



M Ű E G Y E T E M 1 7 8 2
Budapesti Műszaki és Gazdaságtudományi Egyetem
Távközlési és Médiainformatikai Tanszék

Összetett szemantika ábrázolása adatbázisokban

Ph.D. tézisfüzet

Írta:
Surányi Gábor

Tudományos vezető:
Dr. Magyar Gábor

Készült:
Budapest, 2009. december 18.

1. BEVEZETÉS

Alapvetően és tradicionálisan az adatbázisok nagy mennyiségű adattal dolgozó, különféle szoftverrendszerek közös adattáráként szolgálnak. Az elmúlt évtizedek technológiai fejlődése azonban nagy hatással volt az adatbázisok szerepére. Itt csak két jelenséget említek meg.

Az objektumorientált (OO) modellezés végre adatbázisokban is elérhető. Ugyanakkor ezek az adatmodellek még mindig el vannak maradva az általánosan használt OO szoftverfejlesztő eszközök képességeitől.

Adatbázisokat építenek mindenféle számítógép-vezérelt eszközbe, mivel a memóriák és háttértárak átlagos kapacitása úgy növekszik, hogy egységáruk csökken. Az elméleti cél az, amelyet a gyakorlatban igyekeznek minél inkább megközelíteni, hogy minden rendelkezésre álló adatot eltárolnak, és azokat mindenféle, adott esetben tervezéskor előre nem látott módon igyekeznek felhasználni a minőség ill. a nyereség növelése érdekében.

Mindkét jelenség új követelményeket támaszt az adatbázisok *ábrázolási képességével* szemben. Egyrészt *gazdagnak* kell lennie, hogy a szoftverfejlesztő eszközök képességeinek megfeleljenek. Másrészt *nyitottnak* kell lennie, hogy olyan előre nem látott modell- vagy adatelemeket is el tudjon tárolni, amelyek nem felelnek meg a tervezésükkor elképzelt modellnek. [22, C7] Ezt követeli meg az (adat)szemantika, amely egyre összetettebbé válik a szoftverrendszerek intelligenciájának növekedésével. Az érem másik oldala a *lekérdezések támogatása*, amelynek ugyancsak kielégítőnek, azaz az ábrázolási képességnek megfelelőnek kell lennie — nem elegendő valamit úgy eltárolni, hogy nem tudjuk lekérdezni.

Az adatok ábrázolási módja az adatbázisok tervezésekor dől el. Az adatbázisok tervezésekor a következő feladatokat hajtják végre:

1. az adatmodell kiválasztása,
2. a séma meghatározása,
3. a fizikai adatszerzés kialakítása.

Ezen lépések mindegyike vizsgálható, miképpen tudja összetett szemantika ábrázolását támogatni.

Jelen dokumentum az angol változat fordítása (kivéve a hivatkozott irodalmak listáit, amelyeket változatlanul tartalmaz). Amennyiben az értelmezéssel kapcsolatos kérdés merül fel, az angol változat az irányadó.

1.1. Fejlett, objektumorientált adatmodellek

Habár a relációs alapokon nyugvó adatbázisok még mindig gyakoriak, új szoftverrendszerekben előszeretettel alkalmaznak OO adatbázisokat. Előnyeik jól ismertek: az OO szemléletmód magasabb absztrakciós szintet jelent, ugyanakkor intuitív. Valamint mivel az új szoftverek szinte kizárólag OO-ak, nincs szakadék az entitások élő (memóriában levő) és tárolt (adatbázisban levő) ábrázolása között.

Létezik szabvány objektumperzisztencia megvalósítására adatbázisokban, az ún. Object Data Standard (legutóbbi verziója [14]), amelyet az Object Data Management Group (ODMG) hozott létre. Ugyanakkor az OO metamodellek¹ egyre gazdagabbá és gazdagabbá válnak, hogy a modellezni kívánt világokat minél pontosabban leírassák. Az egyik elem, amely hiányzik az OO adatmodellekből (beleértve az ODMG objektummodelljét is), az a kényszerek átfogó használata. Az átfogó használat azt jelenti, hogy nem pusztán az integritáshoz szükséges kényszereket kell kezelni. Ki kell terjednie az OO modellezés minden területére, így az analízisre, a tervezésre, a megvalósításra és a tesztelésre csakúgy, mint az az OO szoftverek fejlesztése esetében történik.

Egy, a legtöbb esetben alkalmazható OO metamodel iránti igény hívta életre a Unified Modeling Language Specification (UML)-t[37, 38] és az Object Constraint Language (OCL)-t[31]. Mindkettőt az Object Management Group, Inc. (OMG) jegyzi, és elmondható mindkettőről, hogy minden más, kevésbé elterjedt OO modelle (így az ODMG szabványra is) nagy hatást gyakorolt.

1.2. Ontológiák: nyitott adatbázissémák

Valóban, a már létező adatmodellek képesek nyitott adatbázissémák ábrázolására, mert már korábban felmerült az igény ún. féligstrukturált adatok[9, 1] ábrázolására. De a DBMS-ek nem adnak semmiféle további támogatást a (következő értelemben) összetett szemantika lekérdezésére: minden adatbázisba eltárolandó adatot elemeire (pl. rekordokra) kell bontani, de ha már el vannak tárolva az adatok, szükségszerű, hogy csak ugyanabban az elemi formában lehet őket lekérdezni? A válasz határozottan nem, ennek ellenére a legtöbb (beleértve a féligstrukturált adatokat is kezelő) DBMS (ld. pl. [10, 1]) így tesz.²

Vegyül például az információ-visszakereső (IR) rendszereket, amelyek szenvedő alanyai ennek. Óriási adatbázisokat építenek a (kereső)kifejezések, erőforrások (maguk a dokumentumok) és a közöttük fennálló kapcsolatok kezelésére. A lekérdezés módszere triviálisnak tűnik: azon erőforrások visszaadása, amelyek kapcsolódnak az adott kifejezésekhez. Ugyanakkor az ilyen módon megalkotott találati halmaz valószínűsíthetően nagy és nem tartalmazza az összes, felhasználót érdeklő erőforrást. Más, kifinomult módszerekkel ezen negatívumok persze

¹ Egy metamodel modellek leírásához ad elemeket, amelyekben már lehetőség van entitások konkrét tulajdonságainak leírására. A mindennapi szóhasználatban a metamodelleket gyakran csak modelleknek hívják, hiszen a szöveggörnyezetből egyértelműen kiderül, melyikről is van szó. Én itt követem ezt a konvenciót, amennyiben nem okoz félreértést.

² Valójában meg kell említeni, hogy amióta tárolt eljárások léteznek, lehetőség van kifinomult lekérdezési módszerek DBMS-ben történő megvalósítására. De ezek a tárolt eljárások is azokat az elemi adatokat használják.

kiküszöbölhetők. Ezek közül az ontológiaalapú IR (OBIR) rendszerek manapság a legkedveltebbek.

Az ontológia egy fogalomalkotási formalizmus.[23, C7] OBIR esetében az ontológia elemei a kifejezések. Az ontológia elemeinek (OE-knek) hasonlóságát a közöttük levő különféle kapcsolatok alapján ítélik meg, és ez a hasonlóság az alapja a kifejezésekből álló keresőkérdés megválaszolásának. Azaz egy OBIR rendszer olyan kérdéseket válaszol meg, mint

Mely rekordokat írnak le a megadotthoz hasonló rekordok? (Q_{similar})

Nyilvánvaló, hogy ez összetettebb, mint a DBMS-ek megszokott lekérdező kifejezései, mert a „hasonló” szó nem értelmezett. Az üzleti logika az, amely végülis a lekérdezéseket a DBMS által érthető kifejezésekké alakítja. (A „leír” csupán egy több-több kapcsolat, amelyet minden DBMS megért.)

A nyílt sémákat az egyszerű (azaz logikamentes) ontológiák általánosításaként képzelhetjük el, hiszen mindössze különféle, rekordok közötti kapcsolatokat ábrázolnak. Itt ugyanez a probléma lép fel: nincs beépített eljárás a kapcsolatok lekérdezésekben történő értelmezésére.

1.3. Fejlettebb fizikai adatbázisok

A fizikai adatszerzés az adatelemek tárolóegységeken való elhelyezésével foglalkozik. Egyetlen célja, hogy a hozzáférési kéréseket gyorsan ki tudja szolgálni. Tradicionálisan pontos találatokat szolgáltat, de az utóbbi időben a közelítő válaszok fontossága megnőtt.[22] Például a relációs világon kívül gyakoriak a halmazértékek³, ahol gyakran pontos találat nem várható, de közelítő (a feltételeket leginkább kielégítő) válaszok megfelelőek.

Az elemek közötti logikai távolságot végső soron maguk az elemek határozzák meg. A kihívás ezen a szinten tehát ennek megfelelően az elemek közötti távolság megragadása és ábrázolása a fizikai adatbázisban.

³ A relációs adatbázisok tervezésének általában része a normalizálás, amelynek előfeltétele, hogy minden attribútum elemi legyen. Így a halmazokat gyakran elemeire bontják fel és további relációk segítségével ábrázolják.

2. KUTATÁSI CÉLKITŰZÉSEK

Az értekezés célja az adatok adatbázisokban történő szemantikai ábrázolásának fejlesztése volt. A kapcsolódó nyitott problémákat az egyes adatbázis-tervezési lépések során azonosítottam. Ezek közül a következőkkel foglalkoztam munkám során.

