind ez ideig feltételeztük, hogy mind a definiciók halmaza, mind pedig az adatok halmaza (az egyes egységek fogadása után) mindig zárt. A definiciós részben ennek nincsen alternativája, a leiró részben azonban lehetőség van arra, hogy egy a felhasználó által nyitva hagyott egységet (feltéve, hogy önmagának nem mondott ellent) utólag zárjunk le. A következő algoritmust lehet (például) alkalmazni:

A leirást szekvenciálisan értelmezzük (az eddigiekkel ellentétben) úgy, hogy valahányszor nemdefiniált objektumnévvel találkozunk, azt rögtön generáljuk a hely által meghatározott legáltalánosabb tipus szerint (és üres attributumokkal). Ha pedig később mégis definiálásra kerül, akkor attributumai értéket kapnak (és esetleg tipusa is specializálódik).

Az efféle automatikus mechanizmus azonban óvatosságra int:

a) Elegendő egy azonositónévben egyetlen betü elirás, és hibajelzés helyett egy keresetlen rejtett uj objektummal állunk szemben.

b) Nincsen egyértelmű deklarációs lista a létező objektumokról.

Mindazonáltal a felhasználó munkáját az ilyen lehetőség mégis jelentősen megkönnyiti. Megengedése esetén természetesen az a minimum, hogy az automatikusan létrehozott objektumokról egy figyelmeztető lista készüljön.

2.5 Néhány gyakorlati probléma

2.5.1 Kommentár

Kommentárok kezelésére többféle lehetőség is kinálkozik. Bevezethetünk pl. egy

(140) concept comment(universal,text);
form universal: comment text;

fogalmat, melynek segitségével kommentárok asszociálhatók tetszőleges objektumhoz. Az ilyen kommentár természetesen az adatbázisban is tárolásra kerül. Lehet beszélni csak az input szövegben megjelenő kommentárokról is. Ezzel itt nem foglalkozunk.

2.5.2 Szinonima

Gondolhatnánk arra, hogy objektumnevek helyett használandó rövidebb ekvivalens szinonimák kezelését a kommentár mintájára lehet megoldani. Ez nincs igy, mert az objektumok nevei nem text attributumok, kezelesüket saját belső mechanizmus (hash vagy akármi) biztositja. Ezzel szemben a gyakorlat számára megfelelő út az, ha maguk az objektumnevek (noha lehetőleg kifejezőek, de) rövidek, ugyanakkor ha szükséges az objektumokhoz hosszabb magyarázó nevet is rendelhetünk (ezt azonban már valóban text, és nem belső név formájában). Megtehetjük pl. azt is, hogy egy közös általános tipust egy text attributummal látunk el:

(141) concept általános(teljes név:text);

majd a rendszerben minden további fogalmat ennek altipusaként definiálunk.

2.5.3 Tipikus fogalmak

Előfordul, hogy bizonyos viszonyokat többféle fogalomra vonatkozóan is célszerű lenne bevezetni. Például

(142) concept subpart(data,of:data);
concept subpart(process,of:process);

Ez azonban ebben a formában egyrészt ellentmondana a fogalomnevek unicitásának, másrészt itt valójában is két különböző tartalmazási relációról van szó.

Mindazonáltal, mégis indokolt lehet az azonos név használata, hiszen ezt a mindennapi beszédben is igy tesszük. Van olyan eset, mikor ezt közös ős-fogalomra visszavezetéssel oldhatjuk meg. Fent azonban nyilvánvalóan nem ez a helyzet (nem akarjuk, hogy adatnak folyamat lehessen része és viszont).

