tanulmányok 190/1986

MTA Számitástechnikai és Automatizálási Kutató Intézet Budapest

Magyar Tudományos Akadémia
Számitástechnikai és Automatizálási Kutató Intézete

A TIPUS FOGALMA, ÉS SZEREPE A MODELLEZÉSBEN

ABSZTRAKT ADATTÍPUSOK ALKALMAZÁSÁNAK ÚJ ELVEIRŐL

HERNÁDI ÁGNES

Tanulmányok 190/1986

Felelős kiadó:

KEVICZKY LÁSZLÓ

E dolgozat szerzője nem intézetünk dolgozója.
Munkáját aspiránsként készítette el.

Aspiráns vezető:

KNUTH ELŐD

ISBN 963 311 219 2
ISSN 0324-2951

T A R T A L O M

1. B E V E Z E T Ő . 5. OLD.

2. ABSZTRAKCIÓ ÉS MODELLEZÉS 7. "

2.1 A PROGRAMOZÁSI NYELVEK SZEREPE A MODELLEZÉSBEN 9. "

2.2 AZ ADATBÁZIS RENDSZEREK SZEREPE A MODELLEZÉSBEN 11. "

2.3 A MESTERSÉGES INTELLIGENCIA-RENDSZEREK SZEREPE A MODELLEZÉSBEN 13. "

3. A TIPUS FOGALMA 15. "

4. ADATTIPUSOK A PROGRAMOZÁSI NYELVEKBEN 20. "

5. ADATTIPUSOK AZ ADATBÁZISOKBAN 31. "

6. ADATTIPUSOK A MESTERSÉGES INTELLIGENCIA-RENDSZEREKBEN 45. "

I R O D A L O M 49. "

1. BEVEZETŐ

A programozási nyelvek, az adatbázis- és a mesterséges intelligencia-rendszerek egyaránt feltételezik, hogy az entitások "tipusokba" sorolhatók. Ugyanakkor "tipuson" nemhogy e három területen, de még az egyes területeken belül sem mindenütt ugyanazt értik.

A három terület kutatóinak problémái és motivációi különbözőek, ezért modellezési technikáik is eltérnek.

- Az adatokra és az adatok közös használatára nagyobb figyelmet forditanak a mesterséges intelligencia-rendszerek és az adatbázis alkalmazások, mint a programozási nyelvek.
- Az implicit számitások iránt jelenleg a mesterséges intelligencia kutatásokban és az adatbázis alkalmazásokban kifejezettebb a gyakorlati érdeklődés, mint a programozási nyelvekben.
- A formális technikák, különösen a specifikációs technikák jelentősebb szerepet játszanak a programozási nyelvekben, mint a másik két területen.
- Nagy, irreguláris, heterogén adatstrukturák többnyire mesterséges intelligencia-rendszerekben jelennek meg, és csak elvétve találhatók adatbázis-alkalmazásokban.

Az információfeldolgozás során adatnak nevezett objektumok várhatóan a valós világ valamely részének modelljeit ábrázolják, modelleken "emberi megismerő strukturákat", azaz gondolati dolgokat értve. A valós világban vagy az

objektumok gondolati világában megtörtént, végbemenő . vagy esetleg bekövetkező folyamatokat pedig operációknak nevezett müveletek ábrázolják. De mind a valós világ részeit, mind az ezeket ábrázoló adatokat folyamatok illetve operációk hozzák létre.

E megállapitásokból következik, hogy a programrendszerek "adatbázisai", "tudásbázisai" és "adatterei" között nincs alapvető különbség, legfeljebb a modelleket reprezentáló fogalmak és technikák térnek el. Ezért nélkülözhetetlen e fogalmak és technikák mélyebb megismerése, megértése.

2. ABSZTRAKCIÓ ÉS MODELLEZÉS

A szoftverfejlesztés és karbantartás területén az egyik fő probléma a rendszerek bonyolultságának kezelése. Az ezzel megbirkózni próbáló programozási módszertanokban és nyelvekben központi szerepet kap az absztrakciókkal foglalkozó eszközök fejlesztése. Az absztrakció egy rendszer olyan egyszerüsitett leirása vagy specifikációja, ami hangsulyozza e rendszer néhány részletét vagy tulajdonságát, mig elnyomja a többit.

Az absztrakció célja, hogy kezelhetőbbé tegye a problémát.

A mesterséges intelligencia területén ezt a paradigmát nevezik a "csaknem helyes tervekben való hibakereséssel történő problémamegoldásnak". Azért, hogy egy olyan könnyen megoldható, egyszerü problémaleirást kapjunk, melynek megoldása felismerhető, absztrahálhatunk, ami aktuálisan probléma-részleteket mellőzhet. A mellőzött részletek bevezetésekor előfordulhat, hogy hibát kell keresnünk az első megoldásban, hogy befogadja az uj részleteket.

Egy jó absztrakció az olvasó illetve a felhasználó számára lényeges információt hangsulyozza, mig a legalábbis pillanatnyilag lényegtelen részleteket nyomja el. Következésképp a jóság a környezet értelmezésétől függ. Az absztrakció mindig ahhoz a felhasználáshoz képest viszonylagos, amelyre a leirásokat alkalmazni kell.

Az, amit a programozási rendszerekben "absztrakciónak" nevezünk, közel áll ahhoz, amit a természettudományokban

és társadalomtudományokban "modellezésnek" neveznek. Sok probléma közös: a rendszer fontos jellemzőinek, a beleértendő változtathatóságnak - azaz a paramétereknek, a használandó leiró formalizmusnak, a modell érvényesitési módjának meghatározása, stb.

A modellezés három intellektuális feladatot ölel fel:
- valamilyen realitás érzékelését,
- egy megfelelő modell kiválasztását, és
- az érzékelt realitás ábrázolását az adott modellnek megfelelően.

Nyilvánvaló, hogy a reprezentáció ismét valamiféle realitás, és igy egy másik reprezentáció tárgya, egy másik modellnek vagy jelölésnek megfelelően.

Mivel az emberi észlelés tökéletlen, és csak bizonyos korlátok között vagyunk képesek a valóság teljes és pontos ábrázolására, bármely ábrázolás a valóság absztrakciója.

Az adatmodellezés paradoxonja, hogy

- alkalmazási szinten - szembekerülve a "valóság töredékeivel" - olyan részleteket érzékelünk, melyek általában ábrázolhatatlanok;
- reprezentációs szinten - szembekerülve a "gépek szintjeivel" - olyan részleteket ábrázolunk, melyek általában nem észlelhetők.

Az absztrakciós módszerek ugy birkóznak meg ezzel a problémával, hogy elnyomják az adott összefüggésben szükségtelen részleteket, és formalizálják, strukturálják a lényeges információt.

Általában elfogadott, hogy egy modell

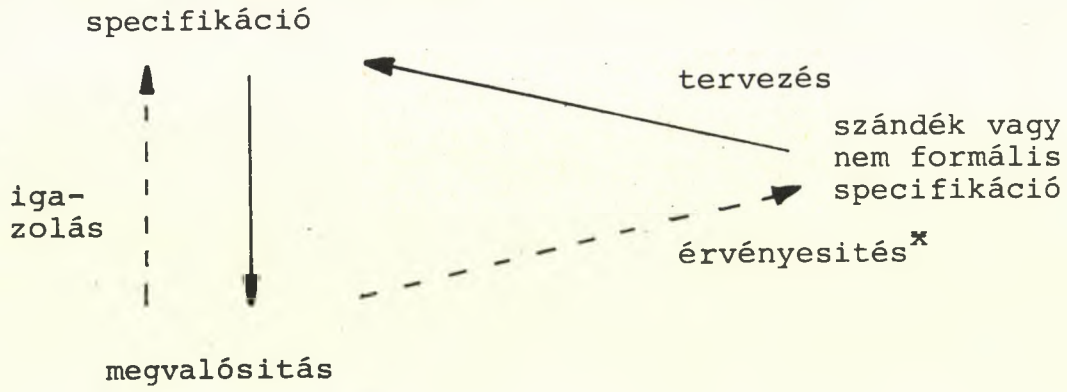
- objektumokból vagy "dolgokból" /pl. entitásokból, operációkból, üzenetekből, eseményekből/,
- objektum-kapcsolatokból,
- /a tipusfogalomhoz szorosan kapcsolódó/ kényszerekből,
- lokális és globális állapotokból,
- akciókból vagy állapotátmenetekből,
- akciókat hivó ágensekből, és
- a rendszer vagy a környezete által megértett célokból

áll.

A rendszerek absztraktabb és kevésbé részletes specifikációjának definiálására törekvő próbálkozások legtöbbje a rendszerállapotok definiálásához "objektumok tartományainak", a rendszerátmenetek definiálásához "objektumok operációinak" fogalmát használja. Predikátumok teljesülése biztositja, hogy az objektumok "jól-formáltak" és az operációk "jól alkalmazottak" legyenek. A tartományok és az operációk, valamint a definiciójukra szolgáló eszközök és technikák az alkalmazási területtől függően igen széleskörben változnak.

2.1 A PROGRAMOZÁSI NYELVEK SZEREPE A MODELLEZÉSBEN

A programozási nyelvek területén a modellezés a programok tervezését és a szoftver rendszerek szerkesztését érinti, amint ez az alábbi, Mary Shaw által javasolt ábrán is látható /1. ábra/. E terület kutatói sok, formális igazolásra és gyakorlati érvényesitésre alkalmas eszközt és technikát fejlesztettek és fejlesztenek ki.