Az UML és az OCL képességeit magába foglaló adatmodellre van szükség. Ez alapvetően azt jelenti, hogy a kényszereket átfogóan támogató OO adatmodellt kell kifejleszteni. Továbbá az adatmodellnek típusrendszerrel kell rendelkeznie. A típusrendszerek alapvető célja, hogy futási időben fellépő hibákat fedjenek fel pusztán a kód analízisa által. Én olyan típusrendszer megalkotásán fáradoztam, amely a hibamentes működést objektuminvariánsok, eljárások és állapotalapú szerek¹ mellett garantálja.

Eljárást kell adni, miképpen lehet időhatékonyan nyílt sémájú adatbázisokat lekérdezni az adatelemek közötti kapcsolatok figyelembevételével. A mesterséges intelligencia már lehetővé tett efféle lekérdezéseket az elemek közötti kapcsolatok megragadásával. Ezek az eljárások rekordpárok közötti egyes kapcsolatokon vagy két rekordhalmaz hasonlóságának megfigyelésén alapulnak. Az utóbbiak természetesen nagyobb rálátással bírnak a teljes szemantikára, és így jobb eredményeket is szolgáltatnak. A hátrányuk, hogy lekérdezés során csak elempárookra alkalmazhatók, ezért nagy léptékben rosszul teljesítenek. A céloom egy módszer kidolgozása volt, amellyel efféle elempárokon alapuló eljárást mégis hatékony lekérdezési eljárásba lehet integrálni. Ez alapján a jövő DBMS-ei is tudnak majd (Q_{similar})-jellegű kérdésekre (ld. 4. oldal) közvetlenül válaszolni.

Olyan fizikai szervezés kialakítására van szükség részbenrendezett halmazok számára, amely hatékonyan támogatja közelítő válaszok és pontos találatok visszaadását lekérdezésekben. A részbenrendezések² nem kaptak eddig sok figyelmet ezen a területen annak ellenére, hogy részbenrendezéseken megfogalmazott lekérdezésekkel gyakori problémák modellezhetők [33].

¹ A szerep kifejezést ebben a munkában az UML 1.5-ben definiált értelmében használom: egy entitás nevesített viselkedése egy adott környezetben [36]. Részletek a 4.1.1. szakaszban.

² A részbenrendezés egy reflexív, tranzitív és antiszimmetrikus bináris reláció. [5]

3. MÓDSZERTAN

Mind a kifejlesztendő adatmodell, mind a típusrendszer matematikai szerkezet, amelyeket meg kellett alkotni. Ennek ellenére igaz, hogy a kerék újbóli feltalálása nem célszerű. Ezért a matematikai szerkezeteim korábbi munkákon alapulnak, úgy mint Fernandes axiomatikus adatmodelljén[18] és Castagna OO kalkulusán, a λ - η [13]. Ez a rész matematikai alapokon nyugszik, amely az elért eredmények analitikus kiértékelését tette lehetővé.

Ezzel szemben az ontológián alapuló hatékony lekérdezési módszer kidolgozása inkább mérnöki feladat. Így főként eredmények adaptálásáról és más területeken való hasznosításáról szól ez a rész. Ez különösen a felhasznált blackboard-rendszer[16, 15] megalkotásánál szembeötlő. Az elért eredmények kiértékelése itt empirikus volt, azaz a célfelhasználók egy csoportja tesztelte az én eljárásomat alkalmazó IR rendszert, és a véleményüket begyűjtöttük.

Egy hatékony fizikai adatszervezés megalkotása bizonyos értelemben a fenti két megközelítés elegye: új dolgok kitalálása és régiak újrahasonosítása, mérnöki ill. tudományos munka. A feladat formalizálásával a problémát matematikai problémára vezettem vissza, és megállapítottam, hogy ott már részletesen elemezték. A kiábrándító általános matematikai eredmények ellenére létezik néhány terület, ahol ilyen fizikai szervezés működik, de általános adatbázisok nincsenek közöttük. Ezért az én fizikai szervezésem tekinthető úgy, mint a korábbi elméleti és gyakorlati eredmények alkalmazása és optimalizálása. Az optimalizálás részben korábbi matematikai eredményeknek és azok számításelméleti alkalmazásainak köszönhető. Valamint a matematikai formalizmus tette lehetővé, hogy eredményeimet kvázirendezésre¹ is kiterjesszem, hiszen bármely kvázirendezésből lehet részbenrendezést készíteni [8].

¹ Az kvázirendezés egy reflexív és tranzitív bináris reláció.[8]

4. ÚJ EREDMÉNYEK

4.1. Kényszerekkel bővített OO modellek

4.1.1. A kényszerekkel bővített axiomatikus OO adatmodell

Létezik már axiomatikus OO adatmodell, amelyet részletesen [18] mutat be. Habár az a modell képes lenne rá, szándékosan nem támogatja az alkalmazáspecifikus, integritást leíró kényszereket, hogy a lekérdezéseket optimalizálni tudja.

Amint azt a bevezetőben írtam, ez a nézőpont már nem tartható. A következőkben megmutatom, hogy alkalmazáspecifikus, integritást leíró kényszerek támogatása elméletileg lehetséges és gyümölcsöző. Ugyanakkor néhány gyakorlati kérdéssel, mint például az eldönthetőség kérdésével nem foglalkozom. Azokat a megfelelő alkalmazási területeken kell megtenni, úgy, ahogy én a 4.1.2. szakaszban teszem.

1. téziscsoport. *Definiáltam egy axiomatikus adatmodellt, amely támogatja az alkalmazáspecifikus integritást leíró kényszereket [J1, C6].*

1. definíció (adatmodellem, adatbázisséma, adatbázis és lekérdezés a modellben). *Legyen \mathcal{L} egy logikai nyelv a következő elemekkel:*

- *változók végtelen halmaza,*
- *konstans- (nulla aritású függvény-) szimbólumok halmaza, amely az osztály- és objektumazonosítókat valamint az elemi konstansokat tartalmazza,*
- *predikátumszimbólumok egy halmaza \mathcal{P} ,*
- *nemkonstans függvénytípusok halmaza \mathcal{F} ,*
- *a zárójelek (és),*
- *a logikai összekötőjelek $\neg, \wedge, \vee, \Rightarrow$,*
- *a kvantorok \forall és \exists .*

A \mathcal{P} halmaz elemei:

- *unáris szimbólumok az egyes atomi típusok számára (pl. integer, string),*
- *az atomi típusokhoz szükséges elemi predikátum- és relációs szimbólumok (pl. =, >),*
- *az unáris szimbólumok class, object,*

- a bináris szimbólumok *specialize*, *instance*,
- bináris szimbólumok minden egyes attribútumnévhez (pl. *name*),
- $(n+1)$ aritású vagy $(n+2)$ aritású szimbólumok minden egyes n argumentumú eljárás nevével.

Az elsőrendű logika (FOL) bármely fentiekben leírt \mathcal{L} nyelvvel az **axiomatikus OO adatmodell, amely támogat alkalmazásspecifikus kényszereket is.**

Legyen \mathcal{A} a következő formulák halmaza:

$$\forall c \text{ class}(c) \Rightarrow \neg \text{object}(c) \quad (4.1)$$

$$\forall o \text{ object}(o) \Rightarrow \neg \text{class}(o) \quad (4.2)$$

$$\forall c_1 \forall c_2 \text{ specialize}(c_1, c_2) \Rightarrow \text{class}(c_1) \wedge \text{class}(c_2) \quad (4.3)$$

$$\forall c \forall o \text{ instance}(c, o) \Rightarrow \text{class}(c) \wedge \text{object}(o) \quad (4.4)$$

$$\forall \text{ class}(c) \Rightarrow \text{specialize}(c, c) \quad (4.5)$$

$$\forall \text{ specialize}(c_1, c_2) \wedge \text{specialize}(c_2, c_1) \Rightarrow c_1 = c_2 \quad (4.6)$$

$$\forall \text{ specialize}(c_1, c_2) \wedge \text{specialize}(c_2, c_3) \Rightarrow \text{specialize}(c_1, c_3) \quad (4.7)$$

$$\forall o \exists c \text{ object}(o) \wedge \text{instance}(c, o) \quad (4.8)$$

$$\forall c_1 \forall c_2 \forall o \text{ specialize}(c_1, c_2) \wedge \text{instance}(c_1, o) \Rightarrow \text{instance}(c_2, o) \quad (4.9)$$

A Π , \mathcal{L} feletti zárt formulákból álló halmaz egy **adatbázis séma**, ha konzisztens és $\Pi \vdash \mathcal{A}$. Az \mathcal{S} struktúra, amely megfelel \mathcal{L} -nek, egy **adatbázis**, ha $\mathcal{S} \models \Pi$.¹ Bármely \mathcal{L} feletti φ zárt formula egy **lekérdezés**. Lekérdezéseknél jelezni kell, hogy min kell kiértékelni:

- $\mathcal{S} \stackrel{?}{\models} \varphi$, azaz a lekérdezést kifejező formula pillanatnyilag igaz-e az adatbázisban vagy
- $\Pi \stackrel{?}{\vdash} \varphi$, azaz a lekérdezést kifejező formula az adatbázis minden állapotában igaz-e.

Az előbbi esetben a DBMS egy változóbehelyettesítést kell, hogy visszaadjon azon egzisztenciálisan kvantifikált változók esetében, amelyeket nem előz meg univerzális kvantor a lekérdezés prenex normálformájában.

1.1. tézis. Megmutattam, hogy az 1. definícióban bemutatott adatmodell tényleg OO, azaz minden alapvető OO fogalomnak megvan a megfelelője: osztály, objektum(példány), öröklődés, polimorfizmus.

