Szemantikai alapú jogi tudás-menedzsment technológiák

PHD Doktori értekezés
Pécsi Tudományegyetem
Állam- és Jogtudományi Kar, Doktori Iskola

Készítette: Kilián Imre

Témavezető: Dr. Balogh Zsolt György PhD.

Tartalomjegyzék

1 Bevezetés

Kezdetben vala az Ige…

Az Ige hozta létre az Embert, aki – a paradicsomi Bűnbeesés egyik értelmezése szerint – addig csupán ösztönös lény volt, és vakon engedelmeskedett az ösztöne és Isten parancsainak. Ez utóbbi nem is okozott gondot, mert a három egy volt: Isten, a Természet és az ösztönök ugyanazt parancsolták. Öntudatlanul léteztünk, mint egy kicsi gyerek, nem volt kétségünk, nem volt gondolatunk, nem volt hozzátenni valónk ahhoz, vagy elvenni valónk abból, ami körülvett. A paradicsomi meztelenség szimbólum: tökéletesen beolvadtunk a Természetbe, nem volt különbség, nem volt elhatárolódás, nem volt különállás, nem volt ellentmondás.

A kígyó a tudás és a kétség szellemét ültette el bennünk: öntudatra ébredtünk, rádöbbentünk, hogy mást, és másképp is lehet, nemcsak élni és megélni szerettük volna, hanem megérteni is a körülvevő világot. Megérteni, használni és kihasználni, a saját céljainkra felhasználni, alakítani és esetleg átalakítani is akartuk…

A Tudás tehát gyűlni kezdett: a saját tudatára, az éntudatra, az öntudatra ébredt ember kezdte megfigyelni magát, és a körülvevő világot. Az összegyűjtött megfigyeléseket rendszerezni kezdte, és a rendszerezett megfigyelések elvonatkoztatásával újabb fogalmakat alkotott, a régi és újabb fogalmak között pedig összefüggéseket fedezett fel. Talán ez lehetett a Természettudomány kialakulása.

A jogtudomány kezdetei és kialakulása a közösségi szervezetek létrejöttének időpontjára datálhatók. Ez azt is jelenti, hogy nem fiatalabb, de nem is sokkal későbbi, mint a Bűnbeesésre datált természettudomány. Az együttélésnek, együttes tevékenységeknek, élelemszerzésnek, utódnemzésnek és -felnevelésnek az emberinél lényegesen primitívebb élőlények közösségeiben is különböző, az állatetológusok által megfigyelt, összegyűjtött és leírt szabályai is vannak – gondoljunk csak a hangyákra, vadludakra vagy farkasokra.

Objektív vagy szubjektív-e a Természettudomány, netán a Jogtudomány, vagy más tudományágak? Az egyes tudományágak objektivitására, vagy szubjektivitására vonatkozólag léteznek meghatározások, de ezek egyike sem megdönthetetlen. A „tudatunktól függetlenül is…”, illetve a „csak a tudatunkban létező” meghatározások helyből sántítanak, hiszen a tudatunk is tőlünk függetlenül létezik. Ezen lehet ugyan jóízűeket vitatkozni, de valóban: a szubjektívnek nevezett tudományok, a művészetek és a jog is olyan, aminek egyfelől komoly természetjogi indítékai és alapjai vannak, másfelől viszont komoly következményei is. Ki állíthatja azt, hogy a tilosban parkolást követő pénzbüntetés szubjektív? Másrészt, főleg a szubjektív ideológiai iskolák szerint minden csak a tudatunkban vetül ki, még merőben objektív dolgokat is gyakran teljesen különbözőféleképpen értékelünk.

Harmadrészt az objektívnak mondott természettudományok fogalmainak további elvonatkoztatásával jöttek létre a matematika, a logika és a filozófia tudományágai, amelyek bizony nagyon nagy százalékban merőben a tudatunkban léteznek: hiszen csupán a matematikusok tudatában tárolt fogalmakkal és összefüggésekkel dolgoznak, és a hétköznapi, megfogható dolgoktól olyan mértékben el vannak vonatkoztatva, hogy a matematika, mint tantárgy nem véletlenül a rémálma már az általános iskolásoknak is.

A jogtudomány semmiképpen sem tekinthető merőben szubjektívnak, mert mindenképpen objektív dolgok elvonatkoztatásával jött létre.

Azok az objektív dolgok, amelyeket a jogtudomány tükröz, magához a Természethez tartoznak. Akkor működőképes egy jogrendszer, ha szilárd természetjogi alapjai vannak. Az eredeti birtok- és országhatárok kialakulása általában valamiféle természetföldrajzi egységet követett (pl. Kárpát-medence): a térképen körzővel-vonalzóval kialakított határok mindig is mesterségesek voltak, és azok is maradnak. A szűk emelkedő úton a felfelé haladónak könnyebb visszagurulnia, felfelé tolatni sokkal nyűgösebb.

A jogrendszer nem más, mint a folytonosan változó társadalmi viszonyok állandónak tekinthető, és ezért rögzített része. A jog tehát tükrözi a társadalmi szokásokat, a szokásjogrendszert, a társadalom általános kulturális, információs, kommunikációs állapotát és kapcsolatrendszerét, a kapcsolatok megszokott és elfogadott, sőt elvárt formáit.

Amikor a tyúklopásért levágták valakinek a kezét, akkor nyilván efféle büntetések voltak a közvélekedésben is elfogadhatók. Amikor keresztre feszítették azt, aki Isten nevét – hiába – a szájára vette, akkor az. Amikor pedig a legvégső büntetési formát mindenféle életfogytiglani büntetésekre cserélték, akkor nyilván az volt az elfogadható.

Manapság a globalizált világban a jogrendszerek között is végbemegy egyfajta kiegyenlítődés, bár mindig vannak és maradnak különbségek. Ezek a különbségek azonban lényegesen szembeötlőbbek voltak a korábbi időkben, amikor az egyes emberek, embercsoportok, népek közötti kulturális és társadalmi különbségek is nagyobbak voltak. Amikor pedig a különböző kultúrák – és különböző jogrendek találkoztak, összeütköztek, akkor általában ez súlyos konfliktusokat eredményezett. Az állattenyésztő Káin megölte a földműves Ábelt (s mint rendesen, most is a gyilkos maradt életben), a sapiens kiirtotta a neandervölgyit, később pedig Vajk, avagy Szent István meghosszabbított jobbkeze – Vecelin német lovag által – megölte Koppányt. Aztán a kisebb népsűrűségre, és ebből fakadóan nagyobb mozgásszabadságra épülő keleti nomád jogrend ütközött a „klasszikusnak" mondható görög-római jogrenddel. A dolognak számunkra érdekes mozzanata, hogy ennek az összeütközésnek az elején még a „nomád" oldalon (hunok, avarok, kalandozások), míg a végén (a tatárjáráskor) már egyértelműen a „klasszikus" oldalon álltunk: Vajk-István királyunk érdeme, hogy felismerte: a történelem és a jövő a „klasszikus" jogrend oldalán áll. Vagy egyszerűen csak döntött, de ha másképp döntött volna, és ha ezután királyaink máskor is a nomád oldal felé döntöttek volna (kunok-jászok, tatárok, törökök), akkor most egészen másképp nézne ki a világ? A kérdés spekulatív és felesleges. A történelmet a győztesek írják, és mi, utódok csak alkalmazkodni tudunk az ezeréves döntésekhez. A nomád és a klasszikus jogrend közötti összeütközésből számtalan okból a klasszikus, szárazföldi, kontinentális jogrend maradt állva, de ez nem az egyetlen hasonló jellegű összeütközés volt.

Az európai szárazföldi jogrend Európa betelepülésével párhuzamosan jött létre, és lényegében sikeresen szabályozta a közös erőforrások (föld, erdő, vizek) használatával és kiélésével kapcsolatos megállapodásokat. Minden komolyabb értéknek, erőforrásnak volt gazdája, aki rendelkezett felette, és aki a gondnoka volt, vagyis gondolt vele. A gazdaváltás pedig békés, vagy erőszakos úton, de lényegileg rögzített peremfeltételek, jogrendszer alapján történt.

A kontinentális jognál mindig is szabadabb volt a tengeri jog, a világóceánok felfedezőinek a joga, amelyet nyilván a tengerparti államok határoztak meg. Ki mit talál azé, aki először tűzi ki valahol a zászlaját, azé. Aki először fedez fel, magáévá tesz, magánosít valami, addig közös, vagy „ismeretlen" erőforrást, azé. Az „Amerika

felfedezése” kifejezést hajmeresztő módon még máig is iskolában tanítjuk, pedig itt is csak két kultúra és két jogrend összeütközését tapasztalhattuk, illetve egy bizonyos fehér fajelmélet bukik rejtve még mindig elő a gimnáziumi tankönyvekből. Nem volt szükséges Amerikát felfedezni, mert az mindig is fel volt fedezve, hiszen az ott élő, és a földrészt otthonuknak valló népek már ősidők óta jól ismerték. Legfeljebb annyit mondhatunk: a fehér ember a maga számára is felfedezte, de nem csak felfedezte, hanem rögtön alkalmazni kezdte a tengeri jogot, amely joghatóságot is jelentett. Az Amerika-dominálta világban ma is beszélünk a második világháborús „zsidó holokausztról”, sőt, tagadását újabban törvény is bünteti. Az Amerika-dominálta világ viszont szemérmesen elfordul akkor, ha valaki indián vagy őslakos holokausztról kezd el beszélni, amely tömeggyilkosság méreteiben minden bizonnyal vetekszik a második világháborússal, csakhogy azt az Amerikába özönlő spanyol, angol és francia népek hajtották végre csupán pár száz éve. Enyhítő körülmény, hogy a halálok az esetek nagy részében betegség, amelyek kórokozóit a fehér felfedező társaink hurcolták magukkal tudtukon kívül – és amelyek ellen az amerikai őslakosság védtelen volt.

A tengeri joghatóság félig-meddig legálissá tette a kalózkodást – még pár száz évvel ezelőtt is jogilag elfogadható volt Kolumbusz eljárásának követése – valamely állam fennhatóságát elismerő kalózokat állami papírok védték, amelyek megengedhetővé tették számukra az ellenséges államok hajóinak – a magánhajóknak is a megtámadását. A tengeri- vagy kalózjog egy érdekes utolsó szimbólumaként foghatjuk fel néhai Neil Armstrong negyven éve a Holdra kitűzött amerikai zászlóját, vagy az egyes bolygók (pl. a Hold) területére manapság tulajdonjogot formáló és bejegyeztető polgárokat. Végső soron pedig a gyarmatosítási törekvéseket megalapozó jogrend utolsó mohikánjaiként tekinthetjük a még mindig létező, közös, és szabályozatlan hozzáférésű erőforrásokon (levegő, víz, föld, stb.) meggazdagodó egyes világcégeket is, de ugyanebbe a kategóriába tartozik a „jogrendet” és a „szabadságot” az egyetlen lehetséges módon, szőnyegbombázással terjesztő világcsendőrállam buzgalma is, mihelyt olyan helyen szól be valaki kicsit keményebben, ami kőolajforrások közelében található.

1.1 Problémafelvetés

A jog rég elszakadt a népességtől, és a közfelfogástól is. A rendszer elképzelhetetlen sebességgel fejlődik, változik, egyre bonyolódik és többszöröződik, egyre nagyobb méretű és áttekinthetetlenebb lesz. A söralátétre ráírható adóbevallás egy még érthetetlenebb és áttekinthetetlenebb nyilatkozat-saláta árán valósítható meg. A „törvény nem ismerete nem mentesít a felelősségre vonás alól” alapelv már nagyon régóta tarthatatlan, illetve csak egyes hatósági önkényeskedésekre szolgálhat egyedül mentségül. A jog eredeti szerepe az lett volna, hogy az élet menetét mederben tartsa – ma inkább szlalompályaként, porcelánboltként tekinthetjük, és bizony igen ügyesnek kell lennie még az átlagembernek is ahhoz, hogy kiselefánt módjára egyetlen véletlen és vétlen farokcsapással le ne sodorjon a polcról, neki ne ütközzön, fel ne borítson alapvetőnek tekintett törvényi akadályokat.

Sajnos a robbanásszerű gyarapodás – az akár napjainkban is tapasztalható törvénygyár – egyik velejárója (ez nemcsak jogi területen szokott így lenni) – a zűrzavar: már egy-egy törvényen belül is, de egyes törvények között még inkább szaporodnak a hibák, a pontatlanságok, az ütközések, a lefedetlen rések.

Mi a szerepe ezek területén egy számítás- és rendszerelmélet tudósnak?

A jogi átláthatatlansággal és áthatolhatatlansággal szemben kétféle módon lehet fellépni: egyrészt tudatos és tervezett bozótirtással, gyomlálással és egyszerűsítéssel,

ami leginkább jogászi feladat. Másrészt pedig számítástechnikai eszközökkel, ami számítástechnikusi feladat.

Bár a szerző meggyőződése, hogy az előbbi, a *jogtisztítás* lenne igazán fontos és hatékony megoldás, a szerzőnek – az előképzettségénél fogva a második témában, a jogi szakterület számítógépes kezelésében van mit mondania. Ez az értekezés tárgya is.

A jogtisztítás azt jelenti, hogy megvizsgáljuk ezeknek a bonyolult jogi rendszereknek a szerkezetét, és a bonyolultság feloldására teszünk javaslatot. A nagyon bonyolult rendszerek bonyolultsága általánosságban többféleképpen is feloldható.

Egyfelől a rendszerek *felbontással, dekomponálással* részrendszerekre, alrendszerekre bonthatók. Az alrendszerek olyan rendszerek, amelyeknek elegendő a felületét, a kölcsönhatásait ismerni, a belső működésüket nem szükséges megértenünk ahhoz, hogy a belőlük felépített szuperrendszerek mégis érthetők, leírhatók és modellezhetők legyenek.

Másfelől a rendszerek valószínűségi elemzésével az elemeikre valószínűségi mértékek fektethetők. A kódtömörítő programok úgy működnek, hogy a szöveges állományok gyakran előforduló betűihez (pl. a magyarban az „e") rövidebb kódot rendelnek, míg a ritkábban előfordulókhoz (q, w, y, x) hosszabbakat. Tudhat eléggé jól magyarul az, aki az idegen betűket még csak fel sem ismeri, és érthet jogi nyelven az, aki nem ismeri a legcsavarosabb és egyben legritkább törvényeket vagy jogi eseteket. Létezhet, és létezik is a nagy rendszereknek olyan felbontása, amely a valószínű eseteket kiemeli, és a ritka eseteket elhanyagolja, mondván, hogy ezzel már az egész rendszer működőképes, mert ezzel a valóságban előforduló esetek túlnyomó része már kezelhető. Ha pedig mégis valamelyik kivétel fordul elő, akkor majd szakembert hívunk. Egy ilyen felbontásnak egy nagyon fontos feltétele van: a rendszerek *káoszmentessége*. Egy rendszert akkor nevezünk kaotikusnak, ha a bemenő adatok icipici módosítására a rendszer kimenete, az eredmény lényegesen, gyökeresen, akár alapvetően is módosulhat. Ha egy pillangó szárnycsapása az egyik kontinensen, képes vihart kavarni a másikon, akkor az kaotikus. Ha egy politikus egy magánbeszélgetésen történő kicsit ügyetlen vagy pongyola (netán szándékosan félrevezető) fogalmazásmódja következményeképpen komoly tőzsdei ingadozások, piaci válságok, bedőlő hitelek alakulnak ki, akkor az kaotikus. Ha az említett ritka kivételek kezelése olyan, hogy szépen belesimul a gyakori esetek kezelési módjába, akkor az elvonatkoztatás jogos. Ha a ritka kivételek elindítják az atomrakétákat, akkor – esetleg öntudatlanul – kaotikus rendszert építettünk fel.

Meggyőződésem, hogy a fenti vizsgálatoknak, megoldásoknak és intézkedéseknek helyük van, sőt, azok előbbre valók, és csak utána – vagy legfeljebb velük párhuzamosan – kell a joghoz való hozzáférést számítástechnikai eszközökkel segíteni. Az efféle vizsgálatokhoz azonban jogász és természettudós szakemberekből álló csapatok kellenek, akik a jogrendszerre különböző metrikákat, mennyiségeket fektetnek, bonyolultságvizsgálatot végeznek, rámutatnak a bonyolultságot rejtő joghelyekre, és javaslatokat adnak a bonyolultság csökkentésére. Ha a számítástudósok törekvéseinek a jogtudósok nem mennek elébe a jogrend egyszerűsítésével, rendszerezésével, akkor már megint csak tünetileg kezelünk valamit, mert a problémát egy másik, még nagyobb problémával akarjuk megoldani.

A jelen dolgozat azonban nem ezt, a csupán látszólagosan könnyebb utat, hanem az egyre bonyolódó joghoz történő hozzáférés *informatikai megkönnyítésének* a ténylegesen is könnyebb útját próbálja kijelölni.

1.2 A dolgozat tárgya

A dolgozat tárgya: a *Számítógéppel Segített Jogkezelés* (Computer Aided Juristics, CAJ). Ennek a lehetőségeit még *messze nem merítettük ki*, a téma még nagyon sok kiaknázatlan lehetőséget rejteget. A kiaknázatlan lehetőségek elsősorban a *természetes nyelvek feldolgozását*, másrészt a *mesterséges intelligencia technológiák* alkalmazását jelentik. A kiaknázatlan lehetőségek mellett a jog tartalmi, értelemközpontú kezelése terén még igen sok *feltáratlan felhasználói igény* is létezhet. Ezek a jelen pillanatban ugyan még rejtettek, de ennek legfőbb oka, hogy a járatos technológiák efféle műveletekre ma még képtelenek. Mindezen technológiai irányok előrehaladásával *eddig számítógépes eszközökkel még meg nem valósított, sőt meg sem közelített feladatosztályok válhatnak megvalósíthatóvá.*

A számítógépes jogkezelés tekintetében felismerhető egy folyamat, amely az alkalmazott technológiák fejlődésével jellemezhető. Ez az úgynevezett mesterséges intelligencia technológiák és a természetes nyelvi feldolgozó technológiák egyfajta *alkalmazási mélységével* írható le. A kezdő technológiák, a szabad szavas keresés (freetext search) és a szóalakelemzés (morfológia) voltak; (ez utóbbi értelemszerűen csak a magyarhoz hasonló, toldalékoló, vagy agglutináló nyelvek esetén). Ezt továbbfejlesztették a tezauruszos kérdéskibővítés megoldásával, amely megoldás még szintén a piacra került. Az ezután következő nemzedék a korlátozott (természetes) nyelvtani (sekély-) elemzéssel (Part Of Speech/POS Tagging), a dokumentumok XML alapú szöveges adatbázisban történő tárolásával működött. Később a pongyola szemantikájú tezauruszok helyett esetleg kicsit feszesebb ontológiák alkalmazásával, ahol még nem volt követelmény az ontológiákban ábrázol szemantikai viszonyok gazdagsága. Ezekben a témákban már lezárt és eredményes kutatások folytak, a konkrét piaci bevezetésről (egyelőre) mégsincs híradás. A technológiák harmadik nemzedéke a nyelvi mélyelemzéssel és az ontológiák még részletesebb és pontosabb alkalmazásával, ontológiai következtetésekkel függ össze. Ez még erősen kutatási fázisban van, a következő években a technológiák területén esetleg várható komoly áttörés egyben áttörést jelenthet az alkalmazói világban, a piacra kerülő szoftverek világában is. Ez az értekezés tárgya: a szerző elvégzett kutatásai, amelyeket elsősorban a technológiák harmadik nemzedékében folytatott.

1.3 A munka módszere

A vázolt problémák megoldása több-tudományszakos (interdiszciplináris, de inkább multidiszciplináris) megközelítést igényel: a megoldás valahol a számítógépes nyelvészet, a matematikai logika, az informatika és a jogtudomány közös határterületén mozog. Az imént körvonalazott hipotézist a következő munkamódszerrel lehet alátámasztani:

1. Megvizsgáljuk, hogy mi volt a természetes nyelvek feldolgozási technológiáinak az eddigi fejlődése, mi az, ami „state-of-the-commerce”, és mi az, ami még csak „state-of-the-art” állapotú. Rámutatunk azokra a pontokra, ahol újszerű, vagy eddig még nem alkalmazott megoldások áttörést vagy legalábbis komoly előrehaladást jelenthetnek.
2. Megvizsgáljuk ugyanezeket a mesterséges intelligencia-technológiákkal kapcsolatban is.
3. Megvizsgáljuk, hogy hasonló kutatási projektek, esetleg termékek esetében milyen eredményeket célszerű átvenni. Itt elsősorban a korábbi kutatások létrehozta szabványok figyelembe vétele és átvétele, illetve egyes, a szabványok

kezelésével, ellenőrzésével és minőségbiztosításával összefüggő szoftvertermékek átvétele lehet érdekes.

4. Saját kísérleti tanulmány-szoftverek kifejlesztése olyan esetekben, ahol ez az elméleti indoklásnál egyszerűbbnek tűnik, vagy ahol a szoftver-elem kifejlesztése további más szoftverek fejlesztéséhez nyit kaput (kulcs-szoftverek). Nem cél, és nem is várható el, hogy az értekezés mellékeredményeképpen egy akár csak prototípus állapotú, teljes jogi tudáskezelő szoftver létrejöjjön.
5. Az eddigi pontokban leírt vizsgálatok és kifejlesztett szoftverek meghatároznak egy szoftver felépítményt, és egy műveletkört. Az eredmények kiértékelésével meg kell adni a megvalósítható műveletek körét, és rá kell mutatni azokra a feladatokra is, amely kihívásaira a vázolt felépítmény nem tud választ adni.

1.4 Az értekezés tagolása

Az értekezés – leszámítva a Bevezetés, Következtetések, Irodalomjegyzék és hasonló, formális fejezeteket, három tartalmilag jól elkülöníthető fejezetre bomlik.

A Bevezetést követi a 2. sorszámú: „Mesterséges intelligencia logikai programozási megközelítésben" c. fejezet. Ebben a mesterséges intelligencia tudományágának a dolgozatban követett paradigmájáról lesz szó. Kicsit szerényebb kifejezéssel élve: a szemantikus modellezés alapjait, a tágan értelmezett logikai programozás tudományágát tekintjük át. Ennek során – bár precíz tételbizonyítások igénye nélkül, mégis bemutatjuk a logikai programozási megközelítés matematikai-logikai alapjait: kezdve a klasszikus ítéletlogikától az elsőrendű logikán át a Szemantikus Háló projektkezdeményezésben ontológia-leírásra használt leíró logikákig,[12] és az általunk ezen túl használni kívánt modális logikákig. Úgyszintén bemutatjuk az ezekkel összefüggő két legismertebb tételbizonyító algoritmus működését: a rezolúciós tételbizonyítást, amelyet általános elsőrendű logikákra alkalmazhatunk, illetve a leíró logikákra alkalmazható tabló algoritmust.

Mindazonáltal ez a fejezet nem csak a matematikai logikai megalapozásról szól: előkerülnek benne a szűken értelmezett logikai programozás (a Prolog programozási nyelv és kiterjesztéseinek) témakörei is,[3] majd az objektum-orientált programtervezési és -készítési megközelítés logikai alapjait tekintjük át. Ez szorosan kötődik a szoftver modellek, metamodellek és meta-metamodellek kérdésköréhez, majd a fejezetet az ontológiák és szoftver környezetüknek a leírása zárja.

Az ezt követő (3. sorszámú) fejezetben az alkalmazható nyelvtechnológiai és szemantikus alapú megoldásokat, illetve a már létező kísérleti megvalósításokat mutatjuk be. Ideértünk mindent, az egyszerű keresőprogramok működésével kezdve, a magyarhoz hasonlóan gazdagon toldalékoló nyelvek helyesírás-ellenőrző programjain keresztül, a különböző korlátozott nyelvi szerkezeteket elemző programokon át, a legmodernebb, diskurzus-szemantikai technológiát alkalmazó programok elemző- és feldolgozó-algoritmusainak működéséig. Itt alapozzuk meg és mondjuk ki a dolgozat egyik legfontosabb tézisét: *a jogi informatikában alkalmazható informatikai módszereket elsősorban az alkalmazható nyelvi és mesterséges intelligencia*

---

[1] T. Berners-Lee - J. Hendler - O. Lassila The Semantic Web Scientific American Magazine. 2001. [BLHL01]

[2] Szeredi P. - Lukácsy G. - Benkő T.: A szemantikus világháló elmélete és gyakorlata [SzLB05]

[3] W. F. Clockshin-C. S. Mellish: Programming in Prolog [CloMe]

technológiák fejlettsége, működőképessége és megbízhatósága határozza meg. Továbblépni – sőt, áttörést elérni elsősorban az alkalmazott technológiai megoldások tökéletesítésével lehet.

Számos kutatási projektum dolgozott már szemantikus modellezési technológiákkal a jogi területen. Ebben a fejezetben többek között egy ilyen projektumot, az Estrellát is bemutatjuk.[4] Az önálló munka során teljesen felesleges mindent az alapoktól elölről kezdeni: a már elért eredményeket persze lehet kritikai elemzés alá venni, mégis a legcélszerűbb a már meglevő eredményeket kiindulópontként felhasználva felépíteni az újabb eredményeket. Azért éppen az Estrellát mutatjuk be részletesebben, mert meggyőződésünk szerint ez a projektum választotta a dolgozatban alkalmazott megközelítéshez legközelebbit. Egyik tézisünk és javaslatunk éppen ezért: *jogi szemantikus rendszerek létrehozásakor az Estrella projektum eredményeit érdemes átvenni*.

Az utolsó (4. sorszámú) fejezet a ténylegesen elvégzett szoftver-tervező, nyelvi elemzési és szemantikus modellépítő munka gyakorlati tapasztalatait és eredményeit mutatja be. Itt számolunk be többek között egy közlekedési-jogi ontológiaépítési kísérlet eredményéről, amely konkrét közlekedési adatokra és egy részletes, közlekedési információt is tartalmazó térkép-adatbázisra építve eldönti, hogy a jármű vajon áthágta-e a közlekedési szabályokat. A szakaszban szót ejtünk a rendszer építéséről és a modell alkalmazási lehetőségeinek gyakorlati kilátásairól.

Ugyanitt írunk a diskurzus-szemantikai alapon működő ℜeALIS nyelvi elemzőprogram[5] alkalmazási tapasztalatairól is. Egy teljes elemzőprogram az értekezés lezárásakor még kifejlesztés alatt áll: a dolgozatban az ezzel kapcsolatban létrehozott megvalósítási tanulmányprogramok tapasztalatait tudjuk csak összefoglalni.

[4] www.estrellaproject.org, Elérés: 21-Dec-2011

[5] Alberti Gábor: ℜeALIS. Interpretálók a világban, világok az interpretálóban [AG11]

2 Mesterséges intelligencia logikai programozási megközelítésben

A Csinált Értelem problémájának körülhatárolására több, különböző indítékú meghatározás született már. A korai, leggyakorlatiasabb változat szerint ez egy kutatási irány, melynek a tárgya a számítógéppel az akkori helyzetben még meg nem oldható feladatok köre. Ez a lista eleinte nem volt rövid, és az egyes elemeiről egyértelmű volt, hogy itt-e a helyük.

Mára a helyzet változott. Egyrészt a lista rövidül – kutatóink fáradhatatlan tevékenységével ismét újabb és újabb, mindaddig megoldhatatlannak hitt problémáról derül ki, hogy akár megoldható is lehetne, ha…, de…, csak…, miközben megoldások, termékek is születnek, amelyek általában nem is tökéletesek, nem is egyértelműek, és gyakran komoly alkalmazási korlátaik is vannak (nézzük például a csapnivaló, de mégis használt internetes fordítási szolgáltatásokat). Vagyis a lista egyrészt rövidül, másrészt az egyes elemek helyzete is gyengül.

A listában valaha megtalálható feladatok jó része valamilyen célfeladattá zsugorodott – így számos konkrét képfeldolgozási vagy jelfeldolgozási feladat ma megoldható, hiszen OCR szoftverek vagy beszéd-előállító, illetve bizonyos beszédfelismerő rendszerek mára a piacon kapható eszközök közé kerültek.

A Csinált Értelem egyik közismert definíciója Alain Turing (1912-1954) angol matematikustól származik. A Turing-próba néven is ismert eljárás lényege: egy számítógép-terminál (ügyfélprogram) társaságában ülünk egy elzárt szobában, és alapvetően parancssoros jellegű írásbeli párbeszédet folytatunk (pl. egy csevegőszobában csevegünk). Hogy kivel? A billentyűn beütjük a mondani- vagy kérdeznivalónkat, amelyet a hálózat valahová továbbít, majd valaki vagy valami, valahol máshol válaszol a kérdésünkre, reagál a felvetésünkre. Ha a kapott válaszokból nem tudjuk eldönteni, hogy tényleg gép válaszolt-e, vagy a gépben benn ült a mechanikus török (aki Kempelen Farkas sakkozógépét is mozgatta), akkor a Csinált Értelem tökéletes. Ha a gépek kiállják Turing próbáját, csak akkor mondhatjuk, hogy megálljunk, mert itt van már…

A Csinált Értelem kérdéseinek középpontjában sokak szerint a Modellkészítés áll. Természetesen, ha az Emberi Értelmet géppel akarjuk utánozni, ahhoz modellezni kell tudnunk. Hogy mit? Az Embert, az Értelmet és a körülvevő Világot. Ha a már említett célfeladatokat a modellkészítés összefüggésében keressük, akkor elegendő egy általános modellkeret konkrét eseteire alkalmazható szakmai modelleket felépítenünk. Ilyen általános kereteket megadó célokat a „logikai programozás”, az „objektumorientált szoftverkészítés” és a „nagy párhuzamosságú rendszerek” megoldási irányvonalai tűztek ki. A jelen dolgozat egyértelműen az előbbi, a logikai utat járja, kézenfekvő hát, hogy a logikai alapfogalmak tisztázásával kezdődjék.

2.1 Matematikai logikai alapfogalmak

A logika minden tudományág ősatyja. Tudományosabban „metatudománynak” nevezik az ilyesmit, ami azt jelenti, hogy a logika eszközeit, vagy legalábbis annak elemeit, részeit gyakorlatilag minden tudományág alkalmazza, a természettudományok ugyanúgy, mint a társadalomtudományok, netán a gazdaságtudomány vagy a jogtudomány. A logika a tiszta és következetes gondolkodás tudománya, amit egyetlen

tudományág sem nélkülözhet, tételeinek kimondásakor és bizonyításakor szándékosan vagy csak ösztönösen minden tudományág logikai lépéseket tesz, a logika szabályait alkalmazza.

Ennek ellenére a logikának vannak tipikus alkalmazási területei.

A matematika sokféle ága a logika eszközeit alkalmazza. A logika már-már matematika, a matematika egyik részterülete. A matematikában, de az egyes természet-tudományokban is a logikát, mint tételek bizonyításának a módszerét és keretét alkalmazzuk.

Az alkalmazott nyelvészeti tudományokban, a természetes nyelvek számítógépes leírásában szintén a logika eszközeit, a magasabb szintű, intenzionális és modális logikákat alkalmazzuk úgy, hogy a nyelvi kifejezéseket az elemzés során általában valamiféle logikai formára alakítjuk.

Az elektronikus technológia, a digitális áramkörök tervezése az elemi logikára, az ítéletlogikára épít.

Végül, de nem utolsósorban, a jogi tudományokban és a jogi napi gyakorlatban is elengedhetetlen, hogy a jogász ne csak ösztönszerűen, hanem tudatosan is alkalmazza az érvelés és a bizonyítás logikai lépéseit, illetve a természetes nyelvekbe bújtatott logikai kétértelműségekről elegendően pontos képe legyen, és efféléket a jogi iratokba csak egyértelműsítve szerkesszen be.

2.2 A logika története

A logika klasszikus tudományág. Ez is, mint oly sok más az ókori görögökkel kezdődött. Arisztotelész Kr.e. 365-340 körül hozta nyilvánosságra „Organon" c. művét, amellyel lefektette a klasszikus logika és a klasszikus görög érveléstudomány alapjait. Ekkor még nem beszéltek matematikai logikáról: Arisztotelész maga a tudományát analitikának nevezte. A logika csupán a filozófia, a retorika része és alkalmazott eszköze volt, és mint a legfontosabb alapelv: nem tartalmazott önellentmondást, ellentmondásmentes volt.

A logika művelését a középkor viharaiban elhanyagolták. A XIX. században a mára matematikai logikaként is ismert ág fejlődése George Boole (tőle származik a Boole-algebra elnevezés), Ernst Schröder, Gottlob Frege és más matematikusok munkássága nyomán kezdődött el. Talán a legmegdöbbentőbb felfedezés ebből az időszakból a paradoxonok felfedezése és vizsgálata volt.

A logikával foglalkozó tudósok azóta sem lankadtak. Itt és most talán három tudós nevét érdemes megemlítenünk, közülük ketten magyarok voltak:

1. Richard Montegue amerikai logikus és filozófus-kutató, aki a természetes nyelvű szövegek logikai értelmezésének alapjait fektette le a múlt (XX.) század közepén.
2. Ruzsa Imre (1921-2008) Széchenyi-díjas filozófus, az ELTE Bölcsészkar professzora volt, aki nemzedékeket vezetett be a logika rejtelmeibe.[6] Érdekes, hogy a bölcsészkaron egy matematikai módszereket alkalmazó tudomány kaphatott ily módon helyet.

[6] Ruzsa Imre: Klasszikus, modális és intenzionális logika [Ru84]

3. Solt Kornél (1917-2002), aki az ELTE Jogi Karán tanított, a logika jogi vonatkozásait kutatta, és jogásznemzedékeket oktatott logikára, a logika jogi környezetben történő alkalmazására és jogbölcseletre.[7]

2.3 Klasszikus logika

A klasszikus logika alatt a görög klasszikusok által lefektetett elvekre épülő, azt csupán kiteljesítő, illetve kiegészítő logikai rendszereket értjük. A klasszikuson túl a következő logikai irányzatokat érdemes megkülönböztetni és megemlíteni:

1. *fuzzy logikák* esetében a logikai kifejezések értéke nem csak igaz vagy hamis lehet, hanem még más is.
   - A *diszkrét fuzzy* vagy *többértékű (multi-valued)* logikai rendszerek esetében az *igaz* és *hamis* logikai alapértékeket továbbiakkal egészítjük ki (pl. talán, is, hibás, stb.).
   - A *folytonos fuzzy* logikai rendszerek esetében egy logikai érték az igaz (1) és a hamis (0) között minden lehetséges értéket felvehet, sőt, esetleg ezekre még valószínűségi mértékek is igazak lehetnek.
2. *modális logikák* esetében a klasszikus logikai mondatokat modális kiegészítőkkel (módhatározókkal, pl. lehetséges, szükségszerű, tudott, stb.) látjuk el.
3. az *intenzionális logikai rendszerek* esetén egy logikai kifejezés értéke egyéb körülményektől függően más és más lehet, ezért a kifejezésnek nem csak az értékére vagyunk kíváncsiak – hiszen az változhat – hanem maga a kifejezés szerkezete az érdekes számunkra, amelyet időről időre ismételten kiértékelhetünk.

A klasszikus logika alapvetően mondatokról, kijelentésekről, ítéletekről szól. A mondatok közül megkülönböztetjük az úgynevezett *nyílt* és *zárt mondatokat*. Zárt mondatok azok, amelyeknek az igazságértéke egyértelműen megállapítható. Nyílt mondatok esetében a mondatok olyan hivatkozásokat (pl. névmásokat) tartalmaznak, amelyek más mondatokra, korábbi információkra utalnak, ezért önmagukban nem vizsgálhatók, nem értékelhetők ki.

Pl. „a viszkis rablót többrendbéli bankrablással vádolják” mondat zárt mondat, hiszen az igazságértéke – a „viszkis rabló” utalás egyértelműsége miatt eldönthető. A „már több éve tölti a börtönbüntetését” mondat viszont nyílt mondat, hiszen csak egy másik mondat (pl. az előző mondat) környezetében értelmezhető egyáltalán.

Egyes ítéletek értéktartalma nagymértékben függ a környezettől, időponttól és egyéb körülményektől. Pl. „az amerikai elnök színesbőrű” mondat értéke 2012-ben, Barack Obama elnöksége idején igaz. A közvetlen kiértékeléssel kapott értéket a mondat *extenziójának* is nevezzük. A mondat Obama megválasztása előtt még sosem volt igaz, és nem tudhatjuk, hogy az, aki a következő választáson elnyeri az elnöki címet, vajon színesbőrű lesz-e vagy sem. Vagyis a mondat extenziója nagymértékben függ a kiértékelés időpontjától és az aktuális körülményektől (esetünkben az aktuális amerikai elnök személyétől). A mondatok tartalmát, vagyis a mondatokba rejtett kiértékelési utasítást (tekintsük az aktuális amerikai elnököt, és vizsgáljuk meg, vajon színes vagy fehér bőrű-e) a mondat *intenziójának* nevezzük.

[7] Solt Kornél: Jogi logika [SoKo96]

A klasszikus logikában csupán zárt mondatok *extenzióját*, azaz a konkrét logikai értékét vizsgáljuk, és a kifejezések szerkezeti felépítésének nem tulajdonítunk jelentőséget. Ezzel összefüggésben a következő alapfeltételezések igazak:

1. Minden mondatnak vagy igaz, vagy hamis igazságértéke lehet. Más értéke nem lehet. Ezt a *kizárt harmadik elvének* is nevezzük.
2. Egyetlen mondatnak sem lehet egyszerre igaz és hamis igazságértéke. Ezt az *ellentmondás-mentesség elvének* is nevezzük.
3. Ha egy mondat értéke nem hamis, akkor a mondat igaz. Ezt a *kettős tagadás elvének* is nevezzük.

2.3.1 Ítéletlogika

Az ítéletlogika a klasszikus logikai kérdéskör legegyszerűbbje, ezért a logika tanulását itt érdemes kezdeni. A logika általában a természetes nyelven megfogalmazott állításokat és a belőlük levonható következtetéseket vizsgálja. Az ítéletlogika az állítások mélyszerkezetét nem vizsgálja, csupán azok felszíni szerkezetét, az *állításokat* és az azokat összekapcsoló *logikai alapműveleteket,* vagy *logikai kapcsolókat* veszi figyelembe.