1. ábra Modellezés a programozási nyelvekben

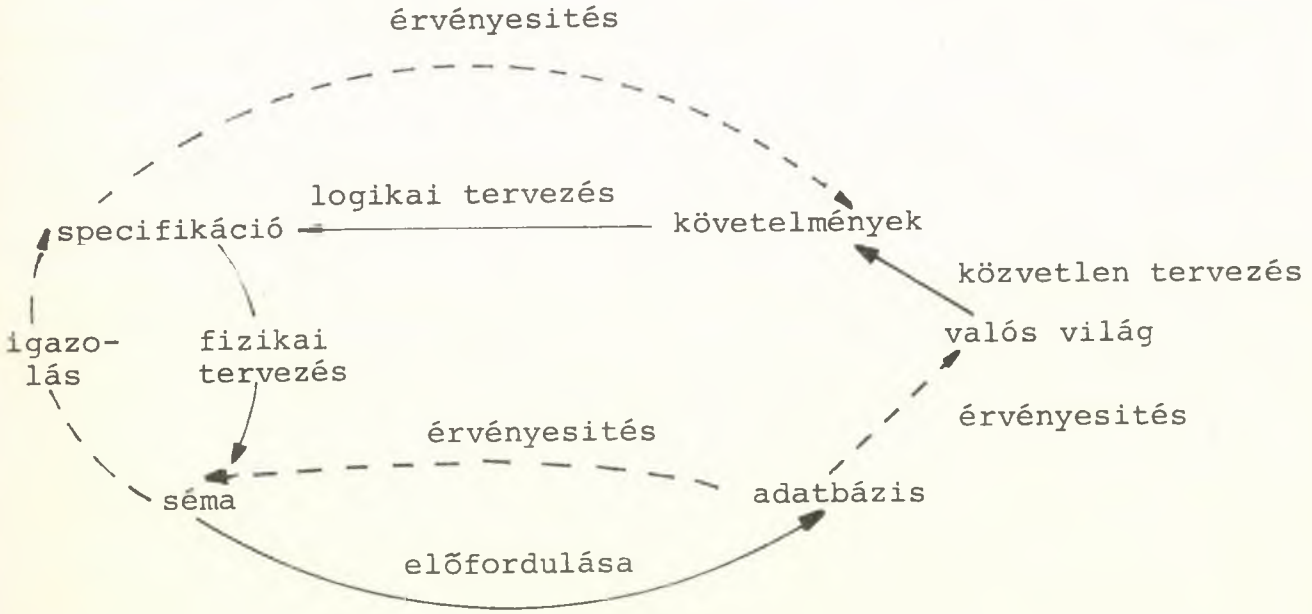
---

* ellenőrző visszacsatolás /validation/

A programozási nyelvek területén a modellek ágensei programok, és ezek olyan operációkat hangsulyoznak, melyek hallgatólagosan definiálják az állapotokat. Szemben a mesterséges intelligencia- és adatbázis rendszerek területein modellezett "valós világi" alkalmazásokkal a programozási nyelvek területén az alkalmazások többsége mesterséges, pl. operációs rendszerek, számitógépek, terminálok képernyős megjelenitői. A "valós világ" modelljei jórészt csak olyan helyzetekben fordulnak elő, melyekben a modell viselkedése fontosabb, mint az ábrázolt információ, pl. nukleáris reaktorok vagy az időjárás szimulációja esetén. A programozási nyelvek területén a modellek szemantikája általában jól ismert; például egy adatabsztrakció specifikációját az absztrakt adattipus tulajdonságainak teljes jellemzése definiálja. A programozási nyelvek területén kifejlesztett modelleket elsősorban szakértőknek és nem naiv felhasználóknak szánják. Bár a legtöbb modellezési eszköz és technika kifejezetten a viselkedéssel foglalkozik, és csak hallgatólagosan érinti a strukturával kapcsolatos kérdéseket, egyre növekvő érdeklődés tapasztalható az adat- és objektum-orientált modellezés iránt.

2.2 AZ ADATBÁZIS RENDSZEREK SZEREPE A MODELLEZÉSBEN

Az adatbázis rendszerek egy felhasználói közösség által használt, állandó, megformált /formatted/ adatbázisban próbálják megragadni azokat a tulajdonságokat, melyek bizonyos összetartozó, "valós világi" alkalmazások számára fontosak. Ezt a folyamatot ábrázolja a következő ábra /2. ábra/. Az adatbázis-alkalmazások szemantikus integritásának /érvényességének/ kimutatása fontos kutatási téma. A valós világgal történő nagyobbrészt intuitiv összehasonlitás helyett számos közbülsõ modellezési szintet és ezeknek megfelelõ érvényesitési technikát vezetnek be. Ehhez az adatbázis kutatások a mesterséges intelligencia-alkalmazásokból és a programozási nyelvek területéről veszik át a megfelelõ fogalmakat.



2. ábra Adatbázis modellezés

Az adatbázis modellek hangsulyozzák az emberekkel, mint a tranzakciók ágenseivel való kapcsolatukat. Az adatbázis alkalmazások nagymennyiségü formált adattal rendelkezhetnek, melyek jelentése elfogadhatóan jól megértett. Egy adatbázist ugy kezelünk, mint egymástól erősen függő adatok tartalom szerint cimezhető tárát, amely lekérdezhető és megváltoztatható.

A hagyományos adatbázis modellezési eszközök egynemüek; kevesebb strukturát és operációt adnak, mint a mesterséges intelligencia és a programozási nyelvek területein használatos eszközök. Adatbázis modellekben az adatstrukturák jellegzetesen fák, hálózatok vagy relációk. Az adatbázis-tervezés az alkalmazás szemantikáját megkisérli kivonni az alkalmazási programokból, és ezeket a tulajdonságokat egy központi sémában ábrázolja. Adatbázis alkalmazások készülnek naiv és szakértő felhasználók számára is, nagy- és kisméretü termelési környezetekben egyaránt. Ujabban nő az érdeklődés az adatbázis alkalmazások viselkedésének modellezése iránt.
A tranzakciókat jórészt figyelmen kivül hagyták az adatbázis modellezésekor, és nem foglalták adatbázis sémába. Ez a helyzet most változik. Az adatmodellezéssel kapcsolatos kutatás és fejlesztés jelenleg három vonalon folyik:

- reprezentációs szinten;

  az alacsony szintü tárolási jellegzetességeknél absztraktabb adatstrukturákat és adatmodelleket javasolnak, hogy közvetlenebbül elégitsék ki az adatmodellezési igényeket.

- alkalmazási szinten;

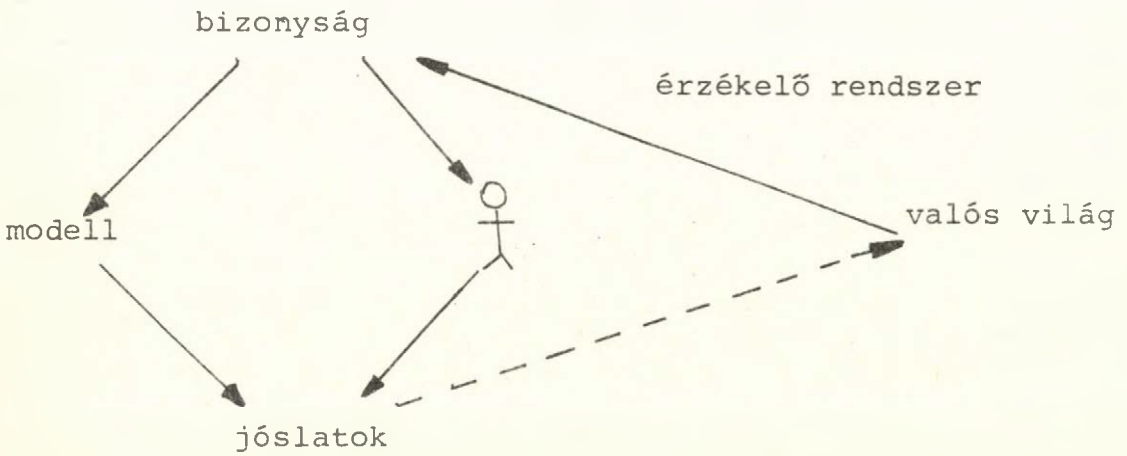
  az adatbázisok tervezését és formális definicióját támogató fogalmi modelleket fejlesztenek ki.

- eszköz szinten;

  a programok és programozási nyelvek nem reprezentáció-orientált eszközeit kiterjesztik és alkalmazzák adat- és adatmodell definiálására.

2.3 A MESTERSÉGES INTELLIGENCIA-RENDSZEREK SZEREPE A MODELLEZÉSBEN

A mesterséges intelligencia-rendszerek az emberi gondolkodást próbálják modellezni. Egy ilyen modell központi célját vázolja az alábbi, Ira Goldsteintől származó ábra /3. ábra/. Az érzékelő rendszer a valós világból vett minták alapján bizonyságokat gyüjt. E bizonyságok alapján mind a modell, mind az ember megjósolja a valós világ állapotát. A modell sikeres - azaz kielégiti az un. Turing tesztet - ha jóslatai ugyanolyan elfogadhatók, mint az emberéi vagy megkülönböztethetetlenek az utóbbitól. A "bizonyság" és a "jóslat" tág értelemben értendő.



3. ábra Modellezés a mesterséges intelligencia-rendszerekben

Például a mesterséges intelligencia azon területén, amelyet automatikus programozásnak neveznek, az a bizonyság, amit egy automatikus programozási modellnek és egy programozónak adnak valójában egy szoftver rendszer-specifikáció; és a jóslat nem más, mint egy futó szoftver rendszer. Természetes nyelvü lekérdező rendszerek esetén a kérdés a bizonyság és a válasz a jóslat. Tételbizonyitáskor pedig az axiómák és a kivánt tétel a bizonyság, és a bizonyitás a jóslat.

A mesterséges intelligencia-modellezésben a bizonyság eltérő, rosszul strukturált gyüjtésekből eredhet. Sok esetben a modellező folyamat kezdetén a bizonyság szemantikája sem értett. A modellező folyamat olyan primitivek definiálásával foglalkozik, melyek segitségével a modell /a személy/ céljai elérhetők, és azután olyan kereső stratégiákat definiál, melyek majd felfedezik e primitivek szükséges kombinációját. Az igy kapott rendszerek többnyire egy sajátos probléma megoldására, vagy egy kis felhasználói csoport számára készülnek.

3. A TIPUS FOGALMA

Nyilvánvaló, hogy az alkalmazott modell preciz definíciója a helyes modellezés nélkülözhetetlen kelléke. De hiába szigoruan definiáltak a használt modellek, ha a modellezési folyamat intuitiv és megbizhatatlan. Nem kevesen tekintik müvészetnek, és valószinüleg annak is fogják tekinteni még sokáig. Meggyőződésünk, hogy a modellezés folyamata algoritmizálható, és a tipus fogalma mint univerzális fogalom, segithet a modellezésben, függetlenül attól, hogy milyen modellt alkalmazunk.

A tipusoknak erős matematikai megalapozásuk van [GO'75], [GO'77], [GO'77a], [EH'78], [TH'78] és általánosan elfogadott, hogy sokmindenre használhatók. Mégsincs rögzitett tipusfogalom vagy tipus-használati mód. Ugy tünik, hogy különböző területeken különféle problémák megoldásának eszköze.

A tipusokat elsődlegesen objektumok vagy entitások osztályozására használják. Az osztályozás módja azonban szélsőségesen változik.

Néhányan értékek halmazaira,

> *Az adattipus egy halmazcsalád, ezen halmazokon értelmezett operációk vagy eljárások gyüjteményével együtt.*
>
> */Thatcher/*

mások szimbólumokon értelmezett predikátumokra,

> *A "t tipusunak levést" egy unáris T predikátummal ábrázolhatjuk.*
>
> */Hayes/*

ismét mások az entitások vagy objektumok tulajdonságaira utalnak, beleértve az operációkat is.

> *A tipus olyan strukturális és viselkedési tulajdonságok preciz jellemzése, amelyekkel egy aktuális vagy potenciális objektumgyüjtemény minden objektuma rendelkezik. A tipus előfordulása egy olyan objektum, ami a tipus karakterisztikus tulajdonságaival bir.*
>
> */Deutsch/*

A fenti definiciókban néhány fontos fogalom, pl. tulajdonság, objektum definiálatlan. Egy tipusrendszer tervezése során kell majd sajátos jelentést választanunk az objektumoknak, a tulajdonságoknak és a jelöléseknek. A gyakorlati alkalmazások számos olyan kérdést vetnek fel, amit nem érint minden tipusdefinició. Ilyen többek között a

- résztipus kérdése,
- a tipusok névszerinti kontra strukturális egyenlőségének kérdése,
- a többszörösen tipizált objektumok lehetősége.

Egy tipusrendszer "haszna" alapvetően az, hogy megengedi az objektumok univerzumának olyan - szükség esetén akár átfedő - felosztását, ami tükrözi a rendszer felhasználója és/vagy tervezője számára fontos eltéréseket.