A bevezetőben leírtaknak megfelelően ezt a tézist az bizonyítja, hogy a modell kompatibilis az UML-konstrukciókkal[37, 38]. Habár az UML maga is definiál megfeleléségi szinteket, inkább kompatibilitásról semmint megfeleléségről beszélek, mert még a legkisebb megfeleléségi szinthez is az UML Basic package minden

¹ Innen ered az adatmodell *axiomatikus* jelzője: az adatbázisséma olyan állításokat tartalmaz, amelyet a konkrét adatoknak ki kell elégíteniük.

elemét le kellene képezni egy megfelelő modellbe. De én csak az alapvető OO fogalmak meglétét vizsgálom, úgy mint osztály, objektum(példány), öröklődés, egységbezárás, polimorfizmus. A Basic package minden elemének leképezése a modellt feleslegesen elbonyolítaná. A kompatibilitás elegendő ahhoz, hogy a modellem UML-megfelelővé bővíthető legyen.

2. definíció (Osztály és objektum(példány)[38]). Az osztály *olyan* objektum-(példány)ok halmazát írja le, amelyek azonos jellemzőkkel, kényszerekkel és szemantikával rendelkeznek. Az osztály egy csoportosító (*classifier*), amelynek jellemzői az attribútumok és az operációk. [...] Az attribútumok bináris asszociációk végződésai is lehetnek.

Az eljárás egy operáció megvalósítása.

A modellemben az objektumok entitások, amelyeket konstansok azonosítanak. Csak ezekre az o konstansokra igaz $object(o)$. Az objektum(példány)oknak különféle jellemzői vannak úgymint

- attribútumok, amelyeket a megfelelő bináris predikátumok adnak meg, pl. $name(object, string)$,
- bináris asszociációk, amelyek az attribútumokhoz hasonlóan vannak ábrázolva,
- n aritású operációk, amelyeknek $(n+1)$ ill. $(n+2)$ aritású predikátumok felelnek meg. Az első argumentum a tulajdonos számára (amelynek környezetében az operáció végrehajtandó), a második argumentum, ha van, a visszatérési érték számára fenntartott. Például $bear(parent_object, integer, child_object)$, amely jelentheti azt, hogy az operáció a gyermekobjektumok új számosságát adja vissza.

Az osztályokat mint modellentitásokat is konstansok azonosítják. Ezekre a c konstansokra $class(c)$ igaz, továbbá különböznek az objektumoktól: (4.1)-(4.2). Azt a tényt, hogy az objektumok osztályokhoz tartoznak, a (4.8) axióma írja le az $instance(c, o)$ predikátum használatával. A predikátumok argumentumainak lehetséges értékeit (4.4) adja meg. Hogy egy osztály minden példánya rendelkezik egy adott képességgel, a következőképp formalizálható:

$$\forall o \forall a_1 \dots \forall a_n \exists r \quad instance(c_o, o) \wedge instance(c_1, a_1) \wedge \dots \wedge instance(c_n, a_n) \Rightarrow \text{FEATURE}(o, r, a_1, \dots, a_n) \wedge instance(c_r, r) \quad (4.10)$$

ahol FEATURE a képesség predikátumszimbóluma, a_1, \dots, a_n csak akkor van jelen, ha a képesség egy operációnak felel meg, nem pedig attribútumnak vagy asszociációnak. Amint már említettem, r elhagyható azon operációk esetében, amelyeknek nincs visszatérési értéke. Egy (4.10) alakú formula a képességet a c_o -val jelölt osztályhoz rendeli.

A fenti formuláknak nem mond ellent, hogy egy példány olyan képességekkel rendelkezzen, amelyeket nem az osztálya(i) ír(nak) elő. Ennek megakadályozására a következő alakú formula használható az adatbázissémában:

$$\forall o \forall a_1 \dots \forall a_n \forall r \quad \text{FEATURE}(o, r, a_1, \dots, a_n) \Rightarrow instance(c_o, o) \wedge instance(c_1, a_1) \wedge \dots \wedge instance(c_n, a_n) \wedge instance(c_r, r). \quad (4.11)$$

A *kényszerek* olyan egyéb² tetszőleges adatbázissémabeli formulák. Például egy kényszer megadhat a konkrét alkalmazásra jellemző, példányokra vonatkozó invarianciafeltételt, operációk elő- és utófeltételeit. De ezeknek a feltételeknek mind FOL-ban leírhatónak kell lenniük.

Kényszerek adják meg az eljárásokat is, amelyek tradicionálisan univerzálisan lezárt implikáció formáját öltik [27] (tehát olyan kényszereknek tekinthetők, amely a bemenet és a kimenet összefüggését írják le):

$$\text{BODY} \Rightarrow \text{OPERATION}(o, r, a_1, \dots, a_n) \quad (4.12)$$

Bármely kényszer c -vel jelölt osztályhoz történő rendelése akkor lehetséges

- ha a formula nem tartalmaz más osztályazonosítót csak c -t és
- minden predikátumszimbólum, amely képességnek felel meg, c -hez van rendelve.

3. definíció (Öröklődés[38]). *Az öröklődés egy taxonómiai viszony egy általánosabb és egy specifikusabb csoportosító között. A specifikusabb csoportosító példányai mind-mind indirekt példányai az általánosabb csoportosítónak. Azaz a specifikusabb csoportosító mintegy örökli az általánosabb csoportosító képességét.*

A modellemben az öröklődésnek a *specialize* bináris predikátum felel meg: (4.3). Az öröklődés megszokott (részbenrendezés-jellegű) tulajdonságait a (4.5)–(4.7) formulák írják le. Az a tény, hogy egy objektum egy általánosabb osztály példánya is egyben, a (4.9) formula írja le. Ily módon, amikor egy általánosabb osztály képességeire hivatkozunk (egy formulában például), egyúttal az is biztosítva van, hogy a specifikusabb osztályok azonos képességeire is hivatkozunk: a képesség öröklődik.

4. definíció (Polimorfizmus[12]). *Polimorf operációk operandusai (tényleges paraméterei) többféle típusúak lehetnek.*

Kétféle polimorfizmus létezik: univerzális és eseti.[12] A gyakorlatban mindkettő fontos, de mivel az eseti csak egy jelölésrövidítési technika véges számú típusra [12], én itt csak az univerzális polimorfizmussal foglalkozom.

Az univerzális polimorfizmus lehet tartalmazási jellegű és parametrikus.[12] Az osztályokra vonatkozó tartalmazási jellegű polimorfizmussal az öröklődésnél már foglalkoztam, és megmutattam, hogy az axiomatikus OO adatmodellem támogatja.

Az univerzális jellegéből adódóan a parametrikus polimorfizmus végtelen számú, de azonos struktúrájú típusal operál.[12] A parametrikus polimorfizmust általában a következő módok egyikén valósítják meg [12]:

- sablonokkal, amelyeket használat során explicit kell alkalmazni konkrét típusokra mint az UML-ben[38] és a C++[11] programozási nyelvben,
- generikus konstrukciókkal, amelyek bármely, egy adott követelményhalmazt kielégítő entitásra alkalmazhatók. Ezt tipikusan funkcionális programozási nyelvek (pl. az ML[29]) alkalmazzák, de az UML is támogatja ezeket az ún. stereotypes[38] típusokkal.

² azaz (4.1)–(4.9)-en felüli és nem (4.10), (4.11) alakú

Az én adatmodellem az egyszerűség és az általánosság miatt az utóbbit támogatja. Csakúgy, mint UML-ben, az osztályfogalom elég rugalmas ahhoz, hogy magába foglaljon egy efféle típusfogalmat is.

Mivel az adatbázisok tradicionálisan hosszú életűek, jó, ha adatmodelljük még egy fogalmat támogat: a szerepeket [21, 19, 36]. [32] A szerepeket kiterjedten alkalmazzák az OO analízis és tervezés során. Sajnos azonban a terminológia nem egységes, még az UML 1.5 és 2.0 is különböző névvel illeti ezt a fogalmat (ld. *ClassifierRole* ill. *ConnectableElement* definícióját [36]-ben ill. [38]-ben).

A terminológiai egység hiánya részben abból ered, hogy kétféle módon lehet szerepeket ábrázolni: *explicit* és *implicit* módon. Explicit ábrázolások során, hogy egy objektum adott szereppel bír, egy dinamikus objektum vagy egy szerepobjektum reprezentálja, amely szükség szerint jön létre és semmisül meg. Szokásos elnevezések még a dinamikus osztályok (ld. pl. [28]) és szereptípusok (ld. pl. [21]).

A szerepek implicit ábrázolása során a szerepet dinamikusán veszi fel az objektum képességei és állapota alapján, nincs szükség további objektumokra ehhez. Innen ered ezen ábrázolásmód másik neve is: állapotalapú szerepek. Ugyanakkor előfordul még a virtuális osztály (ld. pl. [34]), egyszerűen a típus (ld. pl. [C6], ill. *ConnectableElement* [38]-ben) elnevezés is. A típus kifejezés használatát a szerep fogalmára az indokolja, hogy egy szerep valójában nem más, mint követelmények halmaza, amelyeket a példányoknak ki kell elégíteniük. Minthogy egy attribútum megfelelhet annak, hogy egy példány bír-e adott szereppel, az implicit szerepábrázolás magába foglalja az explicitet.

Világos azonban, hogy alapvető különbség van az interfész és az (implicit) szerep fogalmai között. Mindkettő tekinthető kielégítendő követelményhalmaznak, de az interfészeket osztályok és az által objektumok elégítik ki, míg a szerepeket bármely osztály megfelelő példányai veszik fel, amennyiben megfelelnek a követelményeknek.

1.2. tézis. *Megmutattam, hogy az 1. definícióban bemutatott adatmodell osztályfogalma támogatja az implicit szerepeket.*

Ez olyan osztályokkal lehetséges, amelyeket

$$\forall o \text{ CONDITION} \Rightarrow \text{instance}(c_o, o) \quad (4.13)$$

alakú formulák írnak le. CONDITION hivatkozhat képességekre, létezésükre, konkrét értékükre stb.