Az egyes logikai alapműveleteket szokásos *értéktáblázattal* megadni. Az értéktáblázat rögzíti, hogy az összekapcsolt részítéletek összes lehetséges igaz/hamis értékvariációja esetében az összekapcsolt ítélet értéke milyen lesz.

Az alább olvasható összefüggések az egyszerű részítéletek konkrét tartalmától függetlenek, ezért gyakran őket tartalom nélkül – csupán betűkkel vagy emlékeztető jelekkel ábrázoljuk. Az ilyen ábrázolások esetén a részítéleteket szokás *logikai változónak* is nevezni.

2.3.1.1 Logikai értékek

Mint már korábban említettük, *kétértékű logikában* gondolkodunk, ahol csak az *igaz* és *hamis* logikai értékek lehetségesek. Ezeket sok helyen a kezdőbetűikkel, *i* és *h* betűkkel jelölik. Más irodalmak az angol megfelelőiket (*true, false*) használják, esetleg szintén csak a kezdőbetűikkel, *t* és *f* betűkkel jelölve. Szokásos az igaz és hamis értékeket számjegyekkel is jelölni: ilyenkor a 0 számjegy hamist, az 1 számjegy pedig igaz értéket jelent.

2.3.1.2 Logikai negáció

A legalapvetőbb logikai kapcsoló a *negáció*, vagy *tagadás*, amely a következő értéktáblázattal adható meg. A negáció jelölése rendkívül változatos: talán a leggyakoribb a '~' előképzős operátor használata, de helyette néha „–" (mínusz) jelet alkalmazunk. Használatos még a föléhúzás jelölés is.

A	**~A**
1	0
0	1

A negáció egy ítélet hamisságát mondja ki, és a természetes nyelvű tagadásoktól és egyéb hangsúlyozástól eltérően nem utal arra, hogy a hamisság oka miben keresendő. Pl. ha nem áll fenn a „Jácint Gabi házastársa" ítélet, abból nem következtethetünk sem arra, hogy Gabinak esetleg másik házastársa lehet, sem arra, hogy esetleg valamilyen másfajta viszony lenne kettőjük között, sőt még a Gabi szereplő nemére sem.

2.3.1.3 Logikai diszjunkció

A logikai *diszjunkciót* (*megengedő*, inclusive) *VAGY* műveletként is ismerjük, és „∨” jellel jelöljük. A művelet két ítéletet kapcsol össze (kétparaméteres), az eredménye akkor igaz, ha valamelyik rész- (paraméter-) ítélet igaz volt. Ez a meghatározás nem zárja ki azt, hogy akár mindkettő ítélet igaz legyen.

A∨B	0	1
0	0	1
1	1	1

Például „a bűncselekmény indítéka szerelemféltés vagy haszonszerzés volt” állításban a VAGY kapcsoló nem zárja ki, hogy akár mindkettő is igaz legyen: ha a tettes meg kívánta szerezni szerelmi riválisának bizonyos vagyontárgyait is.

A VAGY kötőszó használata a hétköznapi nyelvben meglehetősen többértelmű. Pl. az „út közepén műszaki okból álló gépjárművet jobbról vagy balról is kikerülhetjük” állításban a kötőszót kizáró értelemben használjuk, vagyis mindkét feltétel egyszerre nem teljesülhet.

Kérdő mondatokban a VAGY kötőszót gyakran kizáró alternatívák közötti választást eldöntő, kiegészítendő kérdésként értelmezzük. A Mérő László klasszikusában[8] olvashatók szerint a világhírű mesterséges intelligencia tudós az űrverseny időszakában a következő kérdést intézi a mindentudó számítógéphez:

„A Szovjetunió vagy az USA juttat embert először a Marsra?”

Erre a tudós gép – a VAGY kötőszó tisztán logikai értelmezésével az eldöntendő kérdésre a következőt válaszolja:

„Igen, uram.”

2.3.1.4 Logikai konjunkció

A *logikai konjunkciót ÉS* műveletként is ismerjük, és „∧” jellel jelöljük. A művelet két ítéletet kapcsol össze (kétparaméteres), az eredménye akkor igaz, ha mindkét paraméterítélet igaz volt.

A∧B	0	1
0	0	0
1	0	1

Például a „tettest garázdasággal és hivatalos személy megsértésével vádolják” mondatban a két vádpont egyidejűleg igaz: a teljes mondat akkor igaz, ha a szóban forgó személy ellen mindkét vádpont körében eljárás indult.

2.3.1.5 Egyenértékűség

A *logikai egyenértékűséget* (*ekvivalenciát*) „=” jellel jelöljük. A művelet két ítéletet kapcsol össze (kétparaméteres), az eredménye akkor igaz, ha mindkét paraméterítélet értéke egyforma volt – akár mindkettő igaz, akár mindkettő hamis.

[8] Mérő László: Észjárások [Mé89] 40. old.

A=B	0	1
0	1	0
1	0	1

Szabad szövegben a kapcsolót gyakran „akkor és csak akkor” kifejezéssel jelölik. A matematikusabb változat: „annak szükséges és elégséges feltétele”.

Például: „valaki akkor és csak akkor halott, ha nem mutat életjelenségeket”. Az életjelenségek megszűnése teljes mértékben megegyezik a halál beálltával, a két megállapítás egyidejűleg és csak egyidejűleg lehet igaz.

2.3.1.6 Kizárás (kizáró, exclusive VAGY)

A *kizárás* művelete a logikailag különböző értékek esetén ad igaz eredményt. A műveletnek többféle jelölése is van. Jelölhetjük „≠” (nem egyenlő) jellel, mások gyakran „+” (plusz) jellel jelölik, esetleg a plusz jel köré még egy kört is írnak.

A≠B	0	1
0	0	1
1	1	0

A „+” (plusz) jeles jelölésmód háttere: egybites bináris számok összeadása a kizáró vagy műveletnek felel meg, és a számítógépek aritmetikai egységében éppen így van megvalósítva. Pl. „az amerikai elnök republikánus” + „az amerikai elnök demokrata” állítás értéke Igaz, hiszen a pártállás egymást kizáró fogalom, párttámogatás nélkül nemigen lehet Amerikában választást nyerni. Vagyis a két részállítás közül az egyik teljesen biztosan igaz, a másik viszont hamis. Már a megengedő VAGY-nál is említettük, hogy a hétköznapi nyelv gyakran – vétkesen – összemossa a kétféle művelet közötti különbséget. Kérdések esetében a VAGY kötőszót leggyakrabban kizáró VAGY értelemben, sőt, kiegészítendő kérdésként is felfoghatjuk, amikor a kérdező arra kíváncsi, hogy a két, egymást kizáró lehetőség közül melyik valósul meg. Állító/kijelentő mondatok esetében a kizáró értelmezést a kötőszó megduplázásával „vagy ez vagy az” érjük el, de ha még ez sem elegendő, akkor a két lehetőség kizárólagos voltára utaljuk külön részmondattal!

2.3.1.7 Implikáció

Az *implikáció* a hétköznapi logikában *következtetésként* ismert művelet logikailag denaturált alakja. A művelet nem szimmetrikus: egy F feltételrészt és egy K következményrészt kapcsol össze. Az implikáció egyedül akkor hamis, ha a F feltétel teljesül, ám a K következmény mégsem. Minden egyéb esetben a művelet igaz értéket ad.

Az implikációt legtöbbször „→” nyíl jellel jelöljük (lehet fordított is), amely a feltételrész felé mutat.

F→K	0	1
0	1	1
1	0	1

A fenti pontos logikai definíció sok szempontból nem felel meg a hétköznapi felfogásnak. Lehetetlen feltételből eszerint ugyanis bármilyen következmény levezethető, vagyis: „ha a hó piros, akkor én vagyok a váci püspök” állítás logikailag kifogástalan és Igaz értékű.

A másik ilyen anomália szerint a F→K következmény semmit sem mond az *ok-okozatiságról*, a két dolognak semmi köze sincs egymáshoz, jóllehet a hétköznapi logika a következtetéshez valami efféle háttérjelenséget is társít. Pl.: „Ha fellebbezés történt, akkor megszületett az ítélet" ennek tipikus példája. Ki van zárva az az eset, hogy megfellebbeztek valamit, noha még ítélet sem született – miközben az ok-okozati összefüggés éppen a mondat ellenkezője – előbb születik meg az ítélet, és csak utána fellebbeznek.

Az implikációnak egy másik meghatározása is lehetséges: a két meghatározás egyenértékűségét pedig bizonyítani tudjuk (lásd később). Eszerint F→K implikáció igaz akkor, ha a következmény igaz, vagy a feltétel hamis.

Az F→K implikáció egy harmadik megfogalmazása szerint: „K az F-nek *szükséges feltétele*". Ebben az ok-okozatiság már teljesen felborul: a K következményoldal szükséges feltétele az F-nek, a feltételoldalnak! Az implikáció azonban éppen ezt fejezi ki: Ha a K következmény nem teljesül, akkor az F feltétel nem állhat fenn. Vagyis a F csakis olyankor teljesülhet, ha K is teljesül.

Jogi példát tekintve: „A keresetlevél benyújtása az ítélet szükséges feltétele" Ha ítélet született, akkor korábban keresetnek is kellett lennie.

A szükséges feltétel, vagyis az implikáció nem szimmetrikus művelet, azaz nem fordítható meg a részek sorrendje anélkül, hogy az eredmény logikai értéke megváltozzon. A fenti példában: „Ha keresetet adtak be, akkor ítélet születik…" állítás nem igaz, hamis, hiszen a kereset beadása után még ezernyi olyan ok és körülmény is fennállhat, amely meghiúsíthatja az ítélet megszületését.

Az *elégséges feltétel* a szükségesség megfordítottja az implikációs szerkezetben. Miközben az F→K implikációra nézve K az F-nek szükséges feltétele, addig F a K-nak elégséges feltétele. F fennállása elegendő ahhoz, hogy K is fennálljon, de K esetleg más esetekben is fennállhat.

2.3.1.8 NAND

Különböző célokra egyéb műveleteket szokás használni:

A *NEMÉS* (*NAND*) művelet meghatározása: ~(X ∧ Y), de a meghatározásnak megfelelő értéktáblázat is könnyen felírható.

X NAND Y	0	1
0	1	1
1	1	0

A műveletet a logikai áramkörtervezésben használják/használták nagyon intenzíven, mert a számítástechnika őskorában a legegyszerűbb elektronikus logikai áramkörök (TTL kapuk) NAND műveletet valósítottak meg – egy NAND egyetlen tranzisztorral létrehozható volt.

2.3.1.9 NOR

A NAND-hoz hasonló a NOR (NEMVAGY) művelet ~(X ∨ Y). Értéktáblázata a következő:

X NOR Y	0	1
0	1	0
1	0	0

A NOR művelet szintén a számítástechnikai ősiparban volt használatos, a jelentősége (a NAND-dal) együtt a maguk korában hatalmas volt, de manapság kicsit háttérbe szorultak már.

2.3.1.10 Bináris logikai műveletek száma

Az eddig bemutatottakon túl még van néhány bináris logikai művelet. Felmerülhet a kérdés: van-e a lehetséges kétparaméteres bináris logikai műveleteknek valamiféle számossága? A válasz eléggé kézenfekvő, és az értéktáblázatos ábrázolás kismérvű átalakításából szinte le is olvasható. Ábrázoljuk az összes lehetséges bemenő értékpárt egyetlen koordináta mentén, és az összes lehetséges műveletet pedig a másik mentén. Az összes lehetséges művelet száma az értékpárokhoz rendelhető összes lehetséges bináris értékkel egyezik meg, vagyis:

$N = 2^{2^2} = 16.$

Művelet	00	01	10	11	Megjegyzés
Hamis	0	0	0	0	Mindig Hamis, mindkét változó fiktív
ÉS/Konjunkció	0	0	0	1	NAND negáltja
	0	0	1	0	
	0	0	1	1	Első paraméter értéke, a másodiktól függetlenül
	0	1	0	0	
	0	1	0	1	Második paraméter értéke, az elsőtől függetlenül
XOR/Kizáró VAGY	0	1	1	0	Az ekvivalencia negáltja
VAGY/Diszjunkció	0	1	1	1	NOR negáltja
NOR/Nem VAGY	1	0	0	0	VAGY negáltja
Ekvivalencia	1	0	0	1	A XOR negáltja
	1	0	1	0	Az első paramétertől függetlenül a másodikat negáljuk
	1	0	1	1	
	1	1	0	0	A második paramétertől függetlenül az elsőt negáljuk
Implikáció	1	1	0	1	
NAND	1	1	1	0	ÉS negáltja
Igaz	1	1	1	1	Mindig Igaz, mindkét változó fiktív

1. ábra Kétváltozós logikai műveletek[9]

[9] Demetrovics-Denev-Pavlov: A számítástudomány matematikai alapjai [DePa99] 40. old.

2.3.1.11 Logikai azonosságok

A fenti logikai műveletekkel általában két logikai állítást kapcsolhatunk össze. Az említett állítások azonban nem csak egyszerűek lehetnek, hanem maguk is lehetnek már más állításokból a kapcsolókkal felépített, összetett állítások. Ilyen módon egyszerű logikai állításokból a kapcsolók segítségével többszörösen összetett logikai kifejezések is létrehozhatók.

A logikai kifejezésekkel azután különböző műveletek végezhetők. Egyszerűsítések, értékőrző átalakítások, és normál formák is léteznek. Mindezek alapját a logikai azonosságok rendszere képezi.

Megjegyezzük, hogy a Boole logikát algebrának is hívják. Ez a fogalom viszont a műveletek és az alaphalmazaik közötti összefüggések általános és alaphalmaztól független tárgyalását jelenti. Vagyis az algebra közös állításokat tesz a logikai értékekre, a valós számokra, és más alaphalmazokra is. Az alább leírt bizonyítások, egy magasabb szempontból tekintve, és általános algebrai módszerekkel kezelve esetleg magától értetődők is lehetnek.

A logikai azonosságok bizonyítását kétféleképpen tehetjük meg.

1. *Értéktáblázattal*. Az azonosságokban szereplő logikai egyszerű mondatok (változók) minden lehetséges értékét figyelembe véve bebizonyítjuk, hogy az azonosságok két oldala minden változóérték esetén ugyanazt az értéket veszi fel.
2. *Azonos átalakításokkal*. Már korábban bebizonyított azonosságok felhasználásával az egyenlőségek mindkét oldalát ugyanolyan alakra hozzuk.

A legközismertebb logikai azonosságok:

1. *Felcserélhetőség, vagy kommutativitás* a diszjunkcióra és a konjunkcióra vonatkozóan.
   $A \vee B = B \vee A$, illetve $A \wedge B = B \wedge A$

   Bizonyítás: 1. A konjunkció és a diszjunkció értéktáblázatából is kitűnik, hogy a táblázat tartalma a két műveleti paraméterre nézve szimmetrikus, azok sorrendjétől független.

   Értéktáblázatos bizonyítás:

A	B	A∨B	B∨A
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

   Mint jól látható, az A és B oszlopokban felsoroljuk a változópár összes lehetséges értékét. Az A∨B, mind a B∨A oszlopban, pedig a műveletek eredményei láthatók a korábbi értéktáblázatos művelet-meghatározás alapján. Az is látható, hogy ezek az értékek teljesen megegyeznek egymással. Megjegyezzük, hogy a fenti, diszjunkcióra készített megoldás teljesen hasonlóan elkészíthető a konjunkcióra is.

2. *Csoportosíthatóság vagy asszociativitás* konjunkcióra és diszjunkcióra vonatkozólag
   $A \vee (B \vee C) = (A \vee B) \vee C$, illetve $A \wedge (B \wedge C) = (A \wedge B) \wedge C$

Műveletláncok esetében teljesen mindegy, hogy az egyes részműveleteket milyen sorrendben hajtjuk végre, vagyis mindegy az, hogy az elemeket hogyan zárójelezzük.

Az azonosság nemcsak 3, hanem bármilyen hosszú láncra igaz. A fenti, három hosszúságú lánc esetében egy 8 soros értéktáblázattal lehetne ellenőrizni az állítást, de ettől most eltekintünk.

3. *Disztributivitás (felbonthatóság, kiemelhetőség)* diszjunkcióra és konjunkcióra vonatkozólag.
   $A\wedge(B\vee C)=(A\wedge B)\vee(A\wedge C)$, illetve $A\vee(B\wedge C)=(A\vee B)\wedge(A\vee C)$
4. *De Morgan azonosságok:* Diszjunkció és konjunkció kifejezése a másikkal és negációval.
   $\sim(A\wedge B)=(\sim A\vee\sim B)$, illetve $\sim(A\vee B)=(\sim A\wedge\sim B)$
5. *Ekvivalencia kifejezése implikációval:* Két logikai kifejezés akkor egyenértékű, ha kölcsönösen következnek egymásból.
   $(A=B) = (A\rightarrow B) \wedge (B\rightarrow A)$

2.3.1.12 Kanonikus (normál) alakok

A fenti műveletekből igen sokféle logikai kifejezés létrehozható. Egy adott kifejezés azonban egyenértékű lehet más kifejezésekkel. Az egyenértékű kifejezések közül bizonyos fajtájú / szerkezetű (jól fésült) kifejezéseket kitüntetünk, amelyeket a kifejezés *kanonikus* vagy *normál alakjának* nevezünk.

A következő normál alakokat szokás használni:

$$A\ K = (L_{11}\wedge L_{12}\wedge\ldots\wedge L_{1n_1}) \vee (L_{21}\wedge L_{22}\wedge\ldots\wedge L_{2n_2}) \vee\ldots\vee (L_{m1}\wedge L_{m2}\wedge\ldots\wedge L_{mn_m})$$

…szerkezetű kifejezéseket, (konjunkciók diszjunkciója), *konjunktív normál alaknak* nevezzük. Az L_{nm} kifejezéselemeket literáloknak hívjuk, amelyek vagy egy atomi állítást, vagy egy ilyennek a negáltját jelentik. Tehát a negációkat a VAGY és ÉS műveletek hatókörén belülre hoztuk. Bebizonyítható, hogy minden logikai kifejezés átalakítható konjunktív normál alakúvá, sőt, erre akár (kielégítően gyors) számítógépes program is készíthető.

$$A\ D = (L_{11}\vee L_{12}\vee\ldots\vee L_{1n_1}) \wedge (L_{21}\vee L_{22}\vee\ldots\vee L_{2n_2}) \wedge\ldots\wedge (L_{m1}\vee L_{m2}\vee\ldots\vee L_{mn_m})$$

…szerkezetű kifejezéseket, (diszjunkciók konjunkciója), *diszjunktív normál alaknak* nevezzük. Az L_{nm} elemeket itt is literáloknak hívjuk, amelyek értelmezése a konjunktív normál alak esetében megadottal megegyezik. Itt is bebizonyítható, hogy minden logikai kifejezés diszjunktív normál alakra hozható, sőt erre gyors algoritmus is írható.

A diszjunktív normál alakból kiindulva kapjuk meg az *ítéletkalkulus klóz formáját*. Válasszuk külön egy diszjunkció tagjában a negációt nem tartalmazó, *pozitív literálokat*, valamint a negációt tartalmazó *negatív literálokat*! Jelöljük az előbbieket P_{mn}-nel, az utóbbiakat pedig $\sim N_{mn}$-nel! Ekkor egy diszjunkciós tagot a következtetés (implikáció) műveletre vonatkozó azonosság alapján $P_1\vee P_2\vee\ldots\vee P_{n_p}\leftarrow N_1\wedge N_2\wedge\ldots\wedge N_{n_n}$ alakban (nulladrendű klóz alak) írhatunk fel, a teljes kifejezés pedig az ilyen szerkezetű klózok konjunkciója lesz. A klóz alaknak különös jelentősége van a később tárgyalandó elsőrendű logikai rendszerekben alkalmazott gépi tételbizonyítási technológiáknál, különös tekintettel a Prolog logikai programozási nyelvre. Egy klóz felírásakor a Prolog

programozási nyelv a konjunkciót „**,**” (vessző) jellel, a diszjunkciót „**;**” (pontosvessző) jellel, a jobbról balra implikációt pedig „**:–**” (kettőspont-mínusz) jellel jelöli.

2.3.1.13 Teljes művelethalmazok

A fentebb megadott elemi logikai műveletek közül néhányuk csoportját (halmazát) *teljesnek* nevezzük akkor, ha a műveletek segítségével bármilyen más művelet megvalósítható. Megjegyezzük azt, hogy a megvalósítható műveletek köre tetszőleges, tetszőleges paraméterszámmal, amelyeket – a vonatkozó tételből kifolyólag mindig visszavezethetünk egy és kétparaméteres elemi műveletekre.[10]

A legfontosabb (és legismertebb) teljes művelethalmazok:

- NOT-AND: A negálás és a konjunkció segítségével minden összetett művelet megadható.
- NOT-OR: A negálás és a diszjunkció segítségével is minden összetett művelet megadható. Ezen nagyon nem kell csodálkoznunk: a de Morgan azonosságok segítségével a konjunkció negációval diszjunkcióba alakítható át és viszont.
- NOT-AND-OR: A két előbbi művelethalmaz után nem csodálkozhatunk ezen sem. Bár nem legszűkebb (hanem redundáns) művelethalmaz, talán a legnépszerűbb, és leggyakrabban használt efféle halmaz. Jelentőségét a fentebb említett diszjunktív és konjunktív normál alakok széleskörű használata adja.
- NAND: Egyetlen műveletet tartalmazó teljes művelethalmaz. A számítástechnika úttörő korszakában volt nagy jelentősége, mert igen egyszerű tranzisztoros áramkörrel megvalósítható. Elég volt tehát csak efféle áramköröket, sőt, több ilyet is egy lapkára integrálni ahhoz, hogy abból – ha elegendő van belőle – számítógépet lehessen építeni.
- NOR: Egyetlen műveletet tartalmazó teljes művelethalmaz. Bár nem nehezebb az elektronikus megvalósítása, mint a NAND-é, mégsem terjedt el igazán széles körben.

2.3.1.14 Az ítéletlogika nyelve és értelmezése

Az ítéletlogika nyelve alatt a helyesen képzett (jól formált) ítéletlogikai kifejezések halmazát értjük. Legyen adott az atomi (felbonthatatlan) kifejezések **At** véges halmaza. Az *ítéletlogika (vagy nulladrendű logika) nyelvét* $\mathcal{L}^0$-al jelöljük, és rekurzív módon az alábbi (egyszerűsített BNF) jelöléssel adhatjuk meg (a „|” függőleges vonallal alternatívát, halmaz-diszjunkciót jelölünk):

$\mathcal{L}^0 := \{\phi ::= p \mid \sim \phi_1 \mid (\phi_1) \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2\}$, ahol $p \in At$, és $\phi_1, \phi_2 \in \mathcal{L}^0$

A fenti megadásban a zárójelezés csupán a műveletek sorrendjének meghatározására szolgál, és az egész meghatározás a NOT-AND-OR teljes művelethalmazban van értelmezve. Semmi akadálya annak, hogy további műveleteket vegyünk hozzá, ily módon lehetőséget adva az implikáció (jobbra és balra is), valamint az ekvivalencia műveletének alkalmazására (mind a „=”, mind a „$\leftrightarrow$” jellel). A következő sor egy ennek megfelelő meghatározást jelenít meg.

[10] Demetrovics-Denev-Pavlov: A számítástudomány matematikai alapjai [DePa99] 49. old.

$\mathcal{L}^0$:= {ϕ ::= p | ~ ϕ_1 | (ϕ_1) | $\phi_1 \vee \phi_2$ | $\phi_1 \wedge \phi_2$| $\phi_1 \leftarrow \phi_2$ | $\phi_1 \rightarrow \phi_2$ |
$\phi_1 \leftrightarrow \phi_2$ | $\phi_1 = \phi_2$},
…ahol p∈ At, és ϕ_1, $\phi_2 \in \mathcal{L}^0$

Vagyis az $\mathcal{L}^0$ nyelv nem más, mint ϕ kifejezések halmaza, amelyek részkifejezésekből a {~ (negáció), $\wedge$ (konjunkció), $\vee$ (diszjunkció), $\rightarrow$ vagy $\leftarrow$ (implikáció), $\leftrightarrow$ vagy „=" egyenértékűség} operátorokkal építhetők fel.

Legyen Vv egy, az atomi kifejezésekből az {true, false} igazságérték-halmazba leképező függvény. Úgy is mondhatjuk, hogy Vv az atomi kifejezések *értékvektora*.

Az $\mathcal{L}^0$ logikai nyelv *értelmezése* vagy *interpretációja* alatt ezt, az $\mathcal{J}$:At$\rightarrow${true, false} az atomi kifejezésekből az igazságértékhalmazba leképező függvényt értjük. Ha tehát adott egy logikai nyev értelmezése, akkor az az atomi kifejezések értékvektorát rögzíti. Ebből kiindulva, és a korábban már említett elemi műveletek segítségével ezek után a nyelv tetszőleges összetett kifejezésének logikai értéke kiszámolható.

Egy kifejezést *kielégíthetetlennek* nevezünk akkor, ha az értéke bármely értelmezésben csak **false** értéket vehet fel, *érvényesnek* nevezünk akkor, ha az értelmezése csak **true** értéket vehet fel.

Egy Vv értelmezést a ϕ *kifejezés modelljének* nevezünk akkor, ha a behelyettesítés után a kifejezés értéke **true** lesz, ellenmodelljének, vagy cáfolatának akkor, a kifejezés értéke **false** lesz.

2.3.1.15 Következtetések és levezetések. Helyesség és teljesség.

A logikát szokás a gondolkodás tudományának is nevezni, hiszen célja, hogy segítségével a legkülönbözőbb tudományágakban jussanak új állításokhoz, illetve bizonyítsanak be megsejtett állításokat.[11]

A logika tudományának központi fogalma a logikai következtetés és a logikai következmény fogalma.

Akkor mondjuk, hogy egy F logikai kifejezéshalmaznak a ϕ kifejezés *(szemantikai/logikai) következménye*, ha az F minden modellje egyben modellje a ϕ kifejezésnek is. A logikai következmény viszony jelölése: F $\models \phi$.

A logikai következmény megkeresésére, illetve egy feltételezés következmény voltának bizonyítására számos következtetési eljárást és módszert kitaláltak már. A következtetési módszereket (az elsősorban a kifejezések szintaktikus felépítésére épülő) következtetési módszereket *kalkulusnak* is nevezzük. Egy F logikai kifejezéshalmaz és a belőle egy adott K kalkulus alapján kiszámítható ϕ szintaktikai következménye közötti viszony jelölésére a következő jelet használjuk: F $\vdash_K \phi$.

Egy kalkulus egy számítási rendszer, amelynek működése általában egyáltalán nem nyilvánvaló. Előfordulhat, hogy egy kalkulus olyan állításokat is előállít, amelyek nem következményei az alap állításhalmaznak. Az olyan kalkulusokat, amelyek efféle hibákat nem mutatnak, *helyes (sound)* kalkulusoknak nevezzük. Helyes tehát egy kalkulus, ha a következő állítás igaz:

F $\vdash_K \phi \rightarrow$ F $\models \phi$

Egy K kalkulus rögzítésekor alapvető kérdés a *helyességének* a bizonyítása.

---

[11] Zohar Manna: Programozáselmélet [ZOH81] 88. old.

Sajnos még a helyes kalkulusok segítségével sem feltétlenül állítható elő az összes logikai következmény. Az összes következményt előállítani képes kalkulusokat *teljes (complete)* kalkulusnak nevezzük. Teljes tehát egy kalkulus, ha a következő állítás igaz:

$F \models \phi \rightarrow F \vdash_K \phi$

A helyesség bizonyítása után a következő lépés a kalkulus *teljességének* bizonyítása.

A teljesség meghatározásának kétféle értelmezése lehet. Előfordulhat az, hogy bár a kalkulus elvont szinten minden logikai következmény előállítására képes, algoritmikus okokból ez mégsem teljesül. Itt gyakran nagyon konkrét számítógépes algoritmusokról van szó, és még a legtökéletesebben megtervezett kalkulus is *algoritmikus megoldhatatlansági* kérdésekkel szembesülhet. *Szigorú teljesség* alatt értjük azt, amikor a teljes kalkulushoz tökéletes, véges időben lefutó algoritmus is készíthető.

A kalkulusok levezetési szabályai elvont szinten, választási lehetőségeket nyitva hagyva vannak megfogalmazva, amelyet egy konkrét algoritmus megtervezésekor mindenképpen valahogyan el kell dönteni. Azt a kérdést, hogy egy alapalgoritmus választási pontjaiban mikor, melyik lehetőséget választjuk, *stratégiavezérlésnek* nevezzük. Sajnos az elvontabb logikai nyelvekre igaz: *semmilyen tételbizonyító algoritmushoz nem létezik olyan stratégiavezérlés*, amely minden esetben az *algoritmus sikeres befejezéséhez* vezetne – az állítás végső soron a Turing gép megállási problémából következik.[12]

Az ítéletlogikára (is) alkalmazott legalapvetőbb és legősibb kalkulus az úgynevezett *természetes következtetés* rendszere, ami helyes is és teljes is. A természetes következtetés alapján egy kifejezés levezetéséhez az eredeti F kifejezéshalmazból kiindulva levezetési szabályokat kell alkalmaznunk, azok következményállításaival az eredeti kifejezést bővítenünk kell, majd újabb levezetési szabályokat kell alkalmaznunk, egészen addig, amíg el nem értük a szóban forgó ϕ kifejezést. Az eredeti F állításhalmazból a ϕ célállításig történő eljutás teljes folyamatát *logikai levezetésnek* nevezzük.

Egyetlen ilyen logikai levezetési lépés szerint adott egy igaznak tekintett $\{p_1, p_2, \ldots, p_n\}$ *feltétel-* vagy *premisszahalmaz*, amiből egy *következmény* vagy *konklúzió* helyességére következtethetünk. Az elemi következtetési lépésekre többféle jelölésmód is elterjedt. Az írásbeli jelölésnél szokás az alkalmazott levezetési szabály rövidítését alsó indexben megadni.

$$\frac{p_1, p_2, \ldots, p_n}{k} \qquad\qquad p_1, p_2, \ldots, p_n \rightarrow_{MP} k$$

A hasonló bizonyítások logikáját már szintén az ókorban kidolgozták. Ennek során az alkalmazható *következtetési/levezetési szabályokat/mintákat* is összegyűjtötték. Ezek taglalása előtt annyit jegyezzünk meg, hogy némelyike nyilvánvaló, jól használható, és általában közismert és használt is (pl. a Modus Ponens), de vannak köztük olyanok, amelyek használhatóságára nem könnyű jó példákat felhozni (pl. a destruktív dilemma). Mindez azonban nem von le a szabályok érvényességéből és matematikai erejéből. A természetes következtetés levezetési szabályai tehát a következők:

[12] Bach Iván: Formális nyelvek [BI01] 199. oldal

Logikai elemek *bevezetése (introduction)*

- Diszjunkció bevezetése: $A \rightarrow A \vee B$ Ha egy állítás teljesül, akkor azt tetszés szerint bővíthetjük más állítások diszjunkciójával. Például: ha bizonyított, hogy a vádlott bűnös, akkor a „vádlott vagy a tettestársa bűnös" állítás is bizonyosan igaz.
- Konjunkció bevezetése: $A,B \rightarrow A \wedge B$. Ha két állításról tudjuk, hogy külön-külön igazak, akkor a konjunkciójuk is igaz. A következtetési szabály nyilvánvaló.
- Egyenértékűség/ekvivalencia bevezetése: Ha beláttuk, hogy egy $A \rightarrow B$ állítás és a fordítottja, a $B \rightarrow A$ állítások egyszerre teljesülnek, akkor az A=B állítás, illetve az $A \leftrightarrow B$ állítás is teljesül. A szabályt kiterjedten alkalmazzák a matematikában a „szükséges és elégséges" vagy „akkor és csak akkor" jellegű tételek bizonyításakor. Erre példa lehet: „Ha Jancsi felesége Juliska, akkor Juliska férje Jancsi." Ennek megfordítása: „Ha Juliska férje Jancsi, akkor Jancsi felesége Juliska". Az állítást hibátlannak, a megfordításának a kimondását szószátyárkodásnak érezzük, pedig ennek csakis a házasságra vonatkozó hétköznapi és jogi ismeretünk az egyetlen oka: zsigerből tudjuk, hogy a házasság egy szimmetrikus viszony. Egy egyszerű és általános következtetés fennállásából még egyáltalán nem következtethetünk a megfordítására is. (Pl. a „Ha Juliska ismeri Jancsit, akkor Jancsi ismeri Juliskát." állítás nem feltétlenül nem igaz.) A házasságra vonatkozó fenti két állítás azonban egyenértékű, ekvivalens; logikailag kifogástalan módon pedig a következőképpen hangzik: „annak szükséges és elégséges feltétele, hogy Juliska Jancsi felesége legyen az, hogy Jancsi Juliska férje legyen."

Logikai elemek *kiküszöbölése (elimination)*

- Diszjunkció kiküszöbölése: $(A \vee B), A \rightarrow C, B \rightarrow C \rightarrow C$ Ha egy diszjunktív állítást tudunk csak bebizonyítani, de a diszjunkció mindkét tagja ugyanazt a következményt vonja maga után, akkor nem szükséges azt vizsgálni, hogy a diszjunkció melyik tagja teljesült. Például: „ha egy baleset során személyi sérülés keletkezett, akkor rendőrt kell hívni"… másrészt: „ha egy baleset során nagy értékű kár keletkezett, akkor rendőrt kell hívni". Ha ezek után egy olyan balesetről van szó, amelyben valószínűleg nagy értékű kár, de legalábbis személyi sérülés keletkezett, akkor nem kell vizsgálni, hogy melyik indok megalapozottabb, hiszen mindkettő külön-külön is megköveteli a rendőri intézkedés megtételét.
- Egyenértékűség kiküszöbölése: $A \leftrightarrow B \rightarrow A \rightarrow B$, illetve $A \leftrightarrow B \rightarrow B \rightarrow A$. A szabály valójában két nyilvánvaló szabály; mindkettő az egyenértékűség bevezetésének a fordítottja.
- Konjunkció kiküszöbölése: $A \wedge B \rightarrow A$, illetve $A \wedge B \rightarrow B$. Ha egy konjunktív állításra jutottunk, akkor annak mindkét tagja külön-külön is teljesül. Például: „Ha a vádlottat orgazdaság és devizabűntény bűntettében találták vétkesnek", akkor a devizabűntény és az orgazdaság bűncselekménye a másiktól függetlenül is fennáll (még akkor is, ha a gyakorlatban a két vádat közös eljárásban tárgyalták).

Következtetéssel kapcsolatos szabályok:

- Modus (ponendo) ponens (MP), vagy leválasztási szabály: $(P \rightarrow Q), P \rightarrow_{MP} Q$. Vagyis: ha egy következtetés feltételoldala teljesül, akkor a következmény is teljesül. Például az eljárásjog szerint: „Ha a megszabott határidőn belül egyik fél sem nyújt be fellebbezést, akkor az ítélet jogerőre emelkedik". Ha ezek után tényleg nem nyújtottak be fellebbezést, akkor a határidőn túl az ítéletet hatályosnak kell tekinteni.

- Modus (tollendo) tollens (MT) vagy kontrapozíció: (P→Q), ~Q ➔$_{MT}$ ~P. Vagyis: ha egy következtetés következményoldala nem teljesült, akkor a feltételoldal sem teljesülhetett. Az előző példát alapul véve: „…ha az ítélet tényleg jogerőre emelkedett, akkor nyilván nem nyújtottak be a határidőn belül megfelelő fellebbezést…”. Megjegyezzük, hogy a következtetés maga itt sem jelent ok-okozati, csupán logikai viszonyt.
- Modus ponendo tollens (MPT) vagy kizárási szabály: ~(A∧B), A ➔$_{MPT}$ ~B. Vagyis, ha két lehetőség együttesen nem igaz, de az egyik mégis bebizonyosodik, abból következtethetünk a másik hamisságára. Például: „büntetett előéletű személy nem tölthet be államelnöki pozíciót”. Logikusabban: „nem lehet igaz, hogy valaki büntetve is volt, és államelnök is”. Tehát, ha „Jancsi büntetve volt” igaz, akkor „nem lehet belőle államelnök”.
- Modus tollendo ponens (MTP) vagy diszjunktív szillogizmus: A∨B, ~A ➔$_{MTP}$ B. Ha egyfelől két lehetőség diszjunktív (vagylagos) kapcsolatban áll, ám valamelyik lehetőség fennállása kizárható, abból következtethetünk a másik lehetőség fennállására. Például: „vagy az első vádlott, vagy a második vádlott rabolt bankot”, ám az első (pl. alibivel) tisztázta az ártatlanságát, akkor a második vádlott a bűnös.
- Hipotetikus szillogizmus: A→B, B→C ➔ A→C. Következtetési lánc tranzitív lezárása. Ha egy következtetés feltétele egy másik következtetés következménye, akkor a következtetés a másik következtetés feltételéből kiindulóan is fennáll. Például: „ha 100 km/h-val hajtunk az országúton, akkor túlléptük a sebességhatárt…” és „a sebességmérőt kezelő rendőr túllépés esetén büntetést ró ki” következtetések esetén, ha tényleg 100-zal hajtunk egy országúton, (és láttuk a háromlábút is), akkor bizton számíthatunk a büntetési csekk megérkezésére is.
- Konstruktív dilemma: P→Q, R→S, P∨R ➔ Q∨S. Ha két, egymástól független következtetésünk van, amelyek feltételeinek a vagylagos kapcsolatát sikerült tisztázni, akkor ebből levezethető a következmények vagylagos kapcsolata is. Például: „a nagy értékű lopás bűncselekmény”, „a kis értékű lopás szabálysértés”. Ha ezek után tényként tisztázódik, hogy Jancsi eltulajdonított valamilyen értéktárgyat, aminek az értékét nem tudjuk, akkor az is biztos, hogy „Jancsi szabálysértést vagy bűncselekményt követett el”. Ha az eljárás célja csupán Jancsi feddhetetlenségének tisztázása, akkor itt már meg is állhatunk, hiszen az nyilvánvalóan megdőlt. Ha még a megfelelő büntetési tétel meghatározása is cél, akkor – legalábbis a felhozott példában – az eltulajdonított tárgy értékét is meg kell határoznunk.
- Destruktív dilemma: P→Q, R→S, ~Q∨~S ➔ ~P∨~R. A konstruktív dilemmához hasonló helyzetben, ha a következmények cáfolatának vagylagos kapcsolatát sikerült tisztázni, akkor ebből levezethető a feltételek cáfolatának a vagylagos kapcsolata is. Ha az előző törvényi idézetből indulunk ki, és tisztázzuk, hogy Jancsi lehet, hogy nem követett el szabálysértést, de esetleg nem követett el bűncselekményt sem, akkor mindebből biztosan tudhatjuk, hogy lehet, hogy kis értékben, de lehet, hogy nagy értékben sem tulajdonított el semmit.