Hogy egy tipusrendszer hasznos kölcsönhatásban lehessen egy programozási nyelvvel vagy egy adatbázis rendszerrel, a tipusrendszer tulajdonságainak valamilyen viszonyban kell lenniük a rendszer többi részében lévő operátorokkal vagy kapcsolatokkal. Pl. az egész szám /integer/ fogalma valószinüleg nélkülözhetetlen a programozási nyelvekben, és a tipusrendszernek le kell fednie. Ez az értékek és a változók tipusának, azaz az objektumoktól elválaszthatatlan és a programrészletekben vagy programleirásokban megjelölt tipustulajdonságoknak a megkülönböztetéséhez vezethet.

"Objektumok" alatt nem szabad csupán adatokat értenünk. Definiálható tipusrendszer eljárásokra, és készithető tipusrendszer kapcsolatokra. Eljárásokra sokféleképpen épithető tipusrendszer, pl. egyszerüen az eljárások funkcionalitása, vagy esetleg más, az eljárásokhoz tartozó absztrakt információ alapján. Gondolhatunk egy rendező eljárásra mint eljárástipusra, függetlenül az argumentumok és outputok tipusától. Vagy az eljárás-specifikációkat hierarchiába szervezhetjük a hozzájuk tartozó elő- és utófeltételek alapján is. Ezen tipusok előfordulásait ugy tekinthetjük, mint az eljárás aktuális alkalmazásait.

A tipusok alkalmazásának legalább annyi célja van, mint ahány felhasználó. A programozási nyelvekben a 70-es évek közepe óta erős hangsulyt kapott az operációk absztrakt adattipushcz történő kötése és a reprezentáció integritásának védelme. A programozási nyelvekben tipusok segitségével ellenőrzik, hogy egy operáció használata érvényes-e; de tipusok szolgálnak egy generikus operáció adott alkalmazás által megkövetelt sajátos előfordulásának kiválasztására is. Az adatbázis rendszerekben is használnak tipusokat

operációk érvényességének ellenőrzésére, de tipusok segitségével irnak le információt az objektumokról vagy entitásokról is. A mesterséges intelligencia-rendszerek területén megtalálható az összes programozási nyelvekben és adatbázisokban alkalmazott tipusfelhasználás. Ezekben a rendszerekben a tipus ad növekvő /incremental/ leirásokat a valós világi objektumokról, és tipus vezérli /control/ a keresési teret. A mesterséges intelligencia-rendszerekben ugyanis az volt a cél, hogy a dolgok bizonyos tulajdonságait kivegyék az általános célu következtető gépezetből /inference machinery/, és hatékonyabban hasznositsák őket. Erre szolgálnak például a sokat vitatott IS-A hierarchiák.

A tipus definicióján tul számit az a struktura is, amivel egy tipus-rendszer, ha ugy tetszik tipusok egy halmaza rendelkezik. Három fontos esetre gondolhatunk, bár több is lehet, éspedig a tipusok elkülönülhetnek, alkothatnak fahierarchiát, vagy rácsot.

A programozás alapvető problémája - különösen absztrakt adattipusok esetén - a funkcionális vagy specifikációs szintről a konstruktiv vagy logikai szintre, azaz a tipus implicit definiciójáról az explicit definiciójára való átterés.

A matematikában klasszikusan elfogadott az objektumok implicit bevezetése. Hilbert is azt mondta: nem törődöm azzal, hogy mik az objektumaim. A valós világgal nem törődve csak rendszere ellentmondásmentességét akarja bizonyitani. Egyes kutatók véleménye szerint a számitógéptudományban nem szabadna implicit definiciókat használni, mert egy implicit definicióban sosem leszünk képesek teljesen megragadni egy

alkalmazási világot. Szerintük a gyakorlati számitógéptudomány céljaira az absztrakciós folyamatot algoritmikusan kell megszerkeszteni, és formalizálni kell [WED'81]

A geometriai axiómák pl. implicit módon definiálják a pont és a sik fogalmát. Az explicit definicióhoz más beszédmódra lenne szükség; először beszélnünk kellene arról, hogy mi a pont, és azután kellene felépiteni egy strukturát. Egy explicit definicióban egy karakterláncot egy másikkal helyettesitünk. Pl. egy személyi számot ugy definiálunk, hogy ekvivalens egy névvel és egy cimmel.

Fontos kérdés, hogy hogyan határozzuk meg valamiről azt, hogy egy adott tipus előfordulása-e. Sokan ugy vélik, hogy azért léteznek tipusrendszerek, mert vannak néhány szabály alapján könnyen meghatározható tulajdonságok. Ez nincs mindig igy. Például egy mesterséges-intelligencia-rendszerben előfordulhat, hogy egy objektumról nem tudunk semmit, csak a tipusát. Tekintsünk egy rendszert, melyben három tipus van: "ember", "férfi" és "nő", és nincs előfeltétel megkülönböztetésükre. Csak azt tudjuk, hogy résztipusok. Meghatározhatjuk, hogy János a "férfi" és nem a "nő" előfordulása, de a "férfiről" vagy a "nőről" egy egyszerü tulajdonságot sem állithatunk.

4. ADATTIPUSOK A PROGRAMOZÁSI NYELVEKBEN

A továbbiakban az adatentitások /rész/kategóriájára szoritkozunk. Az adatentitások kategóriája - pontatlanul szólva - a számitógéppel kiszámitható és kezelhető objektumok halmaza. A tipus funkciója az, hogy e kategóriát további megkülönböztethető osztályokba sorolja.

Az, hogy az entitásokat miként osztályozzuk tipusokba, a tipus-osztályozás céljától függ. Széles értelemben véve minden egyazon tipushoz tartozó entitás közös tulajdonsághalmazzal bir. A gyakorlatban a tipusok megkülönböztetésére használható tulajdonság-halmaznak

- illeszkednie kell a tipus-osztályozás céljához, és
- meg kell engednie a tipushoz tartozás egyszerü vizsgálatát.

A programozási nyelvekben az entitások tipusokba osztályozásának legáltalánosabb célja, hogy egy operációnak egy operandusra történő alkalmazásakor igazolni lehessen, hogy a kérdéses operandus az operáció által várt entitáshalmazban, ha ugy tetszik tipusban, van. Itt a "várt" azt jelenti, hogy az operandusra válasz definiált, nem pedig azt, hogy az operáció szükségszerüen eredményt ad vissza az operandusra. Például a verem tetején lévő entitást visszaadó TOP operációt egy üres veremre alkalmazva nem adhat vissza semmit, de ehelyett kivételes feltétel /exceptional condition/ bekövetkezését jelezheti. A lényeg az, hogy a TOP operációt megvalósitó algoritmust ugy irták meg,

hogy üres veremre számit, de arra már nem, hogy verem helyett például egy egész számot kap operandusként. Igy a tipusok arra szolgálnak, hogy elõirják az operációk definiciójának eshetõségeit vagy tartományát.

Tipuson objektumoknak vagy értékeknek a halmazát értjük, operációk egy gyüjteményével együtt. Az operációk közül néhány a halmaz elemeit állitja elő, és a többi - talán - a halmaz elemeit használja. Ezek az operációk más tipusok elemeit is használhatják vagy előállithatják.

Az "operációt" itt széles értelemben használjuk; magában foglalja azt, amit közönségesen is operációnak tekintünk, mint például az összeadást az egész számokon, vagy egy uj elem beszurását az adatstrukturákon; de tartalmazza egy adat-entitás tulajdonságának vagy attributumának fogalmát is. Ez utóbbit néha [DAH'70] az operáció fogalmától különbözőnek tekintik. A tulajdonságoknak megfelelő operációk a tulajdonságokhoz való hozzáférés és a tulajdonságok módositása.

Mint emlitettük, ezeknek az operációknak tipikusan operandusaik vannak, és számos különböző tipusu eredményt állitanak elő. Pl. az az operáció, ami arra szolgál, hogy eldöntse, az adott elem egy bizonyos elemhalmazban van-e, logikai értéket ad eredményül. Következésképpen egy tipus szemantikai definiciója gyakran csak más tipusokkal, tipusok egy családjával összefüggésben adható meg.

A fenti absztrakt definiciótól független a modell vagy realizáció különálló részekbe vagy modulokba csomagolásának fogalma. Ez a fogalom a programozási nyelvek outputjának strukturálásában különösen fontos.

A modul egy vagy több tipusnak és/vagy egy vagy több eljárásnak vagy függvénynek "tokba zárt", számitógépes megvalósitása. A tokba zárás azt jelenti, hogy a megvalósitás sajátos alkotórészeire csak a modulon belüli kiváltságos programrészek hivatkozhatnak, és csak ezek manipulálhatnak velük. Tipusok esetén a modul egy reprezentációt definiál a modulban megvalósitott, implementált egyes tipusok elemeire, és olyan eljárásokat definiál, melyek megvalósitják, implementálják a tipusokra alkalmazható egyes operációkat.

Az absztrakt adattipusok fő felhasználása annak a hézagnak a pótlása, ami egy rendszer által kivánt és egy gép által nyujtott értékhalmaz és operációk között van. E fogalmi hézag hierarchiával történő pótlásához az absztrakt adattipusok specifikációira és konkrét reprezentációira is szükség van. Ezért az absztrakciónak vannak olyan megközelitései is, melyek a megvalósitással is foglalkoznak. Ezek az absztrakt adattipust olyan felhasználó által definiált tipusnak tekintik, ami két részből áll:

- a specifikációból, ami a felhasználó illetve a meghivó program által használható információt tartalmazza: leirja az összes operációt, definiálja azt, hogy a program más részei miként érhetik el és hogyan kezelhetik az objektumokat, és garantálja, hogy az ilyen tipusu objektumok bizonyos tulajdonságuak legyenek;

- és a számitógépes megvalósitásból.

Ehhez olyan védelmi mechanizmus párosul, ami biztositja, hogy a program egy része se rombolhassa le az adatértékek integritását. A megvalósitás sajátos részeire

való hivatkozás megszoritott, a megvalósitás tokba zárt. A programtervezés és fejlesztés folyamatában definiálhatunk egy specifikációt anélkül is, hogy szükségszerüen megvalósitást adnánk.

Egy adattipus specifikációjának, külső nézetének kell definiálnia az értékhalmazt. Az ALPHARD-ban például az értékhalmazt kifejezetten definiálni kell, matematikai entitások /halmazok, vektorok, függvények/ segitségével. Az algebrai megközelitésben az értékhalmaz implicit módon definiált, például egy algebra fölött generált ekvivalenciaosztályokként. Az operációkat állapotátmeneti relációk jellemzik. Az állapotátmeneti relációk adott implicit definiciók, az operációk előtti és utáni lehetséges rendszerállapotokra vonatkozó feltételekkel kifejezve. A feltételeket predikátumok fejezik ki. Az előfeltétel predikátum irja elő az állapotátmeneti reláció értelmezési tartományát. Ha az előfeltétel igaz, akkor van egy absztrakt állapot, ami a kivánt állapotátmenet. Az utófeltétel egy olyan predikátum, ami akkor és csak akkor igaz, ha egy adott utóállapot párosul valamelyik előállapottal az állapotátmeneti relációban.

Egy absztrakt adattipus konkrét reprezentációs szintjén, belső nézetében, adott állapotokkal kiséreljük meg az absztrakt állapotok ábrázolását. Egy konkrét állapotátmeneti reláció a fent definiált absztrakt állapotátmeneti relációval szemben olyan konkrét állapotokkal definiálható, melyeket az operáció számitógépes megvalósitásában alkalmazott konkrét primitivek kompoziciója kapcsol össze.A konkrét primitivek mindegyikének van saját, implicit módon definiált, konkrét állapotátmeneti relációja.