4.1.2. Részleges helyesség bizonyítása kényszereket is tartalmazó OO modellekben

A biztonságot szem előtt tartó tervezés és megvalósítás immár nem csak kritikus szoftverek sajátja. Kényszerek, amelyek invarianciafeltételeket vagy üzenet elő- és utófeltételeit írják le, jó szolgálatot tesznek a tervezési és megvalósítási fázis során vétett hibák megelőzése ill. javítása során.

2. téziscsoport. *Definiáltam két függvénykalkulust, $\lambda\&\mathcal{C}$ -t és $\lambda\&\mathcal{C}'$ -t, amelyekkel bebizonyítható bármely OO program részleges helyessége figyelembe véve az*

értékekre vonatkozó kényszerekkel leírt invarianciafeltételeket, operációk specifikációit és állapotalapú szerepeket. Olyan specifikációkat, ahol valamely operáció bemenete és kimenete közötti kapcsolat is adott, a kalkulusok nem támogatnak.

Bebizonyítottam, hogy a kalkulusaim az ilyen célú alkalmazáshoz szükséges tulajdonságokkal rendelkeznek. [TR, J2]

Munkám kiindulópontjával a λ -kalkulust[13] választottam két okból. Egyrészt minden fontos OO képesség ábrázolására képes (osztályok, operációk precíz kiválasztással (multiple dispatch)³, öröklődés). Másrészt λ egy változata egy további hasznos képességgel van felruházva, a kötött polimorfizmussal (bounded polymorphism) többek között típusmegőrző függvények modellezésére, és egy hasonló technika az én kalkulusaim esetében is alkalmazható lehet a kalkulusok képességeinek bővítésére.

A fő gondolat $\lambda\mathcal{C}$ és $\lambda\mathcal{C}'$ esetében az, hogy a λ -ből ismert (elő)típus fogalmát kényszerhalmazzal egészítettem ki. A kényszerek atomi formulákból a sztenderd összekötőjelekkel felépített, esetlegesen kvantifikált, jólformált elsőrendű formulák. Minden kényszerformula minden szabad változójának annak az (elő)típusnak bal alsó indexében kell megjelennie, amelyhez tartozik. Amint azt sugallja, ezek a változók az általuk jelölt (elő)típus-kifejezéshez tartoznak. Következésképpen az indexváltozóknak mind különbözőnek kell lenniük minden egyes (elő)típusban.

A predikátum- és függvényszimbólumok tényleges halmaza tetszőleges lehet és általában az alkalmazási terület határozza meg, azaz az előtípusok. (De adott esetben kiszámíthatósági okokból, konstansokon kívül más függvényszimbólum nem használható, ld. a 2.4. tézist.) Néhány predikátumszimbólumot, pontosabban minden egyes elemi típushoz egyet-egyet, mindenképp definiálni kell. Jelentésük az, hogy a paraméter olyan típusú (azaz eleme a típus értékkészletének). Ezen predikátumszimbólumok az (elő)típusok kényszerein kívül a levezethetőség alkalmazáspecifikus axiómáiban is szerepelnek.

2.1. tézis (Teljesség [J2]). *Bebizonyítottam, hogy $\lambda\mathcal{C}$ -ben és $\lambda\mathcal{C}'$ -ben bármely kifejezés típusát a kifejezés redukciója nem változtatja meg.*

2.2. tézis (Egyértelmű redukált [J2], állapotalapú szerepekre ld. még [TR]). *Bebizonyítottam, hogy $\lambda\mathcal{C}$ -ben és $\lambda\mathcal{C}'$ -ben a kifejezés teljesen redukált alakja független attól, hogy az egyes redukációs lépéseket milyen sorrendben hajtják végre.*

Mindkét kalkulusom alaposan épít FOL eszköztárára, ugyanakkor FOL-ben néhány probléma eldönthetetlen. Ezért azt is meg kellett vizsgálnom, hogy FOL-alapokon egyáltalán praktikus célokra használhatók-e a kalkulusaim. Két területet azonosítottam, amelyre hatással vannak az eldönthetetlen problémák: a típuskonzisztencia és a levezethetőség kérdése.[TR, J2]

A típusoknak nem kell feltétlenül konzisztenseknek lenniük kényszereik tekintetében, azaz a kényszereik halmaza lehet kielégíthetetlen. Ez azt jelenti, hogy létezhet olyan típus, amelyhez nem tartozhat kifejezés. Ez nem praktikus, hiszen az efféle típusok valamiféle modellezési hibára utalnak és rendszererőforrásokat

³ A precíz kiválasztás azt jelenti, hogy az eljárások kiválasztása az összes argumentum típusa alapján történik, nem csak az üzenet fogadójának típusa alapján.

kötnek le anélkül, hogy bármiféle előnyt szolgáltatnának. Ugyanez igaz azokra a függvényekre, amelyek inkonzisztens típusok kifejezéseit várják el bemenetként. Végül, annak biztosítására, hogy kétértelműség nélkül kiválasztható legyen egy túlterhelt függvény megfelelő változata (branch-e), a definíciók értelmében szükség lehet egy olyan változatra, amely sosem hívódhat meg, mert a bemeneti típusra vonatkozó feltételeknek egyetlen kifejezés sem tud megfelelni.

2.3. tézis (Konzisztens típusok [TR, J2]). *Megmutattam, hogy a típusok konzisztenciája előírható amellet, hogy kalkulusaim megtartják előző két tézisben kimondott tulajdonságaikat.*

A levezethetőség egyike azon problémáknak, amelyek általánosságban eldönthetetlenek FOL-ben, így egy eldönthető alosztályt kerestem kalkulusaim számára. Mivel az OO modellek alkalmazási területe tetszőleges lehet, ezért az eldönthető alosztályokat a formulák elején álló kvantorok alapján, ill. a logikai nyelv predikátum- és függvényszimbólumainak száma alapján kerestem. A [6]-ként hivatkozott könyv nagyon hasznosnak bizonyult erre a célra, hiszen ezen feltételek mentén kimerítően felsorolja a maximális eldönthető és minimális eldönthetetlen osztályokat.

2.4. tézis (Eldönthető kalkulusok [J2]). *Meghatároztam egy elegendő feltételt, amely lehetővé teszi kalkulusaim gyakorlati alkalmazását az eldönthetőség tekintetében.*

Az altípusreláció biztosan eldönthető, ha a modell minden formulája a következő feltételek egyikének megfelel.

Bernays-Schönfinkel-Ramsey-osztály: A prefixformában minden egzisztenciális kvantor megelőz minden univerzális kvantort és konstansokon kívül más függvényszimbólum nincs a nyelvben.

Gurevich-osztály: A prefixformában csak egzisztenciális kvantorok vannak. Tetszőleges aritású függvény- és predikátumszimbólumok használhatók.

Shelah-osztály: A formulák prefixformája mindössze egyetlen univerzális kvantort és legfeljebb egyetlen, unáris függvényszimbólumot tartalmaz. Az egzisztenciális kvantorok és a predikátumszimbólumok száma nem korlátozott.

A típuskonzisztencia miatt (ld. a 2.3. tézist) az utóbbi két osztály nem jöhetett szóba.

A kalkulusaim kifejezőerejét nem befolyásolja, hogy a Bernays-Schönfinkel-Ramsey osztályban nem lehetnek csak konstans függvényszimbólumok, hiszen minden atomi formula

$$p\left(f_1(f_{11}(\dots), \dots, f_{1k}(\dots)), f_2(f_{21}(\dots), \dots, f_{2l}(\dots))\right),$$

amelyben z_1, \dots, z_n változók és konstansok szerepelnek, az általánosság csorbítása nélkül átírható

$$p'(z_1, \dots, z_n)$$

vagy

$$\forall x_1 \forall x_2 p_1(x_1, z_1, \dots, z_n) \wedge p_2(x_2, z_1, \dots, z_n) \Rightarrow p(x_1, x_2)$$

formába megfelelő jelentésű új szimbólumokkal. A legutóbbi kifejezésben az f_i -ket kiváltó p_i -k predikátumszimbólumok, amelyek első argumentuma az eredeti függvény visszatérési értéke. Ugyancsak a függvényszimbólumokat érintő megszorítás volt az oka, hogy az 1. definícióban az operációkat predikátumokkal írtam le, bár elméletileg függvényekkel is lehetett volna.

A fenti eldönthetőségi feltétel csak elégséges, de nem szükséges. Azaz egy konkrét formulahalmaz eredményezhet eldönthető rendszert annak ellenére, hogy nem tartozik a fenti osztályok egyikébe sem.

A kutatási célkitűzésekkel összhangban ez a szakasz azt mutatta meg

- hogy néz ki az én kényszerekkel kiegészített axiomatikus OO adatmodellem és
- az én kalkulusaim hogyan teszik lehetővé OO, értékényszerekkel kiegészített programok részleges helyességének bizonyítását.

Az itt bemutatott rendszerek részletes leírása megtalálható az értekezésben.

4.2. Hatékony lekérdezés-kiértékelés nyílt sémájú adatbázisokban

Az ötlet egyszerű, hogyan lehet eredményes lekérdezés-kiértékelési módszerekből hatékonyat csinálni: nem éri meg azokat az elemeket összehasonlítani, amelyeknek (kapcsolatok szintjén) nem sok közülük van egymáshoz, hanem inkább a rendszer szűrje ki ezeket egy gyors módszerrel, majd a költséges páronkénti összehasonlítást már csak a maradékra végezze el.