2.3.1.16 Érvelési és bizonyítási módszerek

Szintén már az ókori klasszikus logikában kialakultak bizonyos érvelési, bizonyítási módszerek és stratégiák:

Az imént említett módszert, a nyilvánvaló tényekből a levezetési szabályok segítségével végzett logikai levezetéseket *deduktív bizonyításnak* is nevezzük. Ezek érvényessége nem vitatható mindaddig, amíg a levezetési szabályok érvényes lépéseket fogalmaznak meg.

A *reductio ad absurdum* módszer a középiskolából is jól ismert indirekt tételbizonyítás módszere, amely végső soron a kizárt harmadik klasszikus alapelvére épül. Ezek szerint egy tétel vagy igaz, vagy hamis lehet. Ha tehát a tétel ellenkezőjét feltételezve logikus következtetésekkel ellentmondásra jutunk, azzal az ellentett tétel hamisságát bizonyítottuk be, ezért az eredeti tétel igaz. A módszer tehát (pl. jogi) érvelésre, de matematikai precizitású tételbizonyításra is elfogadott.

A *reductio ad ridiculorum* módszer nem logikai bizonyításra, hanem csak *retorikai érvelésre* alkalmas. A lényege: a cáfolni kívánt állításból logikus következtetéssel valamiféle nevetséges következményre jutunk. Ezzel lényegében az országházban az ellenzék javaslata nevetségessé tehető, és mint komolytalan, könnyedén elvethető.

A *reductio ad infinitum* módszert más néven *végtelen regressziónak* is nevezik. A lényege szerint a bizonyítandó állítást egy lépésben egy másik bizonyítandó állításra vezetjük vissza. Ha a visszavezetés a végtelenségig folytatható, és nyilvánvalóan sosem jutunk egy cáfolatig, akkor – a módszer szerint bizonyított lenne az eredeti állítás. A módszert nem fogadják el bizonyításnak, de igazából még érvelésnek sem. A legszebb példája, az ismert Zenón paradoxon alapján nyilvánvaló, hogy miért nem.

A Zenón paradoxon alapkérdése a következő: Eléri-e Akhilleusz valaha is az előtte mászó teknősbékát. Nem érheti el, hiszen mire Akhilleusz eléri a teknősbéka korábbi helyét, addig az továbbmászik. A logikai levezetési lépést végtelenig ismételve azt kapnánk, hogy Akhilleusz a teknősbékát sosem éri el.

Az *indukciós következtetés* során egyedi eseményeket figyelünk meg, és azok közös jellemzőit következményként állítjuk be. Tegyük fel, hogy egy megfigyelő végigsétál kedvenc zsákfalujának, Kutyafának a főutcáján. Ha az illető eközben mindegyik kutyával kapcsolatot teremt, és ennek látható nyomai is maradnak a megfigyelő ruháján, bokáján, nadrágszárán, akkor eléggé megalapozottan állíthatja, hogy a kutyafai kutyák hamisak. Ez még akkor is így van, ha a szomszéd Terka néni palotapincsije a konyhában, a tűz mellett alszik, mint egy macska, és eszébe nem jut bárkit is megharapni, hanem inkább pitizik a harapnivalóért.

Az indukciós következtetési módot szintén nem fogadjuk el érvényes logikai levezetésként, két okból sem. Egyrészt azért, mert az efféle következtetés csupán *valószínűségi jellegű,* és legtöbbször lehetséges rá ellenpéldát állítani. Másrészt azért, mert a következtetés *szubjektív jellegű,* vagyis az érvényessége attól függ, hogy az alanyunk milyen példákat ismert, amelyből a következtetést levonta. (Előfordulhat az is, hogy a kutyafai mellékutcákban csupa barátságos kutya lakik, és csakis a főutcán vannak hamis kutyák.)

Az indukció továbbfejlesztése a *teljes indukció*, amely valamilyen rögzített (legtöbbször rekurzív) szabály által előállítható, megszámlálhatóan végtelen számosságú alaphalmazra vonatkozó általános (univerzális) állításokra használható. A teljes indukció két lépésből áll:

- Az alaphalmaz legegyszerűbb/legelső elemére vonatkozóan belátjuk a tételt. Ez általában nagyon egyszerű, nyilvánvaló dolog szokott lenni.

- Feltételezve, hogy az alaphalmaz tetszőleges elemére igaz a tétel, belátjuk, hogy a halmazelem-képzési szabály alapján kapható következő elemre is igaz lesz.

A teljes indukciót a matematikai pontosságú tételbizonyítási eljárások között tartják számon. A legkézenfekvőbb alkalmazása, amikor az egész számokra, vagy ezekre visszavezethető objektumokra vonatkozó tételre használjuk. Ilyenkor a legegyszerűbb elem legtöbbször a 0 szám. A következő elem képzése szerint pedig feltételezve, hogy a tétel az N számra igaz, a feltételezés alapján a tételt N+1-re is bebizonyítjuk.

Az *abduktív következtetést* szokás *visszavezetésnek* is nevezni. Az abdukció során pontosan ismerjük a rendszerünkben alkalmazható következtetéseket, és tapasztaljuk a következményeik tényszerű fennállását. Egyszerűbben: okozatokból, következményekből és tünetekből kíséreljük meg az okokat, és az okokból az okozatokhoz vezető (legjobb, legnyilvánvalóbb, legrövidebb, leghatékonyabb, stb.) következtetési lépéseket rekonstruálni. A rövid magyarázatból is kiderül: az abdukció egy (vissza-) következtetési stratégia, amelynek helyességét – a dedukcióhoz hasonlóan – elsősorban az eljárás lépéseinek a helyessége határozza meg. Az abduktív bizonyítások használhatósága a való életben (jogi vagy más területen, pl. egészségügyben) szintén nyilvánvaló.

2.3.2 Elsőrendű logika

A nulladrendű logika a logika tudományába való bevezetésre alkalmas, mert pici, egyszerű, minden bizonyítható benne, és létezik hozzá egyszerű és áttekinthető kalkulus, ami ráadásul még teljes is. Sajnos azonban az ilyen egyszerűség komoly hátulütője, hogy valóságos problémák leírására nem eléggé kifejező. Ez abban nyilvánul meg, hogy még egyszerű problémák leírása is akadályokba ütközik: a logikai rendszer leírása az állítások esetleg áttekinthetetlen mértékű szaporodását követeli meg.

Az ítéletlogika egyik általánosítása az *elsőrendű logika* (First Order Logic, FOL). Ez már elegendően kifejező, és az elsőrendű logikára – vagy esetleg részosztályaira - többféle következtetési rendszert is alkalmaznak.

Tegyük fel, hogy adott három halmaz:

- $\boldsymbol{V}$: a változószimbólumok véges halmaza
- $\boldsymbol{P}$: a predikátumszimbólumok véges halmaza. Egy $p_n \in \boldsymbol{P}$ egy predikátumszimbólum, amely kiegészül a rögzített paraméterszámra vonatkozó adattal is, ahol n>=0, egész szám. Érdemes megkülönböztetni az $\boldsymbol{P}_n \subseteq \boldsymbol{P}$ részhalmazokat, melyek az n (rögzített) paraméterszámú függvényszimbólumok halmazát jelölik.
- $\boldsymbol{F}$: a függvényszimbólumok véges halmaza. Egy $f_n \in \boldsymbol{F}$ egy függvényszimbólum, amely kiegészül a rögzített paraméterszámra vonatkozó adattal is, ahol n>=0, egész szám. Érdemes megkülönböztetni az $\boldsymbol{F}_n \subseteq \boldsymbol{F}$ részhalmazokat, melyek az n (rögzített) paraméterszámú függvényszimbólumok halmazát jelölik.

A fenti három halmazt együtt az elsőrendű logikai nyelv *névjegyének* vagy *szignatúrájának* nevezzük, és a következőképpen jelöljük:

$\Sigma = [\boldsymbol{F}, \boldsymbol{P}, \boldsymbol{V}]$

A névjegy értelme: akkor beszélhetünk egyáltalán egy elsőrendű nyelvről, ha a névjegyét, illetve a névjegy elemhalmazait megadjuk, rögzítjük. Ezek után meghatározhatjuk magát a logikai nyelvet.

Az *elsőrendű logika nyelvének,* $\mathcal{L}^1$-nek a pontos megadásához előbb meg kell adnunk a *függvénykifejezések* $\mathcal{LF}$ *résznyelvét.*

$\mathcal{LF}$ = { φ ::= v | f_0 | $f_n(\varphi_1,\ldots,\varphi_n)$ }, ahol v∈$\mathcal{V}$, $f_0\in\mathcal{F}_0$, $f_n\in\mathcal{F}_n$, $\varphi_i\in\mathcal{LF}$, és 1<=i<=n

Ezek után az $\mathcal{L}^1$ nyelvet rekurzív módon az alábbi (egyszerűsített BNF) jelöléssel adhatjuk meg.

$\mathcal{L}^1$ = { ϕ ::= p_0 | $p_n(\varphi_1,\ldots,\varphi_n)$| $\forall$v ϕ_1| $\exists$v ϕ_1| ~ ϕ_1 | (ϕ_1) |
$\phi_1 \vee \phi_2$ | $\phi_1 \wedge \phi_2$| $\phi_1 \leftarrow \phi_2$ | $\phi_1 \rightarrow \phi_2$ | $\phi_1 \leftrightarrow \phi_2$ | $\phi_1 = \phi_2$}, ahol n>=0 egész,
$p_n\in\mathcal{P}_n$, v∈$\mathcal{V}$, $\varphi_i\in\mathcal{LF}$, $\phi_1\in\mathcal{L}^1$, $\phi_2\in\mathcal{L}^1$, és 1<=i<=n

A pontos megadáshoz a következő megjegyzéseket érdemes fűzni:

- A változók *ismeretleneket* jelölnek, tehát olyan dolgokat, amelyeket eleinte nem tudunk azonosítani, az egyes számítási eljárások célja azonban éppen egy olyan behelyettesítési érték meghatározása, amely a logikai megkötéseket kielégíti. Változók azonban az elsőrendű logikában csak kifejezést (az $\mathcal{LF}$ résznyelv mondatait) vehetnek fel értékül.
- Egy n argumentumú függvényszimbólum és n darab másik függvénykifejezés összekapcsolásával egy még összetettebb függvénykifejezést hozhatunk létre.
- A 0 argumentumú predikátumszimbólumokat *predikátumkonstansoknak*, vagy *logikai konstansoknak* a 0 argumentumú függvényszimbólumokat pedig egyszerűen *elemkonstansoknak* nevezzük.
- Egy n argumentumú predikátumszimbólum és n darab függvénykifejezés összekapcsolásával létrehozható kifejezést *predikátumkifejezésnek* is nevezzük. Egy predikátumkifejezést vagy az egyszerű negáltját szokás *literálnak* is nevezni: ha nem tartalmaz negációt, akkor *pozitív literálnak*, ha igen, akkor *negatív literálnak*.

Az elsőrendű nyelvek interpretációjához ismernünk kell a tárgyalási világegyetem $\mathcal{U}$ halmazát. Legyen ezen adottak n változós $\mathcal{UF}_n$, függvények, (amelyek $\mathcal{U}^n$-ből $\mathcal{U}$-ba képeznek le), és az $\mathcal{UP}_n$ n-paraméteres predikátumok (amelyek $\mathcal{U}^n$-ből a {true, false} halmazba képeznek le.

Az $\mathcal{L}^1$ elsőrendű logikai nyelv *értelmezése* vagy *interpretációja* alatt a nyelvben használt predikátumok és függvények leképezését érjük az $\mathcal{U}$ világegyetem feletti függvényekbe és predikátumokba. Formálisabban az interpretáció az $\mathcal{J}$:[V_F:$\mathcal{F}\rightarrow\mathcal{UF}$,$V_P$:$\mathcal{P}\rightarrow\mathcal{UP}$] pár, ahol V_F a függvények leképezése, és V_P a predikátumok leképezése.

Az elsőrendű logikai nyelv változói közül vannak olyanok, amelyek valamilyen kvantor hatáskörében vannak, és vannak kvantorhatáskörön kívül eső változók. Az előbbieket *(kvantorral) kötött*, az utóbbiakat *szabad változók*nak is nevezzük.

Egy elsőrendű, szabad változót nem tartalmazó (összetett) logikai kifejezés értelmezése alatt a $\phi\in\mathcal{L}^1\rightarrow$ {true, false} leképezés értékét, illetve a logikai érték meghatározását értjük. Ez az ismert 0-ad rendű logikai kapcsolók esetében kézenfekvő. A kvantorokra:

- $\exists$x ϕ(x) akkor igaz, (egzisztenciális kvantor) ha találunk legalább egy y∈$\mathcal{LF}$ függvénykifejezést (közte elemkonstansokat is), amelyre ϕ(y)=true.

- $\forall x\ \phi(x)$ akkor igaz, (univerzális kvantor) ha minden lehetséges $y \in LF$ függvénykifejezés (közte elemkonstans) választása esetében $\phi(y)$=true.

Az elsőrendű logikában a nulladrendűhöz hasonlóan beszélhetünk érvényes és kielégíthetetlen kifejezésekről. Egy kifejezés egy konkrét értelmezése pedig a kifejezésnek lehet modellje, vagy ellenmodellje. Teljesen hasonló a nulladrendű meghatározáshoz a logikai következmény fogalma is.

A logikai következmények megkeresésére különféle *kalkulusok* léteznek, amelyek helyességéről és a teljességéről elsőrendű nyelvek esetében is beszélhetünk. Bár a modern megoldások teljesek, ez szigorú értelemben már nem igaz. Az elsőrendű tételbizonyítási algoritmusok már csak *részben eldönthetők*.[13] Ez azt jelenti, hogy létezhet ugyan olyan algoritmus, amely véges idő alatt eldönti, hogy egy A állítás logikailag következik egy T elméletből, de csak akkor, ha az tényleg következik. Ha nem következne, akkor nem tudható, hogy ugyanaz az algoritmus véges idő alatt megáll-e vagy sem. Vagyis, ha egy ilyen bizonyító algoritmus nagyon-nagyon sokáig fut, akkor nem tudjuk megmondani, hogy csak azért fut sokáig, mert még nem találta meg az egyébként véges, de hosszú időben létrejövő eredményt, vagy azért, mert az állítás nem is következik az elméletből.

2.3.2.1 Arisztotelész elsőrendű szillogizmusai

Az ítéletlogika gyengeségét persze a görögök is észrevették, de egységes elsőrendű elméletet nem alkottak meg. Az elsőrendű logika elemeit először Arisztotelész szillogizmusaiban fedezhetjük fel, amelyek az Organon c. többkötetes művében vannak leírva. A görög „szüllogiszmosz" szó érvet, következtetést jelent, és a korábban már tárgyalt természetes következtetési szabályok mellett éppen ezek a szillogizmusok voltak azok az alap-építőkövek, amelyre a görög logikusok fejtegetéseiket építették, és amelyek később az elsőrendű logika fogalomkörének kialakulásához vezettek.

A szillogizmusok tehát egyszerű következtetési szabályokat fogalmaznak meg, mindegyikben két premissza és egyetlen konklúzió van.[14] Az állítások úgynevezett *terminológiai* jellegűek, vagyis valamilyen terminusnak megfelelő halmaz, azaz logikailag egyargumentumú predikátumkifejezés van benne. Másrészt mindegyikük önmagában is egy kategorikus állításnak tekinthető, hiszen a terminológiai állításokra valamilyen egyéb tulajdonság fennállását állítják egyszerű következtetéses (implikációs) formában.

A következtetési szabályok elemei a következő négy sémát követik, amelyeknek egy magánhangzóból álló nevet is adtak.

- A, mint '**A**ffirmo': Minden S az P is, $\forall x(S(x) \rightarrow P(x))$, univerzálisan kvantált változóra vonatkozó *általános pozitív állítás*. Pl. „minden állampolgár jogalany".
- E, mint 'N**E**go': Egyetlen S sem P, $\forall x(S(x) \rightarrow \sim P(x))$, az univerzálisan kvantált változóra vonatkozó *általános negatív állítás* vagy *tagadás*. Pl. „a magzat nem jogalany".

[13] Zohar Manna: Programozáselmélet [ZOH81] 123. old.

[14] Bognár László-Forrai Gábor: Esszéírás és Informális Logika [BF04]

- I: Némely S az P, $\exists x(S(x) \rightarrow P(x))$, az egzisztenciálisan kvantált változóra vonatkozó pozitív állítás, vagy *részleges (partikuláris) állítás*. Pl. „egyes cégek külkereskedelmi tevékenységet is kifejthetnek”.
- O: Némely S az nem P, $\exists x(S(x) \rightarrow \sim P(x))$, az egzisztenciálisan kvantált változóra vonatkozó negatív állítás, vagy *részleges (partikuláris) tagadás*. Pl. „egyes társadalmi szervezetek nem közhasznúak”.

A fenti állításelemek összefüggéseit tekintve:

- Egy általános állításról és tagadás párjáról azt mondjuk, hogy egymással ellentétes (kontrárius) viszonyban vannak. Ez azt jelenti, hogy nem lehetnek egyszerre igazak, de lehetnek egyszerre hamisak. Pl. „minden szervezet közhasznú” $\leftarrow\rightarrow$„egyetlen szervezet sem közhasznú”.
- Egy részleges állításról és tagadás párjáról azt mondjuk, hogy egymással alárendelten ellentétes (szubkontrárius) viszonyban állnak. Ez azt jelenti, hogy lehetnek egyszerre igazak, de egyszerre hamisak nem. Pl. „némely szervezet közhasznú” $\leftarrow\rightarrow$ „némely szervezet nem közhasznú”.
- Egy általános állítás és ugyanarra vonatkozó részleges tagadás párja, illetve egy általános tagadás, és ugyanarra vonatkozó részleges állítás párja egymásnak ellentmondó (kontradiktórius) viszonyban vannak, vagyis egyszerre nem lehetnek sem igazak, sem hamisak. Pl.: „minden szervezet közhasznú” $\leftarrow\rightarrow$ „némely szervezet nem közhasznú”, illetve „egy szervezet sem közhasznú” $\leftarrow\rightarrow$ „némely szervezet közhasznú”.

Az Arisztotelész-féle szillogizmusok további fontos jellemzője még, hogy a terminológiai állítás nem lehet üres. Vagyis a „minden szervezet közhasznú” állítás feltételezi, hogy létezik is a világon legalább egy szervezet. Ez a feltételezés azért fontos, mert Arisztotelész szerint, de a közfelfogás szerint is ez így van. Egyébként a formális logikában az üres terminusra bármilyen tulajdonság igaz. Pl. a „minden jeti emberevő” állítás formálisan tökéletes és igaz, elfogadva a biológusok konzervatív álláspontját a jetikre vonatkozólag (mármint hogy nem léteznek). Kicsit bonyolítva és jogi területre átevezve a „minden vagyonomat jótékony célra fordítottam” állítás is igaz akkor, ha az illetőnek semmilyen ingó vagy ingatlan vagyona sem volt.

A szillogizmusokban három terminus: az alsó, a felső és a középső játszik szerepet, és háromféle általános alakzatuk van. Ezeket az különbözteti meg egymástól, hogy a premisszákban és a konklúzióban milyen szerkezetben alkalmazzuk az egyes terminusokat. Nevezzük el ezeket az elemeket a következőképpen:

- K: legyen a *középső* terminus
- A: legyen az *alsó* terminus. Ez a konklúzió *alanya*.
- F: legyen a *felső* terminus. Ez a konklúzióban *állítmányi* szerepet játszik, vagyis a konklúzió minden esetben: $Qx(A(x) \rightarrow F(x))$ alakú, ahol Q az univerzális vagy egzisztenciális kvantort jelzi, x pedig logikai változó.

A konklúzió minden esetben az alsó és felső terminus között állít valamiféle összefüggést, amelyhez az állítás egy segédfogalmat (a középső terminust) vezet be. Ilyen módon lehetséges megállapítani azt is, hogy egy konkrét szillogizmus-példa milyen típusú.

Mindezek fényében az Arisztotelész-féle három általános alakzat (pontos logikai szerkezet és kvantifikáció nélkül) a következőképpen néz ki:

1. K-F, A-K ➔ A-F
2. F-K, A-K ➔ A-F
3. K-F, K-A ➔ A-F

Az egyes alap-állításfajtákat jelölő magánhangzókat háromtagú fantázianevekbe építve kapjuk az Arisztotelész-féle szillogizmusgyűjteményt. Megjegyezzük, hogy kombinatorikai úton még több alakzat is felírható, de ezek közül csak néhány igaz: konkrétan az első és második alakzatból 4-4, a harmadikból pedig 6. Tekintsük tehát az egyes érvényes szillogizmusokat:

Első alakzatú szillogizmusok:

B**A**rb**A**r**A**: $\forall x(K(x)\rightarrow F(x))$, $\forall x(A(x)\rightarrow K(x))$ ➔ $\forall x(A(x)\rightarrow F(x))$. Például: „minden gazdálkodó szervezet adóalany", „minden betéti társaság gazdálkodó szervezet" ➔ „minden betéti társaság adóalany".

C**E**l**A**r**E**nt: $\forall x(K(x)\rightarrow \sim F(x))$, $\forall x(A(x)\rightarrow K(x))$ ➔ $\forall x(A(x)\rightarrow \sim F(x))$. Például: „az alapítványoknak nincs tagsága", „közhasznú alapítványok is alapítványok" ➔ „közhasznú alapítványoknak nincs tagsága". A szillogizmus a Barbara F-re vonatkozó negáltja.

D**A**r**II:** $\forall x(K(x)\rightarrow F(x))$, $\exists x(A(x)\rightarrow K(x))$ ➔ $\exists x(A(x)\rightarrow F(x))$. Például: „minden betéti társaság gazdálkodó szervezet", „egyes gazdálkodó szervezeteknek külkereskedelmi tevékenységük is van" ➔ „egyes betéti társaságoknak külkereskedelmi tevékenységük is van".

F**E**r**IO**: $\forall x(K(x)\rightarrow \sim F(x))$, $\exists x(A(x)\rightarrow K(x))$ ➔ $\exists x(A(x)\rightarrow \sim F(x))$. Például: „gazdálkodási tevékenység nem adómentes", „egyes társadalmi szervezetek gazdálkodnak is" ➔ „egyes társadalmi szervezetek nem mentesek az adó alól". A szillogizmus a Darii F-re vonatkozó negáltja.

Második alakzatú szillogizmusok:

C**E**s**A**r**E**: $\forall x(F(x)\rightarrow \sim K(x))$, $\forall x(A(x)\rightarrow K(x))$ ➔ $\forall x(A(x)\rightarrow \sim F(x))$ Például: „Egyetlen állat sem jogalany", „Minden ember jogalany" ➔ „Egyetlen ember sem állat" (legalábbis jogi értelemben nem az).

C**A**m**E**str**E**s: $\forall x(F(x)\rightarrow K(x))$, $\forall x(A(x)\rightarrow \sim K(x))$ ➔ $\forall x(A(x)\rightarrow \sim F(x))$ Például: „Gépjárművet csak jogosítvánnyal vezethetünk", „Kerékpárosoknak nincs szükségük jogosítványra" ➔ „A kerékpárosok nem gépjárművezetők" (nyilván egy adott pillanatra vonatkoztatva). A szillogizmus a Cesare K-ra vonatkozó negáltja.

F**E**st**I**n**O**: $\forall x(F(x)\rightarrow \sim K(x))$, $\exists x(A(x)\rightarrow K(x))$ ➔ $\exists x(A(x)\rightarrow \sim F(x))$ Például: „Kiskorú nem büntethető", „Létezik olyan személy, aki büntethető" ➔ „Létezik olyan személy, aki nem kiskorú"

B**A**r**O**c**O**: $\forall x(F(x)\rightarrow K(x))$, $\exists x(A(x)\rightarrow \sim K(x))$ ➔ $\exists x(A(x)\rightarrow \sim F(x))$ Például: „Gépjárművet csak jogosítvánnyal vezethetünk.", „Létezik olyan jármű, amihez nem szükséges jogosítvány." ➔ „Létezik olyan jármű, amely nem gépjármű." A szillogizmus a Festino K-ra vonatkozó negáltja.

Harmadik alakzatú szillogizmusok:

D**A**r**A**pt**I**: $\forall x(K(x)\rightarrow F(x))$, $\forall x(K(x)\rightarrow A(x))$ ➔ $\exists x(A(x)\rightarrow F(x))$ Például: „Minden gépjárműre vonatkozik sebességkorlátozás", „Minden gépjármű jogosítványköteles" ➔ „Létezik olyan sebességkorlátozott gépjármű,

amely jogosítványköteles". Viszont mind a sebességkorlátozott gépjárművek között lehetséges olyan, amely nem jogosítványköteles, (legalábbis elvileg), mind a jogosítványköteles járművek között létezhet olyan jármű, amire viszont nincs sebességkorlátozás (pl. harckocsi).

F**E**l**A**pt**O**n: $\forall x(K(x)\rightarrow\sim F(x))$, $\forall x(K(x)\rightarrow A(x))$ ➔ $\exists x(A(x)\rightarrow\sim F(x))$ Például: „Minden gépjárművel tilos a gyorshajtás", „Minden gépjármű vizsgaköteles" ➔ „Létezik olyan vizsgaköteles jármű, amellyel tilos a gyorshajtás". Létezhet viszont olyan vizsgaköteles jármű, amelyre nincs sebességkorlátozás (pl. vitorlás), és létezhet olyan sebességkorlátozott jármű is (pl. kerékpár), ami nem vizsgakötelezett. A szillogizmus a Darapti F-re vonatkozó negáltja.

D**I**s**A**m**I**s: $\exists x(K(x)\rightarrow F(x))$, $\forall x(K(x)\rightarrow A(x))$ ➔ $\exists x(A(x)\rightarrow F(x))$ Például: „Léteznek megkülönböztetett jelzésű gépjárművek", „Minden gépjármű vizsgaköteles" ➔ „Létezik olyan vizsgaköteles jármű, amely megkülönböztetett jelzést visel".

B**O**c**A**rd**O**: $\exists x(K(x)\rightarrow\sim F(x))$, $\forall x(K(x)\rightarrow A(x))$ ➔ $\exists x(A(x)\rightarrow\sim F(x))$ Például: „Létezik olyan nagykorú állampolgár, akinek nincs jogosítványa", „Minden nagykorú állampolgárnak van személyi igazolványa" ➔ „Létezik olyan személyi igazolvánnyal rendelkező ember, akinek nincs jogosítványa". A szillogizmus a Disamis F-re vonatkozó negáltja.

D**A**t**I**s**I**: $\forall x(K(x)\rightarrow F(x))$, $\exists x(K(x)\rightarrow A(x))$ ➔ $\exists x(A(x)\rightarrow F(x))$ Például: „Minden állampolgár adóköteles", „Létezik olyan állampolgár, aki munkát vállal" ➔ "Létezik olyan munkavállaló, aki adóköteles".

F**E**r**I**s**O**n: $\forall x(K(x)\rightarrow\sim F(x))$, $\exists x(K(x)\rightarrow A(x))$ ➔ $\exists x(A(x)\rightarrow\sim F(x))$ Például: „Egyetlen polgárnak sem lehet egynél több házastársa", „Létezik olyan polgár, akinek gyereke van" ➔ „Létezik olyan gyerekes polgár, akinek nincs egynél több házastársa". A szillogizmus a Datisi F-re vonatkozó negáltja.

Az Arisztotelész-féle szillogizmusoknak komoly jelentősége lehetett a maguk idejében, manapság azonban inkább csak történeti jelentőségük maradt. Egyrészt nem teljesek – vagyis messze nem fedik le a tételek bizonyításához szükséges és használatos logikai szabályok teljes körét, (pl. a többargumentumú predikátumok vagy függvénykifejezések hiánya), másrészt viszont a későbbi fejezetekben tárgyalt gépi tételbizonyítási eljárások „csuklóból" bizonyítják őket.

2.3.2.2 Gépi tételbizonyítás, formulák klóz alakja

A tételbizonyítás alapproblémáját a következőképpen lehet megfogalmazni. Tegyük fel, hogy adott egy T elmélet, amely nem más, mint logikai állítások egy véges halmaza. Ha ezek után adott egy A hipotézis, ez úgy bizonyítható, hogy az $T\rightarrow A$ implikáció érvényességét próbáljuk bebizonyítani. Ez azt fejezi ki, hogy az implikáció a T elméletben leírt összefüggésektől függ, az A állítás igazságát biztosító összes korlát vagy feltétel a T elméletben van modellezve.

A bizonyításra létrehozott modern algoritmusok közül sokan a 'Reductio ad Absurdum' elvet követve, a $T\rightarrow A$ implikáció érvényességének bizonyítása helyett a $T \vee \sim A$ kielégíthetetlenségét bizonyítják. Az ilyen bizonyításokat ezért *cáfoló bizonyításnak*, az efféle algoritmusokat pedig *cáfoló algoritmusoknak* nevezzük. Ez az indirekt

bizonyítási módszer gépi megfelelője: feltesszük az állítás ellenkezőjét, és logikus következtetésekkel ellentmondásra jutunk.[15]

Logikai formulák kielégíthetetlenségének bizonyításakor a korábban már leírt nulladrendű normálformákon kívül jelentőséget nyernek az olyan átalakítások, amelyek logikailag nem feltétlenül egyenértékű, de a kielégíthetetlenséget oda-vissza (akkor és csak akkor) megőrző formulákat eredményeznek. Ilyen az elsőrendű formulák klóz alakja,[16] amely a diszjunktív normál alakhoz hasonló elsőrendű szerkezet, és a következőképpen definiálható:

Egy elsőrendű formuláról akkor mondjuk, hogy klóz alakú, ha a következő formájú:

$\forall x_1 ... \forall x_n \, (C_1 \wedge ... \wedge C_m)$

ahol a C_i elemeket, amelyek csak a kvantált változókat is tartalmazhatják, *klózoknak* nevezzük, és a változók *csak univerzálisan kvantáltak*. Egy C_i klóz alakja a következő:

$L_1 \vee ... \vee L_k$

…ahol az L_i-ket *literálnak* nevezzük, amelyek kvantort már nem tartalmaznak, a változóik pedig mind egységesen univerzálisan kvantáltak. Ha egy literál egy állításelem negációja, akkor *negatív*, egyébként *pozitív literál*.

A klózokkal és klózhalmazokkal kapcsolatosan (részben a Prolog programozási nyelv terjedésével párhuzamosan) még a következő megnevezések használatosak:

- Az eredeti elmélet klózait *alapklózoknak* nevezzük.
- Az egyetlen literálból álló klózokat *egységklózoknak*, az egyetlen pozitív literálból álló klózokat pedig *tényeknek* nevezzük.
- A csak negatív literálokból álló klózokat *célsorozatnak*, *feladatnak* vagy *kérdésnek* nevezzük. Az egyetlen literálból álló célsorozat neve: *célállítás*.

Tétel: Minden elsőrendű F formula átalakítható olyan F' klóz alakú formulává, hogy F akkor és csak akkor kielégíthetetlen, ha F' is az.

A klóz alakba átalakítás lépései:

1. Átalakítás diszjunktív normál alakba.
2. Kvantorok kiemelése a formula elejére. Ez még egyenértékű átalakítás, csupán a kvantorok egymáson történő átemelése (helycseréjük, vagyis a hatásköri sorrendjük módosítása) nem lehetséges.
3. Skolem függvények bevezetése. Helyettesítsük minden $\forall x_1 \ldots \forall x_n \, \exists x \ldots \Phi(\ldots,x,\ldots)$ alakú formulában az x egzisztenciálisan kvantált változót egy $f_{sk}(x_1 ::: x_n)$ újonnan bevezetendő (Skolem-) függvénnyel, ahol az x_i-k az előre kiemelt kvantorláncban az x egzisztenciális kvantort megelőző univerzálisan kvantált változók. Skolem tétele értelmében az eredményként kapott formula akkor és csak akkor lesz kielégíthetetlen, ha az eredeti formula is az volt.
4. A fenti három lépéssel megkapható, úgynevezett *absztrakt klóz alakot* a következőképpen lehet még alakítani:

[15] Zohar Manna: Programozáselmélet [Zoh81] 123. old.

[16] Zohar Manna: Programozáselmélet [Zoh81] 148. old.

- Minden változó univerzálisan kvantált, tehát formálisan elhagyhatjuk a kvantorjelöléseket
- Minden klózban vannak pozitív és negatív literálok. Ha ezeket elemi logikai művelettel egy implikáció szimbólum két oldalára gyűjtjük, akkor megkapjuk a (Prolog nyelvben is használatoshoz hasonló) következő klóz formát.

  $N_1;\ldots;N_n$:- $P_1,\ldots,P_m$.

  …ahol az N_i negatív literálok a következményoldalon, a P_i pozitív literálok pedig a feltételoldalon találhatók, mindkét oldal lehet azonban üres is.
- A klózok közötti konjunkció formális elhagyásával klózok egy halmazához jutunk.

Az előző szakaszban említett klasszikus szillogizmusok bár helyesek voltak, de nem voltak teljesek, vagyis messze nem volt minden olyan állítás velük levezethető, amelyet a természetes következtetési szabályok alkalmazásával levezethetőnek tekintünk. Az első teljes módszer az elsőrendű logikában az azóta már többször is továbbfejlesztett Davis-Putnam (-Longeman) -féle algoritmus[17] [18] szintén az imént említett klózhalmazokon dolgozik. Ez egy iteratív cáfoló algoritmus, amely a változók számára egyre mélyebben és mélyebben beágyazott függvénykifejezéseket generál. Tekintve, hogy minden változó univerzálisan kvantált, egyetlenegy ellenpélda fellelése a teljes klózhalmaz kielégíthetetlenségét is bizonyítja, vagyis az ellenpélda egyben egy pozitív példa arra, hogy az eredetileg bebizonyítandó tétel milyen változólekötés mellett teljesülhet.

Ha egy ellenpéldát már találtunk, akkor az eljárás folytatásával esetleg további ellenpéldák is találhatók, vagyis ezzel az eredeti tétel számára további változólekötések is létrehozhatók.

A fenti leírásból nyilvánvaló az is, hogy az eljárás miért csak félig eldönthető: ha az eredeti T→A állítás mégsem lenne igaz, akkor az egyre mélyülő változóhelyettesítések során sosem találnánk az univerzális kvantálást cáfoló ellenpéldát, vagyis az algoritmus sosem állna meg.

2.3.2.3 Rezolúciós tételbizonyítás

Az *általános rezolúciós módszer* lényege: a *rezolvensképzés művelete* során kapott *rezolvens klózzal* kibővítjük az eredeti klózhalmazt.[19] A rezolvensképzés biztosítja, hogy ha az új klózhalmaz *kielégíthetetlen*, akkor az eredeti klózhalmaz is az volt. Ha a rezolvensképzés ismételt alkalmazása során tehát nyilvánvalóan kielégíthetetlen klózhalmazhoz jutunk, akkor az eredeti klózhalmaz is kielégíthetetlen volt. Ilyen a klózhalmaz például akkor, ha maga a képzett rezolvens kielégíthetetlen lesz, pl. ha az üres klózhoz jutottunk. A rezolúciós tételbizonyítási algoritmus célja tehát az, hogy a

[17] Ч. Чень-Р. Ли: Математическая логика и автоматическое доказательство теорем [ChLee83] 52. old.

[18] Peter Baumgartner: FDPLL – A First-Order Davis-Putnam-Logeman-Loveland Procedure [DPLL00]

[19] Ч. Чень-Р. Ли: Математическая логика и автоматическое доказательство теорем [ChLee83] 76. old.

rezolvensképzés műveletének ciklikus alkalmazása során a nyilvánvalóan kielégíthetetlen *üres klózhoz* jussunk.

A klasszikus rezolúciós módszer által használt *klasszikus rezolvensképző* művelet logikai formulák klóz alakjából indul ki. Tegyük fel, hogy a klózhalmazunkban létezik két olyan klóz,

$C_1 = L_{11},\ldots,L_{1i},\ldots,L_{1n}$ illetve

$C_2 = L_{21},\ldots,\sim L_{2i},\ldots,L_{2m}$

hogy a két klóz egy-egy literálja (a L_{1i} és a $\sim L_{2i}$) ellenkező előjelűek, de az előjeltől eltekintve megegyeznek, vagy *egyesítéssel* (változók behelyettesítésével) azonossá tehetők.

Kicsit pontosítva: két literál a következő feltételek mellett egyesíthető:

- ha a predikátumnevük azonos, és azonos paraméterszámúak, és a paramétereik páronként egyesíthetőek.

Két paraméter a fentebb leírt **LF** kifejezésleíró nyelvhez tartozik. Két ilyen kifejezés akkor egyesíthető, ha:

- egyikük változó, és a másik nem tartalmazza ugyanezt a változót. Ilyenkor a változó felveszi a másik értékét. Ha mindkettő változó, akkor a változók „összekötődnek", az egyikben megjelenő bármilyen érték azonnal megjelenik a másikban is.
- Ha a függvénynevük és a paraméterszámuk megegyezik (ha egyik sem változó), és ha a paramétereik páronként megegyeznek.