A külső és belső nézetet az absztrakciófüggvény kapcsolja össze. Az absztrakciófüggvénynek kell az implementáció-invariánsokat, azaz a megvalósitás által változatlanul megőrzött tulajdonságokat kielégitő konkrét állapotokat az absztrakt állapotokba képeznie ugy, hogy a kezdeti konkrét állapotok a kezdeti absztrakt állapotokba, azaz a lényeges előfeltételt kielégitő állapotokba menjenek. Ha két állapot a származtatott konkrét állapotátmeneti relációban van, akkor absztrakt képeiknek az absztrakt állapotátmeneti relációban kell lenniük.

Ez utóbbi megközelités alapján ugy tünhet, hogy a programozási nyelvekben az absztrakció kapcsolja össze a számitógépes megvalósitást és a specifikációt. Ez azért zavaró, mert mind a megvalósitás, mind a specifikáció adatabsztrakció abban az értelemben, hogy mindkettőre előirt egy operációhalmaz és bizonyos kényszerek.

Valójában a leirásnak két szintje van, és az absztrakció a kapcsolat köztük. Ezért ahogy másodszor definiáltuk, az adatabsztrakció egy specifikációs technika.

A tipusok fogalmát a programozási nyelvekbe olvasztva kellemetlen problémák adódhatnak, ha nem különböztetjük meg a szimbólumokat és az általuk reprezentált objektumokat. A nehézségek gyökere az, hogy a számitógép rendszerek az objektumokat gyakran ábrázolják olyan szimbólumokkal, melyeknek saját belső strukturájuk van. Például a P pontról beszélve a "P" betüt strukturálatlan szimbólumként használjuk a pont jelzésére. De egy számitógép-rendszer P-t olyan szimbólum felhasználásával jelölné, melynek legalább három strukturális tulajdonsága van: egy /n hosszuságu/

bitsorozat az abszcissza ábrázolására, egy /n hosszuságu/ bitsorozat az ordináta ábrázolására, és egy memóriahely.

Egy programozási nyelvben a tipus-információ előirásakor tulságosan gyakran nem világos, hogy az adatentitásokra, az ezeket reprezentáló szimbólumokra, vagy mindkettőre teszünk-e állitásokat. Mit tartsunk egy olyan állitásról, mint "egy pont egy valós számpár"?

Ez a zavar természetesen a programozási nyelvek tervezésének tradiciójából ered. A programozási nyelvek tervezése a korai autókódokkal kezdődött, melyeket az assembly kód magas szintü röviditésének véltek. Az assembler nyelvben több reprezentáció érhető el bizonyos közös adatstrukturákra, nevezetesen - egész és valós - számok. A programozók azonban nem akartak törődni ezekkel a megkülönböztetésekkel, igy bevezették e megkülönböztetések jelzésére a tipusokat - bár erre inkább a forditóprogramnak volt szüksége mintsem a programozónak. Ez a tipusfogalom a nyelvi rendszer által kezelt szimbólumok ill. e szimbólumokat számitógépen megvalósitó adatstruktura belső strukturájára utal, nem pedig a szimbólumok által reprezentált dolgok közti kapcsolatokra.

A zavart csak növeli a nyelvtervezés jelenlegi trendje, ami a SIMULA-ban indult, és folytatódik a SMALLTALK-ban, APIARY-ban, CLU-ban, ALPHARD-ban, stb. Ezek ugyanis a rendszer absztrakt objektumokat ábrázoló, programfutás során keletkező strukturáit objektumokként kezelik. A rendszerbe épitett, és a felhasználók által definiált tipusok egységes kezelésére törekedve összeolvasztják ezeket az objektumokat és a hagyományosabb nyelvi strukturákat, és ugy vélik, hogy a számitógépes megvalósitás

programszövege nemcsak leirja az objektumokat, hanem az objektumokhoz is van rendelve. Ez egy intuitiv képet idéz elő, mely szerint a nyelv által leirt objektumok - a valamilyen tipusu adatentitások - aktiv közvetitők, mindegyiknek saját szövege van, futnak, üzeneteket küldenek egymásnak, stb. Bár ennek a képnek kétségtelenül van valami intuitiv varázsa, nehéz összeegyeztetni bármilyen következetes kapcsolattal, ami a nyelv és valami külső világ között van.

Ez a zavar biztosan nem szükségszerü. Bár ezeknek a belső "objektumoknak" strukturájuk van, ami valamiféle tipushierarchiába rendezi őket, lényegében szintaktikus entitások. A zavar elkerülésére meg kell különböztetni a szimbólumok strukturáját és azt, hogy hogyan használjuk ezeket objektumok reprezentálására.

A hagyományos nyelvekben a tipus egy változó attributuma, ami a változóhoz bizonyos információt társit arról, hogy hogyan használható a változó. A tipust egy létrehozó mechanizmusnak is tekinthetjük, azaz egy sablonnak, amely egy programban használatos változók létrehozására szolgál. A tipus társitható az értékkel, és a tipus-információ a változóval. A "változó: tipus" kapcsolat rendszerint n:1; mert a változók csak egy tipus értékeire hivatkozhatnak. Lehetnek azonban olyan változók, melyek egynél több tipus értékeire hivatkozhatnak. Az APL-hez hasonló nyelvekben a tipus azokhoz az értékekhez társul, melyekre egy változó utal.

Azt a kérdést, hogy egy tipus a változókkal vagy az értékekkel van-e társitva, különböző programozási nyelvekben különbözőképpen oldják meg. Pl. a RUSSELL-ben a tipusok csak a változókkal vannak társitva és az értékekkel egyáltalán

nem. A tipusfogalom teljesen szövegbeli, igy egy objektum tipusának megkérdezése értelmetlen.

Vannak rendszerek, melyekben a tipust tipusként kezelik, azaz vannak tipusokon értelmezett operációk, és vannak rendszerek, melyekben a tipusokon nem értelmeznek operációkat.

A tipus a paraméterezés fogalmával [TH'78] nyeri el teljes erejét, ami az un. "paraméterezett tipusokhoz" vagy "tipuskonstruktorokhoz" vezet. Az /absztrakt/ adattipusokra alkalmazott paraméterezés megengedi, hogy olyan /absztrakt/ adattipusok osztályait irhassuk elő, melyeknek értékhalmazai, sőt részben operáció-szemantikái is különbözők. Tekintsük például az ARRAY[CARRIER,INDEX] paraméterezett /absztrakt/ adattipust, ami nyitva hagyja, hogy egy tömbbe milyen elemek milyen helyekre kerülhetnek. Ezen paramétereket definiálva egy /absztrakt/ adattipust nyerünk.

Mint az eddigiekből kitünik, az absztrakt adattipusok elsődlegesen a programozási nyelvekbeli absztrakciós technikákra összpontositanak. Ez a módszertan kétféleképpen is korlátozott:

- az a modell, amit a programszervezéshez nyujtanak, nem alkalmazható minden program és programrész esetén;

  Néhány esetben a problémák nincsenek kellemesen adattipusokba öntve, vagy a szükséges funkcionalitás nincs kifejezve algebrai axiómákkal. Más esetekben a probléma olyan definicióhalmazt követel, melyek nyilván nagyon hasonlóak, de mégsem fejezhetők ki egy adattipus-definició előforduló-

saival /instantiation/, még generikus definiciókat használva sem. Ilyen értelemben további alternativákat kellene keresni a programabsztrakciók mostanáig jól-megértett két operációjára, a kompozicióra és az előfordulással való szemléltetésre. Uj modellfajták bevezetése természetesen kérdéseket vet fel; hogyan használjunk több modellfajtát hatékonyan egy egyszerü programban.

- az absztrakt adattipus kutatása kizárólag a funkcionális helyességre koncentrált, és figyelmen kivül hagyta a programok egyéb tulajdonságait.

Extrafunkcionális tulajdonságok például az idő- és tárkövetelmények, a tárhozzáférési minták, a megbizhatóság, a szinkronizáció, a folyamat-függetlenség. Olyan specifikációs módszertanra lenne szükség, ami megengedi, hogy

-- *a programozó tulajdonságokra vonatkozó állitásokat tehessen és ezeket igazolhassa, ne egyszerüen a programszöveg elemzésével származtasson pontos értékeket vagy teljes specifikációkat. Ez analóg azzal, hogy a funkcionális specifikációban előirjuk a számitás bizonyos fontos, megőrzendő tulajdonságait, és nem formálisan származtatjuk a program által definiált matematikai függvényt.*

-- *ne adjunk uj fogalmi szerkezetet minden egyes uj tulajdonság-osztályra.*

Mindezt még gyakorlati problémák is tetézik:

- a legtöbb esetben egy egyszerü modulban csak egy tipus definiálható;
- a tipusoknak szigoru hierarchiába kell illeszkedniük;

  Egy ilyen hierarchiában egy objektum nem tartozhat két más nemü tipushoz - bár vannak, akik ezt éppen ellenkezőleg szeretnék.

- problémás két specifikáció "eléggé hasonlóságának" definiálása;

  Az a definició, hogy a két specifikáció jelölje ugyanazt az értékhalmazt, és definiálja rajtuk ugyanazon operációkat, tulságosan leegyszerüsitett. Adott alkalmazás esetén lehet két specifikációnk, melyek értékhalmazai és operációi kissé különbözők. Azonban az alkalmazás által megkövetelt operációk a két specifikáció metszetébe esnek. Igy a két specifikáció különböző, de elég hasonló erre a sajátos célra.

  Az a definició, hogy a két specifikáció ugyanahhoz az elmélethez vezessen /mint például csoportot definiálhatunk osztással és bizonyos axiómákkal vagy szorzással, inverzzel és azonossággal/, tul erős lehet. Néhány alkalmazáshoz a hasonlóság gyengébb fogalmára van szükségünk. Programfejlesztés során a teljesitmény szempontjából jövedelmező lehet egy tipusunk megvalósitásának megváltoztatása. Előfordulhat, hogy az alternativ ábrázolás specifikációja kissé különbözik a helyettesitett specifikációtól, az eltérések azonban nem számitanak abban az alkalmazásban, amiben a tipust használjuk.

- milyenek a használt programozási nyelv tipusellenőrzési szabályai, mikor történik a tipusellenőrzés forditáskor vagy futáskor;

- a számitógépes megvalósitás kérdései, például lehet-e a modulokat külön forditani, stb.

5. ADATTIPUSOK AZ ADATBÁZISOKBAN

Az adatbázistervezés régen csaknem kizárólag adattal foglalkozott; azonban kiderült, hogy az eredményül kapott sémák alkalmatlanok bizonyos kényszerek, főképp az állapotátmenetek leirására. Ezért fordult az adatbázis-kutatók figyelme az adattipusok felé. Már jóideje kisérleteznek az adatbázis-modellezés és az adatabsztrakció fogalmainak összeegyeztetésével, tekintve, hogy a programozási nyelvek adatabsztrakciós világából az információelrejtés és a tokba zárás fogalma átfedi az adatbázisok adatfüggetlenségének fogalmát.

Hogy megértsük hogyan is került a tipus az adatbázisok világába, szükségünk van néhány fogalomra. A reláció fogalmához kapcsolódik a relációs séma fogalma, ami a reláció nevéből, a relációban előforduló attributumok neveiből, és azon tartományokból áll, melyekből ezek az attributumok értékeiket veszik.