Ez az ötlet az OBIR rendszerekre kidolgozott ötlet [30] általánosítása. Az általánosítás azért volt lehetséges, mert egy nagyon általános, logikamentes ontológiadefinícióval éltem (ld. a bevezetőt). A következőkben, akárcsak az értekezésben azonban az érthetőség kedvéért maradok az ontológiák terminológiájánál. A fogalmak egy az egyben megfeleltethetőek a nyílt sémák fogalmainak.

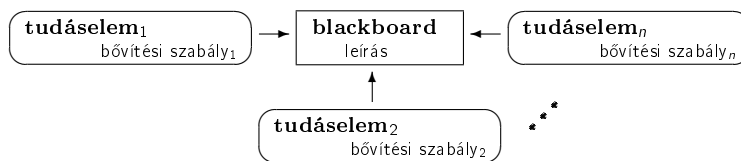
3. téziscsoport (A szóhajóhető elemek meghatározása). *Terveztem egy DBMS (al)rendszert, amely támogatja (Q_{similar})-alakú (ld. a 4. oldalon) lekérdezések hatékony kiértékelését feltéve, hogy*

- a leírások esetlegesen súlyozott OE-k halmazai és
- bizonyos, OE-k halmazára vonatkozó bővítési szabályok a következőkben ismertetett módon adottak. [C5]

Az alrendszer a végső hasonlósági (relevancia-) vizsgálatra szóhajóhető elemeket két lépésben szolgáltatja.

Az első lépésben az alrendszer kibővíti a lekérdezést (súlyok figyelembevétele nélkül). Ezt azonban óvatosan kell tenni, hogy a hatékonyság megmaradjon.

3.1. tézis (Ágensalapú párhuzamosított lekérdezésbővítő). *Erre a célra egy blackboard-rendszert [16, 15] javasoltam, amelyet a 4.1. ábra mutat. A tudáselemeket a konkrét ontológiának megfelelően kell kialakítani (hogy tükrözze az egyes kapcsolatok jellemzőit), azaz a bővítési szabályok az adatbázis részei kell, hogy legyenek.*



4.1. ábra. Lekérdezésbővítés blackboard-architektúrával

A blackboard tárolja a kibővített lekérdezést. Kezdetben azokkal az OE-kkel van feltöltve, amelyek az eredeti lekérdezésben szerepelnek.⁴ Aztán az egyes tudáselemágensek felelősek azért, hogy a blackboard zárt legyen a saját bővítési szabályukat tekintve. Ezt úgy érik el, hogy folyamatosan monitorozzák a blackboard tartalmát és beletesznek további OE-eket, ha szükséges. Nincs külön vezérlés, azaz a bővítési lépésnek akkor van vége, ha a blackboard minden ágens szerint zárt. Az ontológia végessége garantálja, hogy egyszer ez bekövetkezik.

Az eljárás párhuzamosított, hiszen az ágensek függetlenül dolgoznak.

A lekérdezés kibővítése egy alapvetően adatvezérelt és összetett feladat, mivel egy elem felvétele további elemek felvételét vonhatja maga után. Ugyanakkor a blackboard-rendszer sikeresen szétválasztja a funkcionalitást az adattól. Ez azt jelenti, hogy csak a tudáselemek, azaz a bővítési szabályok ontológiaspecifikusak (pontosabban függenek az ontológia kapcsolataitól), az architektúra ontológiafüggetlen.

3.2. tézis (A szóbajóhető elemek meghatározása a kibővített lekérdezés alapján). *A konkrét szűrési lépés megvalósítására a következő javaslatot tettem: A szóbajóhető elemek halmaza azon erőforrások halmaza, amelyet legalább egy, a kibővített lekérdezésben szereplő OE jellemez. A kibővített lekérdezést az előző lépés alapján a blackboard-rendszer szolgáltatja. Ez a lépés nagyon egyszerű, és munkám szerint egyetlen klasszikus adatbázis-lekérdezéssel megvalósítható.*

Természetesen más, kifinomultabb eljárások is alkalmazhatók ebben a lépésben. Ugyanakkor a végfelhasználók tesztje (ld. még az 5. fejezetet) az egyszerű eljárás alapuló OBIR rendszer esetében azt mutatta, hogy a találatok jók és megfelelő időn belül rendelkezésre állnak.

A kutatási célkitűzésekkel összevetve a 3. téziscsoport lehetővé teszi, hogy nagyméretű adatbázisokban is olyan lekérdezéseket hajtsunk végre, amelyekben az attribútumokhoz ontológia kapcsolódik.

4.3. Részbenrendezések megvalósítása fizikai adatbázisokban

Az 1.3. szakasz és a 2. fejezet informális bevezetője után következzen a formalizált feladat, amellyel a fizikai adatbázisok körében foglalkoztam.

⁴ Ez tehát azt jelenti, hogy az esetleges súlyoktól eltekintve induláskor a blackboard tartalma megegyezik a lekérdezéssel.

Bármely A attribútum esetén jelölje d_A d adatelem A értékét és \mathcal{D}_A A értékkészletét. A feladat hatékonyan kiértékelni a következőkben definiált lekérdezéseket figyelembe véve, hogy

- \mathcal{D}_A a \preceq_A relációval egy részbenrendezett halmaz (poset) és
- a v paraméterre $v \in \mathcal{D}_A$.

5. definíció (Elemi lekérdezések részbenrendezett attribútumértékekkel).

$\max_A(v)$: azon d adatelemeket adja, amelyre $d_A \preceq_A v$ és nincs d' adatelem, amelyre $d_A \neq d'_A \wedge d_A \preceq_A d'_A \preceq_A v$.

$\min_A(v)$: azon d adatelemeket adja, amelyre $v \preceq_A d_A$ és nincs d' adatelem, amelyre $d_A \neq d'_A \wedge v \preceq_A d'_A \preceq_A d_A$.

$\text{similar}_A(v) = \max_A(v) \cup \min_A(v)$.

A legutolsóra tekinthetünk úgy, mint a klasszikus keresés közelítő megfelelőjére, mivel azon d adatelemeket szolgáltatja, amelyekre $d_A = v$, ha van ilyen, egyébként azokat, amelyeket bizonyos értelemben v -hez legközelebbinek nevezhetünk. Ugyanakkor ez nem egy tényleges legközelebbi elemeket szolgáltató lekérdezés, mert az eredmény lehet üres halmaz is, habár nem kétséges, melyik elem a legközelebbi. Tekintsünk egy példát. Legyen egy egyattribútumos adatbázis egyetlen adateleme a $\{3, 7\}$ halmaz. Ha a részbenrendezés a részhalmaz reláció, akkor $\text{similar}(\{7, 13\}) = \emptyset$, habár az adatbázis egyetlen elemének is van a lekérdezésben adott paraméterrel közös eleme. Összetett (esetlegesen további elemi lekérdezésekből logikai összekötőkkel épített) lekérdezéseket a szokásos lekérdezés-feldolgozási módszerekkel (ld. pl. [17]) lehet kiértékelni.

A *hatékony válaszadás* azt jelenti, hogy a másodlagos tárhoz való hozzáférések számát (azaz be- és kivitel, IO a médiáról) minimalizálni kell; ez klasszikus adatbázis-kezelési alapelv. A számítások egyszerűsítése érdekében feltételeztem, hogy minden egyes (fő- ill. kiegészítő) adatelem elérése egyetlen hozzáférést igényel. Ezt az indokolja, hogy semmi egyéb információ nem áll rendelkezésre az attribútumok, a rekordok méretéről stb., de az bizonyos, hogy a szükséges hozzáférések száma (azaz az időkomplexitás) ezekkel arányos.

A fent említetten kívül a probléma megoldásának keresése során még a következő feltételezésekkel éltem.

- Az adatrekordok száma az adatbázisban óriási is lehet, valamint ha a poset egyetlen eleméhez több adatrekord tartozik, a poset elemszáma akkor is nagyon nagy lehet. Ez azt jelenti, hogy a teljes poset nem tárolható a fő memóriában. Valóban léteznek alkalmazási területek, amelyeknél a poset mérete felülről nem korlátos. Ilyen például az adatbázisintegráció [26, C2, C4].
- Poset-kódolás (azaz az elemek bitfolyam jellegű leképezése és azokon lekérdezés-kiértékeléskor bitműveletek végzése) [2] nem alkalmazható. Ennek oka, hogy a bitfolyamok kiszámítása és karbantartása drága (és így nem praktikus) adatbázisokban, ahol a posetek óriásiak lehetnek, mert egyetlen adatelem változása maga után vonhatja az összes bitfolyam újraszámításának szükségességét.

- A lekérdezés feldolgozása során nem tárolható külön információ a már feldolgozott posetelemről, mert a cache-elés további, a poset méretével arányos memóriaigényt támaszt. Ez nemkívánatos adatbázisokban, ahol a poset nagy lehet és több lekérdezés is futhat párhuzamosan.

Minden eredményemet ezen feltételek mellett kell érteni.

4. téziscsoport. *Definiáltam két kiegészítő adatstruktúrát adatbázisok számára a hozzájuk tartozó lekérdezési és karbantartási (beszúrási, törlési)⁵ algoritmusokkal, amelyek a fenti elemi lekérdezéseket támogatják. Megvizsgáltam továbbá a konstrukciók tár- és a különböző algoritmusok időigényét. [C3, C4]*

Az első, a naiv kiegészítő adatstruktúra alapvetően egy irányított gráf, amely A elemeinek részbenrendezését reprezentálja, azaz a csomópontok megfelelnek az attribútumértékeknek, az élek pedig a részbenrendezési viszonyoknak. Továbbá a gráf reflexíven és tranzitíven redukált. Ez (amelyet *maggráfnak* neveztem el és G_c -vel jelöltem) további csomópontokkal van kiegészítve, amelyek az adatelemeknek felelnek meg, és élekkel vannak a maggráf azon csomópontjaihoz kötve, amely által reprezentált érték az adatelem A attribútumának értékével megegyezik. [C3, C4] (Az értekezés tartalmazza a formális definíciókat, továbbá szól arról is, hogyan lehet ezt a gráfot blokkszervezésű médián tárolni.)