Ha a konkrét behelyettesítés műveletét a két klózra (kicsit pongyolán) $@_1$-gyel és $@_2$-vel jelöljük, illetve az egész rezolvensképzésre pedig @-vel, akkor a rezolvens klóz a következőképpen néz ki:

$@C_{12} = @_1L_1,\ldots, @_1L_{i-1}, @_1L_{i+1},\ldots,@_1L_n, @_2L_2,\ldots, @_2L_{j-1}, @_2L_{i+1},\ldots, @_2L_m$

Vagyis a rezolvens klózban az egyesített literálok kivételével minden többi literál szerepel, az egyesített (ellentétes előjelű) literálokat kihagyjuk. A rezolúció fenti, klasszikus meghatározását *kétágú (bináris) rezolúciónak* is nevezik, mert a rezolvensképzésben mindig szigorúan *két klóz* vesz részt. Ha ezt a megkötést enyhítjük, és több klóz részvételét is megengedjük a rezolvensképzésben, akkor *általános vagy beágyazott rezolúcióról* beszélhetünk.

Tétel: A rezolúció teljes következtetési módszer.[20]

A rezolúciós tételbizonyítási módszer teljessége még nem jelenti azt, hogy tényleg lehetséges is olyan algoritmust létrehozni, ami minden elsőrendű elméletről és tételről eldönti, hogy az logikai következménye-e az elméletnek. A rezolúciós módszer ugyanis nyitva hagy egy sor olyan kérdést, amelyet egy számítógépes algoritmus készítésekor mindenképpen valahogy el kell dönteni, meg kell válaszolni. Ezen megoldásokat együttesen *rezolúciós stratégiáknak* nevezzük.[21]

A rezolúciós stratégiák legjobban a *rezolúciós háló* fogalmán keresztül érthetők meg, mert ez könnyen ábrázolható grafikusan is.

---

[20] Michael R. Genesereth-Nils J. Nilsson: Logical Foundations of Artificial Intelligence [GeNi87]

[21] Maria Paola Bonacina: A taxonomy of theorem-proving strategies [MPB]

A rezolúciós háló a rezolúció menetét ábrázolja: a csomópontjai az eredeti elmélet alapklózai, és a rezolúciós művelettel nyerhető rezolvens klózok. A rezolúciós háló valahány pontjából akkor vezet él egy újabb ponthoz, ha azokból kiindulva az (általános) rezolvensképzés műveletével éppen a célponthoz rendelt klózt kapjuk rezolvensként. Nyilvánvaló, hogy kétágú rezolúció esetében tehát a rezolúciós háló minden nem forrás csomópontjába pontosan két él vezet.

Anélkül, hogy pontosítanánk azt, hogy különböző kiindulási klózokból nyerhető azonos rezolvenseket a hálóban azonos, vagy különböző elemeknek tekintjük, a következő megállapítások tehetők:

- A rezolúciós háló *forrás csomópontjai* az elmélet eredeti klózai, amelyet *alapklózoknak* is nevezünk
- A rezolúciós háló *nyelői* a nyilvánvalóan kielégíthetetlen *üres klózok* és azok, amelyek már semmilyen további *rezolvensképzésben nem tudnak részt venni* (zsákutca).
- A hálónak lehetnek *végtelen útjai* is. (Pl. létezhet olyan rezolúciós lépéssor, amely minden lépésében az eredeti klózhoz hasonló rezolvenst állít elő, csupán valamely változó kötődik le, minden lépésben egyre összetettebb, de újabb szabad változót tartalmazó kifejezéssel.)
- *Bizonyításnak* nevezzük a teljes háló egy olyan *részgráfját*, amely a források egy részéből egy nyilvánvalóan kielégíthetetlen nyelőig, vagyis egy üres klózig vezet.
- A tételbizonyítási algoritmus célja egy ilyen *bizonyítás megkeresése*.
- *Kétágú rezolúció* esetén minden csomópontba csupán *két él* fut be. Egyébként a kétágú rezolúció a teljes rezolúciós háló egy *részhálója*. A klasszikus (bináris) rezolúció teljességére vonatkozó tétel azt fejezi ki, hogy ha van a rezolúciós hálónak egyáltalán üres klóz végeleme, akkor ez a részháló is elvezet legalább egy ilyenhez. A félig eldönthetőség viszont azt fejezi ki, hogy a rezolúciós hálóra akármilyen építési és bejárási stratégiát is rögzítenénk, ahhoz mindig lehet olyan problémát találni, amely valamelyik végtelen utat járná be még azelőtt, hogy üres klózra találna.

A rezolúciós tételbizonyítási algoritmus a fentiek miatt a hálókban történő útkereső algoritmusok családjába tartozik. A legjellemzőbb különbség mégis az, hogy az útkereső algoritmusok esetében a gráf szigorúan véges, és előre adott. A rezolúciós tételbizonyítási algoritmus ezzel szemben futás közben építi fel a gráfot.

A rezolúció stratégiáját tekintve a következő legfontosabb besorolási szempontokat, és különleges eseteket szokás megkülönböztetni:

- a tételbizonyítási algoritmus lehet *mélységében* vagy *széltében* kereső, attól függően, hogy a következő rezolúciós lépést elsősorban a legutóbbi rezolvens klózzal hajtjuk végre, vagy egy kiválasztott halmaz elemeivel végrehajtjuk az összes lehetséges rezolúciót, és az eredményt egy újabb halmaznemzedékben gyűjtjük össze, majd a rezolúciós lépéseket erre az újabb nemzedékre hajtjuk végre. A mélységében kereső algoritmus – megfelelő választási stratégia esetén – hamarabb rátalálhat egy üres klózra, de a veszélye, hogy a visszalépések során esetleg nem képes egyes végtelenbe vivő utakat elkerülni. A széltében kereső, párhuzamos stratégia alkalmazása esetén egyes szálak végtelen ciklusba esése

nem állítja le a folyamatot, de a kezelt halmaznemzedék mérete gyakran nehezen tartható kézben, és nagy számítási teljesítményt köt le

- *egységrezolúciónak (unit resolution)* nevezik azt a stratégiát, amikor legalább a rezolúcióban részt vevő klózok egyike egyetlen literálból álló, *egységklóz*. Ez biztosítja azt, hogy a rezolvens hossza minden rezolúciós lépésben eggyel csökken a hosszabbik klózhoz képest
- *alaprezolúciónak (input resolution)* nevezzük azt a stratégiát, amikor a rezolvensképzésben legalább egy alapklóz mindig részt vesz. Alapklóznak nevezzük az eredeti elmélethez tartozó klózokat.
- *lineáris rezolúciónak* nevezzük a stratégiát akkor, ha a rezolválás során mindig az előző lépés létrehozta rezolvenshez keresünk rezolválható klózt.
- *előre haladónak (forward chaining)* nevezzük a rezolúciós stratégiát akkor, ha a rezolúciót a csak egyetlen pozitív literálból álló egységklózokkal (tényekkel) kezdjük, és a következtetés közben is elsősorban ilyen rezolvens klózokból indulunk ki.
- *visszafelé haladónak (backward chaining)* akkor nevezzük a rezolúciós stratégiát, ha a rezolúciót negatív literálból álló klózokkal (célállításokkal, illetve célsorozatokkal) kezdjük.

A legfeljebb egyetlen pozitív literált tartalmazó klózokat *Horn klózoknak* is nevezzük. A Horn klózok jelentősége az, hogy ilyenek építik fel a Prolog programokat, és egyes előre haladó produkciós rendszerek is ilyenekre épülnek.[22] Maga a Constraint Handling Rules (CHR) megoldó[23] korlátlogikai eszköz bizonyos értelemben szintén Horn klózok előre haladó végrehajtásának tekinthető. A Prolog nyelv előrehaladó értelmezésének tiszta esetét valósította meg a szerző[24] [25] a Contralog nevű szoftver eszközzel.

2.3.2.3.1 Beágyazott rezolúció

Sok olyan feladat van, amelyekben egyes függvény és állításszimbólumok értelmezését rögzítjük, és ennek megfelelő axiómarendszert is adottnak feltételezünk, ezekre vonatkozóan viszont a teljes logika egy bizonyos, de még mindig eléggé kifejező részére nézve teljes következtetési rendszer adható meg. Ilyenre példa lehet a számok és a számtani műveletek köre, amelyhez különböző egyenletmegoldó algoritmusok és ilyen szoftver csomagok is léteznek. A *beágyazott rezolúció* alapkérdése az, hogy hogyan lehet ezeket a *megoldó (solver) csomagokat* egy általános keretbe beilleszteni úgy, hogy az egész rendszer továbbra is rezolúciós alapon működjön, sőt esetleg a klasszikus bináris rezolúciós alapmódszer is működhessen.

A kérdés megválaszolásához az általános rezolúciós elvből kell kiindulnunk. Ha az általános rezolvensképzés műveletét a rendszerbe bekapcsolt megoldókra bízhatjuk, aminek eredményeképpen a megoldók a rezolúciós elvet kielégítő rezolvens klózokat állítanak elő, és ezek, (valamint esetleg a klasszikus kétágú rezolúciót megvalósító

---

[22] Krauth Péter-Márkus András: OPS5: egy sikeres eszköz szakértői rendszerek készítésére [KrMá85]

[23] Thom Frühwirth: Theory and Practice of Constraint Handling Rules [CHR94]

[24] Kilián Imre: Contralog: egy előre haladó Prolog motor és alkalmazása ℜeALIS nyelvi elemzésre [Ki11.1]

[25] Kilián Imre: Tárgymodell változatok a ℜeALIS nyelvi elemzéshez [Ki11.2]

lépések) során sikerül a nyilvánvalóan kielégíthetetlen üres klózhoz jutni, akkor a feladatot megoldottuk.

Az ilyen módon beillesztett megoldó csomagoknak tehát a következő feltételeket kell teljesíteniük:

1. Fel kell ismerniük azokat a klózokat, (esetleg azok literálszakaszait is), amelyek a rezolvensképzési lépésben szerepet játszhatnak.
2. Végre kell hajtaniuk a rezolvensképzést, és elő kell állítaniuk a rezolvens klózt.

A megoldó csomagokat vezérlő algoritmus feladatai a következőkben foglalhatók össze:

1. nyilván kell tartania, hogy mely klózok mely szakaszait mely megoldók ismerték fel
2. több lehetőség (több megoldó csomag, illetve több klóz-részhalmaz) között a következtetési stratégia által meghatározott fontossági szabályok alapján választania kell
3. meg kell hívnia a kiválasztott megoldót a kiválasztott klózhalmazra
4. a létrehozott rezolvens klózt fel kell vennie az elmélet klózai közé

A megoldók egyes esetekben egyenértékű átalakításokat is végezhetnek, (pl. normál alak képzéssel), és javaslatot adhatnak egyetlen klóz, vagy egy literálszakaszának cseréjére, de ez a logikai működést nem befolyásolja.

A beágyazott rezolúciós elv korlátozott alkalmazását végző szoftvert hozott létre a szerző a SILK projekt modellellenőrző csomagjában.[26]

2.3.2.4 Paradoxonok és a Gödel-féle nemteljességi tételek

A paradoxon szónak többféle értelmezése is van: Paradoxonnak neveznek egyes fogalmazás-, vagy meghatározás-beli pontatlanságokból vagy következtetési hibákból fakadó furcsa eredményeket, végtelen és körkörös definíciókat, vagy csak egyszerűen meghökkentő jelenségeket is. Ez utóbbinak egyik szép példája a *születésnap paradoxon*. Eszerint már egy 23 fős társaságban 50% az esélye, hogy van kettő, az év azonos napján született ember. Ebben semmilyen ellentmondás nincs, az állítás elemi (de nem egyszerű!) valószínűségelméleti összefüggésekkel igazolható.

Ebben a szakaszban a *paradoxonoknak* az *önellentmondást* meghatározó jelentését tárgyaljuk, amit szintén még az ókori görögök fedeztek fel.

A híres *Epimenidész-paradoxon* szerzője krétai filozófus és a sziget Knosszosz nevű épületegyüttesében és szentélyében működő Zeusz-pap volt, aki egy alkalommal a következőket jelentette ki: „Minden krétai hazudik”. Ha minden egyéb értelmezési változattól eltekintve a mondatot úgy értelmezzük, hogy minden krétai minden időben hazudik, és Epimenidész is krétainak minősül, akkor pl. az állítás nem eldönthetetlen. Ha ugyanis feltételezzük, hogy a mondat igaz volt, akkor az Epimenidészre, és az általa kijelentett mondatra is vonatkozik. Tehát egyáltalán nem igaz, hogy minden krétai hazudik, mert a feltételezés szerint legalábbis akkor Epimenidész igazat mondott. Ha viszont a mondat hamis, akkor lehetséges, hogy egyes krétaiak néha-néha mégis igazat mondanak, de ez egyáltalán nem zárja ki, hogy sajátmaga hamis legyen.

---

[26] Kilián Imre: Mixed strategy reasoning. An approach for resolution based verification of OCL constraints in UML models [Kil07]

Az Epimenidész-paradoxon egyik, tényleg eldönthetetlen egyszerűsítése a *hazudós paradoxon*. Ez csupán annyit állít: „Hazudok." Vagy: „Ez a mondat hamis." Ha a mondat igaz, akkor hamis. Viszont ha a mondat tényleg hamis, akkor igaznak kell lennie.

Szintén a hazudós paradoxon egyik változata Mérő László matematikus[27] mondata: „Ebben a mondatba harom hiba van." (sic!) Két hiba – két nyelvtani hiba – még nyilvánvaló, és az is, hogy a harmadik hiba a mondat logikai értelmére vonatkozik. Ha feltételezzük, hogy a mondat igaz, akkor a logikai érték nem adódik hozzá a két nyelvtani hibához, ezért csak két hiba van a mondatban. Tehát a mondat hamis. Ha viszont a mondat hamis, akkor a logikai értéke a harmadik hiba, ami miatt a mondatnak igaznak kell lennie.

Talán még az isten-ellenérv-paradoxont érdemes megemlíteni, bár határozottan nem a vallási – vagy éppen vallásellenes – jelentősége miatt. „Tud-e a Világügyelő akkora követ teremteni, amekkorát nem bír felemelni?" A mindenhatóság elve mindenképpen megdől: ha nem tud, azért, ha tud, azért, mert nem bírja felemelni.

Ez utóbbinak azonban vannak filozófiai következményei: a korlátlanul univerzális tulajdonságokra, a mindenhatóságra, a mindentudásra és effélékre vonatkoznak hasonló paradoxonok, amelyek miatt ezek logikailag nem tarthatók.

1901-ben Bertrand Russel angol polihisztor, matematikus, társadalomtudós, író volt az, aki a hazudós paradoxont és a fentieket halmazelméleti paradoxonná egyszerűsítette le, amit azóta *Russel-paradoxonnak* neveznek. Tétele szerint *nem létezhet mindent magába foglaló (univerzális) halmaz*. Ha ugyanis létezne, akkor részhalmazainak is létezniük kellene. Az univerzális halmaznak viszont nem létezhet azon részhalmaza, amely csak a sajátmagukat nem tartalmazó halmazokat tartalmazza. Ha ugyanis ez létezne, akkor a következő két lehetőség áll fenn. Ha ez a részhalmaz tartalmazná sajátmagát, akkor a definíció szerint nem tartalmazhatná sajátmagát. Ha viszont nem tartalmazná saját magát, akkor a definíció szerint tartalmaznia kellene saját magát. Mivel minden alternatív bizonyítási ágon ellentmondásra jutottunk, ez megdönti a kezdő feltételt, vagyis univerzális halmaz nem létezhet.

A Russel paradoxon felfedezésének és tisztázásának alapvető szerepe volt a modern halmazelmélet megszületésében. Mivel a Russel paradoxon és átfogalmazásai mind abból a feltételezésből születtek, hogy létezhet olyan halmaz, amely saját magának az eleme ($X \in X$), ezért a Russel utáni halmazelméleti és logikai iskolák a halmazok fogalmából ezt a lehetőséget azóta egyértelműen kizárják.

A történelem során felfedezett paradoxonoknak különös elméleti hátteret ad Kurt Gödel német matematikus 1931-ben bebizonyított *első nemteljességi tétele*. Ennek tudományfilozófiai jelentősége felmérhetetlen, legalább akkora, mint a Heisenberg-féle határozatlanságé vagy az Einstein-féle relativitáselméleté a fizikában. A tétel a következőket állítja:

- Tegyük fel, hogy van egy formalizált elméletünk, amely ellentmondásmentes (vagyis nem vezethető le egyetlen formula és ugyanannak a negáltja is).
- Legyen az elméletünkkel kifejezhető az egész számok aritmetikája (vagyis vagy analógiája, vagy bővítése az ismert Peano axiómákkal megadott számelméletnek).

[27] Mérő László: Észjárások [Mé89] 12. old.

- Ekkor teljesen biztosan létezik az elméletnek olyan állítása, amely eldönthetetlen: vagyis nem bizonyítható sem az állítás, sem a negáltja.

Még egyszerűbb megfogalmazással: az egész számok bonyolultságát elérő logikai rendszerekben szükségképpen van eldönthetetlen állítás. Nem tudunk tehát olyan logikai rendszert felépíteni, amiben nem lehetne eldönthetetlen állítást megfogalmazni. A jogi területre vonatkoztatva ezt pedig úgy is érthetjük, *nem lehetséges olyan jogi rendszert felépíteni, amely minden helyzetet egyértelműen és tökéletesen leír.* Még gyakorlatiasabban: rabló-pandúr játék szemlélői vagyunk. A törvények kicselezői ellen egyre bonyolultabb és szövevényesebb törvényi rendszereket hoznak, de minden a törvény csak a tisztelőinek jelent akadályt. Aki tényleg át akarja őket lépni, teljesen biztosan előbb-utóbb megtalálja a kiskaput.[28]

Gödel az első nemteljességi tétele után hamarosan – még ugyanabban az évben – bebizonyította a *második nemteljességi* tételét is. A tétel az első tétel következtetéseit fokozza még tovább, és a következőképpen szól:

- Tegyük fel, hogy van egy formalizált, ellentmondásmentes meta-elméletünk – vagyis olyan elmélet (pl. a matematikai logika valamely részosztálya), amellyel más elméleteket leírhatunk, és amivel legalább a természetes számok aritmetikája kifejezhető.
- Semmilyen efféle meta-elmélet segítségével nem bizonyítható minden formalizált tárgy-elmélet ellentmondás-mentessége.

A második nem-teljességi tételnek még komolyabb következményei voltak a matematikustársadalomra. Lényegében megdőltek a matematikai tudományok ellentmondásmentességére vonatkozó kezdeti álmok: a tétel az efféle célokat kitűző matematikai-logika iskolák lába alól húzta ki egy csapásra a talajt. A matematikatörténetben pedig azóta egyenesen „Gödel-előtti" és „Gödel-utáni" korszakokról beszélnek.

2.3.3 Leíró logikák

A leíró logikák (Description Logics, DL)[29] az elsőrendű logikák olyan részosztályai, amelyekben csak egyparaméteres és kétparaméteres predikátumok léteznek. A fogalom maga a korábbi *terminológia-* vagy *fogalomleíró nyelvek* kikristályosításából alakult ki a 80-as években. A leíró logikákon működő különböző ösztönös bizonyító algoritmusok után a 90-es években fejlesztették ki a *tabló algoritmust*, amelyet, illetve amelynek továbbfejlesztéseit az egyes DL megoldó szoftverekben azóta is használják.[30]

A leíró logikák fejlődését komolyan meglendítette a 2000-es évtizedben a Szemantikus Web elképzelés egyes elemeinek megvalósítása.[31]

A leíró logikák rendszere kicsit hasonlít az objektum-orientált szoftver technológiához, mert az egyparaméteres predikátumok fogalmak valamiféle osztályozására (klasszifikációjára) használhatók, míg a kétparaméteres predikátumok (kétoldalú relációk) a „tulajdonság", illetve „szerep" fogalom matematikai leképezései. Összetett

---

[28] Mérő László: Észjárások [Mé89] 231. old.

[29] F. Baader - D. Calvanese – D. McGuinness et al. eds: The Description Logic Handbook. Theory, Implementation and Applications [BCG03]

[30] Nagy Zsolt: Leíró logikán alapuló tudáskezelő rendszer [Nagy04]

[31] T. Berners-Lee - J. Hendler - O. Lassila The Semantic Web [BLHL01]

függvénykifejezéseket vagy a fentieknél bonyolultabb predikátumkifejezéseket a leíró logikák egyáltalán nem engednek meg.

A nyelv segítségével egyfelől a fogalom-elnevezésekkel és a megengedett megszorítások segítségével újabb fogalom-meghatározások hozhatók létre, de ugyanígy a szerepekből is lehet egyes műveletek segítségével újabb szerepkifejezéseket létrehoznunk, és azon keresztül pedig szerepfogalmakat bevezetnünk.

A leíró logikák egy különböző erejű nyelvekből álló családot alkotnak, amelynek az elemeit a következőkben adjuk meg.[32] [33]

2.3.3.1 Az $\mathcal{AL}$ leíró logikai nyelvcsalád

Az $\mathcal{AL}$ (attributive language) a leíró logikai nyelvcsalád legegyszerűbb tagja. A nyelv terminológiai axiómákból, úgynevezett T-dobozokból (T-box), valamint adataxiómákból, úgynevezett A-dobozokból (A-box) áll. A terminológiai axiómák a nyelv által leírt halmaz- fogalom- terminusrendszert írják le, amelyekre az alábbi formális leírásban sok helyen C_i (class) betűvel hivatkozunk. Ezek között kétoldalú relációk (úgynevezett szerepek) és egyéb kvantifikációs, valamint számossági megkötések állhatnak fenn, amelyekre viszont R_i betűkkel hivatkozunk. Az adataxiómák ezzel szemben az egyes osztályokat és szerepeket azok példányaival együtt nevezik meg. Fejlettebb leíró logikák szerepaxiómák megadását is lehetővé teszik, ezeket egyes szerzők R-doboznak (R-box) nevezik. A leíró logikai nyelvek névjegye ezek miatt a következőképpen adható meg:

$\sum = [\mathcal{T}, \mathcal{R}, \mathcal{J}]$

ahol $\mathcal{T}$ az 1 argumentumú predikátumok (fogalmak, terminusok), $\mathcal{R}$ a 2 argumentumú predikátumok (szerepek, roles), $\mathcal{J}$ pedig az egyedek (individuals) halmaza, amelyeket az egyedi neveikkel (azonosítóikkal) adunk meg.

Az $\mathcal{AL}$ nyelv terminológiai axiómái formálisan a következőképpen határozhatók meg:

$\mathcal{AL} := \{ C_1 \subseteq C_2 \mid C_1 \equiv C_2 \}$, ahol $C_1, C_2 \in \mathcal{AL_e}$

$\mathcal{AL_e} := \{C ::= \bot \mid \top \mid A \mid \neg A \mid C_1 \wedge C_2 \mid \forall R.C \mid \exists R.\top\}$,
ahol $A \in \mathcal{T}$, $R \in \mathcal{R}$ és $C, C_1, C_2 \in \mathcal{AL_e}$

Az $\mathcal{AL_e}$ nyelvet nevezhetjük az *fogalom- vagy osztálykonstruktorok* nyelvének is. Az $\mathcal{AL}$ nyelvet a *terminológiai axiómák* nyelvének nevezzük. A nyelv kétféle axiómából áll: az egyenlőséget rögzítő axiómát szokás *meghatározó-*, vagy *definíciós-*, a részhalmaz viszonyt rögzítőt viszont szokás *tartalmazási axiómának* is nevezni. Megjegyezzük, hogy az osztály-tartalmazási axióma mellett az egyenlőség megadása csupán nyelvtani könnyítés (syntactic sugar), hiszen az egyenértékűség egy tartalmazás-párral is leírható lenne.

A terminológiai axiómák mellett a DL nyelvek *egyed- és szerepaxiómák* megadását is lehetővé teszik, ez az összes leíró logikai változatban egyöntetűen néz ki. Az egyedaxiómák megadása adategyedek és a szerepkapcsolatok megadásával történik. Az

---

[32] Motik, Bruno - Patel, Peter F. - Schneider-Bijan, Parsia (eds, 2009): OWL2 Web Ontology Language Structural Specification and Functional Style Syntax [OWL209]

[33] Szeredi P.-Lukácsy G.-Benkő T.: A szemantikus világháló elmélete és gyakorlata [SzLB05] 181. old.

adategyedek egy adott egyed (példány) egy fogalomhoz tartozását rögzítik a következőképpen:

T(I), ahol $T \in \mathcal{T}$, és $I \in \mathcal{J}$.

A szerepkapcsolatok azt rögzítik, hogy két egyed (példány) egymással valamilyen szerepkapcsolatban van, a következőképpen:

$R(I_1,I_2)$, ahol $R \in \mathcal{R}$, $I_1,I_2 \in \mathcal{J}$

A DL leíró logika nyelvcsalád esetében szokásos az *interpretáció* formális rögzítése is. Ennek során a terminológiai (és a szerep-) axiómákban használt konstruktorok valamilyen konkrét halmazba történő leképezését írjuk le (egyébként eléggé magától értetődő módon).

Legyen adva (az elsőrendű nyelvekhez hasonlóan most is) a tárgyalási világegyetem $\mathcal{U}$ nem üres halmaza, amely egyedekből áll. Az interpretáció egy I függvény, amely az egyes osztálykonstruktorokat az $\mathcal{U}$ világegyetem egy-egy részhalmazába képezi le. (Szerepkonstruktorokat is tartalmazó leíró logikai nyelv-változat esetében egy szerepkonstruktort a világegyetem feletti kétoldalú viszonyba, azaz $\mathcal{U} \times \mathcal{U}$ egy részhalmazába képez le).

Az $\mathcal{AL}$ nyelv interpretációja a következőképpen foglalható össze:

- $I(C) \subseteq \mathcal{U}$, ahol $C \in \mathcal{T}$ — egy osztály a világegyetem egy részhalmaza
- $I(R) \subseteq \mathcal{U} \times \mathcal{U}$, ahol $R \in \mathcal{R}$ — egy szerep a világegyetem feletti kétoldalú viszony
- $I(a) \in \mathcal{U}$, ahol $a \in \mathcal{J}$ — egy egyedmegjelölés a világegyetem egy elemét jelöli
- $I(\bot)=\varnothing$ — a fenékjel az üres halmazt jelöli
- $I(\top)=\mathcal{U}$ — a tetőjel a teljes világegyetemet jelöli
- $I(\sim A)= \mathcal{U}-I(A)$ — a negációjel az osztályok komplementerét jelöli
- $I(C_1 \wedge C_2)= I(C_1) \wedge I(C_2)$ a metszetjel az osztályok metszetét jelöli
- $I(\forall R.C)=\{a \in \mathcal{U} \mid \forall b\ (a,b) \in I(R) \rightarrow b \in I(C)\}$ az osztály azokból az elemekből áll, amelyek minden R szereppel kötött b objektuma egyben a C osztálynak eleme
- $I(\exists R.\top)=\{a \in \mathcal{U} \mid \exists b\ (a,b) \in I(R)\ \}$ az osztály azokból az elemekből áll, amelyeknek létezik az R szereppel kötött b objektuma (de erre vonatkozólag további megkötés nincs)

Az $\mathcal{AL}$ nyelvet különböző újabb axióma-megadási lehetőségekkel bővítve kapjuk a nyelvcsalád többi elemét.

Ha az $\mathcal{AL}$ nyelvhez a *halmazegyesítéses* terminológiai axiómalehetőséget is hozzávesszük, akkor kapjuk az $\mathcal{ALU}$ nyelvet (Attributive Language with Union). Az adataxiómák megadási módja változatlan.

$\mathcal{ALU} := \{\ C_1 \subseteq C_2 \mid C_1 \equiv C_2\ \}$, ahol $C_1, C_2 \in \mathcal{ALU}_e$

$\mathcal{ALU}_e := \mathcal{AL}_e \vee \{\ C_1 \vee C_1\ \}$, ahol $C_1, C_2 \in \mathcal{ALU}_e$

Ha az $\mathcal{AL}$ nyelvhez hozzávesszük a *teljes egzisztenciálisan kvantált* korlátozási lehetőséget, akkor az $\mathcal{ALE}$ nyelvet kapjuk. Az adataxiómák megadása változatlan.

$\mathcal{ALE} := \{ C_1 \subseteq C_2 \mid C_1 \equiv C_2 \}$, ahol $C_1, C_2 \in \mathcal{ALE}_e$

$\mathcal{ALE}_e := \mathcal{AL}_e \vee \{ \exists R.C \}$, ahol $R \in \mathcal{R}$, és $C \in \mathcal{ALE}_e$

Ha az $\mathcal{AL}$ nyelvhez hozzávesszük a *teljes negált fogalommegadás* lehetőségét, akkor (az $\mathcal{ALC}$ nyelvet (Attributive Language with Complement) kapjuk. Az adataxiómák megadása változatlan.

$\mathcal{ALC} := \{ C_1 \subseteq C_2 \mid C_1 \equiv C_2 \}$, ahol $C_1, C_2 \in \mathcal{ALC}_e$

$\mathcal{ALC}_e := \mathcal{AL}_e \vee \{ \sim C \}$, ahol $C \in \mathcal{ALC}_e$

Ha az $\mathcal{AL}$ nyelvhez hozzávesszük a *számosság-korlátozások* lehetőségét, akkor az $\mathcal{ALN}$ nyelvet (Attributive Language with Cardinality Restrictions) kapjuk. Az adataxiómák megadása változatlan.

$\mathcal{ALN} := \{ C_1 \subseteq C_2 \mid C_1 \equiv C_2 \}$, ahol $C_1, C_2 \in \mathcal{ALN}_e$

$\mathcal{ALN}_e := \mathcal{AL}_e \vee \{ \leq nR \mid \geq nR \}$, ahol $R \in \mathcal{R}$, n természetes szám

Ha az $\mathcal{AL}$ nyelvhez hozzávesszük a *funkcionális számosság-korlátozások* lehetőségét, akkor az $\mathcal{ALF}$ nyelvet (Attributive Language with Functional Properties) kapjuk. Ez az előző ($\mathcal{ALN}$) nyelv különleges esetének is tekinthető, az új axiómák csak az 1-szeres, illetve korlátozás nélküli n-szeres többszörösségű kapcsolatok leírására alkalmasak. Az adataxiómák megadása változatlan.

$\mathcal{ALF} := \{ C_1 \subseteq C_2 \mid C_1 \equiv C_2 \}$, ahol $C_1, C_2 \in \mathcal{ALF}_e$

$\mathcal{ALF}_e := \mathcal{AL}_e \vee \{ \leq 1R \mid \geq 2R \}$, ahol $R \in \mathcal{R}$, n természetes szám

Ha az $\mathcal{AL}$ nyelvhez hozzávesszük a *szerephierarchiák* megadásának lehetőségét, akkor az $\mathcal{ALH}$ nyelvet (Attributive Language with Role Hierarchies) kapjuk. A nyelv fogalmi és adataxiómáinak megadása is változatlan, de a szerephierarchiákkal megjelennek újabb szerepaxiómák.

$\mathcal{ALH} := \mathcal{AL} \vee \{ R_1 \subseteq R_2 \mid R_1 \equiv R_2 \}$, ahol $R_1, R_2 \in \mathcal{R}$

Ha az $\mathcal{AL}$ nyelvhez hozzávesszük az *inverz szerepmegadás* lehetőségét, akkor az $\mathcal{ALI}$ nyelvet (Attributive Language with Inverse Properties) kapjuk. A nyelv fogalmi és adataxiómáinak megadása is változatlan, de újabb szerepaxiómák jelennek meg.

$\mathcal{ALI} := \mathcal{AL} \vee \{ R_1 \equiv R_2^{-1} \}$, ahol $R_1, R_2 \in \mathcal{R}$

Ha az $\mathcal{AL}$ nyelvhez hozzávesszük a *számosság-korlátozások* megadásának teljes lehetőségét, egy szerep túloldali fogalmára vonatkozólag is, akkor az $\mathcal{ALQ}$ nyelvet (Attributive Language with Qualified Cardinality Restrictions) kapjuk. A nyelv adataxiómáinak megadása is változatlan.

$\mathcal{ALQ} := \{ C_1 \subseteq C_2 \mid C_1 \equiv C_2 \}$, ahol $C_1, C_1 \in \mathcal{ALQ}_e$

$\mathcal{ALQ}_e := \mathcal{AL}_e \vee \{ \leq nR_S.C \mid \geq nR_S.C \}$, ahol $C \in \mathcal{ALQ}_e$, n természetes szám, $R_S \in \mathcal{R}$, R_S nem tranzitív szerep sem önmagában, sem közvetlenül.

Ha az $\mathcal{AL}$ nyelvhez hozzávesszük az alapvető szerepaxiómákat, a szerep részhalmaz viszony, a szerep reflexivitás és irreflexivitás, valamint a szerep különbözőség (diszjunktság) megadásának teljes lehetőségét, akkor az $\mathcal{ALR}$ nyelvet (Attributive Language with limited complex Role axioms) kapjuk. A nyelv adataxiómáinak megadása is változatlan.

$\mathcal{ALR} := \mathcal{ALQ} \vee \{ R_1 \subseteq R_2 \mid \text{Refl}(R) \mid \text{Irrefl}(R) \mid \text{Disj}(R_1,R_2)\}$, ahol $R, R_1, R_2 \in \mathcal{R}$

Ha az $\mathcal{ALR}$ nyelvhez hozzávesszük a *tranzitív szerepmegadás* lehetőségét, akkor az $\mathcal{ALR}^+$ nyelvet kapjuk. A nyelv fogalmi és adataxiómáinak megadása is változatlan, de megjelennek újabb szerepaxiómák (R-box).

$\mathcal{ALR}^+ := \mathcal{ALR} \vee \{\text{Trans}(R)\}$, ahol $R \in \mathcal{R}$

Ha az $\mathcal{AL}$ nyelvhez hozzávesszük az adattípusok, adattípusú szerepek és adatértékek megadásának lehetőségét, akkor az $\mathcal{ALD}$ nyelvet kapjuk.

A fent jelzett toldalékokkal az $\mathcal{AL}$ alapnyelvet tetszőlegesen (modulárisan) bővíthetjük. Az $\mathcal{ALCR}^+$ nyelvet szokás $\mathcal{S}$-sel is jelölni, és a további nyelvtoldalékokat már csak ehhez képest megadni.

Nevezetes ezek közül a $\mathcal{SHIQ}$, mert ez képezi a ma már szabványnak számító *OWL ontológia-leíró nyelv* alapját. A nyelvnek teljes meghatározása a következőkben olvasható:

$\mathcal{SHIQ} := \{ C_1 \subseteq C_2 \mid C_1 \equiv C_2 \mid R_1 \subseteq R_2 \mid R_1 \equiv R_2 \mid R_1 \equiv R_2^{-1} \mid \text{Trans}(R)\}$,
ahol $C_1, C_2 \in \mathcal{SHIQ}_e$, $R, R_1, R_2 \in \mathcal{SHIQ}_e$

$\mathcal{SHIQ}_e := \{C ::= \bot \mid \top \mid A \mid \neg C \mid C_1 \wedge C_2 \mid \forall R.C \mid \exists R.\top \mid \leq n R_S.C \mid \geq n R_S.C \}$, ahol $A \in \mathcal{T}$, $R, R_S \in \mathcal{R}$ és $C, C_1, C_2 \in \mathcal{SHIQ}_e$

A $\mathcal{SHIQ}$ leíró logikai nyelvnek tehát a következő axióma-megadási lehetőségei vannak:

- Az $\mathcal{AL}$ nyelvet követve fogalomhierarchiákat is megadhatunk
- Szerepmegadás történhet atomi szerep, és inverz szerep segítségével ($\mathcal{I}$).
- Megadhatunk nemcsak fogalom, hanem szerephierarchiákat is ($\mathcal{H}$).
- Tetszőleges számosságkorlátozást köthetünk ki ($\mathcal{Q}$).
- Megadhatunk tranzitív szerepet is ($\mathcal{R}^+$).

Az OWL ontológia-leíró nyelv korábbi változata a $\mathcal{SHIQ}$-hoz hasonló $\mathcal{SHOIQ}$ nyelvet veszi alapul. Ezt támogatja a Protégé ontológiaszerkesztő program korábbi, 3.x változata is. Az O betűvel jelzett nyelvi bővítés egyedfogalmak, azaz tulajdonnevekből képzett egyelemű osztályok létrehozását teszi lehetővé: (pl. {Hungary}).