A relációs modellel kapcsolatos az az ötlet, hogy az oszlopok vagy attributumok valamely kombinációjának azonositási tulajdonsága van; azaz az attributumok ezen kombinációjához társitott értékek megkülönböztetik a reláció bármely két sorát. Ha a választott kombinációban egy attributum sem nélkülözhető, akkor ezt a kombinációt lehetséges kulcsnak nevezzük. Egy adott relációban több lehetséges kulcs is lehet. Megállapodás szerint egyet elsődleges kulcsnak választunk. Ezzel a választással kapcsolatos az az intuitiv elképzelés, hogy az elsődleges kulcs értéke valahogy ábrázol vagy egymástól megkülönböztet bizonyos valós világi objektumokat.

Egy felfogás szerint mind a reprezentáció forrásához, mind a reprezentáció céljához társitottak egy tipust. Ezt a tipusfogalmat használták az operációk, például az összekapcsolás /join/ korlátozására is [CO'79]. Eszerint a rendszernek meg kell akadályoznia két reláció összekapcsolását két különböző tartományu attributum felett - de legalább is riasztania kell a felhasználót, ha ilyesmivel próbálkozna.

A következő tipusfogalom hivatkozási fogalom. Tegyük fel, hogy számos relációnk van, ugyanazon tartomány feletti attributumokkal. Egy tetszőlegesen választott oszlopban tekintsük egy sajátos érték valamely előfordulását. Ez az érték nyilvánvalóan több egy puszta értéknél: fel van cimkézve azon tartomány nevével, ami fölött az oszlop definiált. Ez az érték utaló jellel jelöl meg ugyanezen érték adatbázisában minden más, hasonlóan cimkézett előfordulást. Ezek a kereszthivatkozások nem helyettesithetők egy egyszerü pointerrel, a hivatkozások vagy összefüggések tömege miatt.

Az adatbázisokban eleinte nem volt jó résztipus-fogalom. Bevezetése John és Dianne Smith absztrakciós értekezésének [SM'77b] tulajdonitható. Ha például emberekről van egy adatbázisunk, és az emberek bizonyos részosztályairól speciális információt akarunk feljegyezni, akkor ilyen résztipussal kapcsolatos problémákba bonyolódunk. Az adatbázis rendszernek fontos tudnia, hogy mely dolgok miknek a résztipusai.

A világ modellezésében gyakran találkozunk azzal, hogy egy entitás, mondjuk Kis János "személy", több tipushoz /osztályhoz/ tartozik, és olyan operációkkal

/tulajdonságokkal/ bir, melyek egyes tipusbeli tagságára nézve specifikusak. Például Kis János lehet az egyetem "hallgatója", és egyidejüleg részidős "alkalmazottja" is. Mindkét szerepében az egyetemen ismert "személy". A tipusoknak azt a gyüjteményét, amihez egy entitás tartozik, gyakran általánositó strukturának nevezik. A tartalmazott tipusokat, mint "hallgató" és "alkalmazott" a tartalmazó tipus, a "személy" résztipusainak nevezik. Egy általánositást vagy szupertipust az általánositott entitás tipusára alkalmazható operációk egy részhalmazának kiválasztásával definiálunk. A "személy" általánositásnak lehetnek olyan operációi, melyek elérnek egy állandó cimet, egy telefonszámot, stb. Ezek a "hallgatóra" alkalmazható operációk részhalmazát képezik, melyek között lehetnek osztályzatok és más hallgató tulajdonságok elérésére és módositására szolgáló operációk is. Nyilvánvaló, hogy ugyanazon tipuson többszörös általánositás ugy lehetséges, hogy az operációk különböző részhalmazait választjuk megőrzendőnek.

Van némi hasonlóság az általánositás fenti képzete és a programozási nyelvekben levő unió tipusok fogalma között. Mindkettőnek van résztipus fogalma, vagy talán pontosabban vannak más tipusokba ágyazott tipusaik. A különbség az, hogy a fent definiált általánositások nyilt végüek, azaz, ha egy olyan uj tipust definiálunk, ami rendelkezik az általánositás által megkövetelt operációkkal, akkor ez automatikusan az általánositás részosztályává válik. Az uniók viszont explicit módon azonositott osztályok véges gyüjteményéből állnak. Az is igaz, hogy az unióbeli

osztályoknak nem lehetnek közös operációi. Az uniók reprezentációs problémák megoldásában a leghasznosabbak, például a listák tipusának definiálásakor hasznos, ha más az üres és a nem üres lista reprezentációja. A listareprezentáció a két részeset reprezentációjának uniója lenne.

A reláció tipusának fogalmával kapcsolatban több kérdés vetődik fel. A relációnak összetett értelmezési tartománya van, ami a relációbeli oszlopok értelmezési tartományaiból áll. Ez az összetett tartomány a reláció tipusának tekinthető. Megegyezik-e ez egy n-es vagy egy rekord tipusával? Ha abból az álláspontból indulunk ki, hogy egy tipus objektumokból és operációkból áll, ez a kettő nem ugyanaz, mert a relációkon értelmezett operációk halmaza nem szükségszerüen egyezik meg az n-eseken értelmezett operációk halmazával. Tekintsünk két relációt, legyen mindkettőnek csupán két oszlopa, melyek ugyanazon tartomány felett definiáltak. Ezek a relációk azonos tipusuak? Azonos tipusuak-e, ha az alkalmazható operációk is azonosak?

Helyezkedhetünk olyan álláspontra is, hogy a dolgokat más okok alapján különböző tipusuaknak mondjuk, de a halmazok és operációk szempontjából azt is mondhatjuk, hogy azonos tipusuak. A kérdés az, hogy szükség van-e névszerinti megkülönböztetésre a tipusok között, vagy elég a strukturális megkülönböztetés. A válasz: tipusok azonosságához szükség van mind a névszerinti, mind a strukturális megegyezésre.

Egy relációtipussal társitott értékhalmaz az n-es tipusával társitott - egyedi kulcs feltétellel megszoritott - értékhalmaz hatványhalmaza. Ez azt jelenti, hogy a reláció változhat a teljesen ürestől az összes lehetséges nem redundáns n-est tartalmazóig.

Egy adatbáziskezelő rendszer vagy nevezzük adatmodellnek attól függően, hogy az implementációról vagy az absztrakt modellről beszélünk-e, tipusgenerátorok halmaza. Például egy relációs adatbázisnak olyan tipusgenerátorai lehetnének, amelyek tartományokat és tartományokból relációkat szerkesztenek. Egy CODASYL rendszer lényegében a tipusgenerátorok olyan halmazát adja, mint "area" és "set-of".

Egy tipusgenerátor előfordulása adatbázis-sémát eredményez. Az adatbázistervező készit egy tipus-generátorhalmazt, és létrehoz egy tipushalmazt. Az adatbázis-sémák tervezői valójában a tipusok uj előfordulásait adják meg, az adatbázis felhasználói pedig az objektumokat vizsgálják. Ezek az emberek lefelé exportálják a tipust.

Az átlagos adatbázis modelleknek szabványos reprezentációs alakja vagy strukturája van az entitásokra, ezen entitások tulajdonságaira és a köztük lévő kapcsolatokra. Egy adat-entitás modellje szabványos modellezési primitivek viszonylag kis halmazából áll össze. Például a relációs modellben egy modell tartományokból és relációkból tevődik össze. Az egyes modellek esetén az operációknak egy szabványos halmaza van - create_entity, update_entity, delete_entity, find_entity -, melyek ezen adatmodellben ábrázolható minden adatentitásra alkalmazhatók. Azok az operációk, melyek egyedülállóak az adatentitások egy sajátos osztályára nézve, többnyire nincsenek explicit módon azonositva. Ezt a megközelitést azért használják, mert nehéz, hacsak nem lehetetlen megjósolni az adatokra feltehető összes kérdést. Egy általános célu struktura egy, a teljes gyüjtemény töredékeinek kombinálására alkalmas nyelv esetén megengedi az előreláthatatlan kérdéseket.

Ahogy van egy absztrakt adatfogalom, amit strukturának nevezünk, éppen ugy van egy absztrakt viselkedésfogalom is. A struktura- és operáció-specifikációkban megadott absztrakt tulajdonságok elhatárolhatók a reprezentációtól. Azt, amit strukturának nevezünk, a programozási nyelv kutatásban nem különböztetik meg a reprezentációtól, de az adatbázis-kutatásban igen /belső és fogalmi sémák megkülönböztetése/. Természetesnek tünik például, hogy egy alkalmazottat olyan entitásnak tekintsünk, melynek bizonyos adat- vagy információ-tulajdonságai vannak: név, kor, fizetés, stb.; és ne egy olyan operációsorozat eredményének, mint

előléptet(bér(alk),teherautósofőr).

A programozási nyelvekben a strukturát reprezentációként kezelik, és mint ilyen, nem absztrakt. Az adatbázistervezés lényegét ezek a strukturális absztrakciók képezik.

Sok esetben a modellezett entitásokra alkalmazható operációk egész közvetlenül ezen entitások reprezentációin értelmezett operációkba fordithatók; például egy entitás valamely tulajdonságával társitott érték megváltoztatása az adatbázis módositó /update/ operációjával történik. Vannak azonban olyan esetek, amikor sem a reprezentáció, sem az operációk nem illeszkednek természetes módon az adatmodellbe.

Tekintsük például a sor entitások tipusát, a sor elemeinek beiktatására, eltávolitására és átrendezésére szolgáló operációkkal. A sor ábrázolásához meg kell határozni az elemek reprezentációját, és az elemek rendezésének módját. Egy relációs modellben a sor-elemeket ábrázolhatják n-esek: egy tartomány az elemazonositó, a többi a tartalom. A rendezés ábrázolható egy n-esbeli extra rendezési tartománnyal

vagy egy olyan külön reprezentációval, ami a rendezésben egymást megelőző/követő elemazonositók párjaiból áll. A sorokon értelmezett operációknak nem egyszerü, lekérdező nyelvben megfogalmazott kifejezéseknek, hanem olyan "programoknak" kell lenniük, melyek figyelembe veszik a rendezés választott reprezentációját.

A sor szempontjából a sor elemeinek formája a sor külső tulajdonságaként van előirva, mig a sor rendezésére választott reprezentációnak nem kell kivülről látszania, ehhez csak a sorokon értelmezett operációk reprezentációjának van köze. Igy a fenti reprezentációnak két külön része van:

- *a modellezett entitásban előforduló, külsőleg definiált entitások reprezentációja, és*
- *a modell részleteinek reprezentációja.*

A sorokhoz relációs reprezentációt választva a reprezentációnak ez a két oldala összekeveredik, és mindkettő felszinre kerül. A relációs reprezentáció nem a sor közvetlen modellje; tul sok oda nem tartozó lehetséges kölcsönhatást enged meg. Csak akkor válik a sor modelljévé, amikor a hozzáférés az értelmezett operációkra korlátozott.

A lényeg az, hogy a reprezentáció nem modellezés, és az adatmodell-operációk csak néha modellezik a modellezett entitásra alkalmazható operációkat.