Jelölje a gráf csomópontjai számának felét n (amely, definíció szerint, megegyezik a maggráf csomópontjainak számával) és N az adatbázis adatelemeinek számát. Jelölje a maggráf forrásai (azaz megelőző csomópont nélküli csomópontjai) és nyelői (azaz követő csomópont nélküli csomópontjai) számossága közül a nagyobbikat s és a leghosszabb útját p . Ezekkel a jelölésekkel a következő állításokat tettem.

4.1. tézis (Tárigény[C3]). *Megmutattam, hogy a naiv kiegészítő adatstruktúra $\mathcal{O}(n^2 + N)$ tárat igényel.*

4.2. tézis (Lekérdezés-kiértékelő algoritmusok). *Konstruktív bizonyítást adtam arra, hogy (ellentétben mások hasonló, általános részbenrendezést ábrázoló naiv struktúráinak módszerével) nem szükséges teljes bejárást végezni a lekérdezés-kiértékelés során. Megmutattam, hogy a lekérdezés-kiértékelő algoritmusok, amelyeket a kiegészítő adatstruktúrára definiáltam és a 4.2. ábrán látható eljáráson (egy javított mélységi bejáráson) alapszanak, helyesek. Azt is beláttam, hogy az algoritmusaim, ha a posetben az elemek szomszédjainak száma felülről korlátos, legrosszabb esetben is $\mathcal{O}(s+p + N)$ időben futnak [C3].*

A képletnek természetesen része N , mert az eredményeket, azaz az adatelemeket magukat mindig vissza kell adni.

4.3. tézis (Karbantartó algoritmusok a naiv struktúrán). *Bebizonyítottam, hogy a beszúráások, a törlések a naiv kiegészítő struktúrán általam definiált módon történő megvalósítása helyes és általában a lehető leghatékonyabb, azaz az algoritmusok $\mathcal{O}(n^2 + N)$ időben futnak [C3].*

⁵ Mint szokásos, a módosítást egy törlés és egy beszúráás egymásutánjával meg lehet valósítani.

```

1 function  $\min_A(v)$ 
2   return  $\min'_A(v)$  valamely eleméhez kapcsolódó adatelemek
3 endfun
4 function  $\min'_A(v)$ 
5    $part := \emptyset$ 
6   foreach source által reprezentált elem as node do
7     if  $node \preceq_A v$  then return  $\min'_A(node, v)$ 
8     if  $v \preceq_A node$  then  $part := part \cup \{node\}$ 
9   return  $part$ 
10 endfun
11 function  $\min'_A(node, v)$ 
12   if  $v \preceq_A node$  then return  $\{node\}$ 
13    $part := \emptyset$ 
14   foreach node közvetlen következő szomszédja  $G_c$ -ben as next do
15     if  $next \preceq_A v$  then return  $\min'_A(next, v)$ 
16     if  $v \preceq_A next$  then  $part := part \cup \{next\}$ 
17   return  $part$ 
18 endfun

```

4.2. ábra. $\min_A(v)$ meghatározása

Mivel a naiv gráfrepresentáció adott esetben ugyanolyan lassú, mint a kimerítő keresés (nemcsak csupa összehasonlíthatatlan elemből álló poset esetén, hanem akkor is, ha peches sorrendben látogatja a csomópontokat), javaslatot tettem a poset ténylegesen adatbázisban megtalálható elemeinek láncokként (azaz csomópontfüggetlen utakként) történő ábrázolására.[C4] (Ennek egy konkrét megvalósítására az értekezés mutat példát.)

A [3]-ként hivatkozott munka már láncokként ábrázolta a posetet, azonban nem háttértáron, hanem a fő memóriában. Ez a különbség eltérést okoz az időkomplexitás egységében, amellyel a kiértékelőket összehasonlítjuk. Ugyanez a munka már be is vezette a bináris keresést klasszikus pontos lekérdezések esetére, és empirikusan megmérte a hatékonyságát is. Tehát az én nevemhez fűződő új eredmény itt a láncábrázolás háttértáron történő megvalósítása, a közelítő lekérdezések kiértékelési sebességének javítása bináris kereséssel, és az algoritmusok formális hatékonyságelemzése.

A láncábrázolás a posetnek megfelelő (mag)gráf láncdekompozíciója. Egy ilyen dekompozícióban a láncokon magukon kívül csak a láncok közötti éleket (egészen pontosan minden csomópontból minden másik lánchoz legfeljebb egy-egy élt) elegendő eltárolni. Az ábrázolás egyéb (maggráfon kívüli) részei változatlanok, azaz az adatelemeknek megfelelő csomópontok ugyanúgy léteznek és hozzá vannak kapcsolva a megfelelő maggráfcsomóponthoz csakúgy, mint a naiv gráfrepresentáció esetén.

Figyeljük meg, hogy a láncrepresentáció nem különbözik a naiv reprezentációtól az eltárolt élek (és természetesen csomópontok) tekintetében. Mindössze a maggráf van csomópontdiszjunkt utakra dekomponálva, amelyek teljesen rendezett részalmazok, és mint olyanok különösen alkalmasak sorfolytonos tárolásra.

Jelölje w a láncok számosságát egy láncábrázolásban.

4.4. tézis (Élesebb felső korlát a gráfrepresentációk tárigényére). *Megmutattam, hogy a láncrepresentáció $\mathcal{O}(wn + N)$ helyet foglal, továbbá ugyanez igaz a naiv representációra is, bár ott w értéke explicit nem adott.*

Mivel $n \geq w$ triviálisan mindig teljesül, ez az új korlát élesebb, mint a 4.1. tézisben szereplő. Ezen kívül ismert, hogy:

- a legkisebb lehetséges w (amelyet itt a továbbiakban w_{\min} jelöl) egyenlő a legnagyobb független csomóponthalmaz méretével (Dilworth tétele[5] véges gráfokra),
- [35] alapján: $1 > q > 0$ valószínűségű véletlen gráfok esetén

$$E(w_{\min}) = \mathcal{O}\left(\frac{\ln(nq)}{q}\right),$$

ahol E a várható értéket jelöli.

4.5. tézis (Lekérdezés-kiértékelési algoritmusok láncrepresentációval). *Átalakítottam eredeti algoritmusaimat, hogy bináris kereséssel működjenek (vö. 4.3. ábra). Megmutattam, hogy helyesek és hogy időigényük*

$$\mathcal{O}(w + w \log_2(p+1) + N)$$

a konkrét posettől függetlenül.

Abban a speciális esetben, amikor teljes rendezésről van szó, azaz $w=1$ és $n = p+1$, a láncokkal történő lekérdezés-kiértékelés ugyanolyan hatékony, mint a klasszikus indexek használata.

A naiv gráfrepresentációhoz hasonlóan a láncrepresentáció karbantartása (adat-elemek beszúrása-törlése) alapvetően a maggráf, azaz itt maga a láncdekompozíció karbantartásáról szól, hiszen a gráf további része nagyon egyszerű. A láncdekompozíciók karbantartását már alaposan megvizsgálták, ld. pl. [25, 3, 24, 7]. A kapcsolódó irodalom olyan algoritmusokat ad, amelyek kis módosításokkal magától értetődő módon alkalmazhatók az én láncrepresentációm esetében is.

A kutatási célkitűzésekkel összevetve ez a szakasz hatékony, doménfüggetlen fizikai szervezést mutatott be részbenrendezések számára. Ebben a környezetben a közelítő válaszok megtalálásával is foglalkoztam.

```

1 function  $\min_A(v)$ 
2   return  $\min'_A(v)$  valamely eleméhez kapcsolódó adatelemek
3 endfun
4 function  $\min'_A(v)$ 
5    $part := \emptyset$ 
6   for  $i := 1, \dots, w$  do
7      $node := chains[i][1]$ 
8     if  $node$  nem forrás then continue
9     if  $v \preceq_A$   $node$  által reprezentált elem then  $part := part \cup \{node\}$ 
10    if a  $node$  által reprezentált elem  $\preceq_A v$  then
11      return  $part \cup \min'_A(i^{th} \text{ chain}, 1, v)$ 
12    return  $part$ 
13 endfun
14 function  $\min'_A(chain, item, v)$ 
15    $node := chain[item]$ 
16   if  $v \preceq_A$   $node$  által reprezentált elem then return  $\{node\}$ 
17    $part := \emptyset$ 
18    $last := \text{binary\_search}(v, chain, item, \text{length of chain})$ 
19   if  $v \preceq_A$   $chain[last]$  által reprezentált elem then return  $\{chain[last]\}$ 
20   foreach  $c \in$  még a teljes algoritmus által meg nem látogatott lánc do
21      $item := \text{neighbour}(chain, last, c)$ 
22     if  $0 = item$  then continue
23      $next := c[item]$ 
24     if  $v \preceq_A$   $next$  által reprezentált elem then  $part := part \cup \{next\}$ 
25     if a  $next$  által reprezentált elem  $\preceq_A v$  then
26       return  $part \cup \min'_A(c, item, v)$ 
27   return  $part$ 
28 endfun
29 function  $\text{binary\_search}(v, chain, first, last)$ 
30   if  $first = last$  then return  $first$ 
31    $item := \lceil \frac{first+last}{2} \rceil$ 
32   if  $chain[item] \preceq_A v$  then
33     return  $\text{binary\_search}(v, chain, item, last)$ 
34   else return  $\text{binary\_search}(v, chain, first, item-1)$ 
35 endfun
36 function  $\text{neighbour}(chain, item, next)$ 
37   return a  $next$  lánc azon elemének indexe, amely éllel elérhető
     $chain[item]$ -ből, ill. 0, ha nincs ilyen
38 endfun

```

4.3. ábra. $\min_A(v)$ meghatározása láncokkal

5. AZ EREDMÉNYEK ALKALMAZÁSA

A **kényszerekkel bővített axiomatikus OO adatmodell** valójában egy (a legutóbbi kor igényeit is kielégítő) keretrendszer objektumok és kényszerek modellezésére adatbázisokban. A keretrendszer maga nem foglalkozik praktikus alkalmazhatósági kérdésekkel (mint pl. eldönthetőség, komplexitás), ezeket a konkrét alkalmazásokban kell megvizsgálni. Hasznos ehhez a FOL alkalmazásainak bőséges irodalmát tanulmányozni, mert minden efféle optimalizálási problémát megvizsgáltak már.