A 2009-ben nyilvánosságra hozott OWL2 szabvány háromféle kifejezőerejű nyelvet rögzít,[34] amit a Protégé ontológiaszerkesztő környezet[35] 4.x változata támogat is:

---

[34] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler: OWL 2: The next step for OWL [CHMP08]

[35] http://protege.stanford.edu, a Protégé ontológiaszerkesztő program honlapja. Elérés: 31-Jan-2013

- Az OWL-Lite nyelvváltozat a korábbiak szerint meghatározott $\mathcal{SHIF}^{(\mathcal{D})}$ nyelvvel egyenértékű, vagyis megadhatunk:
    - osztály és szerephierarchiákat
    - adattípusokat és adatszerepeket
    - funkcionális szerepeket
    - tranzitív szerepeket,
- Az OWL-DL változat a $\mathcal{SHOIQ}^{(\mathcal{D})}$ nyelvvel egyenértékű, vagyis az OWL-Lite-hoz képest megadhatunk egyedfogalmakat, azaz tulajdonnevekből képzett egyelemű osztályokat is.
- Az OWL-Full pedig a $\mathcal{SROIQ}^{(\mathcal{D})}$ nyelvvel egyenértékű, vagyis az $\mathcal{R}$ jelölés miatt megadhatunk szerepaxiómákat (R-boxokat), vagyis különböző szereptulajdonságokat is:
    - Az OWL-DL változathoz képest ez a szerepösszetételek lehetőségét jelenti többletként. (Pl.: $\texttt{helyszíne o része} \subseteq \texttt{helyszíne}$)

2.3.3.2 Tételbizonyítás a tabló algoritmussal

A tabló algoritmus általánosságban egy tételbizonyító algoritmus, amely összetett logikai kifejezések kielégíthetőségét bizonyítja különböző erejű logikák esetén. Az algoritmus kifejlesztése több szerző nevéhez köthető, a helyességével és a bonyolultságával (hatékonyságával) összefüggő matematikai bizonyításokat Baader és Sattler cikkeiből olvashatjuk el. [36] [37]

Az algoritmus a futása közben egy *szemantikus tablót* állít elő, amelyet nulladrendre alkalmazva *döntési fának* is nevezhetünk. A fa egyes csomópontjaihoz logikai kifejezések halmazát rendeljük, a kielégíthetőséget pedig a fa egyes csomópontjainak (levélelemeinek) a vizsgálatával lehet eldönteni. Mivel a fa szélességét és mélységét is korlátozzák a kiértékelendő kifejezés elemei, ezért ez mindig egy véges adatszerkezet, *az algoritmus minden esetben véget ér*. Általános esetben viszont a faszerkezet előállítása és a vizsgálata akár *exponenciális idő- és tárigényű is lehet*, a tényleges bonyolultság komolyan függ a bizonyítandó probléma szerkezetétől és a konkrét logikai nyelvtől. Míg az $\mathcal{AL}$ és $\mathcal{ALC}$ nyelvosztályt eldöntő algoritmus PTIME (polinomiális), illetve PSPACE osztályba tartozik, addig a $\mathcal{SHIF}$ nyelvosztály eldöntése már EXPTIME bonyolultságú, az OWL-DL-nek megfelelő $\mathcal{SHOIN}$ nyelvosztály pedig a NEXPTIME bonyolultságot is eléri.[38] Ha pedig az OWL-Full-nak megfelelő $\mathcal{SROIQ}$ nyelvosztályt tekintjük, akkor a nyelv eldönthetetlenné válik. Megjegyezzük, hogy mindezek felső becslést jelentenek, valós esetekben ezeknél lényegesen kedvezőbb idő- és tárigényű futási eredmények is előfordulhatnak.[39]

---

[36] F. Baader – U. Sattler: An Overview of Tableau Algorithms for Description Logics [BaSa00]

[37] F. Baader - D. Calvanese – D. McGuinness et al. eds: The Description Logic Handbook. Theory, Implementation and Applications [BCG03]

[38] A nyelvosztályra vonatkozó bizonyító algoritmus futása a bemenő mondat hosszával exponenciális arányú időt vesz igénybe determinisztikus Turing gépen (EXPTIME), illetve nemdeterminisztikus Turing gépen (NEXPTIME) – Lovász: Computational Complexity [Lov04]

[39] Egy érdekes – és hitelesnek tűnő – internetes alkalmazás a http://www.cs.man.ac.uk/~ezolin/dl/ címen (elérés: 1-Aug-2012) a pontos nyelvkonstrukció ismeretében a megadja a hozzátartozó bonyolultsági

A tabló algoritmus fája nem előzetesen (a priori) adott, hanem az algoritmus futása közben jön létre. A fa csomópontjai egyedeket jelképeznek, amelyhez az algoritmus fogalom-kifejezések egy halmazát rendeli, méghozzá éppen azt a halmazt, amelynek a csomópont-egyed az egyik példánya. Az éleihez pedig konkrét szerepeket rendelünk úgy, hogy akkor köt össze két csomópontot egy él, ha a két példány között az említett szerepkapcsolat fennáll. További részalgoritmusok döntik el, hogy a szerepkapcsolatokban megjelölt kvantifikációs feltétel teljesül-e vagy sem. Úgy is fogalmazhatunk, hogy az algoritmus egy konstruktív bizonyító algoritmus, amely – amennyiben a terminológiai viszonyok leírta elmélet ellentmondásmentes – absztrakt egyedek felett a szereprelációk segítségével egy faszerkezetet épít fel.

Kezdetben a fa csupán a gyökércsomópontból áll, amelyhez azt a fogalom-kifejezést rendeljük, amelyre a kielégíthetőséget vizsgálni kívánjuk. A fa az algoritmus egyes lépéseiben újabb csomóponttal bővülhet: az ilyen lépéseket *kiterjesztő lépéseknek* nevezzük. A faépítés utáni lépésben egy vizsgálat következik, amely a fa egyes ágainak kielégíthetőségét dönti el. Ha egy ág kielégíthetetlen, akkor a fa az adott irányban nem nő tovább. Ha a fa összes ága kielégíthetetlen, akkor az eredeti kifejezés is az.

Amennyiben egyenlőségeket, illetve egyenlőtlenségeket is kívánunk az algoritmussal kezelni, úgy a futás közben folyamatosan tárolunk egy $I=\{x\neq y|x,y\in V$, a facsomópontok halmaza$\}$ egyenlőtlenségekből álló halmazt: ez az egyes csomópontok összevonhatóságát (ld. az algoritmus leírását) akadályozhatja meg.

Az algoritmus egyes lépéseiben az alább részletezett szabályok alkalmazhatók. A szabályalkalmazást szokás *tüzelésnek* is nevezni. Amennyiben egy adott lépésben több szabály is tüzelőképes, akkor tetszőlegesen választhatunk a szabályok között. Ezt a viselkedést *„don't care nondeterminism"*, vagy *tetszőleges választás* néven nevezik. Az alább pontosan leírt szabályok azonban bizonyos esetekben több átalakítást is rögzítenek. Ezek *tényleges nemdeterminisztikus választások*, amelyeket viszont széltében vagy mélységében keresős stratégiával kezelni kell, vagyis a változatok mindegyikét ki kell értékelni és be kell járni.

Az algoritmus működését a *negációs normál alakban* levő kifejezéseken magyarázzuk el. Ez diszjunkciókat és konjunkciókat tartalmazó alak, melyben a negációk csak (fogalom vagy szerepmegkötésből származó) literálokra vonatkozhatnak. Ezt az alakot az **ALCN** nyelvre a következő azonosságok alkalmazásával lehet létrehozni; az átalakítások végső célja, hogy összetett kifejezések esetén a negációt a szerkezetben minél beljebbre vigyük. A negációs normál alakot létrehozhatjuk egy előfeldolgozó fázis keretében, de az állítások feldolgozásának idejében is.

$\sim\sim A$	➔	A	kétszeres negáció kiküszöbölése
$\sim(A \vee B)$	➔	$\sim A \wedge \sim B$	átemelés diszjunkción (De Morgan)
$\sim(A \wedge B)$	➔	$\sim A \vee \sim B$	átemelés konjunkción (De Morgan)
$\sim(\forall R.A)$	➔	$\exists R.\sim A$	átemelés univerzális kvantoron
$\sim(\exists R.A)$	➔	$\forall R.\sim A$	átemelés egzisztenciális kvantoron
$\sim(\leq nR)$	➔	$\geq(n+1)R$	átemelés számosság korláton
$\sim(\geq nR)$	➔	$\leq (n-1)R$	átemelés számosság korláton

---

mértéket. Mindamellett a honlap egy nagyon részletes hivatkozáslistát is tartalmaz a Szemantikus Világháló és a leíró logikák témájában.

Az algoritmus futása kicsit pontosítva:[40]

1. Az algoritmus a kiinduló tablószerkezet létrehozásával kezdődik. Ez csupán egyetlenegy csomópontot jelent, amelyhez a kiindulási fogalom-kifejezést rendeljük hozzá. Ez lesz a felépítendő fa gyökérpontja.
2. ÉS szabály: ha egy csomópont egy $A \wedge B$ konjunkciót tartalmaz, de sem az A, sem a B kifejezést külön nem tartalmazza, akkor a csomóponthoz hozzávesszük az A és a B kifejezést is.
3. VAGY szabály: ha egy csomópont egy $A \vee B$ diszjunkciót tartalmaz, de sem az A, sem a B kifejezést külön nem tartalmazza, akkor a csomóponthoz nemdeterminisztikusan hozzávesszük az A, majd a B részkifejezést.
4. LÉTEZIK (kiterjesztő) szabály: ha egy x csomópont a $\exists R.C$ kifejezést tartalmazza, de nem létezik olyan x-ből kiinduló él, amely címkéje R lenne, és célcsomópontja címkéiben pedig C szerepelne. Ilyenkor felveszünk egy új csomópontot, amelyet C-vel, és a hozzá vezető, x-ből induló élt pedig R-rel címkézzük.
5. BÁRMELY szabály: ha egy x csomópont a $\forall R.C$ kifejezést tartalmazza, de az x-ből kiinduló, R-rel címkézett éleknek van olyan célcsomópontja, amelyek nem tartalmazzák a C fogalom-kifejezést, akkor az összes ilyen célcsomóponthoz hozzávesszük a C kifejezést is.
6. NAGYOBB-EGYENLŐ (kiterjesztő) szabály: ha egy x csomópont a $\geq nR$ kifejezést tartalmazza, de nincs legalább n darab, x-ből kiinduló, R-rel címkézett, nem összevonható él. Ilyenkor a hiányzó számban felveszünk új éleket, melyeket R-rel címkézünk meg, amelyek célcsomópontjaihoz az üres halmazt rendeljük. Egyenlőségvizsgálat kezelése esetében az egyenlőtlenségeket tároló I halmazt bővítjük: az új csomópontok nem egyeznek meg sem egymással, sem a már meglevő szomszédjaikkal.
7. KISEBB-EGYENLŐ szabály: ha egy x csomópont a $\leq nR$ kifejezést tartalmazza, de legalább n darab x-ből kiinduló, R-rel címkézett éle van, amelyből legalább kettő összevonható. Ilyenkor az összevonható éleket páronként nemdeterminisztikusan összevonjuk a következőképpen:
    - A csomópontok halmazából eltávolítjuk a pár egyik elemét
    - A megmaradó csomópont címkehalmazait egyesítjük
    - A törölt csomópontba vivő élt töröljük
    - A törölt csomópontból kiinduló éleket a megmaradó csomópontból indítjuk, és a kiinduló élhalmazokat egyesítjük.
    - Az egyenlőtlenséghalmazban a törölt csomópont minden előfordulását a megmaradóra változtatjuk, az esetleges halmazelem-kettőződéseket pedig töröljük.
8. Az algoritmus minden lépése után *ütközésvizsgálatot* végzünk. Egy x facsomópontot ütközőnek nevezünk, ha

[40] Szeredi P.-Lukácsy G.-Benkő T.: A szemantikus világháló elmélete és gyakorlata [SzLB05] 231. old.

- a hozzá vezető úton valamely csomópontnál megtaláljuk a `false` állításkonstanst, illetve az annak megfelelő ⊥ fenékjelet.
- A hozzá vezető úton egy állításelem és a negáltja is megtalálható.
- Valamely x csomóponthoz (≤nR) jellegű korlát van rendelve, viszont x-ből legalább n+1 darab nem összevonható kiinduló él található.

9. A fenti szabályokat addig alkalmazzuk, amíg van alkalmazható szabály. Ha nincs ilyen, akkor a tabló-állapotot *teljesnek* nevezzük, és a szabályalkalmazás leáll. A szabályalkalmazás leállhat még akkor is, ha a fában egyetlenegy ütközésmentes ág sincsen.
10. Ha a fenti szabályalkalmazások végén, a nemdeterminisztikus választások legalább egyik ágán egy teljes, de ütközésmentes ág található, akkor a *kezdeti fogalom-kifejezés kielégíthető*, és az ütközésmentes ág a kielégíthetőség feltételeit is rögzíti.

A fentiekben általánosan bemutatott tabló algoritmus működését egy igen-igen egyszerű büntetőjogi példán ábrázoljuk.[41] A példa megértéséhez két alapfogalmat kell ismernünk:

`Károsult`	A fogalom-kifejezés a bármilyen okból, pl. tolvajlás vagy más káresemény útján megkárosultakat takarja. Egyelőre nyitva hagyjuk azt, hogy vajon csak élő, tényleges, természetes személyeket gondolunk ide, vagy egyéb szervezeteket, közösségeket is. Megállapodás szerint a fogalom-kifejezéseket nagy kezdőbetűvel írjuk.
`lop`	a szerepkifejezés a lopásban résztvevő aktív személyeket köti össze azokkal, akiktől a lopás történt (valaki valakitől lop). Megállapodás szerint a szerepkifejezéseket kis kezdőbetűvel írjuk.

A példa futása az alábbi fogalommegadásokkal dolgozik:

`X = IT ∧ T ∧ NS ∧ NK`	van-e olyan, aki ismétlő tolvaj, de nem sorozatosan ismétel, aki vagy maga is károsult, vagy nem károsulttól lopott?
`IT = (>=2lop)`	Ismétlő Tolvaj, aki legalább kétszer lopott, függetlenül attól, hogy Károsultat lopott-e, vagy sem
`T = ∃lop.Károsult`	Tolvaj: aki legalább egy Károsultat meglopott
`NS = (<=2lop)`	Nem Sorozatosan ismétlő tolvaj, aki legfeljebb kétszer lopott, de a célszemélyről nem tudunk semmit sem
`NK = NT ∨ K`	Károsultakat Nem meglopók (Nem Tolvajok) és Károsultak egyesített halmaza
`NT = ∀lop.~Károsult`	akik loptak ugyan, de egyetlen Károsultat sem loptak meg (hanem valaki/valami mást, pl. csókot egy szép hölgytől), vagyis akik Nem Tolvajok
`K = Károsult`	károsultak halmaza

[41] Szeredi P.-Lukácsy G.-Benkő T.: A szemantikus világháló elmélete és gyakorlata [SzLB05] alapján 249. old

A bizonyítási eljárás célja annak eldöntése, hogy az X fogalom kielégíthető-e, azaz – a példányadatoktól függetlenül, csupán az elmélet logikai összefüggései alapján – létezhet-e olyasvalaki, aki ismétlő tolvaj, de nem sorozatosan ismétel, aki vagy maga is károsult, vagy nem károsulttól lopott. A bizonyítás illusztrációjánál a fogalom-meghatározások pontos szemantikáját nem tárgyaljuk, hanem feltételezzük, hogy azok makrószerűen kifejtődnek – mindegyik akkor, amikor már nem halogatható tovább, vagyis akkor, amikor egy egyszerű fogalomnevet tartunk a kezünkben, egy olyat, amelyre létezik meghatározás. Miután viszont kifejtettük a makrókat, a táblázatban a magyarázat tömörsége végett ismét a rövidítéseiket használjuk.

Megjegyezzük, hogy a nemdeterminisztikus választásokat mélységi kereséses (visszalépéses) módon kezeljük, és az egyes alternatívákat az előfordulási sorrendjükben vesszük. A táblázatban szürke festéssel jelöljük a nemdeterminisztikus szabályok változatait.

Lépés	Csomópont	Címkehalmaz	Szabályalkalmazás Magyarázat
0	a	{X=IT∧T∧NS∧NK}	ÉS szabály
1	a	{X,IT,T∧NS∧NK}	ÉS szabály
2	a	{X,IT,T,NS∧NK}	ÉS szabály
3	a	{X,>=2lop,T,NS,NK}	>= szabály Új üres b és c csomópontok
4	a	{X,IT,∃lop.Károsult,NS,NK}	∃ szabály Új d {Károsult} csomópont
5.1	a	{X,IT,T,<=2lop,NK}	<= szabály 1. változat b egyesítése d-vel.
5.1.1	a	{X,IT,T,NS,NT∨K}	∨ szabály, 1. változat
5.1.2	a	{X,IT,T,NS,∀lop.~Károsult}	∀ szabály ~Károsulttal bővítjük b és c címkehalmazát
5.1.3	b	{Károsult,~Károsult}	Ütközés, visszalépés
5.1.4.	a	{X,IT,T,NS,NT∨K}	∨ szabály, 2. változat
5.1.5	a	{X,IT,T,NS,Károsult}	Nincs további lépésre lehetőség, visszalépés
5.2	a	{X,IT,T,<=2lop,NK}	<= szabály, 2. változat, c egyesítése d-vel
5.2.1	a	{X,IT,T,NS,NT∨K}	∨ szabály, 1. változat
5.2.2	a	{X,IT,T,NS,∀lop.~Károsult}	∀ szabály ~Károsulttal bővítjük b és c címkehalmazát
5.2.3	c	{Károsult,~Károsult}	Ütközés, visszalépés
5.2.4	a	{X,IT,T,NS,NT∨K}	∨ szabály, 2. változat
5.2.5	a	{X,IT,T,NS,Károsult}	Nincs további lépésre lehetőség,

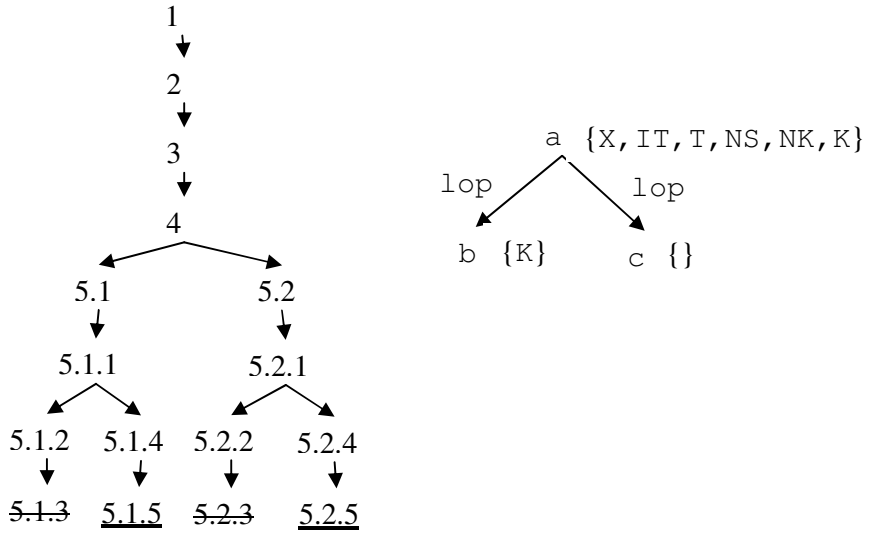
A fenti táblázatos algoritmus-leírásban nem tüntettük fel a csomópontokhoz tartozó egyenlőtlenséget, amely mindenütt b<>c.

A futás egyik eredménye egy olyan elképzelt egyedmegadás (A-box), amely kielégíti a terminológia megkötéseit. Az adategyedek a gráf csomópontok nevei lesznek, a gráf éleihez rendelt címkék pedig a közöttük fennálló szerepkapcsolatokat rögzítik. A csomópontokhoz rendelt fogalomkifejezés-halmaz azokat a kifejezéseket adja meg, amelyek az adott egyedre teljesülnek. Ha a futás több nemdeterminisztikus eredményt is szolgáltat, akkor mindegyik eredmény a kiinduló osztálykifejezés egy-egy külön, párhuzamos modelljét hozza létre.

Fontos megkülönböztetni a probléma faszerkezetű *keresési terét*, az algoritmus által felépített *adatmegkötési* vagy *példánygráfjától*. A keresési tér az algoritmus állapotai

között jön létre, lényegében a programlépések fája, úgy, hogy a nemdeterminisztikus választások képezik az elágazásokat. Az adatmegkötések viszont tetszőleges gráfot kiadhatnak, és a program a nemdeterminisztikus eredmények miatt többféle ilyen eredményt is létrehozhat.

Alább bemutatjuk a fenti példa keresési terét és példánygráfját. A keresési teret a példaprogram állapotainak azonosítóval címkéztük. A levélelemek között aláhúzással jelöltük a teljes konzisztenseket és áthúzással az ütközéseseket. Az első teljesnek levélelemnek megfelelő példánygráfot jobbra mellette láthatjuk. A második példánygráf az elsőnek `b`-re és `c`-re nézve szimmetrikus tükörképe lenne.



2. ábra Keresési tér és példánygráf

2.3.3.3 Következtetés terminológia axiómákon

Az eddigiekben ismertetett algoritmus csak az ***ALCN*** nyelvnek megfelelő összetett osztálykifejezésekre működött. Mint a példa leírása során említettük is: az egyenlőségeket csupán rövidítésként tekintettük, amit tetszés és igény szerint helyettesíthettünk be, vagy vissza. Ebben a szakaszban megmutatjuk, hogy az alapalgoritmust hogyan szükséges bővíteni ahhoz, hogy tényleges terminológiai axiómahalmazra is működtethető legyen.[42] A tényleges terminológiai axiómahasználat és a fenti megoldás között az alapvető különbségek a következőkben foglalhatók össze:

1. Terminológiai axiómákban a „=” és a „⊆” (halmazegyenlőség, illetve részhalmaza) jeleket is használhatjuk. Ezek a tényleges logikai egyenértékűségnek és implikációnak, illetve a meghatározó és tartalmazó axiómáknak felelnek meg.
2. A terminológiai axiómák tetszőlegesen tartalmazhatnak ciklikusságot, rekurziót, azaz az axiómák bal és jobboldalán tetszőlegesen, közvetlenül vagy közvetve hivatkozhatnak ugyanarra a fogalomra is (`Magyar=∀szülője.Magyar` …azaz magyar az, akinek mindegyik szülője magyar).

[42] Nagy Zsolt: Leíró logikán alapuló tudáskezelő rendszer [Nagy04] 31. old.

A terminológiai axiómákon működő következtetés alapfeladata szerint azt mutatjuk meg, hogy egy X kifejezés egy adott T háttérelmélet, azaz terminológiai axiómakészlet (T-box) felett is kielégíthető. Ez akkor teljesül, ha az X kifejezés nemcsak önmagában, hanem a T háttérelmélettel együtt is kielégíthető. A tabló algoritmusra vonatkoztatva ehhez elegendő az $X \wedge T$ konjunkció kielégíthetőségét megmutatnunk, amennyiben a T háttérelmélet önmagában felírható egyetlen logikai kifejezésként.

A T elmélet egyenértékű logikai kifejezésként történő átalakítását a *belsősítés (internalization)* műveletével végezhetjük a következő lépések szerint:

1. Első lépésben bővítsük ki a meghatározó axiómákat két tartalmazási axiómával, azaz minden $A = B$ axiómából csináljunk egy $A \subseteq B$, $B \subseteq A$ axiómapárt.
2. A második lépésben minden $A \subseteq B$ axiómából hozzunk létre egy $\sim A \vee B$ logikai kifejezést.
3. Végül vegyük minden axiómából létrehozott logikai kifejezés konjunkcióját. Így az eredeti elméletet kiindulási pontként véve létrehozhatunk egy újabb, $C_T = (\sim A_1 \vee B_1) \wedge (\sim A_2 \vee B_2) \wedge \ldots (\sim A_n \vee B_n)$ fogalom-meghatározást. Ennek a jobboldala pontosan az eredeti T elmélet egyenértékű alakja, egyetlen kifejezésbe sűrítve.
4. A példánygráfot bővítő lépések esetében – a T elmélet feletti meghatározás miatt – az új csomópontok mindegyikének ki kell elégíteni az elméletet is. Ezért az új csomópontok minden esetben öröklik az T elméletet is, illetve annak belsősített C_T formáját.
5. A konjunkcióval összekapcsolt kifejezéseket tekinthetjük egyidejűleg igaznak és kielégíthetőnek gondolt kifejezéshalmazként is.

Mindezek elegendőek egy rekurzió-mentes terminológia feletti tökéletes következtetéshez. Amennyiben azonban az elméletünk rekurziót is tartalmaz, az alapalgoritmus végtelen ciklusba eshet. A rekurzió átalakításos megközelítéssel általánosságban nem kiküszöbölhető, ehhez az alapalgoritmust kell kismértékben módosítanunk.

Az új működést *blokkolásnak* vagy *megállításnak* nevezzük. A megállítás bizonyos *megállító feltételek* jelzését, és a feltételek beállta esetén bizonyos *szabályfajták alkalmazásának a tiltását* jelenti. Az általános módszer a logikai nyelv egyes további bővítései esetén is alkalmazható – ilyenkor természetesen a megállító feltételek, és a tiltó szabályok mások. A következőkben a megállítás műveletét az ***ALCN*** nyelvre vonatkoztatva mutatjuk be.

A megállítás műveletét mindig a létrehozott keresési fa egy csomópontjára vonatkoztatjuk. A megállítás műveletéhez – és így a teljes ***ALCN*** bizonyító algoritmushoz –a nemdeterminizmus tetszőleges kezelésének elve már nem tartható tovább. A művelet megvalósítását ugyanis sokkal egyszerűbbé teszi az, ha a csomópontokra a kiterjesztő szabályokat csak legvégül alkalmazzuk, miután a többi, „közönséges” szabályokat már mindet alkalmaztuk.

Nevezzük a keresési fa egy csomópontját *stabilnak* akkor, ha az alatta (gyökér felé) létrehozott csomópontokra már nincsen egyetlen alkalmazható szabály sem, és saját magára vonatkozólag már csak kiterjesztő szabályok ($\exists$ és >= szabályok) várnak alkalmazásra.

A megállító feltétel nyilvánvalóan a kiterjesztő szabályokra vonatkozik – hiszen éppen attól félünk, hogy az algoritmus végtelen ciklusban generálja a fa újabb és újabb, de a megelőzőhöz képest változatlan ágait és rétegeit. Az ***ALCN*** megoldó algoritmusban úgynevezett *részhalmazos megállítást (subset-blocking)* alkalmazunk. Eszerint kiterjesztő szabályok alkalmazását egy újonnan létrehozott csomópontra vonatkozólag akkor tiltjuk meg, ha az új csomóponthoz rendelt fogalomhalmaz részhalmaza valamelyik ősének. Nyilvánvaló, hogy ilyen esetben az új, utód csomópontból kiinduló részfa részfája lesz az ősből kiinduló részfának, vagyis ilyenkor az új csomópont és az ős közötti faszakasz végtelen ismétlődésére számíthatunk.

A teljes ***ALCN*** megoldó algoritmusban tehát a ∃ és >= szabályok megvalósításához két újabb dolgot kell hozzávenni:

- Az alkalmazási feltételeihez hozzá kell vennünk az „aktuális x csomópont stabil és nincs megállítva” feltételt is.
- Az új csomópontok minden esetben öröklik az eredeti `T` elmélet belsősített, C_T alakját is.
- A szabályalkalmazás végén minden újonnan generált csomópontra megállásvizsgálatot kell végeznünk: ha az új csomópont kifejezéshalmaza részhalmaza valamelyik ősének, akkor a csomópontot megállítjuk.

Lássunk egy példát az elmondottak alkalmazására![43] Tegyük fel, hogy a tranzitivitást nem ismerő rendszerünkben a tranzitivitást az alapreláció rekurzív megadásával kívánjuk leírni. Álljon a `T` elmélet egyetlen axiómából, amely szerint (eltekintve a különféle birtokolható értéktárgyaktól) egy közös tulajdon tulajdonosának minden társtulajdonosa is közös tulajdonos, de csak akkor, ha egyáltalán létezik társtulajdonos (ez különbözteti meg az egyedi tulajdonlástól).

`T = KözösTulaj ⊆ ∀társ.KözösTulaj ∧ ∃társ`

Vizsgáljuk most meg a `KözösTulaj` fogalmat, hogy kielégíthető-e a fenti `T` elmélet szerint. Bemutatjuk, csupán gondolati úton, hogy megállítás nélkül a bizonyítás végtelen ciklusba fordulna, míg a megállítás műveletét alkalmazva a végtelen ciklus felismerhető, és a végtelen ciklust okozó szabályalkalmazás letiltható, így a bizonyítás végigvihető.

Ha a tabló algoritmus alkalmazásával feltesszük, hogy létezik egy `KözösTulaj(a)` egyed (gráf csomópont), akkor a ∃ szabály alkalmazása miatt léteznie kell egy `KözösTulaj(b)` társtulajdonosának is, akire viszont az eredeti T elmélet az öröklődés miatt ismét teljesül. Erre a b társtulajdonosra ismételten alkalmazhatnánk a ∃ szabályt, ami a megállítás művelet nélkül így egy végtelen objektumláncolatot hozna létre.

A megállítás műveletét a ∃ szabály `b` egyedre történő alkalmazásakor hasznosíthatjuk. A `b` egyedre ugyanis nyilvánvalóan ugyanazok a fogalmak érvényesek (a csomópont a létrehozáskor ugyanazokat örökli), mint a szülő `a` egyedre. Az új egyedre vonatkozó fogalmak tehát részhalmazai a szülő egyedének, tehát teljesül a ∃ szabály megállítási feltétele. A megállítás nem változtatja meg a teljes fa kielégíthetőségét, hiszen az új

[43] Szeredi P.-Lukácsy G.-Benkő T.: A szemantikus világháló elmélete és gyakorlata [SzLB05] 259. old.

részfák ugyanolyan feltételek mellett tartalmaznak (vagy nem tartalmaznak) ütközést, mint a szülő csomópont által létrehozott részfa.

2.3.3.4 Következtetés adataxiómákon

Az eddigiekben azt mutattuk be, hogy a csupán terminológiai axiómákon hogyan lehet a tabló algoritmus segítségével különféle következtetési műveleteket megvalósítani. Most azt mutatjuk be, hogy adataxiómák hozzávételével ez hogyan történhet.[44] Most is csupán a kielégíthetőség-vizsgálat bemutatására szorítkozunk: a többi, elképzelhető adat-következtetési feladat vagy a kielégíthetőség vizsgálatra visszavezethető, vagy pedig ennél lényegesen magától értetődőbb algoritmussal megvalósítható.

Az adatdobozok és terminológiai axiómák feletti együttes kielégíthetőség-vizsgálat alapvetően három fázisból áll:

1. Az adatdobozokból egy *kezdeti tablót* építünk fel a következők szerint:
    a. A kezdeti tabló csomópontjai az egyednevek lesznek, az élei pedig az egyedeket összekötő szerepnevek.
    b. A kezdeti tabló csomópontjaihoz hozzárendeljük az elmélet belsősítéséből származó C_T fogalmat, valamint az összes olyan C fogalmat, amelyhez tartozónak egy megfelelő, $C(x)$ formájú adat-axióma állította.
    c. Ha szükséges, az egyenlőtlenség-rendszerhez hozzávesszük az összes egyednévből alkotott párt.
2. A kezdeti tablóból kiindulóan *futtatjuk* az előző szakaszban bemutatott tabló algoritmust a következő megszorításokkal:
    a. Csak kiinduló tabló csomópontjaira alkalmazhatunk kiterjesztő szabályt, az azokból létrehozott újabb csomópontokra már nem.
    b. Ha két csomópontot összevonunk, és az egyik a kiinduló tablóhoz, azaz az eredeti adatdobozhoz tartozott, akkor az eredeti csomóponthoz vonjuk az újonnan létrehozottat, semmiképp sem fordítva.
    c. A két eredeti csomópontot vonunk össze, akkor az összevont csomópontban nem csak két eredeti csomópontból kiinduló, de az oda befutó éleket is egyesítenünk kell.
3. A futtatási fázis most is nemdeterminisztikus, amely ezért létrehozhat eredményváltozatokat is. A kielégíthetőség-vizsgálat során mindegyik eredményt meg kell vizsgálnunk a következőképpen:
    a. Az algoritmus futtatása során az egyes csomópontokhoz hozzárendelt fogalom-kifejezések halmaza bővülhet. A tabló minden egyes eredeti adatcsomópontjához elkészítjük a hozzárendelt fogalom-kifejezések metszetét. Ezek száma nyilván véges, hiszen a csomópontokhoz rendelt egyedek számossága is véges.
    b. Az algoritmus lefuttatásával minden egyes ilyen fogalomkifejezés-metszetre eldöntjük, hogy kielégíthető-e. Amennyiben egy adott eredménytabló mindegyik csomópontja kielégíthető, akkor a tabló által

[44] Szeredi P.-Lukácsy G.-Benkő T.: A szemantikus világháló elmélete és gyakorlata [SzLB05] 264. old.

létrehozott új fogalom-példány megkötések az eredeti adatdobozzal együttesen kielégítik a háttérelméletet.

c. Ha egy háttérelmélet nem kielégíthető, akkor a többi, alternatív tabló kielégíthetőségét is megvizsgáljuk. Minden egyes kielégíthető tabló az eredeti adatdoboz és a háttérelmélet egy megoldását adja. Ha egyetlen kielégíthető eredménytabló sincsen, akkor az adatdoboz a háttérelmélettel nem kielégíthető.

Az előző szakaszokban bemutattuk, hogy hogyan lehet a tabló algoritmus alkalmazásával az ***ALCN*** adat- és terminológiai axiómáin következtetési műveleteket végezni, és bemutattuk a tabló-algoritmust ebben az alapkiépítésben. Megállapítottuk, hogy az egyre bonyolódó leíró logikai nyelvekhez léteznek a tabló algoritmus egyre bonyolódó (és egyre erőforrás-igényesebb) kiterjesztései. Ez egészen a ***SHOIN*** nyelvosztályig igaz, mert az OWL-Full-nak megfelelő ***SROIQ*** nyelvosztály már nem is eldönthető. Mindezek részletes tárgyalására nem térünk ki, mint ahogy az egyes algoritmus-fokozatok után szokásos matematikai precizitású helyesség és teljességbizonyításra sem.

2.3.3.5 Az SWRL szabályleíró résznyelv

Az SWRL (ejtsd: swirl, azaz örvény) nyelv a korábbi RuleML és az OWL nyelvek egyfajta metszete,[45] illetve az OWL-nak kiterjesztése és kiegészítése.

A nyelv a lényegét tekintve egy függvénymentes Horn logikán alapuló szabálynyelv (a Dataloghoz hasonló). Amennyiben az OWL nyelvvel való szigorú metszetet tekintjük, akkor csupán 1 és 2 argumentumú állításelemeket használhatunk (fogalmak, illetve szerepek). A nyelvet könnyű tetszőleges argumentumszámú predikátumok elfogadására is alkalmassá tenni: ez a nyelv erejét nem módosítja, és nem emeli. Másrészt az SWRL bizonyos –nem túl szoros – korlátok mellett a teljes elsőrendű logika (***FOL***) irányába is kiterjeszthető.[46]

Maga a tiszta SWRL a DL $\mathcal{SHOIN}^{(\mathcal{D})}$ nyelv elemeit tartalmazza, és a Dataloggal való metszetét tekintve eldönthetetlen. A nyelv alkalmazásának a teljesség matematikai fogalmát figyelmen kívül hagyó, erősen gyakorlati szempontjai voltak: részben az OWL egyes hiányosságait akarták pótolni, részben pedig egy eljárásközpontúan is könnyen értelmezhető kiskaput próbáltak az OWL nehezen átlátható tételbizonyító algoritmusai mellett kinyitni. Egyes szerzők[47] dolgozatukban részletesen tárgyalva az SWRL nyelv Dataloggal való kapcsolatát, meg is rázzák – ha nem is a vészharangot, de legalábbis a vészcsengettyűt a nyelv kicsit pongyola tervezése felett. Az SWRL talán egyik legfontosabb kínálkozó lehetősége, a felnyithatósága és kiterjeszthetősége: talán az egész OWL-SWRL infrastruktúrában az SWRL az a pont, ahol a legkönnyebben lehet bővítéseket eszközölni.

---

[45] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, Mike Dean: SWRL: A Semantic Web Rule Language Combining OWL and RuleML [SWRL04]

[46] Peter F. Patel-Schneider: A Proposal for a SWRL Extension towards First-Order Logic [PS04]

[47] Bijan Parsia, Evren Sirin, Bernardo Cuenca Grau, Edna Ruckhaus, Daniel Hewlett: Cautiously approaching SWRL [Pa05]

Az SWRL által is használt Horn logikák legismertebb alkalmazása a később részletesen ismertetett Prolog nyelv. A Prologhoz képest az SWRL a következő kiterjesztéseket és módosításokat tartalmazza:

- A nyelv *negációt* is tartalmaz, amelyet – a megvalósítási javaslat szerint – nem a Prolog kudarckezelő (nem teljes) megoldásával, hanem egy teljes és konkrét negációkezelő részalgoritmussal valósítanak meg.
- A Prolog kvantálatlan változókat tartalmaz, melyeket a logikai háttér miatt univerzálisan kvantáltnak tekinthetünk (az egzisztenciális értelmezés csupán a megvalósító algoritmus nemdeterminisztikussága, valamint a nem logikai eszközök, főleg a vágó („!") alkalmazása miatt lehetséges). Az SWRL – az OWL-ból történő kiindulás miatt - mindkét fajta, az *egzisztenciális kvantálás* alkalmazását határozottan lehetővé teszi.
- Az SWRL, még a kiterjesztett értelmezés szerint *sem használ függvényeket* (összetett kifejezéseket), így efféle adatszerkezeteket létrehozó, esetleg klasszikusnak tekinthető algoritmusokat (pl. listák, fák, gráfok kezelését) a nyelv segítségével egyáltalán nem lehet készíteni.
- A nyelvet elsősorban *előrehaladó szabályértelmezőkkel* valósítják meg, de ez nincsen megváltoztathatatlanul rögzítve. A Prolog programnyelv visszafelé haladó kiértékelő algoritmust alkalmaz, bár éppen a logikai programozásról szóló fejezetben rámutatunk majd egy Prolog nyelvtanú előrehaladó következtető rendszer lehetőségére.

Az SWRL-hez (éppúgy, mint az OWL-hoz magához) számos végrehajtó és következtető csomag létezik, de ezek a csomagok is sok esetben nem támogatják a teljes nyelvet, vagy esetleg valamilyen rögzített, de nem teljes stratégiát valósítanak meg. A népszerű Protégé ontológiaszerkesztő rendszer 4.x változatában a Clark & Parsia amerikai vállalkozás Pellet nevű, szabadon használhatóg következtető gépét[48] használják SWRL értelmezésére, de más is bedugaszolható, és a korábbi változatok a Pelletet magát is csak dugaszként[49] ismerték.

2.4 Modális logikák

A klasszikus logika keretei, főleg a szigorú logikai kétértékűség már nagyon régóta okozott fejfájást a logikát, mint tudományágat művelő tudósoknak. Az egyik – bár időrendben későbbi – kitörési pont a *homályos (fuzzy) logikák* világa, amely végleg búcsút mond a kétértékűségnek; (pl. valamilyen berendezés állapota lehet Igaz/Hamis/Üzemen kívül/Ismeretlen). Amennyiben a logikai értékek számát csak valamilyen egész számig bővítjük, akkor kapjuk a *diszkrét fuzzy* logikákat. Ha akár valós intervallumot is megengedünk értéktartományul, akkor pedig a *folytonos fuzzy* logikákat kapjuk. Mindkét részterület igen érdekes, önállóan is kutatható, hatalmas előzménytárral, irodalommal, tapasztalatokkal rendelkezik, de a jelen értekezésnek egyik sem tárgya.