Az adatbázis-alkalmazásokkal foglalkozó kutatók felismerték, hogy egy entitás modellezéséhez a lényegi reprezentációnál több kell. Ez látható az adatmodellekkel kapcsolatos

utóbbi munkákban [CO'79], [SM'77a], [SM'77b], ahol a modellbe több deklarativ információt épitenek be ugy, hogy az adatmodell szabványos operációi, mint például a törlés /delete/, pontosabban tükrözzék a modellezett entitás szemantikáját. Amikor egy entitást számos "normalizált" alkotórésszel modelleznek, a normalizálási információt használják fel annak biztositására, hogy a felső szintü alkotórész törlésekor az összes alkotórész törlődjön. Ez a megközelités növeli azon entitások számát, melyeknek természetes modelljük van az alkalmazott adatmodellben, de még ilyen szemantikai bővitések mellett is esztelenség lenne azt hinni, hogy az adatreprezentációk bármely rögzitett halmaza természetes modelleket ad minden entitáshoz. Ezt a legtöbb adatmodell-tervező még csak nem is igényli.

Az adatmodell reprezentációban az implicit tulspecifikálás megoldására az adatbázis-kutatók a választott reprezentációt integritási kényszerek halmazával bővitik. Az integritási kényszer ötlete onnan ered, hogy a szabványos adatmodell-operációk csak akkor válnak megengedetté, ha az eredmények ki fogják elégiteni a modellezett entitások reprezentációjának formájára és tartalmára vonatkozó kényszerek /vagy invariánsok/ halmazát. Tipikus kényszer lehetne az, hogy egy adott részlegbeli összes alkalmazott fizetésének összege nem haladhatja meg e részleg teljes fizetési költségvetését. Ekkor egy alkalmazott fizetésének növelésére szolgáló módositó operáció csak akkor lenne megengedett, ha a részleg fent megfogalmazott költségvetési kényszerének eleget tenne.

A kényszer képzete nagyon hasonló a programozási nyelvekben alkalmazott adatabsztrakciók megvalósitásában lévő invariáns fogalmához. Mindkettő a modellezési tér olyan alterét

definiálja, mely elégséges a modellezett entitások ábrázolásához. Abban különböznek, hogy az invariáns kielégitését az entitásokra alkalmazható operációkat realizáló adatmodell-operációk tokba zárt halmazaira vonatkozóan állapitják meg, azaz az invariánsok kielégitését egyszer és mindenkorra megállapitják, és nem kell ismételten ellenőrizni. Ezzel ellentétben a kényszereket sokszor kell ellenőrizni, a reprezentáción értelmezett, egymással összefüggésben lévő operációk minden egyes halmaza után. Ez nem jelenti azt, hogy a kényszereknek /vagy invariánsoknak/ minden időpontban teljesülniük kell. Csak az adatbázis-operációk egymással összefüggésben lévő halmazai között /e halmazok kezdetekor és végekor/ kell teljesülniük. Az operációk egymással összefüggő halmazát tranzakciónak nevezzük, ez az adatabsztrakció operációjának megfelelője. E hasonlóság, és az invariánsok kielégithetőségének nagyobb hatékonyságu bizonyitása alapján, értelmes lenne tokba zárni az adatmodellbeli reprezentációt egy olyan operációhalmazzal együtt, melyek egy adatabsztrakciót definiálnak.

Hogy továbbra is kezelni tudjuk az előre nem látható kérdéseket, a tokba zárásnak nem kéne teljesnek lennie. A kényszerek kielégitettségének biztositásához csak az szükséges, hogy a reprezentáció állapotát megváltoztató összes operációt tokba zárjuk. Lehetséges és helyénvaló megengedni viszont, hogy a lekérdező szabványos adatmodell-operációk közül néhány vagy mind félig átlátszódjon a tokon. Ez alatt azt értjük, hogy egy adatabsztrakció reprezentációjára alkalmazható operációk közül néhány közvetlenül elérhető /exported/, mint az adatabsztrakcióra alkalmazható operáció.

A javasolt adatmodellek bármelyikének választása olyan absztrakciók egy bizonyos rögzitett halmazának kiválasztását jelenti, melyeket azután az összes entitás modellezésére fogunk használni. Ahogy a sor példája is mutatja, adott /absztrakt/ reprezentációs konstruktorok bármely rögzitett halmaza esetén mindig lesznek olyan entitások, amiknek nincs természetes modelljük abban a halmazban. Ezért lenne szükség egy olyan adottságra, hogy tokba zárt adatabsztrakciókat definiálhassunk ezen kivételes helyzetek kezelésére.

Megengedve, hogy a felhasználók a tipusok gyüjteményét egy adatabsztrakciós, tokba záró mechanizmuson keresztül kiterjeszthessék, sok meglévő adatmodellezési megközelités, de legalábbis számitógépes realizációjuk kiterjesztésére kényszerülünk. Ahelyett, hogy a tulajdonságok - azaz a rekordbeli vagy n-esbeli mezők - értékeire rögzitett adattipushalmazunk lenne, az adatmodellnek meg kell engednie, hogy a tartomány-tipusok halmaza nyilt végü legyen. Emiatt kénytelenek vagyunk

- megváltoztatni az alapvető egységek, ugymint a rekordok, az n-esek és a szegmensek tárolását; és
- az indexterületet a tartományok ezen uj fajtáira kiterjeszthetővé tenni.

Ezzel szemben az adatbázis-alkalmazások egy irányzata kételkedik abban, hogy a tipusok elég erőteljesek volnának minden szempont megragadásához. Az idézett ellenpéldák többek között

- a kényszerek,
- a strukturák és operációk bonyolultsága,
- az adat élettartama,

- a megbizhatóság hardver müködési hiba esetén,
- kivétel-kezelés /exception handling/,
- párhuzamos elérések.

Következésképpen a tipusok "szemantikai képességeinek" kiterjesztését kérik. Sokan ugy vélik, hogy a programozási nyelvekbeli tipusfogalmat az adatbázisrendszerekbeli tranzakció fogalmába kell forditani [WED'81] A fogalmak ugyan megfelelnek egymásnak, de nem azonosak.

A tipus-specifikáció és a megvalósitott tipus támogatásának jelenlegi helyzetét tekintve ez a követelés gyakorlati szempontból jogos.

A tipusok adatbázis célokra való felhasználásának fő akadálya az univerzális, formális, átfogó és gyakorlat-orientált tipus-specifikálási technikák jelenlegi hiánya. Az irodalomban a megközelitések gazdag választékát tárgyalták, de mindezideig ezek egyike sem vált szélesen elfogadottá vagy használttá. A széleskörü alkalmazhatósággal biró, átfogó technikák többnyire nem eléggé formálisak a helyesség- és teljességbizonyitások megengedéséhez. Ez a követelmény közben a gyakorlatban is megjelent. Ezzel szemben a formális technikák néha nem eléggé univerzálisak, vagy bonyolult és áttekinthetetlen specifikációkhoz vezetnek. Főként a jelenlegi favoritok, az algebrai technika és a vele összefüggő, átirányitási rendszereken alapuló megközelitések szenvednek a tisztán rekurziv operáció-definiálásoktól. Számos algoritmikusan könnyen kezelhető fogalom válik nehézkesen kezelhetővé; gondoljunk csak a rekordtörlés javaslatára a DBTG hálózatos adatbázisokban.

A kételkedők azonban elméletileg rossz feltételezésből indulnak ki. Ha elfogadjuk, hogy az adatobjektumok a valós

világ valamely részét modellezik, és operációk eredményei, akkor az adattipusok bevezethetők ugy [GO'75] [GU'80] hogy rendelkezzenek a szükséges képességekkel. Például az összes felsorolt "ellenpélda" kifejezhető adat és operációk segitségével, és igy megragadhatók, ha a megfelelő adattipus-specifikációs technikák rendelkezésre állnak.

Az adatmodellek is "dolgok", és mint ilyenek modellezhetők. Azt a megközelitést, hogy az adatbázisokat is tekintsük ugy, mint tipusértékeket, [EH'78] vázolta fel. Ez számos érdekes lényegi problémát vet fel.

- Az adatmodellek paraméterezett absztrakt adattipusokká válnak, "csak" az a kérdés, hogy hogyan specifikáljuk ezeket.
- Az adatbázis sémák normális paraméter-értékadássá válnak, és az a kérdés, hogy milyen szintaktikai eszközökre van szükség megfogalmazásukhoz.
- A kényszerek normális operáció-tulajdonságokká válnak, és az a kérdés, hogy hogyan fogalmazzuk meg az operáció definicióját, hogy átfogó és teljes legyen. Ekkor majd a különböző "alacsonyabb" tipusok - mint a DBTG hálózat modellben lévő halmazok /Set/, területek és rekordok - közti kölcsönviszonyokat érintő különféle kényszerek tisztán és nyiltan azonosithatók lesznek.

Probléma az is, hogy egy adatbázis rendszerben lévő egyszerü adatabsztrakciónak tipikusan több reprezentációja van. Ez az adatfüggetlenség elvének következménye, ami előirja, hogy a felhasználó adatról alkotott nézetének nem szabad változnia azért, mert az alapjául szolgáló

reprezentációt a hozzáférés hatékonyságának javitása céljából megváltoztatták. Ez maga után vonja, hogy az adatmodell-absztrakcióban tökéletesen kifinomult reprezentáció-forditó eszközök csomagjának kell lennie.

A preciz tipus-szerkezetre való hivatkozás másik előnye az, hogy a mindezideig gyenge fogalmak formálisabban hozhatók kapcsolatba, definiálhatók és tanulmányozhatók. Például, ha a "fogalmi modellezést" intellektuális modellépitő, és ezzel kapcsolatos folyamatok /általánositás, aggregáció, osztályozás, stb./ végrehajtásaként értelmezzük, akkor ezeknek a folyamatoknak tipus-szintü alkalmazásaik vannak, a tipusok közti morfizmusok alakjában. E morfizmusok nem lépik tul az eddig emlitett keretet, mivel olyan adattipus operációként kezelhetők, melynek értékei maguk is adattipusok. Ez az elgondolás már [BRO'80] -ban is megjelenik.

Sok még a tennivaló a megfelelő és talán speciális célra orientált technikák kifejlesztésében, melyek együtt, egy erősen tipus-középpontu szemlélet alapján "eszköz-készletet" adnának az adatbázisokhoz.

A tipusnak az a fogalma tehát, ami univerzális modellezési fogalomként megfelelőnek tünik, a tipusokat objektumosztályoknak tekinti, a megfelelő osztály objektumain értelmezett operációkkal együtt. Ez a fogalom második alkotóelemként felöleli a tipusok primitivebb tipusokból való kompoziciójának fogalmát, és következésképp az objektum- és operációkompoziciók fogalmait. Ez a tipusfogalom bármely modellezést támogat, mivel kötelezővé teszi a megfigyelt realitásbeli dolgok

- osztályozását,
- változékonyságuk elemzését, és
- strukturájuk elemzését a sajátos modelltől függetlenül.

E jellemzők miatt a tipusfogalom a következőképpen támogatja a modellezést:

- Egy adatmodellnek megfelelő reprezentáció:

  Az adatmodellt, például a relációs adatmodellt paraméterezett tipusnak tekinthetjük. Ekkor egy relációs adatbázisbeli összes reláció "reláció" tipusunak tekinthető.

- Adatstrukturák reprezentációja:

  Az adat strukturáló modelljei, mint az aggregáció és az általánositás, tipuskompoziciókként ábrázolhatók.