A **függvénykalkulusok**, λ & \mathcal{C} és λ & \mathcal{C}' kényszerekkel felvértezett OO programok részlegeshelyesség-bizonyításának alapjául szolgálnak (vö. a kapcsolódó tézisekkel a pontos képességek és korlátok tekintetében). Érdemes megemlíteni, hogy a kalkulusok szélesebb alkalmazási területtel bírnak: nem csak adatbáziskezelőkben, hanem általános OO szoftverfejlesztések során is alkalmazhatók.

Részleges helyesség bizonyítása a kalkulusokkal a következő lépéseken keresztül lehetséges.

1. Az elemi típusok azonosítása.
2. A predikátumok tulajdonságainak formalizálása. A matematikai logikában ezt axiomatizálásnak hívják. Minden bizonyítás (\vdash) ezekre az axiómákra fog támaszkodni.
3. Az osztályok (beleértve az állapotalapú szerepeket), a specifikációk (elő)típusokká és az eljárások (függvények) λ & \mathcal{C} - ill. λ & \mathcal{C}' -kifejezésekké történő újraírása.
4. Minden egyes term valódi V típusának meghatározása és összehasonlítása a szándékolt (adott) W típussal. Akkor és csak akkor, ha $V \leq W$ igaz, a kifejezés kielégíti a vele szemben támasztott követelményeket.

Egy konkrét illusztratív példát mutat be [J2].

Amint azt az értekezésem egy fejezetében megállapítottam, mind az adatmodell, mind a függvénykalkulusok használhatnak FOL helyett leíró logikát.

A **nyílt sémájú, adatelemek közötti kapcsolatokat figyelembe vevő hatékony lekérdezés-kiértékelési eljárást** egy webalapú prototípus OBIR rendszerben¹ alkalmaztam Európához köthető történelmi szövegek visszakeresésére. A rendszer részletes leírását [C5] adja, itt csak egyes elemeit emelem ki a lekérdezés-kiértékelési időre tekintettel.

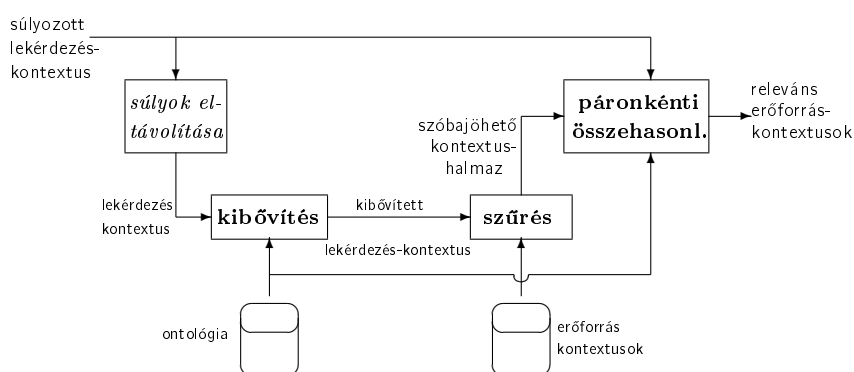
A rendszer erőforrásleírói (amelyeket, akárcsak bármely OE-halmazt, kontextusnak neveztünk az ott alkalmazott terminológia szerint) két részből álltak: egy időintervallumból és egy fogalom- (OE-) halmazból. A leírók minden eleme nulla és

¹ webcím: <http://www.eurohistory.net>

egy közé eső valós számokkal volt súlyozva. Az ún. kontextuskezelő komponensnek két feladata volt:

1. új erőforrásokhoz leírók előállítása,
2. bármely két leíró hasonlóságának kiszámítása.

Minden lekérdezés indirekten a másodikra támaszkodott, amely során valójában nem az összes erőforrásleíróra, csak annak egy részhalmazára történt meg a kiértékelés. A részhalmaz kiválasztása céljából az én szűrő eljárásomat építettük be. Azaz az egész lekérdezés-kiértékelési (leírójuk alapján releváns erőforrásokat kereső) eljárás az 5.1. ábra szerint működött.



5.1. ábra. A releváns adatelemek meghatározása OBIR-ben [C5]

Az OBIR rendszer csak egy volt az Európai Unió 5. keretprogramjának Information Society Technologies által finanszírozott Visual Contextualisation of Digital content (VICODI) projektjének² eredményei közül. A projekt célja végső soron az internet digitális tartalmának emberek által történő megértésének növelése volt. Ezért a rendszer az erőforrásokat automatikusan kontextualizálta, majd a kontextusokat bejárhatóvá és kereshetővé tette. A keresési eljárás is újszerű volt: OE-kre lehetett kattintani, amelyeket ekkor a rendszer felvett a keresőkérdésbe, majd rögtön megmutatta az új keresőkérdésre adott találatokat. A konkrét erőforrások tárolt példányainak megjelenítésekor pedig a kontextushoz köthető elemeket a rendszer kiemelte. [C5]

A projekt céljával összhangban egy végfelhasználói tesztet terveztek az egész webportálra. Az első ilyen tesztre még az összes funkcionalitás elkészülte előtt, de már a véglegesnek szánt keresőmotor beépítése után került sor. Ennek megfelelően a teszt nem pusztán a keresőgépre, hanem a teljes keresőrendszerre vonatkozott. Éppen ezért annak teljesítményét nem lehetett a klasszikus IR metrikákkal mint precizitás és visszahívás³ jellemezni.

Összességében elmondható, hogy a teszt eredményei az elvárt eredményt hozták: a felhasználók elégedettek voltak az OBIR rendszerrel. Ez az eredmény alapvetően a végső páronkénti összehasonlító eljárást jellemzi, azaz az biztosan jó precizitással és visszahívással rendelkezik, de áttételesen szól arról is, hogy

² webcím: <http://www.vicodi.org>

³ *Precizitás*: a találatok között a releváns találatok aránya. *Visszahívás*: a releváns találatoknak az összes tárolt releváns erőforrásokhoz viszonyított aránya.[4]

1. az első (szűrő) lépés, amely a szóbjöhető erőforrásokat határozza meg, nem befolyásolja észrevehetően ezeket a tulajdonságokat,
2. a válaszdő megfelelő.

Az első megállapítás elosztat minden kétséget, amely megkérdőjelezi, hogy egy egyszerű lépéssel érdemes a kibővített lekérdezésből a szóbjöhető elemeket kiszámítani. Természetesen ennek ellenére kifinomultabb módszert is lehetne itt alkalmazni.

A második megállapítás az általam kitűzött cél (hatékonyság növelése) fényében fontos. Meg kell azonban említeni, hogy néhány felhasználó nem volt maradéktalanul elégedett a lekérdezés sebességével. Azt várták, hogy a találati oldal megjelenítésének nagyságrendjébe fog esni, mint ahogy az a népszerű megszokott webes keresők esetében van. A mi rendszerünk ezzel szemben néhány másodperc alatt szolgáltatja az eredményeket, amely még elfogadható mind abszolút értékben, mind pedig az alkalmazott programozási nyelv (Java[20], tárolt eljárások nélkül) és a rendelkezésre álló számítási kapacitás (belépő szintű PC-kiszolgáló) figyelembevételével.

Az kvázi- és részbenrendezéseket közvetlenül ábrázoló fizikai szervezést alapvetően nagy, ritkán módosított adatbázisokban érdemes alkalmazni, mivel a beszűrés, módosítás és a törlés csak költségesen valósítható meg. A lekérdezéskor elérhető időnyereség viszont jelentős lehet az adatbázis méretéből adódóan. A jövőben meg vizsgálni, milyen további tulajdonságoknak kell fennállniuk az elő- vagy részbenrendezéseken ahhoz, hogy az adatbázis módosítása is gyors legyen, és hol alkalmazható egy ilyen rendszer.

Az általam kialakított fizikai szervezés igény esetén közelítő válaszokat is tud adni. Mivel a részalmaz reláció egy kvázi rendezés, és a hierarchiák halmazelemeken értelmezett kvázi rendezések, a leggyakrabban előforduló esetekben, amikor pontos találat hiányában közelítő válasz elegendő, ez a fizikai szervezés alkalmazható.