A *modális logikák* a klasszikus igaz/hamis értéktartományt nem közvetlenül bővítik, hanem az úgynevezett *logikai operátoraik* révén.[50] Ezek a mondatban alkalmazott

[48] A Pellet következtetőgép honlapja: http://clarkparsia.com/pellet/, elérés: 31-Jan-2013

[49] Dugaszfelépítmény (plug-in architecture) egy alapszoftverhez dinamikusan, akár futásidőben is becsatolható bővítményeket jelent.

[50] Ruzsa Imre: Klasszikus, modális és intenzionális logika [Ru84]

módhatározószók jelentésével kapcsolatosak (*Szükségszerű, Lehetséges, Tudott, Hitt, Kötelező, Tilos*), amelyek segítségével olyan mondatok is logikailag leírhatókká és értelmezhetőkké válnak, mint pl.:

- Koppány győz**het**ett volna.
- **Szerintem** holnap havazik.
- A madarak **szükségszerűen** gerincesek.
- **Tilos** az átjárás

Az egészre másik példát felhozva: modális operátorok azok a szavak, esetleg rövid kifejezések, amelyeket a „Juliska ____ boldog.” mondatban módhatározóként beszúrhatunk vagy értelemszerűen behelyettesíthetünk. Melyek ezek? Például a (szükségszerűen, esetleg, Jancsi szerint, azt hiszi, hogy, muszáj, hogy ~ legyen, engedély szerint, most, később ~ lesz) halmaz valamelyike.

2.4.1 Intenzionális és extenzionális logika

A modális logika kialakulása nem az ókori görögökkel kezdődött. Arisztotelész kategorikus szillogizmusaiban semmi modalitás nem fedezhető fel. Arisztotelész mintájára modális szillogizmusok először a középkorban születtek, és ugyanez idő tájt fedezték fel egyes mondatok *de re* és *de dicto* olvasata közötti különbséget is.

A de re és de dicto értelmezések a mondatok extenziójával és intenziójával is összefüggésbe hozhatók. Tegyük fel, hogy igaz a „Péter várja a háziorvost.” állítás. Mindamellett tudjuk, hogy a háziorvost Gazsinak hívják, és ő a cigányzenekar brácsása is. Következik-e mindebből, hogy a „Péter várja Gazsit, a brácsást” állítás is igaz lesz? Nyilvánvalóan nem, még akkor sem, ha Péter egyébként tudna Gazsi zenészi időtöltéséről: betegsége esetén csakis a Gáspár nevű háziorvost várhatja, vagy bárki mást, Gáspár helyettesét, vagy az utódját akkor, ha Gáspár Gazsi elfoglaltsága vagy bármilyen akadályoztatása miatt nem lenne elérhető. A példában a „háziorvos” kifejezés azonnali kiértékelésével kapjuk a mondat *de re* olvasatát. Ha a kiértékelést – nagyon helyesen – később végezzük, akkor a mondat *de dicto* olvasatát kapjuk. Az efféle kifejezéseket *intenzionálisnak* nevezzük: a beteg szempontjából mindegy, hogy ki az orvos, az meg pláne mindegy, hogy az orvos más környezetben másféle funkciókkal is bírhat. Az intenzionális kifejezéseket nem értékeljük ki azonnal, hiszen a kifejezésmód lényege nem az, hogy rámutathassunk a megfelelő személyre, objektumra vagy igazságértékre, hanem ehelyett a kifejezést eltároljuk, és a használat minden egyes pillanatában újra és újra kiértékeljük. A kifejezés egy konkrét aktuális kiértékelés szerinti értéke az extenziója, és az eszerint történő értelmezése a de re olvasat. A kifejezés, amit minduntalan kiértékelünk, maga a kifejezés intenziója, és az a de dicto olvasat.

Ehhez hasonló jelenség a „jövőre az elnök szocialista lesz” példamondatban érhető tetten. Ha az „elnök” kifejezést *mohó* módon, azonnal *kiértékelem* (eager evaluation, de re olvasat), akkor a mostani elnökről állítottam valamit, de a jövőre vonatkozólag – azt, hogy feltehetőleg pártot vált. Ezzel szemben, ha *lusta kiértékelést* (lazy evaluation, de dicto olvasat) alkalmazunk, akkor a mondat a jövő évben éppen hivatalban lévő elnökről szól, aki éppenséggel lehet szocialista is, abban az esetben például, ha közben történt egy választás, amit a szocialista párt nyer meg.

Az ezzel összefüggő iskolai példamondat a következőképpen hangzik: „Minden közlegény lehet tábornok.” Elemezgessük kicsit a mondatot!

- „Minden közlegény esetében lehetséges, hogy tábornok legyen.” a modális segédigét csak ott alkalmazva, ahol a mondatban is megjelenik, megkapjuk a mondat de re olvasatát
- „Lehetséges az, hogy minden közlegény egyben tábornok is.” a modális segédigét az egész mondatra kiterjesztve kapjuk a mondat de dicto olvasatát
- és esetünkben a legvalószínűbb olvasat a harmadik, ami már temporális logikai háttérrel értelmezhető csak helyesen: „minden mai közlegényből válhat később tábornok...”, de a lehetőséget csak kevesen ragadhatják meg, nyilván kellő tanulás, törekvés és egyebek után...

További példaként vegyük még a következő mondatokat:

„Az amerikai elnök nős.” a mondat nem rögzíti, hogy az aktuálisan kormányzó elnökről van-e szó, vagy a mondat általános érvényű lenne. A de re olvasat azonnal kiértékeli a mondatot, és az aktuális elnökre vonatkoztatja. A de dicto olvasat szerint a mondatot a mindenkori elolvasásának és értelmezésének időpillanatában kell kiértékelni, vagyis a mondat az elnökök nőtlenségéről általános érvényű (de nem feltétlenül igaz) állítást fogalmaz meg.

„A református pap nős.” az előzőhöz hasonló mondat nem rögzíti, hogy melyik református papról is van szó. Mégis úgy érezzük, a de re olvasat szerinti, a helyileg megtalálható papról szólhat inkább az állítás, de ezt a félreértések elkerülése végett mindenképpen pontosítani kell.

A múlt század elején, David Lewis volt az, aki munkáiban a modális operátorok fogalmát bevezette, de ezeket akkor még csak ítéletkalkulusbeli formulákra alkalmazta. Az eredmények körét azonban ítéletkalkulusból elsőrendűvé általánosítani nem sikerült gond nélkül: az egész számtalan paradoxonszerű állításkört eredményezett. Ezek feloldását végül is Saul Kripke (17 évesen!) adta meg, a modális szemantika rögzítésével.

A modális logikák sokféle modalitáskört és fogalmat megragadhatnak:

- a legegyszerűbb, *aletikus logikában* a lehetségességről és szükségszerűségről tehetünk állításokat (alétheia: görögül igazság).
- a megismerhetőség kérdéseit az *episztemikus logika* vizsgálja (episztemé: görögül tudás). Ennek speciális esete, amikor több szereplő által esetleg csak részlegesen tudott/állított elméletet vizsgálunk. Ennek különös jelentősége lehet jogi ítélethozatali problémák esetén a különböző jogi szereplők állításai közötti ellentmondás felfedezésében, és egy esetlegesen objektív állítás megkeresésében.
- a hit és meggyőződés kérdéskörét a *doxasztikus (doxatikus)* logika vizsgálja (doxa: görögül hit).
- valamilyen normarendszernek megfelelés vagy ellentmondás kérdéseivel a *deontikus* logika foglalkozik (deon: görögül kötelez).
- az egyes állítások időbeli érvényét vagy korlátozottságát a *temporális* logika képes leírni.

A modális operátorokat lehetséges csupán önmagukban, de lehet további paraméterektől függően is értelmezni (pl. az episztemikus logikák esetén meg kell adni azt is, hogy ki

az a szereplő, aki a tudás birtokában van). Ha a modális rendszerben kettőnél több modális operátort használunk, akkor *polimodális* logikáról beszélünk.

Az egyes modalitások egymástól függetlenül vizsgálhatók. Ha egy logikai rendszerben szükség van többféle modalitáskör megengedésére és alkalmazására is, akkor *multimodális* rendszerről beszélünk.

Jogi szemantikus rendszerek megépítéséhez feltétlenül szükség van a modalitás fogalmának, és gépi következtetési rendszerekben való alkalmazhatóságának módjait megismerni. Ebben a körben különösen fontos az alábbi modalitások kezelése, méghozzá multimodális megközelítésben:

- A normarendszernek való általános megfelelést vagy ütközést tárgyaló deontikus modalitás a jog-eset viszony alapkérdéseit tárgyalja.
- Az időbeli viszonyokat tárgyaló temporális modalitás kezelése azért elengedhetetlen, mert szinte minden jogi megállapításnak valamiféle időbeli hatálya van.
- Nem kevésbé fontos az egyes – egymásnak akár ellent is mondó – információforrások (pl. vallomások) által rögzített világszerkezetekben rendet vágó episztemikus modalitások kezelése.

A modális logika a fentinél lényegesen tágabb részterületekre bontható fel, amely nem lehet tárgya a jelen értekezésnek. A most következő fejezetben a fent említett modalitások bevezetését tárgyaljuk.

A fenti modalitások alapja, egyben a legegyszerűbb modalitáskör az aletikus modalitásé, ezért a modális logikák tárgyalását rendszerint ezzel érdemes kezdeni. Az aletikus modalitás két *alapoperátorral* dolgozik, nevezetesen:

- a □ operátort, az *erős modalitásnak* vagy *szükségszerűségnek* nevezzük. A □P állítást úgy olvassuk ki: „szükségszerű, hogy P”.
- a ◊ operátort, a *gyenge modalitásnak* vagy *lehetségességnek* nevezzük. A ◊P állítást úgy olvassuk ki: „lehetséges, hogy P”.

Az erős és gyenge modalitás fogalmát a legtöbb modális rendszer alkalmazza, sokan úgy, hogy pontosan a fent nevezett operátorszimbólumokat használják, de mindig az adott modalitás által rögzített értelmezésben. A különbséget az adott modalitáskör rögzítette/megkövetelte axiómakészlet és az axiómák alkalmazása teszi egyértelművé.

Kézenfekvő az alábbi axiómák elfogadása:

1. *Modális következtetés axiómája*. Ha egy következmény „tisztán” is igaz, akkor „erősen-modálisan”, azaz „szükségszerűen” is igaz lesz.

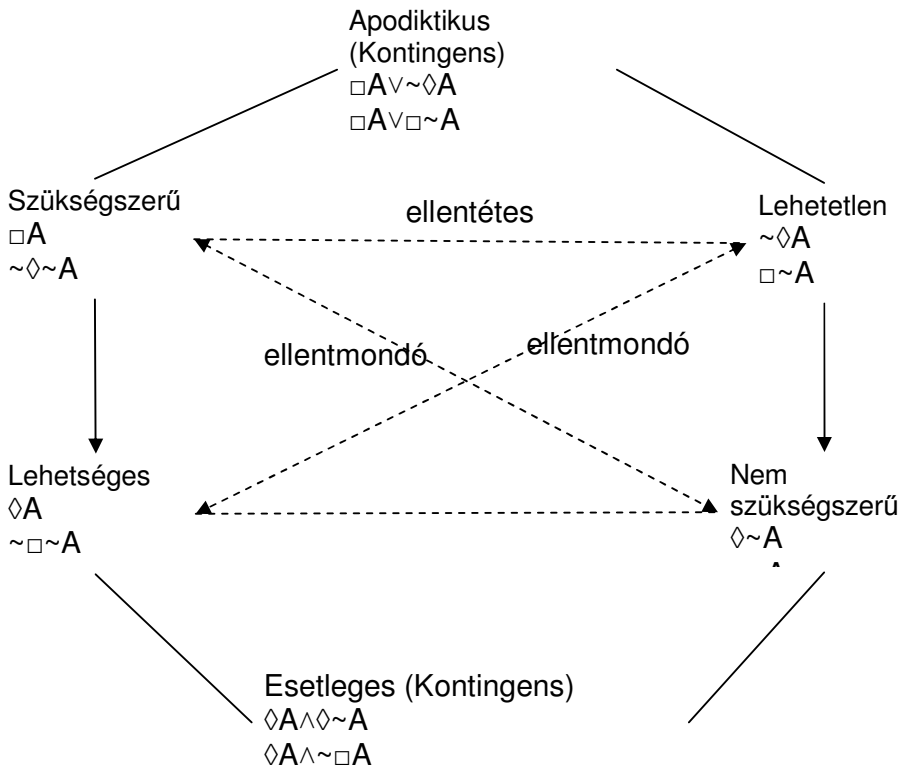
   $(A_1, A_2, \ldots A_n \rightarrow B) \rightarrow (\Box A_1, \Box A_2, \ldots \Box A_n \rightarrow \Box B)$

2. *Modális dualitás* elve:

   1. $\Diamond P \leftrightarrow \sim\Box\sim P$ vagyis valami akkor és csak akkor lehetséges, ha a negáltja nem szükségszerű, illetve:
   2. $\Box P \leftrightarrow \sim\Diamond\sim P$ vagyis valami akkor és csak akkor szükségszerű, ha a negáltja nem lehetetlen

Az aletikus modalitás alapösszefüggéseit jól ábrázolja az alábbi *aletikus modális négyszög*.[51] (Két ponttal, a legfelsővel és a legalsóval kiegészítve ez már hatszög, amit hagyományosan mégis négyszögnek neveznek.)

[51] Ruzsa Imre: Klasszikus, modális és intenzionális logika [Ru84]



3. ábra Aletikus modális négyszög

A négyszöget tanulmányozva a következő megállapítások tehetők:

1. Az átlók végein található Szükségszerű és Nem szükségszerű minősítések egymásnak ellentmondanak, csakúgy, mint a Lehetséges és Nem lehetséges minősítések. Ezt a viszonyt latin eredetű kifejezéssel *kontradiktóriusnak* is nevezzük.
2. A Szükségszerű minősítésnél gyengébb a Lehetséges minősítés, a Lehetetlennél gyengébb a Nem szükségszerű. Ezt a viszonyt *alárendeltségnek* is nevezzük: a fentiekből következik az alsó. Ha valami Szükségszerű, akkor az egyben Lehetséges is, ha viszont valami Lehetetlen, akkor Nem szükségszerű is.
3. A Szükségszerű és a Lehetetlen minősítések egymásnak ellentmondanak, hiszen egyszerre valami nem lehet szükségszerű és lehetetlen is, de lehet mindkettő hamis. Ezt a fogalmat *kontráriusnak* is nevezzük.
4. A Lehetséges és a Nem szükségszerű pár elemei egyszerre lehetnek igazak, de egyszerre hamisak nem. A viszonyt *szubkontráriusnak* is nevezzük.
5. A Szükségszerű és Lehetetlen minősítések diszjunkciójával kapjuk az *Apodiktikus* (kontingens) esetet, amely – ismerve a kiinduló állapotokat – inkább egyfajta kizáró VAGY jellegű helyzetet jelent, amikor nincs középút, amelyben mindkét kiinduló minősítés egyidejűleg nem lehet igaz.
6. A Lehetséges és a Nem szükségszerű minősítések konjunkcióba hozhatók: így kapjuk az *Esetleges* (kontingens) minősítést.

2.4.2 Kripke-féle modális szemantika

Bár a modális logika első használata szintén Arisztotelészhez köthető, és ezt később, a középkorban is számosan fejlesztették tovább, kerek elmélet csak a világháborúk után született. A kora modális logikai világban uralkodó bizonytalanságot egy csapásra szüntette meg Saul Aaron Kripke, aki még főiskolai hallgatóként írt cikkében a modális logikákhoz meghatározott feszes háttérszemantikájával sok korábbi terméketlen szakmai vita alól kihúzta a talajt.[52] A szemantikájának a lényege egyfajta *kvantifikáció a világok felett*:

- □P: a szükségszerűséget értelmezzük úgy, mint valamit, amely az *összes, közvetlenül elérhető világban igaz.*
- ◊P: a lehetségességet pedig értelmezzük úgy, mint valamit, amely *legalább egy lehetséges, közvetlenül elérhető világban igaz.*

Mindezek értelmezéséhez pedig meg kell adni a világok felett egyfajta *elérhetőségi/alternatíva relációt*. Az elméletben *Kripke-keretnek* (frame) vagy *világszerkezetnek* nevezünk egy $\mathcal{F}$=<W,R> párt, ahol:

- W: az értelmezhető világok (véges) halmaza.
- R: a világok feletti kétoldalú viszony. Ezt *elérhetőségi- / kompatibilitási- / alternatívarelációnak* is nevezzük. A viszony elemeit közbeékelődő (infix) operátoros jelöléssel is jelölhetjük. Vagyis $<w_1 \in W, w_2 \in W> \in R$ kifejezés helyett $w_1 R w_2$-t is írhatunk.

Az egyes világok jelentősége abban áll, hogy bár mindegyik világban ugyanazokat a logikai szerkezeteket használhatjuk, azok kiértékelése más és más lehet. Az elérhetőségi reláció pontos felállítására Kripke nem adott iránymutatást, az teljesen alkalmazásfüggő. Egy lehetséges javaslat: amennyiben a világok az egyes szereplők elmeállapotait jelzik, akkor az alternatívareláció jelezheti az adott szereplő elmeállapotával kompatibilis (nem ellentmondó) szereplők elmeállapotát, mint világot. A világok közötti elérhetőségi reláció a világok között egy gráfot feszít ki, amelyet a legcélszerűbben szintén grafikusan ábrázolhatunk.

Egy L^{m0} *nulladrendű modális logikai nyelvet* a következőképpen adhatunk meg:

$L^{m0} = \{\phi \mid \phi ::= p \mid \sim \phi_1 \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \rightarrow \phi_2 \mid \Box\phi \mid \Diamond\phi \}$

Vagyis a nyelv nem más, mint ϕ kifejezések halmaza, amelyek részkifejezésekből a {~ (negáció), $\wedge$ (konjunkció), $\vee$ (diszjunkció), $\rightarrow$ (implikáció)} operátorokkal építhetők fel. Külön említjük a részkifejezésekre alkalmazható modális operátorokat. A legegyszerűbb kifejezések a „p" atomi kifejezések. Ezek halmazát az L^{m0} nyelv teljes megadásához szintén pontosan meg kell adnunk. A fenti nyelvdefinícióban említett négy műveleti jel egyfelől bővebb, mint egy teljes műveleti rendszer, másrészt pedig alkalmasint más műveleteket is használni szeretnénk. Ennek semmi akadálya, a nyelvdefiníció természetesen tetszés szerint bővíthető.

Ha adott egy L^{m0} nulladrendű modális logikai nyelvünk, akkor $V=At \rightarrow 2^W$ függvényt a nyelv kiértékelésének nevezzük, ahol:

- At: a nyelv *atomi formuláinak* halmaza (esik, fúj, kék, szocialista, stb.)

---

[52] Saul Aaron Kripke: „A Completeness Theorem for Modal Logic" Journal of Symbolic Logic, 1959.

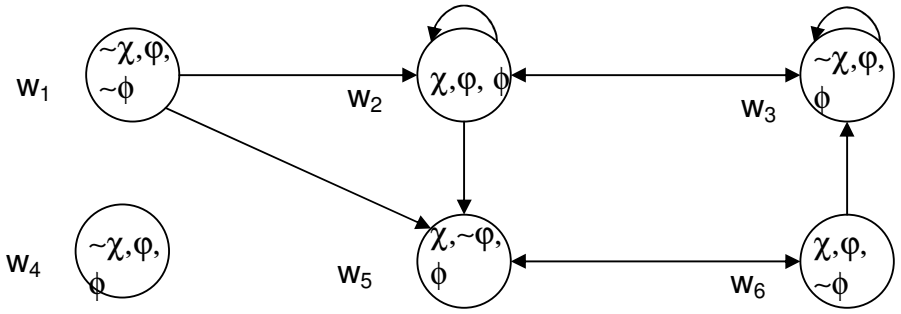
- A 2^W halmaz a világok hatványhalmaza: vagyis a függvény az egyes f∈ At atomi formulákhoz azokat a világokat adja meg, ahol az f atomi formula igaz.

Egy adott L^{m0} nulladrendű modális logikai nyelv *Kripke-modelljén* vagy *interpretációján* az M=<F,V> párt értjük. Vagyis meg kell hozzá adnunk:

- a világszerkezetet
- és azt, hogy az egyes formulák értéke hogyan alakul az egyes világok esetén

Ha egy A formula igaz az M modell w világában, azt így jelöljük: (M,w) |= A. Ha a modell nyilvánvaló, akkor azt el is hagyhatjuk: w |= A. Ha az ellenkezőjét akarjuk kifejezni, vagyis egy adott w világban az A formula hamis, azt a következőképpen jelöljük: w ~|= A.

Lássunk egy példát:[53]



4. ábra Egy elsőrendű modális nyelv Kripke-modellje

Tegyük fel, hogy logikai rendszerünk a fenti, $\{w_1,\ldots,w_6\}$ világokat tartalmazza, amelyeket a fenti ábra szerinti elérhetőségi viszonyok kapcsolnak össze. Logikai nyelvünkben a $\{\chi,\varphi, \sim\phi\}$ atomi állításokat használjuk, melyek kiértékelése az egyes világokban szintén az ábrán látható. Mindezek alapján a következő megállapítások tehetők:

- $w_1 \models \Box\chi, \Box\phi$ hiszen w_1-ből a w_2 és w_5 világok elérhetők, ezek mindegyikében igazak a χ és ϕ atomi állítások (még akkor is, ha saját világukban nem azok).
- $w_2 \models \Box\phi$ hiszen w_2-ből a w_2, a w_3 és a w_5 világok érhetők el, mindhármukban viszont csak a ϕ atomi állítás igaz.
- $w_3 \sim\models \Box\chi$ hiszen w_3-ból a w_2 és a w_3 világok érhetők el, de w_2-ben χ igaz, w_3-ban viszont hamis.
- Az *elszigetelt* w_4 világban: $w_4 \models \Box\chi$, még akkor is, ha $w_4 \models \sim\chi$. Az ilyen esetre azt mondjuk: *szükségszerű, de lehetetlen*.
- A fenti példa is bizonyítja a következő általános tételt: *Elszigetelt világban minden szükségszerű.*
- Ismételt modális operátorok vizsgálatához a több lépésben elérhető világokat kell vizsgálni. Pl.: $w_2 \models \Box\Box\phi$, hiszen a w_2-ből 2 lépésben elérhető világok: $\{w_2, w_3, w_5\}$, ezekben pedig ϕ igaz.

[53] Ruzsa Imre: Klasszikus, modális és intenzionális logika [Ru84] nyomán

A világszerkezet alternatívarelációjára vonatkozólag különféle metarelációs követelmények köthetők ki. Ezek egyenértékűek a logikai következtetésre vonatkozólag valamiféle újabb axiómával. Az egyes követelmények és axiómák modális logikai axiómarendszerek egy összefüggő halmazát adják. Az ezen meta-követelmények és az ezzel egyenértékű pótlólagos axiómák összefüggéseit vizsgáló modális logikai résztudományt *korrespondencia elméletnek* is nevezik.

2.4.3 Aletikus logikai rendszerek

2.4.3.1 A Kripke-féle „K" modális kalkulus.

A legegyszerűbb modális következtetési rendszer a Kripke-féle K modális kalkulus, amit a következő axiómákkal határozunk meg.

1. **K** axióma, amit *Kripke-féle axiómának* is neveznek.

   $\Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$

   Az axióma – sokban hasonlítva a korábban tárgyalt modális következtetés axiómájához – az állítja: ha egy következtetés szükségszerű, akkor az előfeltétel szükségszerűségéből is következik a következmény szükségszerűsége.

2. A Modális Általánosítás (Modal Generalization) **MG** axiómája:

   Ha A levezethető, akkor $\Box A$ is az.

   $A \rightarrow \Box A$.

   Az axióma szerint, ha egy állítás a világszerkezettől függetlenül levezethető, akkor az szükségszerűen levezethető. Másképp fogalmazva: a tiszta logikai levezethetőségnél még a szükségszerűség sem erősebb.

A K-kalkulus a Kripke által eredetileg megadott és használt rendszer, amely a lehető legegyszerűbb, és amely éppen ezért az összes többi rendszer kiindulópontja.

2.4.3.2 A „T" modális kalkulus.

A *T modális kalkulus* a K kalkulusból származtatható úgy, hogy feltételezzük az *alternatívareláció reflexivitását*. Vagyis eszerint minden világ elérhető sajátmagából.

Ezen kikötés az eddigieken túl egy újabb axióma felvételét indokolja; eszerint, ha *valami szükségszerű, akkor az úgy is van*.

T: $\Box A \rightarrow A$

Ez tulajdonképpen a **MG** axióma megfordítottja. Az axiómát szokás **T** axiómának is nevezni. A **T** axióma másik neve: *aletikus séma*, és szűkebb értelemben a T logikát, illetve további megszorításait nevezik csak aletikus modális logikának. Aletikus tehát egy modális logika akkor, ha tartalmazza az aletikus sémát, mint axiómát. Mivel a T logika további megszorításokat nem tartalmaz, egyben ez a logika az aletikus logikák leggyengébbike is.

2.4.3.3 A „K4" modális kalkulus.

A K4 modális kalkulus szintén a K kalkulusból származtatható úgy, hogy feltételezzük az alternatívareláció tranzitivitását. Ez azt fejezi ki, hogy ha valami minden elérhető világban igaz, akkor ezekben szükségszerűen is igaz lesz, vagyis minden tovább elérhető világban is igaz lesz.

Az ennek megfelelő új axióma:

tra: □A ➔ □□A

Az új axiómát szokás *tranzitivitási axiómának*, vagy ***tra*** *axiómának* is nevezni.

2.4.3.4 Az „S4” modális kalkulus.

A fenti két kalkulus, a T és a K4 kalkulusok összekombinálásával kapjuk az S4 modális kalkulust. Ebben tehát az alternatívareláció reflexív és tranzitív is, és a következtetési axiómákat úgy kapjuk, hogy a K rendszer axiómáihoz hozzávesszük mind a **T**, mind a **tra** axiómákat.

A rendszer egyik lehetséges értelmezési kerete episztemikus jellegű. Ha ugyanis az operátort most úgy értelmezzük, hogy „tudom, hogy…”, akkor a két axióma a következő kézenfekvő megállapításokat fejezi ki:

T: □A ➔ A ha valamit tudok, akkor az úgy is van

tra: □A ➔ □□A ha valamit tudok, akkor azt is tudom, hogy tudom

Már az alap modális rendszer is tartalmazza a modális dualitás szabályait. A csak modalitásokat és negációt tartalmazó kifejezéseket általánosságban *modalitásnak*, a csak ezeket tartalmazó axiómákat pedig modális redukciós szabályoknak nevezzük. Ezek az S4 rendszerben – már a **T** és **tra** axiómák következményeképpen számbelileg lényegesen felszaporodnak a következők szerint:

1. □A ➔➔ □□A
2. □A ➔➔ ◊□A
3. ◊A ➔➔ ◊◊A
4. ◊A ➔➔ □◊A

2.4.3.5 A „B” modális kalkulus

Ha a T rendszerbeli, csupán reflexív alternatívarelációról még azt is feltételezzük, hogy szimmetrikus, akkor a B rendszert kapjuk. A szimmetria eredményeképpen, ha egy formula valamely világban igaz, akkor minden közvetlenül elérhető világban létezik legalább még egy közvetlenül elérhető világ (ha más nem, akkor a kiindulási világunk), ahol ez szintén igaz lesz.

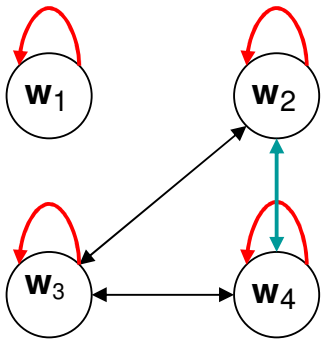
Axiómák szintjén ez azt jelenti, hogy a T rendszer **T** axiómájához még egyet, a **B** axiómát is felvesszük. Ez a következőképpen formalizálható:

B: A➔□◊A

2.4.3.6 Az „S5” modális kalkulus

Az S5 kalkulust a B és az S4 logikákból származtatjuk úgy, hogy mind a B rendszer reflexív és szimmetrikus alternatívarelációját, mind az S4 rendszer reflexív és tranzitív alternatívarelációját kikötjük, ezáltal az alternatívarelációt ekvivalencia relációvá tesszük. Ez a kikötés biztosítja, hogy ha egy formula valamely világban igaz, akkor minden (akár közvetve) elérhető világban létezik legalább még egy (akár közvetve) elérhető világ (a kiindulási világunk), ahol ez szintén igaz lesz.

Az S5 kalkulusban ezáltal nincsen új axióma, csupán a B és S4 logikák axiómáit benne együttesen alkalmazzuk.



5. ábra Példa az S5 ekvivalenciarelációt mintázó világszerkezetére

T: □A ➔ A

B: A➔□◊A

tra: □A ➔ □□A

Szokásos az S5 kalkulust közvetlenül a T-ből származtatni úgy, hogy annak axiómáit csupán egyetlen újabbal, az euklideszi axiómával egészítjük ki.

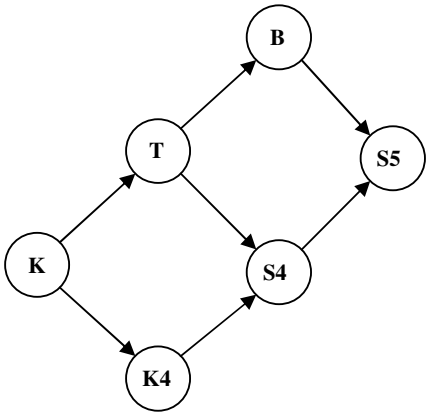
eukl: A➔□◊A

Az euklideszi axióma az alternatívareláció topológiájára nézve köti ki az euklideszi tulajdonságot. (Euklideszinek nevezünk egy R relációt, ha wRw_1 és wRw_2 fennállása esetén w_1Rw_2, vagy a fordítottja, w_2Rw_1 biztosan fennáll.)

A két származtatási mód egyenértékű, az ennek megfelelő tétel precíz bizonyításától most eltekintünk.

2.4.3.7 Aletikus modális logikák összefüggése

A fentebb tárgyalt modális logikák között a bennük alkalmazott axiómák halmaz-részhalmaz viszonyai a grafikusan is jól ábrázolható alábbi viszonyrendszert feszítik ki. Az ábrázolásban a nyilak vége mutat az erősebb, több axiómával megadott logikai rendszer felé. Ez egyben azt is jelenti: az erősebb rendszerben több formula levezethető, bebizonyítható.



6. ábra Aletikus modális logikák összefüggése

2.4.4 Deontikus (norma-) logika

A deontikus vagy normalogika alapelveit Ernst Mally osztrák filozófus fektette le 1926-ban megjelent „Die Grundgesetze des Sollens" művében. Az eredeti forrásanyaghoz hozzáférni nem könnyű, de modern feldolgozása és recenziója megjelent többek között a Stanford Egyetem „Stanford Encyclopedia of Philosophy" interneten is elérhető honlapján.[54] [55]

A deontikus logikában használhatjuk ugyanazokat a modális operátorokat, amelyeket az aletikus logikában megismertünk, de kicsit másféle értelmezésben:

- a □ operátort, az erős modalitást *kötelezettségnek* nevezzük. A □P állítást úgy olvassuk ki: „kötelező, hogy P". Más rendszerek – főleg multimodális környezetben – ezt az operátort az „obligatory" szóból képezve ***O*** P-vel jelölik.
- a ◊ operátort, a gyenge modalitást pedig *engedélyezettségnek* nevezzük. A ◊P állítást úgy olvassuk ki: „engedélyezve van, hogy P". Más, esetleg multimodális rendszerek a „permitted" angol szót rövidítve ezt a fogalmat ***P*** P-vel jelölik.

[54] Lokhorst, Gert-Jan: Mally's Deontic Logic [Lo08]

[55] McNamara, Paul: Deontic Logic [Na10]

– a deontikus modalitás használhatja a $\mathcal{F}$ tiltás-operátort is. $\mathcal{F}$ P a „forbidden” angol szóból rövidítve azt jelenti: P tilos. Ezt külön nem mindig említik meg, mert $\mathcal{F}$ a következő axiómával kifejezhető:

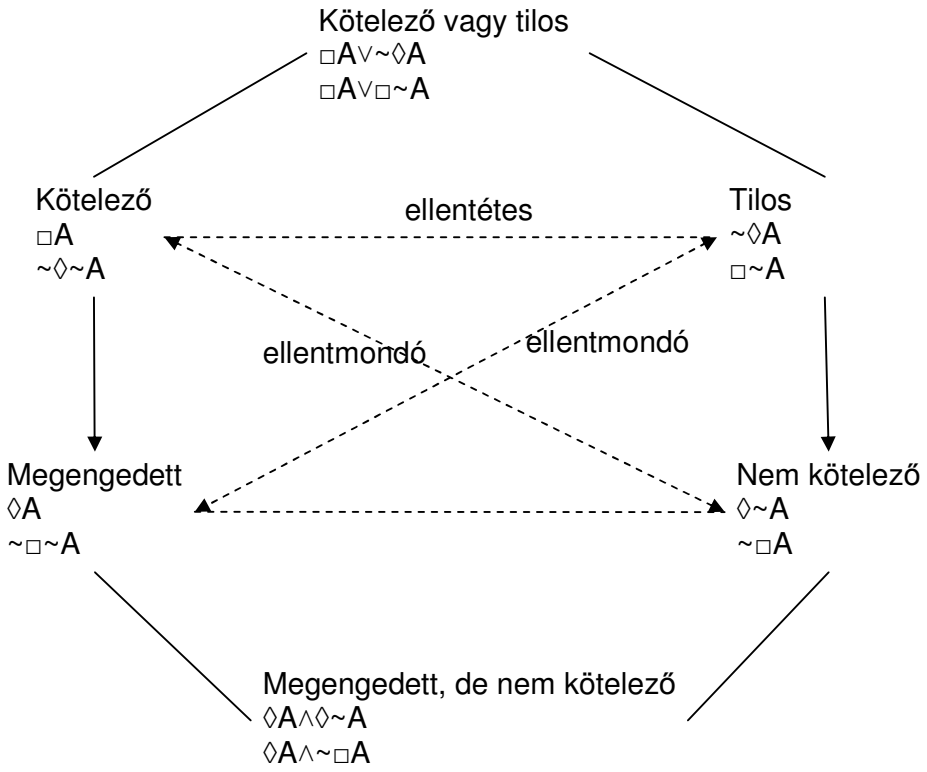
  $\mathcal{F}$ P = ~ $\mathcal{P}$ P, vagyis ami nincs engedélyezve, az tilos. Képezzük az axióma negáltját! (~$\mathcal{F}$ P = $\mathcal{P}$ P) Ez a liberális gondolkodók alaptételét fejezi ki: mindent szabad, ami nem tilos.

– Ritkán használják az $\mathcal{OM}$ operátort (omissible, elhagyható). Az elhagyhatóság a többiből származtatható: ($\mathcal{OM}$ P = ~$\mathcal{O}$ P), azaz, elhagyható az, ami nem kötelező.

Az aletikus logikához hasonlóan a deontikus logikában is felrajzolható a deontikus modális négyszög (négyszögből származtatott hatszög). A négyszöget tanulmányozva a következő megállapítások tehetők:

1. Az átlók végein található Kötelező és Nem kötelező minősítések egymásnak ellentmondanak, csakúgy, mint a Megengedett és Tilos minősítések. Ezt a viszonyt itt is nevezhetjük *kontradiktóriusnak* is.
2. A Kötelezőnél minősítésnél gyengébb a Megengedett minősítés, a Tilosnál gyengébb a Nem kötelező, amely a viszonyt *alárendeltségnek* is nevezzük: a fentiekből következik az alsó.
3. A Kötelező és a Tilos minősítések egymásnak ellentmondanak, hiszen egyszerre valami nem lehet kötelező és tiltott is, de lehet mindkettő hamis, vagyis vannak nem kötelező, de nem is tiltott dolgok. Ezt a fogalmat *kontráriusnak* is nevezzük.
4. A Lehetséges és a Nem szükségszerű pár elemei egyszerre lehetnek igazak, de egyszerre hamisak nem. A viszonyt *szubkontráriusnak* is nevezzük.
5. A Szükségszerű és Lehetetlen minősítések diszjunkciójával kapjuk azt a kontingens esetet, amely – ismerve a kiinduló állapotokat – inkább egyfajta kizáró VAGY jellegű helyzetet jelent, amikor minimális a személyes szabadság, hiszen minden szabályozva van, nincs középút, vagy tilos valami, vagy kötelező. amelyben mindkét kiinduló minősítés egyidejűleg nem lehet igaz.
6. A Lehetséges és a Nem szükségszerű minősítések konjunkcióba hozhatók: így kapjuk a *Választható* (kontingens) minősítést. Ez a személyes szabadság területe, olyan kérdés, amelyet jogi eszközökkel nem szükséges szabályozni. Ez normahiányt jelent, ez a joghézag kifejezés pontos értelmezése.
7. Ha a Tilos és Kötelező minősítéseket konjunkcióba hoznánk (az ábrán nincs jelölve), akkor kapnánk a *jogi normaütközés (kollízió)* esetét, amikor is különböző törvények ugyanarról a dologról össze nem egyeztethető dolgokat követelnek meg. A hétköznapi beszédben ezt a helyzetet szokás – helytelenül – joghézagnak nevezni.

A deontikus logikában szintén többféle axiómarendszer használatos, amely az aletikus logikához hasonlóan a világszerkezeten értelmezett alternatívarelációra nézve megtett kikötésekkel van összefüggésben.



7. ábra Deontikus modális négyszög

2.4.4.1 SDL: Szabványos Deontikus Logika

Az SDL, a szabványos deontikus logika (Standard Deontic Logic) a korábban megismert aletikus modalitásfajtákhoz képest a következő különbségeket rögzíti:

- Törli a T alapvető aletikus logikában használatos aletikus sémát (**T** axiómát)

  □A ➔ A …hiszen ez az aletikus logikában azt jelentette „ami szükségszerű, az úgy is van". Ennek megfelelője a deontikus logikában a következő lenne: „ami kötelező, az úgy is van", ami nyilvánvalóan nem tehető fel. Ennek megfelelően az alternatívareláció a deontikus logikában *sosem reflexív*.