A megoldást az azonos relativ formák nyújtják, mely természetesen megengedett, és pl. a következő módon valósitható meg:

(143)
```
concept data subpart(data,of:data);
form data: part of of;

concept process subpart(process,of:process);
form process: part of of;
```

2.5 Példa az eszközök alkalmazására

Induljunk ki most abból, hogy pl. az alábbi leírást akarjuk tárolni:

(144)
```
process X;
   belongs to system component S;
   uses data D1, D2;
      to derive entity E;
         under condition C;
   uses data D3;
      to update information I;
   uses data D4,D5;
   produces data D6;

data D7;
   is used by process Y;
   produced by process Z;
   stb.
```

mégpedig úgy, hogy

a) a szövegösszefüggésből láthatóan ekvivalens formák valóban ugyanazt a fogalmat takarják,

b) a mellé- és alárendelések a szöveg tagolásával összhangban legyenek.

Ennek elérésére a következő definiciók alkalmasak:

(145)
```
concept system component;
concept process;
concept data;
concept entity;
concept condition;
concept information;

concept belong(process,system component);
form process: belongs to system component;

concept use(process,data);
form process: uses data;
form data: is used by process;

concept produce(process,data);
form process: produces data;
form data: produced by process;

concept purpose(use);

concept derivation is purpose(entity);
form use: to derive entity;
```

```
concept update is purpose(information);
form use: to update information;

concept derivation condition(derivation,condition);
form derivation: under condition;
```

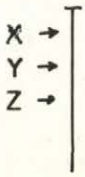
Mármost ezeket a fogalmakat figyelembe véve a (144) szöveg fogalmi sémára való fordítása a következő formát ölti:

(146)
```
process X;
system component S;
data D1;
data D2;
data D3;
data D4;
data D5;
data D6;
entity E;
condition C;
information I;

belong (X,S);

use A1(X,D1);
use A2(X,D2);

derivation A3(A1,E);
derivation A4(A2,E);

derivation condition (A3,C);
derivation condition (A4,C);

use A5(X,D3);
update (A5,I);

use (X,D4);
use (X,D5);

produce (X,D6);


data D7;
process Y;
process Z;

use (Y,D7);
produce (Z,D7);
```

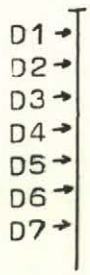
(Az itt "A"-betüvel kezdődő azonositók "belső azonositók" melyek az eredeti szövegben nem szerepeltek).

Végül bemutatjuk a nyert adatok képét reláció táblázatok formájában:

PROCESS:



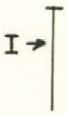
DATA:



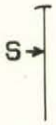
ENTITY:

CONDITION:

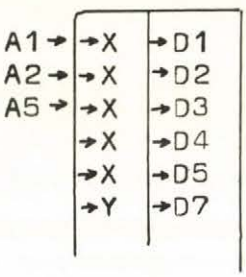
INFORMATION:



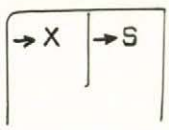
SYSTEM COMPONENT:



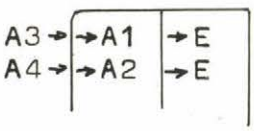
USE:



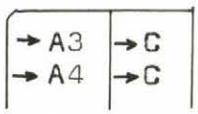
BELONG:



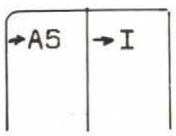
DERIVATION:



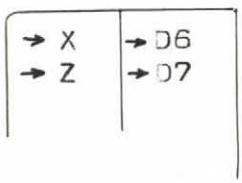
DERIVATION CONDITION:



UPDATE:



PRODUCE:



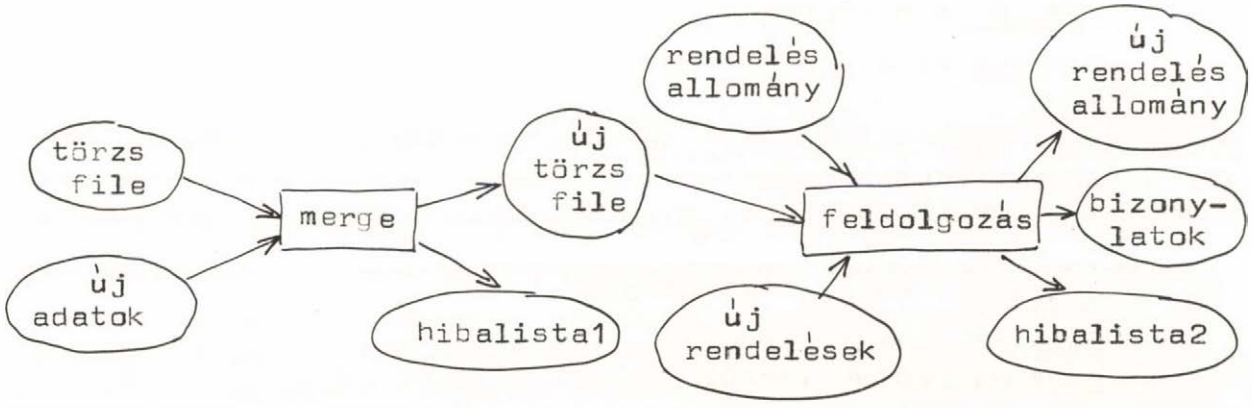
22. ábra

3. Néhány alkalmazási irány illusztrációja

3.1 Kauzális hálók

A kauzális hálók matematikai szempontból sok érdekességet rejtő, szerkezetükben azonban rendkívül egyszerű struktúrák. Segitségükkel csupán a szerkezetleirásokhoz való hozzáállás különféle lehetőségeit kívánjuk illusztrálni.

Egy kauzális háló (némileg leegyszerűsitve) egy irányitott párosgráf, melynek kétféle szögpontjait mondjuk példaképpen "adatok" és "eljárások"-ként interpretálhatjuk, és amely rendelkezik azzal a speciális tulajdonsággal, hogy az egyik fajta szögpontnak (most adatnak) legfeljebb egy be- és kimenete lehet. Alább egy egyszerű példát mutatunk:



23.ábra

Tekintsünk most néhány lehetőséget ilyenfajta strukturák formális leirására. Ennek egyik módja például a szögpontok atomi fogalmakként való bevezetése és az élek kapcsolatként való értelmezése:

```
(147)   concept data;
        concept procedure;
        concept consume(procedure,data);
           function of data;
           form procedure: consumes data;
        concept produce(procedure,data);
           function of data;
           form procedure: produces data;
```

Ebben az esetben a 23.ábrán látható példa leirása következő lehet:

(148) data törzsfile, új adatok, ... stb. ;

```
procedure merge;
   consumes törzsfile, új adatok;
   produces új törzsfile, hibalista1;

procedure feldolgozás;
   consumes új törzsfile, új rendelések, rendelés áll.;
   produces bizonylatok, hibalista2, új rendelés áll.;
```

Van azonban más lehetőség is. Kihasználhatjuk pl. azt, hogy az adatok kapcsolatai egyediek, ezért ezeket attributumként is hozzájuk rendelhetjük. (Ez egyébként főképpen a visszakeresések módját fogja befolyásolni). Ekkor tehát:

(149) concept procedure;
concept data(producer:procedure,consumer:procedure);

(Ezesetben a (147)-beli function integritásra sincs szükség, minthogy az automatikusan teljesül.) A leirás most (pl. abszolút mondatokkal) a következő lehet:

(150) procedure merge, feldolgozás;
data törzsfile(,merge);
data új adatok(,merge);
data új törzsfile(merge,feldolgozás);
data hibalista1(merge,);
stb.

Végül ugyanez

(151) concept data(producer:procedure,consumer:procedure);
form absolute: produced by producer used by consumer;

feltételezésével igy is irható:

(152) törzsfile: produced by nil used by merge;
új adatok: produced by nil used by merge;
új törzsfile: produced by merge used by feldolgozás;
stb.