Hivatkozások

- [1] Serge Abiteboul. Querying semi-structured data. In Foto N. Afrati and Phokion G. Kolaitis, editors, *ICDT*, volume 1186 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 1997.
- [2] Hassan Ait-Kaci, Robert Boyer, Patrick Lincoln, and Roger Nasr. Efficient implementation of lattice operations. *ACM Transactions on Programming Languages and Systems*, 11(1):115–146, January 1989.
- [3] Franz Baader, Berndhard Hollunder, Berndhard Nebel, Hans-Jürgen Profitlich, and Enrico Franconi. An empirical analysis of optimization techniques for terminological representation systems — or: Making KRIS get a move on. *Applied Intelligence*, 4(2):109–132, May 1994.
- [4] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. ACM Press, 1999.
- [5] Garrett Birkhoff. *Lattice Theory*, volume XXV of *American Mathematical Society Colloquium Publications*. American Mathematical Society, Providence, Rhode Island, third (new) edition, 1967.
- [6] Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Springer-Verlag Telos, 1st edition, January 15 1997.
- [7] Bartłomiej Bosek and Piotr Micek. On-line adaptive chain covering of up-growing posets. In René David, Danièle Gardy, Pierre Lescanne, and Marek Zaionc, editors, *Computational Logic and Applications, CLA '05*, volume AF of *Discrete Mathematics and Theoretical Computer Science Proceedings*, pages 37–48, 2006.
- [8] I. N. Bronshtein, K. A. Semendyayev, G. Musiol, and H. Mühlig. *Handbook of Mathematics*. Springer, 4th edition, June 14 2004.
- [9] Peter Buneman. Semistructured data. In *PODS*, pages 117–121. ACM Press, 1997.
- [10] Peter Buneman, Susan B. Davidson, Gerd G. Hillebrand, and Dan Suciu. A query language and optimization techniques for unstructured data. In H. V. Jagadish and Inderpal Singh Mumick, editors, *SIGMOD Conference*, pages 505–516. ACM Press, 1996.
- [11] *ISO/IEC International Standard 14882: Programming Language – C++*. International Organization for Standardization, 2003.

-
- [12] Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys*, 17(4):471–522, 1985.
- [13] Giuseppe Castagna. *Object-Oriented Programming: A Unified Foundation*. Progress in Theoretical Computer Science. Birkhäuser, Boston, 1997.
- [14] R. G. G. Cattell, Douglas Barry, Mark Berler, Jeff Eastman, David Jordan, Craig Russell, Olaf Schadow, Torsten Stanienda, and Fernando Valez, editors. *The Object Data Standard, ODMG 3.0*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, San Francisco, California, USA, 2000.
- [15] Daniel D. Corkill. Collaborating software: Blackboard and multi-agent systems & the future. In *Proceedings of the International Lisp Conference*, New York, New York, USA, October 2003.
- [16] Iain Craig. *Blackboard Systems*. Ablex Publishing Corporation, Norwood, New Jersey, 1995.
- [17] C. J. Date. *An Introduction to Database Systems, Volume I*. Addison-Wesley Publishing Company, 5th edition, 1990.
- [18] Alvaro Adolfo Antunes Fernandes. *An Axiomatic Approach to Deductive Object-Oriented Databases*. PhD thesis, Heriot-Watt University, Department of Computing and Electrical Engineering, Edinburgh, Scotland, UK, September 1995.
- [19] Giorgio Ghelli. Foundations for extensible objects with roles. *Information and Computation*, 175(1):50–75, May 2002.
- [20] James Gosling, Bill Joy, and Guy Steele. *The Java™ Language Specification*. The Java Series. Prentice Hall PTR, 3rd edition, June 14 2005.
- [21] Georg Gottlob, Michael Schrefl, and Brigitte Röck. Extending object-oriented systems with roles. *ACM Transactions on Information Systems*, 14(3):268–296, 1996.
- [22] Jim Gray. The next database revolution. In Gerhard Weikum, Arnd Christian König, and Stefan DeBloch, editors, *SIGMOD Conference*, pages 1–4. ACM Press, June 13–18 2004.
- [23] Thomas R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies*, 43(5–6):907–928, 1995.
- [24] Selma İköz and Vijay Kumar Garg. Online algorithms for Dilworth’s chain partition. Technical report, The University of Texas at Austin, USA, 2004.
- [25] Ragesh Jaiswal and Kapil Narula. Lattice theoretic algorithms. Technical report, Indian Institute of Technology, Kanpur, India, 2002.

-
- [26] Zsolt Tivadar Kardkovács. *Heterogén adatbázisok lekérdezése strukturált nyelvi szerkezetek komplex szemantikai feldolgozásával (in Hungarian, Querying Heterogeneous Databases via Processing Complex Semantics of Structured Grammatical Phrases)*. PhD thesis, Budapest University of Technology and Economics, 2009.
- [27] Robert Kowalski. Algorithm = logic + control. *Communications of the ACM*, 22(7):424–436, 1979.
- [28] Liwu Li. Extending the Java language with dynamic classification. *Journal of Object Technology*, 3(7):101–120, July-August 2004.
- [29] Robin Milner. A proposal for standard ML. In *Proceedings of Symposium on Lisp and Functional Programming*, pages 184–197, New York, New York, USA, August 6–8 1984. ACM Press.
- [30] Gábor Nagypál. Private communication.
- [31] *OCL 2.0 Specification*. Object Management Group, Inc., May 1 2006.
- [32] M. P. Papazoglou and B. J. Krämer. A database model for object dynamics. *The VLDB Journal*, 6(2):73–96, 1997.
- [33] Darrell R. Raymond. *Partial Order Databases*. PhD thesis, University of Waterloo, Waterloo, Ontario, Canada, 1996.
- [34] Elke A. Rundensteiner. *MultiView: A methodology for supporting multiple views in object-oriented databases*. In Li-Yan Yuan, editor, *Proceedings of the 18th International Conference on Very Large Data Bases*, pages 187–198. Morgan Kaufmann, August 23–27 1992.
- [35] Klaus Simon. An improved algorithm for transitive closure on acyclic digraphs. In Laurent Kott, editor, *Automata, Languages and Programming – 13th International Colloquium*, number 226 in Lecture Notes in Computer Science, pages 376–386, Rennes, France, July 1986. Springer-Verlag.
- [36] *OMG Unified Modeling Language Specification, Version 1.5*. Object Management Group, Inc., March 2003.
- [37] *Unified Modeling Language: Infrastructure, Version 2.0*. Object Management Group, Inc., March 2006.
- [38] *Unified Modeling Language: Superstructure, Version 2.0*. Object Management Group, Inc., August 2005.

Tudományos közlemények

Folyóiratcikkek

- [J1] Sándor Gajdos and Zsolt Tivadar Kardkovács and Gábor Mihály Surányi. Deduktív objektumorientált adatbázis-kezelők tervezése és megvalósítása (in Hungarian, Designing and implementing deductive object-oriented database management systems). *Híradástechnika (Journal on C⁵)*, L(11):18–24, November 1999.
- [J2] Gábor Mihály Surányi. An object-oriented calculus with term constraints. *Journal of Functional Programming*, 17(3):353–386, May 2007.

Konferenciatickek

- [C1] Zsolt Tivadar Kardkovács and Gábor Mihály Surányi. Ubiquitous access to deep content via web services. In Juan Manuel Cueva Lovelle, Bernardo Martín González Rodríguez, Luis Joyanes Aguilar, José Emilio Labra Gayo, and María del Puerto Paule Ruiz, editors, *ICWE 2003*, volume 2722 of *Lecture Notes in Computer Science*, pages 208–211. Springer-Verlag, 2003.
- [C2] Zsolt Tivadar Kardkovács, Gábor Mihály Surányi, and Sándor Gajdos. Application of catalogues to integrate heterogeneous data banks. In Robert Meersman and Zahir Tari, editors, *OTM Workshops*, volume 2889 of *Lecture Notes in Computer Science*, pages 1045–1056. Springer-Verlag, 2003.
- [C3] Gábor Mihály Surányi, Zsolt Tivadar Kardkovács, and Sándor Gajdos. Catalogues from a new perspective: a data structure for physical organisation. In Georg Gottlob, András A. Benczúr, and János Demetrovics, editors, *AD-BIS*, volume 3255 of *Lecture Notes in Computer Science*, pages 204–214. Springer-Verlag, 2004.
- [C4] Zsolt Tivadar Kardkovács, Gábor Mihály Surányi, and Sándor Gajdos. Towards building knowledge centres on the world wide web. In Tatyana M. Yakhno, editor, *ADVIS*, volume 3261 of *Lecture Notes in Computer Science*, pages 139–149. Springer-Verlag, 2004.
- [C5] Gábor Mihály Surányi, Gábor Nagypál, and Andreas Schmidt. Intelligent retrieval of digital resources by exploiting their semantic context. In Robert Meersman and Zahir Tari, editors, *CoopIS/DOA/ODBASE (1)*, volume 3290

of *Lecture Notes in Computer Science*, pages 705–723. Springer-Verlag, 2004.

- [C6] Zsolt Tivadar Kardkovács and Gábor Mihály Surányi. An axiomatic model for deductive object-oriented databases. In *Proceedings of the 5th International Symposium of Hungarian Researchers on Computational Intelligence*, pages 325–336. Budapest Tech — Hungarian Fuzzy Association, 2004.
- [C7] Gábor Mihály Surányi. Neue Semantikrepräsentationen im Datenbankbereich (in German, New representations of semantics in databases). In *Wissenschaftliche Mitteilungen der 17. Frühlingsakademie*, pages 122–128. Technische und Wirtschaftswissenschaftliche Universität Budapest, Institut für Ingenieurweiterbildung, May 4–8 2005.

Technikai dokumentáció

- [TR] Gábor Mihály Surányi. The λ & \mathcal{C} -calculus and its basic properties. Technical report, Budapest University of Technology and Economics, June 2004. Available on-line: <http://db.bme.hu/suranyi/calculus.pdf>.