- Dinamikus adat-tulajdonságok reprezentációja:

  Egy tipushoz társitott operációk, és néhány alkalmazhatósági kényszer az adat dinamikus tulajdonságainak reprezentációjaként tekinthető.

Az adatbázis rendszerek modellezésében a fenti tipusfogalom három absztrakció-tipust támogat:

- a realitásbeli dolgok modellezésekor egy adott modell jegyében történő absztrakciót,
- egy adott reprezentációnak más reprezentációval való modellezésekor egy másik modell jegyében történő absztrakciót, végül
- egy adott reprezentációnak egy matematikailag formális metajelöléssel való modellezésekor végrehajtott absztrakciót.

6. ADATTIPUSOK A MESTERSÉGES INTELLIGENCIA-RENDSZEREKBEN

Az aktuális számitási folyamat sikján a programozási nyelvekkel kapcsolatos tipikus témákról gondolkozunk. E sik felett, lenézve erre a sikre olyan kérdéseket tehetünk fel, mint "Van ebben a sikban olyan objektum, ami részt vesz bizonyos kapcsolatban?". Ez egyfajta adatbázis szempont. A mesterséges intelligencia szempontja valahol e rendszer egészén kivül helyezkedik el, mert a rendszerbeli tipusok, és a rendszerre vonatkozó kényszerek felismerésével foglalkozik.

A mesterséges intelligenciával foglalkozó kutatókat felismerési problémák foglalkoztatják. A hagyományos adatbázis-alkalmazásokban amikor egy n-esünk van, a tartományok tipusai jól-definiáltak, és közvetlenül a tipusleirásból ismertek. Ugyanez igaz a programozási nyelvek világában is. A felismerési probléma abból áll, hogy egy objektum tipusát az objektumról felgyülemlett bizonyitékokból kisérelјük meg azonositani. Ezzel kapcsolatban további kérdések merülnek fel:

- az alapértelmezés /default information/ tárolása,
- az adatoperációkra előirt kényszerek ábrázolása, és a
- dinamikus ujraosztályozás problémája.

Ez utóbbi esetben elég információt kaptunk arra, hogy egy objektumhoz egy másik tipust társitsunk. Lehet egymásbaskatulyázott leiráshalmazunk is.

Ez eltér a programozási nyelvek konstruktiv szemléletétől, miszerint az objektumok logikai entitások

halmazából jönnek létre operátorok segitségével. A programozási nyelvekben ezért az objektumokon értelmezett operációknak ismerniük kell, hogyan vannak szerkesztve az objektumok. A következtetés hatékonyságának javitása és az objektumok szerkesztésének ismerete két merőben eltérő tipus hierarchia-fogalomhoz vezet. Például ha van egy axiomatikus számelméletünk, akkor semmi sem szól az ellen, hogy az egész számokat a valós számok részhalmazának tekintsük. Ha azonban az egész és valós számok meg vannak szerkesztve, mint például az ALGOL-ban vagy az axiomatikus halmazelméletben, akkor különféle entitások.

Nézzünk egy példát. Legyen a feladat a mesterséges égitestek mozgásának szimulálása.

Egy olyan funkcionális programozási nyelvben, mint a LISP, egy függvény-gyüjteményünk lenne a mozgatáshoz és gyorsitáshoz. Egy szimulációs programozási nyelvben, mint a SIMULA, létrehoznánk egy adattipust a mesterséges égitestekre, és társithatnánk azokkal a sajátos viselkedésekkel, melyekkel akarjuk, hogy rendelkezzenek.

A SMALLTALK-ban ezt a szimulációt egy un. osztállyal /absztrakt adattipussal/ implementálnánk. A tényleges mesterséges égitestek az osztály előfordulásai /instance/ lennének. Lehetne egy sajátos előfordulásunk, amit "SZALJUT"-nak hivnak. Ennek lenne egy sajátos állapota, bizonyos pozíciója, bizonyos sebessége, gyorsulása, stb. A mesterséges égitestek osztály minden előfordulása esetén a mesterséges égitest nevezi meg az állapotváltozókat és definiálja a módszert. A módszer valójában egy eljárás, ami például kidolgoz egy uj helyzetet, sebességet, stb. Az itteni helyzet-,

sebesség- és gyorsulás-függvények megfelelnek az absztrakt adattipusbeli operációknak. Minden osztály egész csomó módszert támogathat. A belső megvalósitások más módszerekkel is definiálhatnak módszereket, és fizikai ismereteink alapján kitalálható, hogy az objektum helyzetének, sebességének és gyorsulásának primitiv függvényei nem függetlenek.

Az ürben repülő mesterséges égitesteket kezelhetjük olyan adatbázisként is, melytől megkérdezhető, hogy "Melyek az adott szektorban lévő mesterséges égitestek". A SMALLTALK-ban ennek eléréséhez másfajta módszerek kellenek, melyek megvizsgálják az osztályt. Mostantól már nem kivánatos, hogy egy osztály megvalósitsa az adattipus összes viselkedését, de hasznos, hogy megvan az osztály szó, mert itt kapjuk meg az adatmodell néhány viselkedését. A mesterséges égitest viselkedése magában foglal egy függvényt annak eldöntésére, vajon a helyzet egy bizonyos szektorba esik-e. A különböző kérdésekhez különbözőféle támogatásokra gondolhatunk. Az osztálynak rendeznie kell előfordulásait, ha sok van belőlük. Egy absztrakt adattipus általában nincs adatbázisszerüen szervezve, előfordulásaira mutató indexekkel. Egy adatmodell viszont igy van szervezve, és hogy feleletet kapjunk a kérdésekre, különféle szervezési elveket kell bevezetnünk. A jelenlegi SMALLTALK-ban az előfordulások száma kicsi, igy tipikusan lineáris keresést végeznek.

Fontos kiemelni, hogy a SMALLTALK-ban nincs általános rendszer-információs nyelv. Minden lokálisan definiált, és ez igaz az absztrakt adattipusra is. Ha van olyan következetesség, ami kiterjed az egész rendszerre, akkor az megegyezés következménye. Mivel a SMALLTALK világa hierarchikus, ha van egy objektum nevü adattipusunk, akkor az objektumokkal kinált viselkedés a rendszerben minden objektumra vonatkozik, feltéve, hogy egy részosztállyal nem irjuk felül.

I R O D A L O M

BOR'81 Borgida,A.T.
Data and Activities: Exploiting Hierarchies of Classes Proceedings of the Workshop on Data Abstraction,
Databases and Conceptual Modelling,
Pingree Park, Colorado, June 23-26, 1980
SIGPLAN Notices, Vol. 16, No. 1, January 1981, pp. 98-100

BR'80 Brachman,R.J., Smith, B.C.
Special Issue on Knowledge Representation
SIGART Newsletter, No.90, February 1980

BR'83a Brachman,R.J.
What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks
Computer, Vol. 16, No.10, October 1983, pp.30-36

BR'83b Brachman, R.J., Fikes, R.E., Levesque, H.J.
Krypton: A Functional Approach to Knowledge Representation
Computer, Vol.16, No.10, October 1983, pp.67-73

BRO'78 Brodie,M.L.
The Application of Data Types to Databases
IFI-HH-B-51/78, Fachbereich für Informatik
Universität Hamburg, Dezember 1978

BRO'80 Brodie,M.L.
The Application of Data Types to Database Semantic Integrity
Information Systems, Vol.5, No.4, 1980, pp.287-296

BRO'81 Brodie,M.L.
Data Abstraction for Designing Database-Intensive Applications
Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling,
Pingree Park, Colorado, June 23-26, 1980
SIGPLAN Notices, Vol.16, No.1, January 1981,
pp. 101-103

BRO'81a Brodie,M.L.
Association: A Database Abstraction for Semantic Modelling
In: Entity-Relationship Approach to Information Modelling and Analysis, P.P.Chen ed., ER Institute,
Los Angeles, Oct. 1981

BRO'82 Brodie,M.L.
Axiomatic Definitions for Data Model Semantics
Information Systems, Vol.7, No.2, 1982, pp.183-197

BRO'84 Brodie,M.L., Mylopoulos, J.,Schmidt, J.W., eds.
Conceptual Modelling
Springer-Verlag New York, 1984

CA'81 Carbonall,J.G.
Default Reasoning and Inheritance Mechanisms on Type Hierarchies
Proceedings of the Workshop on Data Abstraction, databases and Conceptual Modelling,
Pingree park, Colorado, June 23-26, 1980
SIGPLAN Notices, Vol.16, No.1, January 1981, pp.107-109.

CHE'76 Chen,P.
The Entity-Relationship Model - Toward a Unified View of Data
ACM Transactions on Database Systems, Vol.1, No.1, March 1976, pp. 9-36

CO'70 Codd,E.F.
A Relational Model of Data for Large Shared Data Bases
CACM, Vol.13, No.6, June 1970, pp.377-387

CO'79 Codd,E.F.
Extending the Database RElational Model to Capture More Meaning
ACM Transactions on Database Systems, Vol.4, No.4, December 1979, pp.397-434

DAH'70 Dahl,O.J., Myhrhaug, B., Nygaard,K.
SIMULA 67 Common Base Language
Publication S-22 Oct. 1970, Norwegian Computing Center, Forskningsvien 1B Oslo 3, Norway.

DEM'82 Demetrivics,J.,Knuth,E.,Radó,P.
Specification Meta Systems
IEEE Computer, Vol.15, No.5, 1982, pp.29-35

DEM'85 Demetrovics,J.,Knuth,E.
A Knowledge Based Interactive Way to Systems Design
MTA SZTAKI Working Paper, 85 julius II/69

EH'78 Ehrig,H.,Krekowski,H.J.,Weber,H.
Algebraic Specification Shemes for Database Systems Proceedings of the Fourth International Conference on Very Large Databases, West Berlin, September 13-15, 1978, pp.427-440

FAR'85 Farmer, D.B.,King,R.,Myers,D.A.
The Semantic Database Constructor
IEEE Transactionson Software Engineering,
Vol.SE-11, No.7, July 1985, pp.583-591

GANG'82 Gangopadhyay,D.,Dayal,U.,Browne,J.C.
Semantics of Network Data Manipulation Languages:
An Object-Oriented Approac
Proceedings of the Eighth International Conference
on Very Large Databases, Mexico City, Mexico,
8-10 September, 1982, pp.357-369

GANN'79 Gannon,J.D.,Rosenberg,J.
Implementing Data Abstraction Features in a
Stack-based Language
Software-Practice and Experience, Vol.9,1979,
pp. 547-560

GO'75 Goguen,J.A.,Thatcher,J.W.,Wagner,E.G.,Wright,J.B.
Abstract Data Types as Initial Algebras and
Correctness of Data Representations
Proceedings of the Conference on Comp. Graphics
and Pattern Recognition and Data Structures,
pp. 89-93

GO'77 Goguen,J.A.
Abstract Errors for Abstract Data Types
Semantics and Theory of Computation Report 6,
UCLA, and in: Formal Description of Programming
Concepts, E.Neuhold, ed., North-Holland Pub.
Co., 1978

GO'77a Goguen,J.A.
Algebraic Specification
Semantics and Theory of Computation Report 9, UCLA

GU'77 Guttag,J.V.
Abstract Data Types and the Development of Data Structures
CACM, Vol.20, No.6, June 1977, pp.396-404

GU'78a Guttag,J.V.,Horning,J.J.
The Algebraic Specification of Abstract Data Types
Acta Informatica, Vol.10, 1978, pp.27-52