3.2 SADT

Az SADT [4] modell esetén is többféle lehetőség kínálkozik a leíró nyelv ill. az adatbázis struktura kialakitására. Az alábbiakban egy ilyen lehetőség kiindulópontjait mutatjuk be.

Megjegyezzük, hogy az SADT modell számos kézenfekvő lehetőséget kínál (ld. pl.[9]) szemantikai jellegű integritások és kényszeritések alkalmazására. Ez azonban már egy külön dolgozatot igényelne. Egyelőre tekintsük a következő fogalmakat:

(153)
```
concept arrow;
concept box;
comment "arrow" általános értelemben (hálózatként) értendő;

concept actionbox is box;
concept actionarrow is arrow;

concept databox is box;
concept dataarrow is arrow;

concept subaction(part:actionbox,of:actionbox);
   function;
   form absolute: subactions of of are part;
   form of: subaction are part;

concept input(arrow,box);
concept output(box,arrow);

concept use is input;
   function;
   form box: uses arrow;

concept control is input;
   function;
   form box: controlled by arrow;

concept mechanism is input;
   function;
   form box: mechanism arrow;

concept generation is output;
   function;
   form box: generates arrow;

stb.
```

A most következő leirás a fent megadott fogalmak segitségével a [7] 69. oldalán lévő ábrákon szereplő példának egy leegyszerüsitett formalizálása:

(154) allokáció szimulátor: action;
subactions are vezérlés, tárkezelés, nyomtatás;

vezérlés: action;
controlled by kezelő válasz;
generates kezelési kérés, nyomtatási kérés,
szimulációs eredmény;

tárkezelés: action;
controlled by kezelési kérés;
generates kezelő válasz, allokált terület;
mechanism tárolo algoritmus;

nyomtatás: action;
controlled by nyomtatási kérés;
generates tárolási kép;
uses allokált terület;

subaction of vezérlés are órakezelés, blokk méretezés,
nyilvántartás, felszabaditás,
feltételkezelés;

órakezelés: action;
controlled by szinkron impulzus;
generates idő, látszblagos idő;

stb.

3.3 ISDOS

Nincsen akadálya annak, hogy akár az egész PSL nyelvet [3] eredeti formájában definiáljuk. Ez azonban itt nem célunk, csupán illusztráljuk az irányzatot néhány fontosabb PSL-beli fogalom (az egyszerüség kedvéért némileg módositott és kissé más szemléletmódú) megadásával:

(155) concept information;

concept logical information is information;
concept data is information;

concept set is logical information;
concept entity is logical information;

```
concept input is entity;
concept output is entity;

concept group is data;
concept element is data;

concept relation(e1:entity,e2:entity);
form e1: is related to e2;

concept association is relation(associated:data);

concept process;

concept use(process,information);
   form process: uses information;

concept purpose(use,information);

concept update is purpose;
   form use: to update information;

concept derivation is purpose;
   form use: to derive information;

stb.
```

Az e fogalmak felhasználásával nyerhető leirási forma pedig pl. az alábbi jellegű lehet:

(156)
```
process X;
   uses group G, input I;
      to update set G;
      to derive entity E;

input I1;
   is related to output P;
      associating data D;

stb. stb.
```

3.4 Data-flow jellegű struktúrák

Ismeretes, hogy a data-flow jellegű struktúrák a hagyományos Neumann-i architektúrák esetén is jól használhatók, többek között real-time rendszerek logikai sémajaként (példaképpen utalunk az MTA SZTAKI Folytonos Folyamatok Osztályán készitett rendszerre [8]).