- Ezzel szemben viszont az alternatívareláció *definitív*: azaz ha valami minden elérhető világban igaz (kötelező), akkor kell lennie legalább egy ilyen világnak. A világszerkezet topológiájára nézve ez azt jelenti, hogy legalább 1 elérhető világ minden világból létezik. Tilos tehát a világszerkezetben az elszigetelt világok, valamint a zsákutca (nyelő) világok szerepeltetése.

  Mindezek következményeképpen egy új és fontos axiómát vehetünk a rendszerhez.

  D: □A➔◊A

  Ezt az axiómát szokás *deontikus sémának*, vagy *szerialitási axiómának (**ser**)* is nevezni, és azt a kézenfekvő igazságot rögzíti, hogy ha valami kötelező, akkor az már meg is van engedve. Ez az axióma a deontikus rendszerek alapja, azaz bármely deontikus rendszernek tartalmaznia kell az SDL-t és a fenti deontikus sémát (D axiómát) is.

2.4.4.2 SDL+: a Szabvány Deontikus Logika továbbfejlesztése

Az SDL-t megvizsgálva gondot jelent, hogy néhány olyan kifejezést nem tekint igaznak, amelyet a közfelfogás annak tart. Ezek a következők:

- „…elvárt, hogy ha valami kötelező, akkor az úgy is legyen…" vagy másféle értelmezésben „....a kötelező dolgokat kötelező megvalósítani is…"
- „…ha valami kötelezően kötelező, akkor az már önmagában is kötelező…" vagy másképp: ha valamilyen törvény meghozatalát magasabb rangú törvény – pl. alkotmány – előírja, akkor a törvényben rögzítendők már önmagukban is – még a törvény tényleges meghozatala nélkül is kötelezők.

Ehhez hasonló axiómákat a világszerkezetre vonatkozó *másodlagos definitivitás* tulajdonság kikötésével lehet elérni. Ez topológiailag azt rögzíti, hogy a bárhonnan elérhető világokból saját maguk mindig elérhetők legyenek. Ez egyfajta korlátozott reflexivitást jelent: a kikötés nem kötelező a világszerkezet beérkező éllel nem rendelkező / forrás elemein.

A topológiai kikötés a következő axiómával egyenértékű:

SDL+ vagy **ser2** axióma: $\Box(\Box A \rightarrow A)$

2.4.4.3 Az OS4 deontikus kalkulus

Az OS4 deontikus modális kalkulus a továbbfejlesztett szabványos deontikus logikából (SDL+) származtatható úgy, hogy annak világszerkezetére vonatkozólag a definitivitás kikötését még *tranzitivitással* is kiegészítjük. Ez a K4 aletikus rendszerben is használthoz hasonló axiómával bővíti ki az eredeti rendszert a következőképpen:

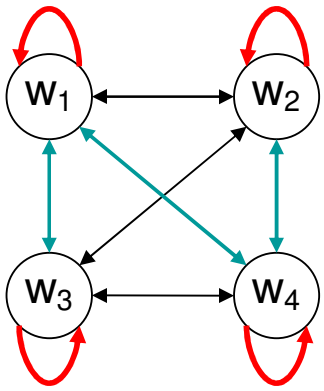
tra: $\Box A \rightarrow \Box\Box A$

Ez valami lelkiismereti kötelezettségfogalomhoz hasonló szerkezettel bővíti az szabványos deontikus rendszert. A **tra** axióma értelmezhető ugyanis úgy: Ha valami kötelező, akkor azok betartása is kötelező – azaz a kötelezettségek betartását mindenki elvárja.

Az axióma a dualitás miatt a következő axiómával egyenértékű:

tra: $\Diamond\Diamond A \rightarrow \Diamond A$

Az ilyen megfogalmazás értelmezése: ha megengedett, hogy valamit megengedjünk, akkor az már meg is van engedve.



8. ábra Példa az OS5 rendszer világszerkezetére

2.4.4.4 Az OS5 deontikus kalkulus

Az OS4 kalkulus esetében – hasonlóan az aletikus K5-höz – a rendszerünkben az elérhetőségi relációra vonatkozólag az OS4-ben kikötötteken túl még a *szimmetriát* is kikötjük. A topológiai kikötésnek megfelelő új axióma (az aletikus **B**-nek megfelelő):

sym: $A \rightarrow \Box\Diamond A$

Vagyis: ha valami igaz, akkor kötelező, hogy meg legyen engedve…

Megjegyezzük, hogy e rendszernek a deontikus hozzáállás (normarendszernek való megfelelés) szempontjából nincs sok jelentősége. Egyfelől a **sym** axióma a valóságos

világokban egyáltalán nem kézenfekvő (léteznek törvényszegő esetek). Másrészt viszont kézenfekvő, hogy ezek után az OS5-ben teljesül az aletikus séma (**T** axióma), vagyis a **sym** axióma a **T** axióma törlési követelményének ellentmond.

2.4.5 Modális időlogika

A világ egyes entitásainak, de a logikai állításoknak, és természetesen a jogi normarendszerek normáinak is valamiféle *időbeli hatályuk* van. Bár léteznek idő független, állandó, örök igazságok is, az állításaink jó része nem ilyen. Az „Időlogika" (temporal logic) kifejezés a fenti problémára megoldásul adott logikai rendszerek összefoglaló neve. A következő szakaszban a *modális időlogikáról* lesz szó, amely az idő kezelését a modális logikák keretein belül, megfelelő modális operátorok bevezetésével oldja meg.[56]

A modális időlogika a múltra és a jövőre vonatkozóan 2-2 modális operátort vezet be, tehát a korábbi meghatározások értelmében polimodális rendszerről van szó. Tekintve, hogy ezek közül csak kettő független egymástól, a logika *bimodális*. A bevezetett operátorok a következők:

P x: *Gyenge múltbeli* modalitás (P, mint Past), a jelentése: valaha igaz volt, hogy x.

H x: *Erős múltbeli* modalitás, a jelentése: mindig igaz volt, hogy x.

F x: *Gyenge jövőbeli* modalitás (F, mint Future), a jelentése: valamikor igaz lesz, hogy x.

G x: *Erős jövőbeli* modalitás, a jelentése: mindig igaz lesz, hogy x.

Az időbeli modalitásokra érvényes a *múlt-jövő felcserélhetőségi (szimmetria) szabály*. Ez kimondja, hogy amennyiben egy állításban a múlt és a jövő függetlenek egymástól, akkor az azonos erősségű múltbeli és jövőbeli operátorok kicserélhetők egymással.

Az aletikus logika modalitásaihoz hasonlóan a megfelelő gyenge és erős modalitások összefüggenek, amit az *időbeli dualitás axiómáinak* nevezünk.

P x = ~H~x …valami valaha akkor és csak akkor történt meg, ha az, hogy nem történt meg, soha nem volt igaz.

F x = ~G~x …valami valamikor akkor és csak akkor fog megtörténni, ha az, hogy soha nem fog megtörténni, nem igaz.

Ehhez még a következő általános axiómákat tehetjük hozzá:

H x ➔ P x …ha valami mindig megtörtént, akkor valaha megtörtént (az erős modalitásból következik a gyenge)

G x ➔ F x …ha valami mindig meg fog történni, akkor valamikor meg fog történni

A modális időlogikán belül – az aletikus és a deontikus modalitáshoz hasonlóan – az alábbi szakaszokban tárgyalt különböző axiómarendszerek léteznek.

2.4.5.1 A TL0 modális időlogikai rendszer

A TL0 (Temporal Logic 0) időlogikai kalkulus a következő axiómákat rögzíti:

[56] Galton, Antony: Temporal Logic [Ga08]

1. A modális általánosításnak (Modal Generalzation, **MG**) megfelelő időlogikai axióma az időáltalánosítás (Temporal Generalization, **TG**). Ez azt állítja, hogy ha valami időfüggetlen/örök igazság, akkor az a múltban is, és a jövőben is mindig igaz volt és igaz is marad.

   TG_F: A ➔ H A.

   TG_P: A ➔ G A.

2. A Kripke-féle axiómák megfelelői a modális időlogikai rendszerben. Ez azt mondja ki, hogy ha egy következtetés a múltban vagy a jövőben mindig is igaz volt vagy lesz, akkor, ha a feltételoldal a múltban vagy a jövőben mindig is fennállt vagy majd fennáll, akkor a következményoldal is.

   K_F: H(A→B) ➔ (HA→HB)

   K_P: G(A→B) ➔ (GA→GB)

3. A múlt és a jövő mindkét irányban összekapcsolható, de ellentétes értelemben. A megfelelő axiómát múlt- vagy jövőbeli antiszimmetria axiómának is nevezik, mert a múlt és jövő ellentétét fejezik ki. A múltbeli azt mondja ki, hogy ha valami igaz, akkor mindig igaz volt, hogy valamikor igaz lesz. Párja, a jövőbeli viszont azt állítja, hogy ha valami igaz, akkor mindig igaz lesz, hogy valaha igaz volt.

   $antisym_F$: A ➔ G P A.

   $antisym_P$: A ➔ H F A.

2.4.5.2 A TL1 modális időlogikai rendszer

A TL1 modális időlogikai rendszer a TL0-hoz még a tranzitivitást is hozzáveszi. Ez most is kettő, a jövőre, és a múltra is vonatkozó – egyébként teljesen kézenfekvő – axióma felvételét jelenti:

tra_F: G A ➔ G G A — vagyis, ha valami mindig igaz lesz a jövőben, akkor a jövő minden pillanatában az is mindig igaz lesz, hogy az adott pillanathoz képest a jövőben mindig igaz lesz.

tra_P: H A ➔ H H A — vagyis, ha valami mindig igaz volt a múltban, akkor a múlt minden pillanatában az is mindig igaz volt, hogy az adott pillanathoz képesti múltban mindig igaz volt.

2.4.5.3 A TL2 modális időlogikai rendszer

A TL1-ben még semmi nem zárta ki alternatív, közös múltú vagy jövőjű világok létezését (csak részben rendezést kötöttek ki az axiómák). Ez egyes fantasztikus történetek alapötlete, a mi szempontunkból azonban célszerű kizárni az efféléket. A kizárás módja: a TL1 rendszerhez még az időre vonatkozó trichotómia-axiómák hozzávételével az időre vonatkozólag teljes rendezést rögzítünk.

tri_F: $FA \wedge FB \rightarrow F(A \wedge B) \vee F(A \wedge FB) \vee F(FA \wedge B)$ Ez azt rögzíti, hogy ha két állítás a jövőben valamikor igaz lesz, akkor vagy a jövőben valamikor mindkettő egyszerre lesz igaz, vagy a jövőben valamikor az egyik, majd ehhez képesti jövőben valamikor a másik lesz igaz, vagy fordítva.

tri_P: $PA \wedge PB \rightarrow P(A \wedge B) \vee P(A \wedge PB) \vee P(PA \wedge B)$ Ez azt rögzíti, hogy ha két állítás a múltban valaha igaz volt, akkor vagy a múltban valaha mindkettő egyszerre volt

igaz, vagy a múltban valaha az egyik, és ehhez képesti múltban a másik volt igaz, vagy fordítva.

2.4.5.4 A TL5 modális időlogikai rendszer

A TL2 rendszer még nem rögzítette, hogy az időpontok tetszőlegesen sűrűk. Vagyis másképp fogalmazva: nincsen csak a két végpontjából álló időszakasz, mert ezeknek mindig van belső pontjuk.

A TL5 rendszer a TL2-ből származtatható úgy, hogy hozzávesszük a tetszőleges idősűrűségre vonatkozó axiómákat.

den$_F$: FA → FFA Ha valami a jövőben valamikor igaz lesz, akkor létezik egy olyan jövőbeli időpillanat is, amihez képest ugyanaz a dolog szintén a jövőben lesz valamikor igaz.

den$_P$: PA → PPA Ha valami a múltban valaha igaz volt, akkor létezett egy olyan múltbeli időpillanat is, amihez képest ugyanaz a dolog szintén a múltban volt valaha igaz.

2.4.6 Több-szereplős episztemikus/doxasztikus logika

Az *episztemológia a tudás tudománya*, amelyet már a paradicsomi Bűnbeesés kezdőképe gerjeszthetett, de ezt az ókori görög filozófusok tudományos eszközökkel is művelték, valamint magyar népmesék is gyakorta érintik. A kicsit szétszórt kezdetek után a modern episztemikus modális logika alapjait Hintikka finn filozófus fektette le a múlt század hatvanas éveiben.

Az episztemikus logikát, amelynek központi fogalma a tudás, együtt említik a *doxasztikus* logikával, amely analóg eszközöket használ, a különbség annyi, hogy a tudás helyett a központi fogalom a hit.

Az episztemikus-doxasztikus modális logika alapvetően kétféle modalitásfogalmat, és ennek megfelelően két modális operátort vezet be:[57]

- A □ operátort ritkán használjuk. Helyette azt írjuk: K P, (K, mint Knows) amit úgy olvasunk ki: „tudjuk, hogy P” – az alany általános, nincs jelentősége, hogy pontosan ki, általában mindenki tudja, hogy P.
- A ◊ operátort egyáltalán nem használjuk. A használatos másik operátor B P ugyanis a *hitről* szól. A B P állítást (B, mint Believes) ennek megfelelően úgy olvassuk ki: „azt hisszük, hogy P”.

Bár az episztemikus/doxasztikus modalitásfogalom és a felette értelmezhető kalkulus a fenti két operátorral is felépíthető, ennek jelentősége lényegesen kisebb annál, mintha a fenti, leegyszerűsített fogalmakat kiterjesztjük egy többszereplős környezetre. Ekkor a fentiek a következőképpen módosulnak:

- a K operátor helyett a K_c operátort használjuk. Itt a c paraméterrel a szóban forgó szereplőre, a tudás birtokosára hivatkozunk, a K_c P állítást pedig így olvassuk ki: „c tudja, hogy P”.
- a B operátor helyett a B_c operátort használjuk. A c paraméterrel a szóban forgó szereplőre, tudás birtokosára hivatkozunk, a B_c P állítást pedig így olvassuk ki: „c azt hiszi, hogy P”.

[57] Hendricks, Vincent and Symons, John: Epistemic Logic [HeSy09]

Több szereplős környezetben a világszerkezet értelmezése kissé változik. A *Kripke-keret* (frame) vagy *világszerkezet* fogalmát ugyanis egy c szereplőre vonatkoztatjuk, azaz minden szereplőre egy külön világszerkezetet értelmezhetünk. Így egy c szereplő világszerkezete alatt azt az $\mathcal{F}_c$=<W, R_c> párt értjük, ahol:

- W: az értelmezhető világok (véges) halmaza.
- R_c: pedig a világok feletti *elérhetőségi- / kompatibilitási- / alternatíva-reláció*. A viszony elemeit most is jelölhetjük közbeékelődő (infix) operátoros módon: $w_1 R_c w_2$.

Az egyes világokban a logikai szerkezetek és kifejezések kiértékelése még ugyanazon szereplő számára is más és más lehet, a különböző szereplők számára azonban ez a különbség még határozottabb. A világok közötti elérhetőségi relációcsokor a világok között egy olyan gráfot feszít ki, amelyben az éleket célszerű a szereplő nevével címkézni. A gyakorlati szempontból a világszerkezetet és az elérhetőségi relációkat olyan módon vesszük fel, hogy az egyes szereplők által biztosan tudott információk az adott szereplő részgráfjának mindegyik világocskájában ugyanolyan logikai értékű legyen. Vagyis, az adott szereplő részgráfja az általa határozottan nem ismert világocskákat/állapotokat köt össze.

Egy adott L^{m0} nulladrendű modális logikai nyelv *Kripke-modelljén* vagy *interpretációján* az M=<F_c,V> párt értjük. Vagyis meg kell hozzá adnunk:

- a szereplőfüggő világszerkezetet
- és azt, hogy az egyes formulák értéke hogyan alakul az egyes világok esetén.

Ha egy A formula igaz az M modell w világában, azt így jelöljük: (M,w) |= A. Ha a modell nyilvánvaló, akkor azt el is hagyhatjuk: w |= A. Ha az ellenkezőjét akarjuk kifejezni, vagyis egy adott w világban az A formula hamis, azt a következőképpen jelöljük: w ~|= A.

Az egyszereplős Kripke-féle értelmezéshez hasonlóan egy K_c A kifejezésről egy adott w világban akkor mondjuk, hogy igaz ((M,w) |= K_c A), ha a c szereplő számára elérhető összes világban igaz ($\forall$w': wR_cw'→(M,w')|=A).

Az episztemikus/doxasztikus világban a leggyengébb aletikus rendszereknek (K,S4) megfelelőket nem használjuk. A gyakorlatban az alább következő rendszereket szokás megkülönböztetni.

2.4.6.1 A KT4 episztemikus modális rendszer

A KT4 episztemikus modális rendszer az S4 aletikus rendszer megfelelője. Ebből adódóan a következő axiómákat értelmezi:

1. a *Kripke-féle axiómának* is nevezett **K** axióma. Ez a klasszikus modus ponenshez hasonló következtetési szabály, amely azt állítja, hogy ha tudomásunk van egy következtetésről, valamint a következtetés feltételének teljesüléséről, akkor tudomásunk van a következmény teljesüléséről is.

   K: $K_c(A \rightarrow B) \rightarrow (K_c A \rightarrow K_c B)$

   A **K** axiómát megkérdőjelezi a *logikai mindentudás (omniscience) problémája*. Használata egyáltalán nem kézenfekvő, túl erősnek érezzük az axióma többszörös alkalmazását (tranzitív lezárását). Vagyis a kérdés az, ha egy szereplő valamely következtetést ismer, és a feltételeket is ismeri, akkor a következtetést

is ismeri-e. (Vajon Paris Hilton tudja-e, hogy végtelen sok prímszám van, csak azért, mert tanult számtant az iskolában?)

2. az aletikus **T** axióma episztemikus megfelelője. Eszerint minden világ elérhető sajátmagából, mert az alternatívareláció reflexív. Ha valamit biztosan tudok, akkor az úgy is van. Az axiómát szokás *igazolhatóságnak* (*veridicality*) is nevezni.

   T: $K_cA \rightarrow A$

3. az aletikus **tra** axióma megfelelőjének hozzávétele. Ezt az episztemikus logikában szokás **4.** axiómának is nevezni, mert ez a KT4 rendszer névadója.

   tra: $K_cA \rightarrow K_c K_c A$

   Az axióma azt fejezi ki: ha tudok valamit, akkor azt is tudom, hogy tudom. Emiatt szokás a *pozitív önismeret (positive introspection)* axiómájának is nevezni. Az elérhetőségi relációra nézve – az aletikus logikához hasonlóan itt is – az axióma topológiailag tranzitivitást követel meg.

2.4.6.2 A KT5 episztemikus modális rendszer

A KT5 rendszer az aletikus S5 rendszer megfelelője, amelyet úgy kapunk, hogy a KT4 rendszert kiterjesztjük az újabb, **5** axiómával.

5: $\sim K_c A \rightarrow K_c \sim K_c A$

Az **5** axióma azt fejezi ki: ha valamit nem tudunk, akkor azt biztosan tudjuk is. Az axióma a **tra** pozitív önismeret axióma egy változata, amit ezért a *negatív önismeret (negative introspection)* axiómájának is nevezünk. Az alternatívareláció topológiájára nézve az **5** axióma az euklideszi reláció fennállását követeli meg.

2.4.6.3 A KD4-KD5 doxasztikus modális rendszerek

Doxasztikus rendszerek a legkönnyebben az episztemikus rendszerekből származtathatók úgy, hogy a K operátort kicseréljük a B operátorral. Ilyen módon az episztemikus rendszerekhez hasonlót kapunk, amelyben az analógia miatt az egyes megállapítások is érvényben maradnak.

A doxasztikus modalitásban használt KD4-KD5 logikai rendszerek az analóg KT4-KT5 rendszerekből a következőképpen származtathatók:

1. Töröljük a T axiómát, az igazolhatóságot, vagyis ha valaki valamit hisz, akkor az egyáltalán nem biztos, hogy helytálló.
2. Felvesszük viszont a deontikus modalitásban használt **D** axióma doxasztikus megfelelőjét. Ez azt az igazságot fejezi ki, hogy ha *valamit hiszünk*, akkor *nem hihetjük az ellenkezőjét* is. Az alternatívarelációra vonatkoztatva ez most is szerialitást fejez ki, vagyis ha valamit tudunk, akkor van legalább egy világocska, ahol ez igaz is. Topológiailag tehát tilos az elszigetelt, illetve a nyelő világok használata.

   D: $B_cA \rightarrow \sim B_c \sim A$

2.5 Logikai programozás

A *logikai programozás* a Mesterséges Intelligencia (MI) tárgykörébe tartozó problémák egyik megközelítési iránya a szuperpárhuzamos-neurális számítási módok és egyéb irányok mellett. Ez a Világ modellezését és az emberéhez hasonló intelligens

viselkedésmódot nem az emberi agy biológiai szerkezetére vonatkozó modellalkotáson keresztül próbálja megragadni, hanem a modellalkotás alapvető eszközeként a matematikai logika eszközrendszerét használja.

A Logikai Programozás tehát ebben a megközelítésben a külső valóság – kellően bonyolult, kellően részletes – modellezése feszes szemantikájú, matematikai logikára alapuló eszközökkel, végső soron egy olyan modellalkotási megközelítés (egy paradigma), amely az Objektum Orientált Modellezés, a hagyományos Logikai Programozás és a Logikai Modellezés részterületeit egyaránt magába foglalja. A jelen értekezés fontos tézise, hogy *a logikai és az objektumorientált programozás látszólag független témakörei valójában közös tőről fakadnak*, sőt a feladatok bizonyos korlátok elismerése mellett egymásba át is alakíthatók.

A „logikai programozás" hagyományos értelmezésében a Prolog programozási nyelv és egyes kiterjesztéseinek, változatainak megvalósításait, illetve azok gyakorlati programkészítésre történő alkalmazásának a megoldásait, módszereit foglalta magába.[58] Ez az eredeti értelmezés egy gyakorlati és Prolog központú meghatározás volt, amit az értekezésben most a saját céljainkra kicsit szélesítünk, és eltérő összefüggésbe helyezünk.

2.5.1 Meghatározás

A *logikai programozás* a valóságközeli, tartalmilag széles modellezési területet felölelő feladatok olyan megoldási megközelítése, amely a *valóságot matematikai logikai alapú modellekkel* írja le, a feladatok megoldását pedig ilyen modelleken végzett – és matematikailag tökéletesen és szabatosan leírható és értelmezhető – *műveletekkel, átalakításokkal, és tételbizonyítási módszerekkel* adja meg.

- a feladatok *valóságközeliek*, mert elsősorban nem gépi fogalmak és nem is nagyon elvonatkoztatott vagy nagyon szűken értelmezett szakterületre vonatkozó feladatok megoldását keressük (nem program-hibakeresőt, nem mátrixinvertálót és nem is bérszámfejtő programot írunk).
- a feladatok *széleskörűek*, mert lehetőleg a valós fogalomrendszerek egy széles halmazát ölelik fel. Ezt modellezzük, és ezt oldjuk meg az elkészített szoftver eszközökkel.
- a valóság modellezésének – a görög filozófusok óta – az egyik megközelítése a *matematikai logika*, amelyben a való világ egyes objektumainak különféle logikai fogalmakat feleltetünk meg
- egyes feladatok már pusztán *átalakításokkal* is megoldhatók: ezen feladatok a modellben ábrázolt ismeretanyagot nem bővítik, csupán a megjelenítés és kiértékelés céljaira formai változtatásokat hajtanak végre. Új ismeretek létrehozását és következtetések levonását *gépi tételbizonyítási módszerek* alkalmazásával lehet megtenni.

A logikai programozás fenti meghatározása egyrészt szélesebb, mint a konkrét programkészítés kérdésköre. Felölel minden olyan matematikai elméletet, tételbizonyítási technikát, modellezési módszert és programozási nyelvet is, amely a matematika logikai tudományágára épül, és amely ennek a módszereit, és ezt a megközelítést alkalmazza.

---

[58] W. F. Clockshin-C. S. Mellish: Programming in Prolog [CloMe]

Másrészt tehát egy sor olyan programozási nyelvet a témakörbe emel, amelynek – a Prolog nyelvtől függetlenül, vagy éppen annak kiterjesztéseként – matematikai logikai alapjai vannak. Így például – a jelen dolgozat céljaival teljes összhangban – az objektum-orientált megközelítést is részben ide tartozónak tekinti. Olyan részben, amennyiben az elsősorban a való világról készített modellalkotási célokat szolgál, és amennyiben a modellalkotás matematikai logikai tisztaságát tartjuk szem előtt.

A logikai programozás a dolgozatban alkalmazott megközelítésben egyszerre *tárgy és eszköz*. Tárgy, mert az objektum-orientált modellalkotásról kizárólag a fenti megközelítésben szól, de eszköz is, mert mondanivalóját végső soron matematikai logikai eszközökkel fejti ki, és az említett alkalmazási példák megvalósítása is logikai programnyelvek segítségével történt, illetve csak azokkal történhetne meg.

2.5.2 A Prolog és a szoftverkészítés

A Prolog programozási nyelv első definíciója 1972-ből, Alain Colmerauertől származik. A hetvenes években a világ néhány helyén, köztük 1975-ben hazánkban is Prolog megvalósítások születtek, amelyre építve aztán különféle problémamegoldó kisalkalmazásokat írtak a lelkes programozók.

1981-ben az ötödik generációs számítógépekre vonatkozó japán bejelentések fényében, (amelynek az alapszoftvere a Prolog nyelvre épült volna), a világon több helyen is nekiláttak a Prolog nyelv ipari szoftverek előállítására történő továbbfejlesztésének. Ezek között a Szeredi Péter vezette hazai MPROLOG csapat a korábbi kutatási eredmények fényében meglehetősen jó helyen állt. Ezt az előnyös helyzetét azonban – részben a piac gyors beszűkülése miatt, részben az ország akkori gazdasági és műszaki elszigeteltségének és lemaradásának tulajdoníthatóan az évtized végére elvesztette.

Funkcionális szempontból a 80-as években a Prolog nyelv alapdefiníciójához eszközrendszer készült. Már az alapdefiníció is tartalmazta azonban a DCG (Definite Clause Grammar) környezetfüggetlen attribútumnyelvtan elemzőjét, ami egyrészt a rekurzív leszállásos elemzési módszer visszalépéses kibővítése. Másrészt az elemző lehetővé teszi a nemterminális szimbólumok Prolog-fákkal, mint attribútumokkal történő kiegészítését.

A 90-es évek elején állapodtak meg a nyelv szabványos definíciójáról, de ebben az időszakban terjedt el a „Korlátos Logikai Programozás” (Constrained Logic Programming, CLP) irányzata is.[59]

A nyelvnek a CLP-n kívül is számos kiterjesztése született. Ezek általában a vezérlési mechanizmust vagy a Prolog meglehetősen primitív beépített tételbizonyítási stratégiáját bővítik. Ilyenek, pl. a szerző által publikált Pro-Contra-Log (PC-LOG),[60] [61] amely a visszafelé haladó stratégiával működő Prologot az alapnyelv előre haladó változatával egészíti ki, vagy a GCLA,[62] amely a Horn klózok egy általánosítására, és egyben heurisztikusan befolyásolható következtető mechanizmusra épül. Hasonló kiterjesztések számosan léteznek még, de ezek teljes áttekintése a jelen értekezés céljain mindenképpen kívül esik.

---

[59] Roman Barták: Constraint Programming: in Pursuit of the Holy Gral [Bar99]

[60] Kilián Imre: Contralog: egy előre haladó Prolog motor és alkalmazása ℜeALIS nyelvi elemzésre [Kil11.1]

[61] Kilián Imre: Tárgymodell változatok a ℜeALIS nyelvi elemzéshez [Kil11.2]

[62] Per Kreuger: GCLA II. A Definitional Approach to Control [GCLA92]

2.5.3 A Prolog tételbizonyítási stratégiája

Az elemi logika fejezetének rezolúcióról szóló szakaszában részletesen tárgyaltunk egyes tipikus rezolúciós stratégiákat. A Prolog nyelv a Horn klózok alapjára épül. Ez az elsőrendű logika egy olyan részosztálya, amelyben a következményoldali diszjunkció nem megengedett, így ott legfeljebb egy literál szerepelhet.

A pontosan egy következményoldali literált tartalmazó klózokat szokás *szabályoknak* vagy *definit klózoknak (definite clause)* is nevezni, bár ez utóbbi kifejezés a magyar szaknyelvben nemigen terjedt el. Az egyetlen következmény-literált sem tartalmazó klóz neve *célállítás*, az egyetlen feltételliterált tartalmazóé pedig *tényállítás*.

Az egyetlen következményliterálra vonatkozó megkötésnek a rezolúciós tételbizonyítás szempontjából fontos szerepe van: ha egy feltételoldalon álló negatív literálhoz találunk egy illeszkedő fejű (következményrészű) állítást, akkor az egyesítés után annak feltételrészét kell behelyettesíteni a negatív literál helyére. Ezt a Prolog szakirodalom *redukciós lépésnek* is nevezi. Az aktuális állításssorozat hossza akkor rövidül, ha a behelyettesített állításban nincs negatív literál – vagyis ha tényállítást helyettesítünk be.

A Prolog alapvető tételbizonyítási stratégiája *visszafelé haladó*. Ez azt jelenti, hogy a programfutást a csak negatív literálból álló célállítások indítják el. A célállítást magát hipotézisnek is felfoghatjuk, a Prolog programot pedig olyan elméletnek, amely a programot alkotó állításokból, illetve ezek konjunkciójából épül fel. A programfutás ezek után nem más, mint egy indirekt tételbizonyítás: a célállítás negáltjáról bebizonyítjuk, hogy az elmélettel együtt inkonzisztens, vagyis ellentmondásra vezet. Az ellentmondás bizonyítása *konstruktív*: melynek során az egyes univerzálisan kvantált változókra vonatkozólag ellenpéldát keresünk: olyan változóértékeket, amelyre az ellentmondás nyilvánvaló.

A visszafelé haladó tételbizonyítás során még további nem rögzített *döntési pontok* is felmerülnek, amelyeket a konkrét *rezolúciós stratégia* válaszol meg:

- Egy negatív literálsor melyik elemével kezdjük a rezolválást? A Prolog ezt minden esetben a legelsővel (a bal szélsővel) kezdi.
- Ha egy negatív literálra több illeszkedő pozitív literál is található, akkor melyikkel kezdjük a rezolválást? A Prolog minden esetben a szövegesen legkorábban előfordulóval kezd, majd a bizonyítás sikertelensége esetén visszalépésesen veszi a többit is.

A Prolog tételbizonyítási stratégiáját a LUSH (Leftmost, Uppermost Selection Heuristic) betűszóval is jellemzik, ami az SLD (Selective Linear Definite Clause) rezolúció egyik fajtája. Ez a következőket jelenti:

- Lineáris rezolúció, vagyis a rezolváláskor mindig az előző rezolúciós lépés eredménye (a rezolvens) az egyik rezolválandó klóz.
- Input-rezolúció is, vagyis a másik rezolválandó klózt mindig az alapklózok (az eredeti elmélet klózai) közül vesszük.
- A literálok balról jobbra és a klózok felülről lefelé történő kiválasztási szabályát már említettük.

2.5.4 Contralog: a Prolog előrehaladó végrehajtása

A Contralog a Prolog egy kiterjesztésének tekinthető. Tervezésekor cél volt, hogy az általa megvalósított előre- és a Prolog eredeti visszafelé haladó működésmódja

integrálható legyen úgy, hogy a logikai forrásnyelv ugyanaz (a Horn klózok nyelve). Ezt a közös forrásnyelvet részben maga a Prolog visszafelé haladóan, részben pedig az előrehaladó motor akként értékelhet ki. A kétféle rezolúciós stratégia pedig a programozó által vezérelhetően váltható: egyrészt a Prologból legyen meghívható az előrehaladó motor, másrészt az előrehaladó végrehajtásból legyen meghívható a Prolog.

A Contralog programnyelv a Horn klózok nyelvét (a Prolog nyelvet) előrehaladó stratégiát megvalósítva képezi le a Prolog nyelvre magára úgy, hogy egy inkrementális fordítóprogram a beolvasott Contralog szabályokat Prolog szabályokká fordítja le, és a szabványos Prolog futtatókörnyezetben működteti.[63]

Az így létrehozott rendszerben tehát minden fordítva működik, mint a Prologban:

- A következtetés *adat-vezérelt*, amit nem a célállítások, hanem a *tények* indítanak.
- ha van olyan szabály, amelynek feltételrészében egy adott tény szerepel, akkor megvizsgáljuk, hogy a feltétel többi részét már sikerült-e bebizonyítani korábban. Ha igen, akkor a *szabály tüzel*, vagyis a következményrészt sikerült bebizonyítanunk.
- A bebizonyított következmény újabb *egységklóz rezolvenseket* (bebizonyított tényeket) jelent, amelyet a *munkatáblán* (blackboard-on) tárolunk, és ezzel a ténnyel folytatjuk a bizonyítást.
- A következtetési folyamatot a *célállítások* állítják le.
- Célállítás elérésekor, vagy ha bármi okból a bizonyítás az adott láncon tovább nem folytatható, a rendszer visszalép, és egy korábban nyitva hagyott alternatíva mentén próbálkozik újra.

A Prolog-Contralog kapcsolatot kétféleképpen lehet működtetni:

- a Contralog szabályok feltételrészében a `{}/1` literál közvetlen Prolog cél meghívását eredményezi.
- A Contralog *importok* azok a tények, amelyek egy modul következtetési láncát elindítják. Ez az indító tényeknek megfelelő Prolog tüzelési szabályok *exportját* jelenti.
- A Contralog *exportok* viszont azok a predikátumok, amelyeket az előre haladó stratégia szerint tényként kikövetkeztettünk, és vagy másik modul *importját* elégítjük ki vele, vagy a Prolog futtatórendszer egy predikátumát hívjuk meg. A Contralog exportokból Prolog importok lesznek, (bár ezt a fogalmat a szabványos Prolog nem ismeri).

A fent ismertetett alapműködésen túl az elburjánzó következménytények megakadályozására, illetve törlésére logikán kívüli eszközöket vezettünk be:

- minden tárgymodulban létrehoztunk egy, a *munkatáblát teljesen törlő* Prolog eljárást, amit a `MODULE:clean` hívással indíthatunk.
- egyes tények kikövetkeztetésekor *letilthatjuk a következtetést* az adott szálon (a tényt a munkatáblán tároljuk ugyan, de a megfelelő tüzelő eljárásokat nem hívjuk meg). Ezt a működést a `:- lazy NAME/ARITY.` deklaráció hatására válthatjuk ki.
- egyes tények kikövetkeztetésekor az *azonos névjegyű tényeket mind töröljük* a munkatábláról (`:- var NAME/ARITY.`), vagy egyes argumentumokat – a

[63] W.F.Clockshin-C.S.Mellish: Programming in Prolog [CloMe]

relációs technológiához hasonlóan – kulcsként tekintve, csak az azonos kulcsú tényt töröljük. Ezt pedig a `:- key(NAME(KEYVECTOR)).` deklarációval válthatjuk ki, ahol a `KEYVECTOR` szerkezet egy argumentumlista, ahol a „+” jel azt jelzi, hogy az argumentum kulcsként szerepel, a „–” pedig azt, hogy nem.

Az előre haladó következtetés alapproblémája, hogy a klózok feltételrészén több elemi feltétel is szerepelhet. Amikor ezek közül nem mindegyik elégül ki, a hiányzókat meg kell várni, és a következmény tüzelését csak akkor indítjuk, ha az utolsó feltétel is kielégült. Ezt úgy érjük el, hogy a már kielégülteket dinamikus állításokként tároljuk, és egy Contralog szabály összes feltétel-literáljához létrehozunk egy külön Prolog szabályt, ami ellenőrzi, hogy a többi feltétel már korábban teljesült-e. Vegyünk egy egyszerű példát, tekintsük a következő Contralog szabályt!

```
a:-b, c.
```

Ha a `b` vagy a `c` feltételek kielégültek, akkor az eredményként kapott tények a megfelelő `b/0`, illetve `c/0` dinamikus állításokban találhatók. Mindegyik feltételhez létrehozunk egy `fire_NAME` tüzelő, és egy `test_NAME` ellenőrző Prolog predikátumot. Az előbbi tárolja a kikövetkeztetett tényt, majd meghívja az utóbbit. Az utóbbi pedig ellenőrzi, hogy a többi Contralog feltétel teljesül-e, és ha igen, akkor meghívja a következményhez tartozó tüzelő eljárást.

A fenti esetben ez a következő Prolog kód létrehozását jelenti (az `assert` beépített eljárás a kikövetkeztetett tényt tárolja):

```
fire_b:- assert(b), test_b.

fire_c:- assert(c), test_c.

test_b:- c, fire_a.

test_c:- b, fire_a.
```

A fenti tárgymodellben továbbra is a Prologhoz hasonló *visszalépéses keresés* történik. Választási pontok többféleképpen is keletkezhetnek:

- Ha egy feltétel több Contralog szabályban is szerepel, akkor annyi Prolog alternatíva jön létre belőle, ahány szabályban a feltétel szerepel.
- Ha egy feltétel többször is teljesül, akkor ugyanannyi *dinamikus tény* jön létre belőle – feltéve, hogy az adott feltételre nem teljesülnek a következtetési ágak megnyirbálását célzó deklarációk.
- A modul összes statikus tényállításának a tárolása úgy történik, hogy a Prolog modul célállítása visszalépésesen meghívja az összes *statikus tény tüzelő eljárását*. Vagyis, ha valamilyen feltétel nem teljesül, akkor végső soron akár egészen a Prolog célállításig is történhet egy visszalépés.

A nyitott választási pontokra a visszalépések során kerül a vezérlés. Visszalépés szintén többféleképpen bekövetkezhet:

- Ha valamelyik feltétel az adott pillanatban *nem teljesül*. Ez lehet Contralog feltétel, de a feltételek közé beszúrt Prolog feltétel meghiúsulása is.
- Ha egy Contralog célállítás elérésekor (a Prologhoz hasonlóan) újabb megoldások kérésével *visszalépésre kényszerítjük* a rendszert.