GU'78b Guttag,J.V.Horowitz,E.,Musser,D.R.
Abstract Data Types and Software Validation
CACM, Vol.21, No.12, December 1978, pp.1048-1064

GU'80 Guttag,J.V.
Notes on Type Abstraction /Version 2/
IEEE Transactions on Software Engineering,
Vol. SE-6, No.1, January 1980, pp. 13-23

HAY'81 Hayes,P.J.,Hendrix,G.G.
A Logical View of Types
Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling,
Pingree Park, Colorado, June 23-26, 1980,
SIGPLAN Notices, Vol.16, No.1, January 1981,
pp.128-130

HAM'75 Hammer,M.,McLeod,D.J.
Semantic Integrity in a Relational Data Base System
Proceedings of International Conference on Very Large Data Bases, Framingham MA, 22-24 September, 1975

HAM'76 Hammer,M.
Data Abstraction for Data Bases
Proceedings of Conference on Data: Abstraction, Definition and Structure,
ACM SIGPLAN Notices, Vol.8, No.2, 1976, pp.58-59

HAM'80 Hammer,M.,Berkowitz,B.
DIAL: A Programming Language for Data-Intensive Applications
Proceedings of the ACM SIGMOND International Conference on the Management of Data, Los Angeles CA, May 1980

HAM'81 Hammer,M.,McLeod,D.
Database Description with SDM: A Semantic Database Model
ACM Transactions on Database Systems, Vol.6, No.3, September 1981

HAR'81 Hardgrave,W.T.,Deutsch,D.R.
Processing Data Model Abstractions
Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling,
Pingree Park, Colorado, June 23-26, 1980
SIGPLAN Notices, Vol.16, No.1, January 1981, pp. 126-127

HO'69 Hoare,C.A.R.
An Axiomatic Basis for Computer Programming
CACM, Vol.12, No.10, October 1969, pp.576-580

HO'72 Hoare,C.A.R.
Proofs of Correctness of Data Representations
Acta Informatica, Vol.1, No.1, 1972, pp.271-281

HO'72a Hoare,C.A.R.
Notes on Data Structuring
APIC Studies in Data Processing No.8,
Academic Press, New York 1972.

JO'76 Jones,A.K.,Liskov,B.
A Language Extension for Controlling Access to Shared Data
IEEE Transactions on Software Engineering,
Vol SE-2, No.12, December 1976, pp.277-285

KR'81 Kreps,P.
Relativism and Views in a Conceptual Data Base Model,
Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling,
Pingree Park, Colorado, June 23-26, 1980
SIGPLAN Notices, Vol.16, No.1, January 1981,
pp. 141-143

LAC'81 Lacroix,M.
Associating Types with Domains of Relational Data Bases
Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling,
Pingree Park, Colorado, June 23-26, 1980
SIGPLAN Notices, Vol.16, No.1, January 1981,
pp. 144-146

LAM'77 Lampson,B.W.,Horning,J.J.,London,R.L.,Mitchell,J.G., Popek,G.J.
Revised Report on the Programming Language Euclid
SIGPLAN Notices, Vol.12, No.2, February 1977

LEA'81 Leavenworth,B.
A Data Abstraction Approach to Database Modelling
Proceedings of the Workshop on Data Abstraction,
Databases and Conceptual Modelling,
Pingree Park, Colorado, June 23-26, 1980
SIGPLAN Notices, Vol.16, No.1, January 1981,
pp. 147-149

LI'74 Liskov,B.,Zilles,S.
Programming with Abstract Data Types
SIGPLAN Notices, Vol.9, No.4, April 1974,
pp.50-59

LI'77 Liskov,B.,Snyder,A.,Atkinson,R.,Schaffert,C.
Abstraction Mechanisms in CLU
CACM, Vol.20, No.7, August 1977, pp.564-576

LI'78 Liskov,B.,Moss,E.,Schaffert,C.,Scheifler,B.,
Snyder,A.
CLU Reference Manual
Massachusetts Institute of Technology,
Laboratory for Computer Science, Computation
Structures Group, Memo 161, July 1978

LO'78 London,R.L.,Guttag,J.V.,Horning,J.J.,Lampson,B.W.,
Mitchell,J.G.,Popek,G.J.
Proog Rules for the Programming Language Euclid
Acta Informatica, Vol.10, 1978, pp.1-26

MARK'81 Mark,W.
Use of Database Organization in the Consul System
Proceedings of the Workshop on Data Abstraction,
Databases and Conceptual Modelling,
Pingree Park, Colorado, June 23-26, 1980
SIGPLAN Notices, Vol.16, No.1, January 1981,
pp.155-157

MARKO'84 Markowitz,V.M.,Raz,Y.
An Entity-Relationship Algebra and Its Semantic Description Capabilities
Journal of Systems and Software, Vol.4, No.2, 3, July 1984, pp. 147-162

MAY'81 Mayr,H.C.
Make More of Data Types
Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling, Pingree Park, Colorado, June 23-26, 1980, SIGPLAN Notices, Vol.16, No.1, January 1981, pp. 158-160

ME'81 Meyer,B.
A Three-Level Approach to the Description of Data Structures, and Notational Framework
Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling, Pingree Park, Colorado, June 23-26, 1980, SIGPLAN Notices, Vol.16, No.1, January 1981, pp. 164-166

MC'83 McCalla,G.,Cercone,N.
Guest Editors' Introduction: Approaches to Knowledge Representation
Computer, Vol.16, No.10, October 1983, pp.12-18

ML'81 McLeod,D.,Smith,J.M.
Abstraction in Databases
Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling, Pingree Park, Colorado, June 23-26, 1980, SIGPLAN Notices, Vol.16, No.1, January 1981, pp.19-25

NÁ'78 Náray,M.
Az ALPHARD nyelv lehetőségeinek áttekintése
SZKI, Elméleti Laboratórium, Budapest, 1978
szeptember

NE'84 Neisser,U.
Megismerés és valóság
Gondolat Kiadó, Budapest, 1984

NO'76 Nordström,B.
An Outline of a Mathematical Model for the
Definition and Manipulation of Data
Proceedings of Conference on Data: Abstraction,
Definition and Structure,
ACM SIGPLAN Notices, Vol.8, No.2, 1976, pp. 1-11

PAL'78 Palmer,I.
Record Subtype Facilities in Database Systems
Proceedings of the Fourth International Conference
on Very Large Databases, West Berlin, 13-15 September,
1978

PAO'81 Paolini,P.
Abstract Data Types and Data Bases
Proceedings of the Workshop on Data Abstraction,
Databases and Conceptual Modelling,
Pingree Park, Colorado, June 23-26, 1980
SIGPLAN Notices, Vol.16, No.1, January 1981,
pp. 171-173

SCMI'77 Schmidt, J.W.
Some High Level Language Constructs for Data of
Type Relation
ACM Transactions on Database Systems, Vol.2,
No.3, September 1977, pp.247-261

SCMI'78 Schmidt,J.W.
Type Concepts for Database Definition
Databases: improving usability and responsiveness,
B.Shneidermen ed., Academic Press, 1978,
pp.215-244

SCMI'81 Schmidt,J.W.
Data Abstraction Tools: Design, Specification
and Application
Proceedings of the Workshop on Data Abstraction,
Databases and Conceptual Modelling
Pingree Park, Colorado, June 23-26, 1980
SIGPLAN Notices, Vol.16, No.1, January 1981,
pp.186-188

SHA'81 Shaw,M.
Abstraction, Data Types and Models for Software
Proceedings of the Workshop on Data Abstraction,
Databases and Conceptual Modelling,
Pingree Park, Colorado, June 23-26, 1980,
SIGPLAN Notices, Vol.16, No.1, January 1981,
pp.189-191

SHI'81 Shipman,D.
The Functional Data Model and the Data Language
DAPLEX ACM Transactions on Database Systems,
Vol.6, No.1, March 1981

SM'77a Smith,J.M.,Smith,D.C.P.
Database Abstraction: Aggregation
CACM, Vol.20, No.6, June 1977, pp.405-413

SM'77b Smith,J.M., Smith,D.C.P.
Database Abstractions: Aggregation and Generalization ACM Transactions on Database Systems, Vol.2, No.2, June 1977, pp.105-133

TH'78 Thatcher,J.W.,Wagner,E.G.,Wright,J.B.
Data Type Specification: Parameterization and the Power of Specification Techniques.
Proceedings of the SIGACT Tenth Annual Symposium on Theory of Computing, May 1978, pp.119-132

WA'81 Wasserman,A.I.
The Extension of Data Abstraction to Database Management
Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling,
Pingree Park, Colorado, June 23-26, 1980,
SIGPLAN Notices, Vol.16, No.1, January 1981,
pp. 198-200

WEB'81 Weber,H.
Are Data Types Universal Modelling Concepts for Data Base Systems?
Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling,
Pingree Park, Colorado, June 23-26, 1980,
SIGPLAN Notices, Vol.16, No.1, January 1981,
pp. 201-202

WED'81 Wedekind,H.H.
Constructive Abstract Data Types /CAD/
Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling,
Pingree Park, Colorado, June 23-26, 1980,
SIGPLAN Notices, Vol.16, No.1, January 1981,
pp. 203-206

WEL'84 Weller,D.L.,York,B.W.
A Relational Representation of an Abstract Type System
IEEE Transactions on Software Engineering, Vol. SE-10, No.3, May 1984, pp.303-309

WIE'84 Wiederhold,G.
Knowledge and Database Management
IEEE Software, Vol.1, No.1, January 1984, pp. 63-73

ZI'81 Zilles,S.N.
Types, Algebras and Modelling
Proceedings of the Workshop on Data Abstraction, Databases and Conceptual Modelling, Pingree Park, Colorado, June 23-26, 1980, SIGPLAN Notices, Vol.16, No.1, January 1981, pp. 207-209

1986-BAN EDDIG MEGJELENTEK:

179/1986 Terlaky Tamás: Egy véges criss-cross módszer és alkalmazásai

180/1986 K.N. Čimev: Separable sets of arguments of functions

181/1986 Renner Gábor: Kör approximációja a számitógépes geometriai tervezésben

182/1986 Proceedings of the Joint Bulgarian-Hungarian Workshop on "Mathematical Cybernetics and Data Processing" Scientific Station of Sofia University, Giulecica /Bulgaria/, May 6-10, 1985 /Editors: J. Denev, B. Uhrin/ Vol I

183/1986 Proceedings of the Joint Bulgarian-Hungarian Workshop on "Mathematical Cybernetics and Data Processing" Scientific Station of Sofia University, Giulečica /Bulgaria/, May 6-10, 1985 /Editors: J. Denev, B. Uhrin/ Vol II

184/1986 HO THUAN: Contribution to the theory of relational databases

185/1986 Proceedings of the 4th International Meeting of Young Computer Scientists IMICS'86 /Smolenice, 1986/ /Editors: J. Demetrovics, J. Kelemen/

186/1986 PUBLIKÁCIÓK - PUBLICATIONS 1985 Szerkesztette: Petróczy Judit

187/1986 Proceedings of the Winter School on Conceptual modelling /Visegrád, 27-30 January, 1986/ /Editors: E. Knuth, A. Márkus/

188/1986 Lengyel Tamás: A Cluster analizis néhány kombinatorikai és valószinüségszámitási problémája

189/1986 Bernus Péter: Gyártórendszerek funkcionális analizise és szintézise