Ilyen fajta rendszerek logikai szerkezetének leírásához pl. az alábbi ötleteket lehet sugallni:

(157)
```
concept data;

concept integer is data;
concept bit is data;

:
:
stb.

concept array is data(size:integer);
   comment figyelem: nem aláhúzott integer!;
   form absolute: is of size integer;

concept part(data,of:data);
   form of: consists of data;

concept condition;
concept operation;

concept pre(condition,operation);
   form operation: precondition condition;

concept post(condition,operation);
   form operation: postcondition condition;

concept reception(data,operation);
   form operation: data received data;

concept production(data,operation);
   form operation: data produced data;

stb.
```

Melynek használata pl. az alábbi formájú lehet:

(158)
```
operation X;
   precondition P1, P2;
   postcondition S1, S2, S3;
   data received D1;
   data produced D2, D3;

data D1;
   consists of integer I, array A, bit B;

array A;
   is of size I;
```

FÜGGELÉK

1. Belső szintaxis

Az alábbiakban a logikai sémanyelv szintaxisának fontosabb részeit foglaljuk össze. A leirást a könnyebb érthetőség kedvéért csak heurisztikusan, nem formalizáltan adjuk meg.

sémanyelv=definiciós rész,adat rész.

1.1 Definiciós rész

definiciós rész=def egységek nem üres sorozata.

def egység=defunit,deklarációk pontosvesszővel elválasztott sorozata,endunit.

deklarácio=fogalom|különálló integritás|különálló kényszerités.

1.1.1 Fogalom

fogalom=concept,fogalomnév,is,fölérendelt fogalom neve, attributum rész,kényszerités és vagy integritás esetleg.

attributum rész=üres|attributum megadások vesszővel elválasztva zárójelben esetleg.

attributum megadás=szelektor,kettőspont,tipus.

1.1.2 Tipus

tipus=érték tipus|hivatkozási tipus.

érték tipus=integer|real|text.

hivatkozási tipus=deklarált fogalomnév|universal.

1.1.3 Közvetlen kényszerités és integritás

kényszerités=imply,egy másik fogalom neve,eredeti fogalom egyes szelektorai vesszővel elválasztva zárójelben.

integritás=function,szelektorok felsorolása esetleg.

1.1.4 Reláció kifejezés

rel kif=fogalomnév|nagyitás|kicsinyitès|kivàlasztás|
illesztès|halm mű̈v.

nagyitás=rel kif,pont,oszlop megjelölès.

kicsinyitès=rel kif szögletes zárójelben.

kivàlasztás=oszlop megjelölèsek vesszővel elválasztva
zárójelben,rel kif.

illesztès=rel kif,csillag,rel kif.

halm mű̈v=rel kif,konnektiva,rel kif.

konnektiva= $\cap$ | $\cup$ | $\setminus$.

1.1.5 Kényszeritès és integritás

különàlló kényszeritès=constraint:,rel kif,oszlop azonositok megadàsa zárójelben,nyil,fogalomnév,behelyettesitett oszlopazonositók.

különàlló integritás=integrity:,int leirás.

int leirás=függőség|bin tul|halm tul.

függőség=rel kif,function of,oszlop megjelölèsek vesszővel elválasztva esetleg.

bin tul=binaris rel kif,tulajdonság.

tulajdonság=antisymmetric|irreflexive|identity|hierarchic|
precedence|lattice.

halm tul=rel kif,rel jel,rel kif.

rel jel= $\supset$ | $\subset$ | $\setminus$ | = .

1.2 Adat rèsz

adat rèsz=mondatok sorozata.

mondat=adat egysèg|módositàs|kérdès.

1.2.1 Adat egysèg

adat egysèg=dataunit,adat mondatok pontosvesszővel elválasztott sorozata,endunit.

adat mondat=alàhúzott fogalomnév,objektumnév esetleg.
obj kifejezèsek vesszővel elválasztott sorozata zárójelben esetleg.

1.2.2 Objektum kifejezés

obj kifejezés=elemi kif|obj kifejezés,pont,szelektor.
elemi kif=objektumnév|konstans.
konstans=nil|érték.

1.2.3 Modositás

modositás=átutalás|törlés.
átutalás=obj kifejezés,assign,obj kifejezés.
törlés=cancel,obj kifejezés.

1.2.4 Kérdés

kérdés=általános|standard.
általános=list,kifejezés.
kifejezés=rel kif|rel kifbe helyettesitett obj kifejezések.

2. Felhasználói nyelv

Csupán a főbb eltéréseket soroljuk fel:

2.1 Definiciós rész

felhasználói fogalom=fogalom esetleg is-rész nélkül, relativ formák pntosvesszővel elválasztva esetleg.
relativ forma=form,szelektornév,kettőspont,szelektorok alahúzott szövegbe ágyazva.

2.2 Adat rész

felhasználói adat mondat=adat mondat|beágyazott mondat.
beágyazott mondat=objektumnév kettősponttal esetleg, obj kifejezések alahúzott beágyazó szövegben.
felhasználói reláció kifejezés=relkif|beágyazott mondat.

Irodalom:

1. Holbaek,H.E.,Handlykken,P.,Nygaard,K. System description and the DELTA language. NCC, 1975. Delta Proj.Rep.4.

2. Dahl,O.J.,Myhrhaug,B.,Nygaard,K. SIMULA 67 Common Base Language. NCC, 1970. S-22.

3. Hershey,E.A.,Teichroew,D.,Berg,D.L.B.,Winter,E.W.,Bastarache, M.J. Problem Statement Language. ISDOS WP.68. 1974.

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

5. Kangassalo,H. Logical and semantical properties of environmental situation for stating information requirements. Univ. of Tampere, Dept. Math. Rep. A33. apr. 1979.

6. Knuth,E., Rényi,L. Closed reference schemes. MTA SZTAKI Working Paper II/7 , jun. 1979.

7. Bartha,J.,Farkas,Zs.,Garami,P.,Keresztély,Zs.,Koppány,L., Kosztolányi,Z.,Németi,T.,Sántáné-Toth,E.,Simor,G. A rendszertervezés korszerű módszereinek áttekintése.

8. Almásy,G.,Huppert,A.,Sztanó,T. Szóbeli közlés.

9. Bernus,P.,Hatvany,J. Computer aids to the design of integrated manufacturing systems.

A TANULMÁNYOK sorozatban 1979-ben megjelentek:

88/1979 Renner G. - Gaál B. - Hermann Gy. - Horváth L. - Várady T.: Szoborszerü felületek tervezése és megmunkálása

89/1979 Ruda Mihály: A SIS77 statisztikai információs rendszer /a felhasznált számitástechnikai eszközök, a rendszer szerkezete és programjai/

90/1979 Bányász Cs. - Keviczky L.: Optimum Insensitivity of the Linear-continuous Transformation

91/1979 Téli iskola /Szentendre/

92/1979 Andor László: Kisgépes adatbázis kezelő rendszer

93/1979 Bolla M. - Csáki P. - Fische J. - Herodek S. - Hoffman Gy. - Kutas T. - Telegdi L. - Witmman I.: A balatoni ökoszisztéma modellezése

94/1979 Gertler János: Egy statisztikus szürési eljárás számitógépes folyamatirányitásához

95/1979 Báthory M. - Galló V. - Kovács E. - Mérő L. - Siegler A. - Vajta L.: Festőrobot vezérlésére alkalmas alafelsimerési berendezés

96/1979 Mérő László: Konturkeresés zajos digitalizát képekben

97/1979 Pásztorné - Matavovszky T.: Boole-függvény kezelőrendszer

98/1979 Kecskés Zsuzsa: Három dimenziós tárgyak drótvázának ábrázolása vonalrajzoló grafikus berendezésekkel

99/1979 Ivics József: KGST Riga

100/1979 Téli iskola

101/1979 Gerencsér L. - Hangos K.: Diszkrét lineáris sztochasztikus rendszerek önhangoló szabályozása