Az előre haladó tárgymodell esetében a *szabályalkalmazási rohamokat (burstout)* az egyes tények felvétele (beérkezése) indítja. A tények érkezhetnek *aszinkron módon*, időben elcsúsztatva, sőt akár tetszőleges sorrendben is: egy következtetési lépés akkor

történik meg, ha minden feltétel megérkezett és rendelkezésre áll. Bár van lehetőség a *következtetési fa ágainak nyírbálására*, a következmények a teljes gazdagságukban előállnak, ha ezekből néhány illeszkedik a megadott célállításokra, akkor a következtetés leáll.

A modell előnye, hogy az egyszer bebizonyított tényeket tároljuk, és azokat akárhányszor fel lehet még használni. Ezzel a modell lényegében a *dinamikus programozásnak* nevezett algoritmuskészítési paradigmát támogatja.

2.5.5 ProType: egy Prolog típusleíró résznyelv

- Bonyolult Prolog alkalmazások, pl. természetes nyelvi megvalósítások egyik sarokköve az adatszerkezetek pontos leírási módja. Ezt célszerűen valamilyen nyelvleíró formális nyelven tehetjük meg. Az adatszerkezet leírás alatt a Prolog egy résznyelvét értjük, ami nem más, mint az alapnyelvtan (a Prolog kifejezések általános nyelvtana) egyfajta alkalmazói megszorítása. Mivel a Prolog típustalan, ezért erre a célra egy Prolog *típusleíró nyelvkiegészítést* (ProType) valósítottunk meg. A típusleíró nyelvkiegészítést a természetes nyelvi alkalmazásokra való tekintettel rögzítettük: a fő cél az volt, hogy a típusleírás segítségével a későbbi fejezetekben megemlített ℜeALIS természetes nyelvi rendszer adatszerkezeteit képesek legyünk leírni. A leírandó nyelvek alapvetően *jegyszerkezetesek (feature structured)*, egy jegyszerkezet mátrix megadása alapvetően Prolog listában, `JEGY:ÉRTÉK` párokkal lehetséges.

A típusleírás működtetéséhez egy Prolog forrásfájlt kell betöltenünk (`proType.pl`), amely a típusdeklarációt a Prolog `term_expansion/2` programkampójának (hook predicate) megadásával valósítja meg. A típusdeklaráció ezek után a program bármelyik Prolog forrásfájljában megtörténhet a…

```
:-type(TYPE=DECL).
```

…deklaráció segítségével. A deklarált típusokra vonatkozólag később, de már futásidőben ellenőrizhetjük, hogy egy Prolog objektum megfelel-e a típusnak. Ezt a típusellenőrző forrásfájlban közzétett `check(?STRUCT,+TYPE).` eljárás meghívásával tehetjük meg.

A `TYPE` típusnév a deklarációban és az ellenőrző eljárásban is mindenképpen változómentes kifejezés, célszerűen Prolog azonosítónak kell lennie. A `DECL` deklarációs kifejezésben a következő elemek használhatók:

- Alapértelmezett skaláris típusok: `integer, real, string`
- Tetszőleges/rögzített aritású Prolog kifejezés: `expr, expr(N)`, ahol `N` természetes szám.
- Adott típusú kifejezések listája, illetve lista, de a tagkifejezés egymagában is állhat: `list(TYPE), slist(TYPE)`
- Típusegyesítés (diszjunkció): `TYPE1; TYPE2`.
- Jegyszerkezet-mátrix (jegygeometria): `[FEATURE:TYPE||LIST]`, ahol `FEATURE` az adott jegy neve, `LIST` pedig a jegyszerkezet-mátrix folytatása, ami önmagában is jegyszerkezet-mátrix típusú, esetleg üres lista (`[]`).
- A típus futásidejű értékére vonatkozó extra Prolog feltétel: `TYPE:X^COND`.

Ehhez az általános leíráshoz képest még a következő *bővítéseket* és *nyelvtani könnyítéseket (syntactic sugar)* tesszük lehetővé:

- Ha egy jegy értéke szintén összetett, és a jegygeometriában megadott összes jegyet tartalmazza, akkor a jegynevek megadása nem kötelező, és a Prolog listakifejezés helyett kerek zárójelekkel *teljes Prolog kifejezés* is megadható. Pl. `agr:[pers:1, nr:sing]` helyett `agr(1,nr)` is írható.
- Azonos értékek (KIG összefutó élek) jelölésére (fordításidejű egyesítés) *Prolog változókat*, és a `=/2` funktort használjuk. Pl.: `PRED=desire(SUBJ,OBJ).`

A fordításidejű egyesítés mellett a `:=/2` funktorral a *jobboldal kiértékelésére és futásidejű egyesítésre* is lehetőséget adunk. Pl. a…

```
REF:=[argn(ord(-7,nei),cat(+2,noun),
        case(+2,nom)),
        argd(cat(+7,gqd))]
```

…kifejezés futásidőben egyesíti a `REF` Prolog változót a jobboldal kiértékeléséből származó kifejezéssel. A kiértékeléshez alkalmazói eljáráskampóként (kívülről) biztosítani kell a `:=/2` névjegyű Prolog eljárást, amely a jobboldal kiértékelését, és az eredmény egyesítését végzi a baloldali változóval.

2.5.6 Korlát-logikai programozás

A *korlát-logikai programozás (Constraint Logic Programming, CLP)* a logikai programozás egyik – sokak szerint legtöbbet ígérő – továbbfejlesztése. Korlátok – vagy talán kicsit érthetőbb magyarításban – egyes mennyiségek közötti megszorítások és összefüggések kielégítésének vizsgálata a számítástechnikai problémák között mindig is igen fontos helyen állt. Az ilyen jellegű célalkalmazásokat általánosabb, matematikailag is jobban megalapozott rendszerek követték.

A logikai programozás kialakulásának egyik ismert jelszava annak deklaratív jellege volt: a feladat puszta leírása alapján – kis túlzással – a rendszer maga találja meg a feladat megoldását. Vagyis a programkészítés „hogyanja” helyett a programozónak elég a „mit” kérdésre összpontosítania. Ha a deklaratív jelleg igaz volt az alapszintű logikai programozás esetében, akkor a CLP esetében ez még inkább igaz. Bár korlátmegoldó algoritmusokat sokféle programozási nyelvbe be lehet építeni, sőt ilyet akár önálló alkalmazásként is létre lehet hozni, talán éppen ez a deklaratív jelleg is az oka, hogy sokan az ilyen eszközöket egyértelműen a logikai programozási eszközök kiterjesztésének, egyfajta továbbfejlesztésének tekintik.

2.5.6.1 Korlát-probléma és megoldása

Egy általánosságban vett korlát problémát a következőkkel adhatunk meg:[64]

- logikai változók (ismeretlenek) egy halmaza
- minden egyes változóhoz egy alaphalmaz megadása
- egy megszorításhalmaz, amely az ismeretlenek által felvehető értékeket korlátozza

A korlát probléma eredménye a változókhoz történő érték-hozzárendelés. Egy változóhoz önmagában is rendelhetünk egyetlen értéket, vagy egy értékhalmazt.

[64] Szeredi Péter-Benkő Tamás: Nagyhatékonyságú logikai programozás [SzPBT02]

Értékhalmaz hozzárendelés esetén az adott változó az értékhalmazból tetszőleges elemet felvehet értékül a megszorítások teljesüléséhez. A változókészlethez pedig ilyen hozzárendelés-készlet egy halmazát is rendelhetjük, azzal a jelentéssel, hogy a halmaz bármelyik elemének behelyettesítése kielégíti az eredeti megkötéseket.[65] Az eredményt meghatározó programcsomagot, illetve algoritmust *megoldónak (solver)* is nevezzük.

Az egyes értelmezett alaphalmazoktól és a megszorítások megengedett megadási eszközeitől függően különféle korlát-megoldó rendszerek léteznek. Ha az érdeklődésünket a továbbiakban a logikai programozási alapokon létrehozott rendszerekre szűkítjük le, akkor egy ilyen rendszer a {***D***, ***F***, ***R***, ***S***} négyes szerkezettel írható le, amelyben:

- ***D***: egy értelmezési tartomány, amely a változók által felvehető értékek alaphalmazát adja meg. A gyakorlati rendszerekben nem engedik meg a változók tipizálását, azaz az alaphalmaz egyedi megadását.
- ***F***: a ***D*** felett megadott függvényjelek halmaza, amelyet a megszorítások megadásánál használhatunk.
- ***R***: a ***D*** felett megadott relációk halmaza, amelyet szintén a megszorítások megadásánál használhatunk
- ***S***: egy megoldó algoritmus, amely a ***D*** értelmezési tartomány felett és az ***F*** függvény, illetve az ***R*** relációjelek figyelembe vételével előállítja a korlát-probléma eredményét.

2.5.6.2 Korlát-logikai rendszerek és működésük

A korlát-logikai rendszerek a logikai programozás során használatos fogalmakat és a korlát-probléma fogalmait együttesen, keverve használják. A megoldás egyetlen G cél/korlát sorozatból indul ki, amelyben Prolog célok és korlátok vegyesen előfordulhatnak. A futás célja a Prolog célokban és a korlátokban található változók lekötési értékeinek egyidejű meghatározása úgy, hogy

T |– G

teljesüljön, vagyis hogy a G cél-korlát sorozat levezethető legyen a T elméletből. Míg azonban tiszta Prolog esetében a T elmélet megegyezik a tárolt Prolog program állításaival, addig a korlát-logikai alkalmazások esetében ehhez még hozzá kell venni a megoldó által alkalmazott matematikai elmélet állításait, vagyis azokat a tételeket és összefüggéseket, amelyeket a megoldó létrehozója a szoftver csomagba – burkolt módon – beleépített.

A korlát-logikai megoldó csomagokat CLP(ϰ) jellel jelölik, ahol ϰ az alaphalmazra és/vagy a rajta értelmezett megszorításokra vonatkozó jelzés. Megjegyezzük, hogy a létező megoldó rendszerek mellett a Prolog sajátmaga is egyfajta megoldónak tekinthető, amelyben a ***D*** alaphalmaz a logikai függvénykifejezések által meghatározott faszerkezetek halmaza, az ***F*** függvényjelek halmaza üres, az ***R*** relációjelek halmaza pedig az egyesítésre bevezetett kétargumentumú „=” Prolog predikátumból áll. Az ***S*** megoldó algoritmus maga a Prologban is használt egyesítési algoritmus. Mindezek felhasználásával, ha a célsorozatba a „=/2” predikátumszimbólummal képzett „logikai

[65] Roman Barták: Constraint Programming: in Pursuit of the Holy Gral [Bar99]

faegyenletet" írunk, akkor a rendszer válaszul megadja a megszorítások által kikényszerített változólekötéseket.

A piacon vagy kutatólaboratóriumokban elérhető CLP rendszerek legfőbb jellegzetessége az értelmezési tartománya. Ezek alapján a legelterjedtebbek a Q racionális számokon (természetes számpárokon), vagy R valós számokon működő CLP(Q) vagy CLP(R) megoldó csomagok, amelyek a kézenfekvő Gauss eliminációs algoritmust használják megoldóként. Emellett még léteznek a CLP(B) Boolean alaphalmazon, illetve a CLP(FD) véges alaphalmazon működő rendszerek, a Prolog listafogalmát halmaz, csomag és kompakt listafogalommal kiegészítő olasz {log} (ejtsd: Setlog) rendszer[66], [67], valamint Cliff Walinsky CLP(Sigma) rendszere[68], amely reguláris halmazokon működő megoldó. Külön érdekesség a SICStus Prologba (is) beépített, Thom Frühwirth által a müncheni egyetemen létrehozott, Constraint Handling Rules (CHR) nevű megoldó,[69] amely egy önálló, előrehaladó következtetést végző programozási nyelvként is tekinthető, a korlát-logikai megoldók általános működési módszereit alkalmazó rendszer. Ez egyfajta metarendszer, rögzítetlen alaphalmazzal, amelynek beprogramozásával más megoldókat is nyerhetünk, mint ahogy a rendszer a példaprogramjai között is közöl ilyen megoldásokat.

A Prolog rendszerek közül talán a SICStus Prolog[70] rendelkezik a legtöbb beépített CLP megoldó csomaggal. Mindenképpen megemlítésre érdemes még a CHIP rendszer, amely FD, Q és B megoldókat tartalmaz, és az ECRC müncheni kutatóintézet, valamint a Cosytec francia cég rendszereibe is be van építve, a PrologIA marseille-i cég Prolog III. valamint Prolog IV: rendszereibe beépített Q, B, és FD megoldók, melyek közül a Q a lineárisnál bonyolultabb egyenletrendszerek megoldására is alkalmas. Piaci célokra ezek közül a SICStus megoldóit, a CHIP rendszert és a Prolog III-IV rendszert használják. A szakma hatalmas gazdasági sikereket elért cége az ILOG, amely nemzetek feletti, maga 8 országban tart fenn irodát.

2.5.6.3 Tudásbázis kiegészítés. Zárt világ

Logikai programok az általuk leírt világ logikai modelljének is tekinthetők. Ez általában ugyan nem igaz a program teljes egészére, de jól megírt logikai programokban még mindig a teljes programszöveg meghatározó részét teheti ki a deklaratív, leíró jelleggel megírt programrész, a procedurális, vagy végrehajtható módon megírt rész mellett.

Az ilyen, leíró jellegű logikai programokban bevett szokás néhány olyan feltételezés megtétele, amelyek a tudásbázis logikai eszközökkel történő kiegészítésének tekinthetők.[71] Ezek a logikai eszközök az alkalmazott rendszerben általában nem elérhetők, ezért az alkalmazásuk a nyelv végrehajtása során a logikán kívül történik.

A legfontosabb ilyen eszköz a *zárt világ feltételezése (Closed World Assumption, CWA)*, amely a következő megállapítások alapján történik.

---

[66] Agostino Dovier-Alberto Policriti-Gianfranco Rossi: A uniform axiomatic view of lists, multisets and sets, and the relevant unification algorithms [DPR98]

[67] Agostino Dovier, Carla Piazza, Gianfranco Rossi: A uniform approach to constraint-solving for lists, multisets, compact lists and sets [DPR00]

[68] Clifford Walinsky: CLP(Sigma) [Wal89]

[69] Thom Frühwirth: Theory and Practice of Constraint Handling Rules [CHR94]

[70] M.Carlsson-J.Widen: SICStus Prolog 3.9. Users Manual [SICS02]

[71] Michael R. Genesereth-Nils J. Nilsson: Logical Foundations of Artificial Intelligence [GeNi87]

Egy logikai elméletet a logikai következményfogalomra nézve *lezárhatunk* úgy, hogy az elmélethez hozzávesszük azokat a tényállításokat, amelyek az elméletből levezethetők. Egy logikai elméletet *teljesnek* nevezünk akkor, ha minden lehetséges alapliterál (tényállítás), vagy a negáltja az elmélethez tartozik. Egy elmélet általánosságban véve többféleképpen is teljessé tehető.

Egy *elmélet zárt világ feltételezés melletti teljessé tétele* alatt értjük azt a műveletet, amikor:

- az elméletet a logikai következményfogalomra nézve lezárjuk, vagyis hozzávesszük az elméletből levezethető tényállításokat
- a lezárt elméletben nem szereplő alapliterálokat negatív literálként az elmélethez hozzávesszük

Bizonyos esetekben azonban nem szükséges a zárt világot az egész elméletre nézve feltételezni. A módszer egyik gyengítése, hogy csak *bizonyos predikátumokra* végezzük el a kiegészítést.

A zárt világ feltételezés tehát az elméletből le nem vezethető tényállításokról, vagy azok egy részéről feltételezi, hogy hamisak. Ezt a feltételezést használhatjuk pl. adatbázis-kezelő rendszerekben, amikor feltételezzük, hogy az adatbázisban, de legalábbis annak egyes részeiben az összes olyan adatot tároljuk, amely valamilyen halmazba beleesik. Ez bizonyos esetekben kézenfekvő is (pl. egy vállalat dolgozói esetében), más esetben azonban nyilvánvalóan hamis feltételezés (pl. az APEH állampolgári bevételeinek esetében). A Prolog nyelvben beépített a `\+ / 1` eljárás, amely zárt világban éppen ezt csinálja: ha a paraméterében megadott hívás nem bizonyítható be, akkor hamisnak veszi. Az eljárást negációnak, az egész módszert pedig *kudarcalapú negálásnak (Negation As Failure, NAF)* nevezik.

A zárt világgal szemben a *nyílt világot* feltételezi éppen a dolgozat tárgyát képező modellalkotási kérdéskör. Szoftver modellek készítése során ugyanis elvonatkoztatunk a tényleges példányoktól, és a leírtak nem egy-egy konkrét példányhalmazra, hanem a megkötéseket kielégítő bármely példányra vonatkoznak. Példányfüggő információk kezelésére sok esetben nincs is mód, vagy legalábbis hangsúlyozottan ellenjavallt.

Zárt világot feltételezünk a Prolog nyelvben, méghozzá leggyakrabban a tényállításokból álló predikátumok esetében. Pl. az alábbi tényállításokból álló predikátumok esetén feltételezhetjük, hogy törökverő vezérünknek több gyermeke egyáltalán nem volt.

```
apja(hunyadi-janos, hunyadi-laszlo).
apja(hunyadi-janos, hunyadi-matyas).
```

Vagyis a zárt világ feltételezése a logikailag egyenértékű

```
X= hunyadi-laszlo;X= hunyadi-matyas →
   apja(hunyadi-janos,X).
```

logikai állításhoz hozzáveszi a

```
\+ apja(hunyadi-janos, hunyadi-janos).
```

tényállítást. A fentieket általánosítva a következőt is írhatjuk

```
\+ (X= hunyadi-laszlo;X= hunyadi-matyas) →
   \+ apja(hunyadi-janos,X).
```

Az eredeti és az új állítások összevonásával kapjuk az

```
(X= hunyadi-laszlo;X= hunyadi-matyas) ≡
    apja(hunyadi-janos,X).
```

…állítást, vagyis a következmény a művelet hatására kiegészült a másik irányú párjával.

Megjegyezzük, hogy a zárt világ feltételezés ilyen, a következtetést egyenértékűségbe átvivő hatása csak a program, mint elmélet világegyetemében érvényes. Ez tényállítások esetében könnyen bizonyítható, de nem általánosítható.

2.6 Objektumorientált rendszerek és logikai modelljük

Ha az objektum-orientált programozási paradigma megjelenését a Smalltalk-80-tól számítjuk, akkor is több mint három évtizedes múltra tekinthet vissza. Az objektum-orientált technológia pályafutása diadalmenet volt – szabványok, eszközök százai születtek, amit több programozási nyelv is átvett, és amivel ma fejlesztők milliói dolgoznak. A technológia elterjedése egyfajta káoszt is eredményezett: ennek egyik következményeképpen a témában is felmerültek szabványosítási törekvések, amiből elsősorban az Object Management Group (OMG) és az Object Data Management Group (ODMG) az említésre méltók. Ez előbbi inkább egyes alkalmazási területek szabványos szoftver felületeinek leírására, az utóbbi pedig objektumorientált adatbázisok megvalósítására koncentrál. Az OMG tevékenységének egyik eredménye az általuk létrehozott UML szoftvertervezési módszertan, amelyet ma már számos CASE tervező eszköz is támogat.

A témában kiváló magyar nyelvű bevezető tankönyvek léteznek,[72] de ha mélyebb vagy pontosabb ismereteket kívánunk szerezni, akkor mindenképpen célszerű valamelyik szabványleírás[73] [74] beszerzése.

Az objektumorientált szoftverkészítésnek igen sok nyelvjárása létezik, nemcsak formai, hanem esetenként tartalmi-értelmezési különbségekkel is. Ezek a különbségek azonban – programozói gyakorlatból mondhatom – szinte mindig áthidalhatók, az áthidalás pedig legtöbbször könnyed módosításokat jelent, és csak igen-igen ritkán igényel izzadságos körbeprogramozást.

2.6.1 UML modellek és OCL megszorítások

Talán az egyetlen, ipari szabványnak is tekinthető objektum-orientált tervezési módszer az UML, amelyet a téma öregjei, Jim Rumbaugh, Ivar Jacobson és Grady Booch hoztak össze 1996-ban, az OMG égisze alatt. A módszer legfontosabb olyan jellemzői, amelyek a versenytársaikkal szemben előnyt biztosítottak számára:

- a módszer egyfajta szabványgyűjtemény, amelynek elemei egymást részben át is fedik. Az egész a használó számára szinte felmérhetetlen szabadsági fokot biztosít. Ezen szabadsági fokon belüli tájékozódás céljából a módszer alkotói, illetve más csoportok különböző módszertanokat is alkottak, amelyek az alapmódszert kiegészítve annak alkalmazására vonatkozó javaslatokat foglalnak magukba.
- A módszer a szoftver-tervezés folyamatát támogatja meg egy grafikus nyelv biztosításával. Ezzel a programozók és programtervezők képességeit gátló egyik

[72] Angster Erzsébet: Az objektumorientált tervezés és programozás alapjai [Ang97]

[73] Rumbaugh-Jacobson-Booch: Unified Modelling Language Reference Manual [UML99]

[74] R.G.G.Cattell-D.Barry és mások: The Object Data Standard: ODMG 3.0 [ODMG99]

igen régi akadályt bontják le: a használatos programnyelvek szöveges, vonalszerű, egydimenziós természetét a második dimenzióba, ábrákba is kiterjesztve, nagyméretű rendszerek lényegesen könnyebben áttekinthetők.

Az UML több mint fél tucat diagramfajtával siet a szoftvertervezők segítségére. A használatieset-diagramok, osztálydiagramok, állapotdiagramok, tevékenység-diagramok, szekvencia-diagramok, együttműködési diagramok, összetevő-diagramok és telepítési diagramok közül a jelen értekezésben azonban csak a leglényegesebbet, az osztálydiagramokat tárgyaljuk, mert ezekkel lehetséges egyes való világbeli fogalmak és azok összefüggéseinek a modellezése.

A továbbiakban ezen diagramfajta modellelemeinek az ismeretét – legalább alapszinten – adottnak feltételezzük. Ezen alapszintű ismeretkészletre építve tárgyaljuk – sajátmagát használva formalizmusként – az UML összetettebb és részletesebb fogalmait.

A jelen értekezés elsősorban *adatmodellek* tulajdonságait elemzi, ezért az osztálydiagramokban ábrázolható függvények és ezek viselkedése kissé háttérbe szorul.

Az Object Constraint Language (OCL) *megszorítás-leíró nyelv*[75] az UML osztálydiagramok grafikai nyelvének kiegészítése: olyan megszorítások is megfogalmazhatók vele, amelyek az osztálydiagramok eszközeivel nem lehetségesek. Mindazonáltal egy *pontosan meghatározott szemantikájú nyelv* is, amely miatt még a fenti grafikus megkötéseket és összefüggéseket is gyakran OCL segítségével adják meg pontosan. Bár a hajszálpontos meghatározottságba mégis csúsztak kisebb pontatlanságok, amire később még utalunk, de ezek az állítás erejét csak alig csorbítják.

Az OCL nem végrehajtható, hanem csupán *specifikációs nyelv*. Ezért logikai programvezérlés megadására alkalmatlan. Olyan modellkészítő nyelv, amivel csakis bizonyos programállapotok írhatók le. Éppen ezért az OCL-re vonatkoztatva mindenféle végrehajtási és értelmezési kérdés teljes mértékben érdektelen és értelmetlen, viszont ugyanezen okból felvethető a logikai eszközökkel történő kiértékelés és feldolgozás kérdése.

Az OCL *típusos objektumközpontú nyelv*, melynek a szerkezetei nagymértékben hasonlítanak más OO nyelvéire, ezért a legalapvetőbb szerkezetek semmiképp sem igényelnek különösebb elemzést vagy magyarázatot.

OCL kifejezéseket az UML osztálydiagramokban a következő helyzetekben használhatunk:

- *invariáns feltételek* megfogalmazásakor elsősorban osztályok és kapcsolatok, de általánosságban más modellelemek esetében is
- eljárások és függvények *elő- (pre) és utó- (post) -feltételeinek* megadására
- *eljárások és függvények* megadására
- mint *navigációs nyelv*
- *őrkifejezések* megadására, amelyek állapotdiagramokban az állapotátmenetekre vonatkozó feltételek megfogalmazására szolgálnak

Az OCL nyelv segítségével a használatos objektumorientált nyelvekhez hasonló navigációs szerkezetek és feltétel-kifejezések fogalmazhatók meg, amelyekkel a fent

[75] Damien Pollet-Didier Vojtisek-Jean-Marc Jezequel: OCL as a Core UML Transformation Language [OCL02]

említett programfordulatok leírhatók. Ezen túl a következő jellegzetességek emelhetők még ki:

- *Gyűjteménytípusok kezelése* alaptípusokként: a nyelv a `Set`, a `Bag` és a `Sequence` (halmaz, csomag, és sorozat) alaptípusokat rögzíti, amelyek rendre megfelelnek a Clark-féle gyűjtemény-axiómarendszer `Set`, `Multiset` és `List` típusainak,[76] vagyis míg halmazokban az egyenértékűség ellenőrzésekor az elemek sorrendje és többszörös előfordulásuk is közömbös, addig a csomagok esetén csak az elemek sorrendje közömbös. Sorozat alaptípusok esetében pedig csak pontosan ugyanolyan gyűjtemények tekinthetők egyenértékűnek, vagyis sem az elemek sorrendje, sem a többszörös előfordulásuk nem hanyagolható el.
- *Metainformációk elérése*. A nyelv `OclType` alaptípusán, annak tulajdonságain, valamint az ilyen eredményt adó egyéb függvényeken keresztül a modell típusainak az összefüggései maguk is lekérdezhetők.

A továbbiakban feltételezzük az OCL ismeretét – legalább alapszinten. Erre, mint specifikációs nyelvre erősen építeni is fogunk, sőt a szabvány OCL-t néhány nyelvi fordulattal (pl. konstruktorokkal) kiegészítve fogjuk használni, de mindez feltehetőleg az olvasó számára nem jelent majd komoly akadályt, mert a nyelv ismeretlenül is könnyen olvasható. Bár úgy gondoljuk, hogy a dolgozatban bevezetett újításoknak magától értetődőnek kell lenniük, mégis, egy későbbi fejezetben ezeket a biztonság kedvéért össze is foglaljuk.

2.6.2 Modellek és metamodellek

A modellalkotás a tudomány régi eszköztárához tartozik, ami a számítástechnikai kultúrában különös jelentőséget nyert. Ez ugyanis éppen a napi élet számtalan területén próbál a szellemi feladatok megvalósításához segédeszközt nyújtani. A modellalkotás egyfajta elvonatkoztatás, amelyben a vizsgált dolog vagy jelenség egyes részleteitől eltekintünk annak érdekében, hogy a jelenség megmaradó vonásainak tárgyalása révén a lényegre összpontosíthassunk. A modell általában már megértett analógiákra, a leggyakrabban matematikai szerkezetekre épül. A modellalkotás akkor sikeres, ha a megfigyelt dolgokból az analógiák felhasználásával és csak a modellre vonatkozó mozgási szabályok alkalmazásának útján olyan következtetéseket tudunk levonni, amelyek aztán a valósággal ismét összevetve is megállják a helyüket. A modellalkotás közben a következő legfontosabb szempontokat szokás figyelembe venni:

- Mi a feladat, mi a modellezendő objektumok köre, mi legyen a vizsgált *alaphalmaz (univerzum)*? A szűken vett feladatkiíráshoz a modellezés céljából a leggyakrabban még hozzá kell vennünk néhány dolgot, néhány jelenséget, néhány vonást, de ha ezek száma túlzottan megszaporodik, akkor aligha lehetséges a lényeget jól megragadó, tömör és egyszerű modellt építenünk.
- Mi a modellezendő világ részleteinek a szükséges *finomsága (granularitása)*? A túl durva modell nem írhatja le azokat a részleteket, amelyeket a feladat megkövetel, a túl nagy részletgazdagság viszont elfedi a vizsgált jelenség lényegét, és túlságosan elbonyolítja a modellt.
- Mivel a modellalkotás elvonatkoztatás, ezért annak elemei nem egyedi dolgok vagy objektumok, hanem azok halmazai: mégpedig olyan halmazai, amelyek a

[76] Agostino Dovier-Alberto Policriti-Gianfranco Rossi: A uniform axiomatic view of lists, multisets and sets, and the relevant unification algorithms [DPR98]

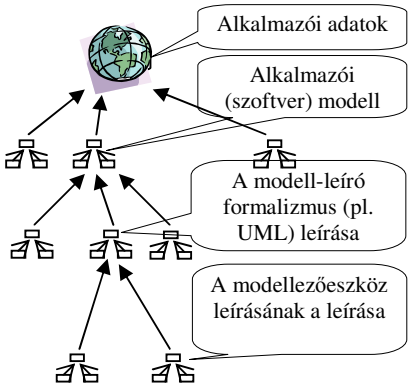
modell kikötéseit teljesítik. Egy modellnek életciklusa van, melynek során általában a modellezés finomsága, és a kikötések száma nő. A modellezés során sosem mondjuk, hogy a modell által nem megragadott dolgok nem is léteznek, így – néhány kivételtől eltekintve *–nyílt világot feltételezünk*.

Objektumorientált modellek készítésének alapelve a modell osztályainak és azok kapcsolatainak meghatározása. Ez a művelet nem más, mint az említett osztályozás, és elvonatkoztatás. Ennek során a világ egyes objektumaiból kiemeljük bizonyos lényeges és közös vonásaikat. A lényeges vonásokban közös objektumokat „egy osztályhoz tartozónak" tekintjük, a közös vonások leírását pedig osztályleírásnak nevezzük. Eközben elvonatkoztatunk az objektumok bizonyos lényegtelen vagy érdektelen vonásaitól.

Az *osztály* fogalom alatt tehát a világ azon objektumainak összességét értjük, amelyek az *osztályleírás* kikötéseit kielégítik. Az osztály egyes elemeit a *példányainak,* vagy egyszerűen *objektumoknak* nevezzük, függetlenül attól, hogy ezek ténylegesen léteznek (pl. a számítógépünk tárában) vagy csak létezhetnének, netán létezésük kívánatos is, ezért létrehozzuk őket. Egy osztályleírást tehát a példányai mintájának (sablonjának) is tekinthetünk. Magát az osztályleírást, illetve annak hivatkozását vagy azonosító nevét az adott példány *típusának* is nevezzük.

Ehhez kicsit hasonlóan, az osztálydiagram *kapcsolatai* is elvonatkoztatások, tényleges kapcsolati példányok összessége. Az osztályokat és a kapcsolatokat, valamint az osztálydiagramok eddig még nem részletezett elemeit együttesen *objektumorientált modellnek* nevezzük, míg a valójában létrehozott objektumok valamely hálózata pedig a *modell egy példánya*. Az elnevezés sugallja azt is, hogy a modellben általában nincsenek konkrét adatok, csupán azok elvonatkoztatás útján keletkezett osztályai, kapcsolatai.

A modellek egyes elemei azonban maguk is tekinthetők úgy, mint egy ősosztály, pl. a „modellElem" osztály példányai, amit, mint általános fogalmat tovább finomíthatunk osztályokra, kapcsolatokra, tulajdonságokra, eljárásokra stb. Azt az UML modellt, amely az UML modellelemeit, valamint azok összefüggéseit tartalmazza, az *UML metamodelljének* nevezzük. Az UML rendszer – fordítóprogramok bootstrapjéhez hasonlóan – megadható a saját metamodelljével, vagyis a metamodell grafikus, OCL-ben leírt vagy verbális leírásával. Ezért az UML legrészletesebb és legtökéletesebb leírása éppen a szerzők által készített ilyen leírás.[77]



9. ábra Az OMG négyrétegű metamodell szerkezete

Az UML szabványos metamodelljén kívül azonban egyéb, az UML-t különböző céllal leíró, ezért különböző metamodell-változatok hozhatók létre. A különböző metamodellek vagy változataik egységes leírására és ábrázolására azok modelljét használhatjuk, amelyet *meta-metamodellnek* is nevezhetünk. Természetesen egy rögzített meta-metamodell elemeivel nemcsak az UML, hanem más rendszerek (pl. programnyelvek) metamodellje

[77] Rumbaugh-Jacobson-Booch: Unified Modelling Language Reference Manual [UML99]

is leírhatók, amelynek így a példányai az egyes konkrét metamodellek. A meta-metamodelleket a különböző metamodellek leírásán túl *modelltárház* (Repository) jellegű, illetve különböző metamodellű rendszereken keresztülívelő alkalmazásokhoz használjuk.[78]

Az [alkalmazói adatok, alkalmazói modell, metamodell, meta-metamodell] négyest nevezik *négyrétegű metamodell szerkezetnek*. Bár a meta-metamodell fölé még tetszőleges számú metamodell-réteg elképzelhető lenne, egy ötödik réteg már – a rendszer megalkotói szerint – nem lenne eléggé kifejező.

Általánosságban szólva a rendszer n-edik rétegbeli modellje *példánya* egy n+1-edik rétegbeli modellnek. Itt a szerzők nem tartották szükségesnek rögzített metamodell-rendszer alkalmazását. Vagyis bármelyik rétegbeli modell példánya lehet több metamodellnek is. A meta-metamodellre az OMG ajánlást is ad,[79] ez azonban nem kizárólagos érvényű, más célokra esetleg más meta-metamodell is megadható.

Ha gyakorlatban bevált dolgokhoz matematikai modellt építünk, akkor ez a legritkábban sikerül ideális módon, úgy, hogy elegánsan tömör megfogalmazással a rendszert teljesen és tökéletesen lefedné. Ugyanilyen a helyzet az objektumorientált rendszerek matematikai logikai modellje esetében is. Az ilyen helyzetekben szokásos eljárás az elvonatkoztatás, vagyis az, hogy a modell a vizsgált világnak csak a túlnyomó és lényeges részét fedi le. A lefedett rész természetesen növelhető, de ez legtöbbször vagy újszerű modellezési eszközöket, vagy a modell mennyiségi növekedését, de legalábbis az elbonyolódását eredményezi.

Objektumorientált rendszerek és UML modelljeik matematikai logikai modellezéséhez a következő alapfeltevéseket kötjük ki:

- Logikai modellek alapvető tulajdonsága a csak *egyszeres értékadás*. Vagyis bizonyos értékeket, mennyiségeket logikai változókkal ábrázolunk, amelyek egy következtetési eljárás során csak egyszer kaphatnak értéket. Tilos tehát a változók felülírása. Minden olyan jellegű információt kizárunk tehát a modellkészítésből, amely ezeknek ellentmond.
- A modellezett világban az *eljárások*, és különösen a mellékhatással rendelkezők pontos *modellezése nehéz*. Ezért a következő fejtegetések általában az eljárások hatásainak elhanyagolásával kapott, úgynevezett *adatmodellre* érvényesek csupán.
- Az UML megengedi az egész *osztályra érvényes hatáskörű tulajdonságok* használatát is. A jelen értekezés ezek logikai modellezésétől is eltekint.

A következő fejezetekben az UML logikai modellezését olvashatjuk.

2.6.3 Osztályok és osztályszerkezetek

Egy objektum mindig egy osztályhoz tartozik, annak egy példánya. Az osztályból az objektumot példányosítással állítjuk elő, amely lehet korai, vagy késői. A *korai példányosítás* (korai típuslekötés) *fordításidejű*, a *késői* pedig *futásidejű objektum-előállítást* jelent.

---

[78] Joaquin Miller-Jishnu Mukerji: Model Driven Architecture OMG Document July-2001 [MDA01]

[79] Joint Revised Submission: Meta Object Facility (MOF) Specification [MOF97]

Az adott osztály egy programtípust is megad: az osztályból példányosítható lehetséges objektumok halmaza megegyezik az osztály típus-értékhalmazával.

2.6.3.1 Példányok megvalósítása. Osztály és példányspecifikus elemek.

Egy osztály példányosításakor létrejön annak a *példánya*, a megvalósítása. A megvalósítás a programmemóriában elhelyezett objektumokból áll. Ezek között vannak *osztályspecifikus* és *objektumspecifikus* adatok. Osztályspecifikusak azok, amelyek az osztály minden példánya számára közösen, csupán egyetlenegy példányban jönnek létre, és objektumspecifikusak azok, amelyek minden egyes létrejött objektumhoz egyedileg tartoznak.

Osztályspecifikus adatok a következők:

- Az osztály *műveleteinek (metódusainak) vektora*. Ez nem más, mint egy eljáráscím-ugrótábla, aminek a konkrét formátumát az adott gépi megvalósítás szabja meg.
- Az *osztályspecifikus (statikus) tulajdonságok vektora*. Ez egy adatrekord, amely egyetlen példányban jön létre minden egyes osztályhoz.

Példányspecifikus adatok a következők:

- A példányspecifikus *tulajdonságértékek vektora*. Ez egy adatrekord, amely minden egyes osztálypéldányhoz külön létrejön.
- A példányspecifikus adatok között tárolunk mutatót az osztályspecifikus információkra is:
    - a műveletvektorra mutatót *virtuális függvénytáblának* is nevezzük (vtable).
    - ugyanitt tároljuk az *osztálytulajdonságok vektormutatóját* is

2.6.3.2 Öröklés és elvonatkoztatás

Jelöljük $c(X_1,\ldots,X_n)$–nel azt a logikai állításelemet, ami azt fejezi ki, hogy az $X_1,\ldots,X_n$ tulajdonságértékekből alkotott n-es a „c” osztályhoz tartozik. Ha feltételezzük, hogy az $X_1,\ldots,X_n$ objektumtulajdonságok $t_1,\ldots,t_n$ típusúak, akkor az osztálydefiníció a következő elsőrendű logikai állításnak felel meg:

$$t_1(X_1),\ldots,t_n(X_n) \equiv c(X_1,\ldots,X_n)$$

A meghatározás szerint egy osztálypéldány minden esetben tartalmazza a tulajdonságértékeket, amelyek mindig a kikötött típusúak. A másik logikai irány miatt, ha adottak a kikötött típusú elemek, akkor az mindig meghatároz egy objektumpéldányt is – ha ilyen még nem létezne, akkor ez a nyílt világ miatt legalábbis létezhetne, és létre is hozható.

Mindez úgy is érthető, hogy az osztálydefiníció – az osztályképző predikátumon (itt: c) keresztül – a tulajdonságtípusok halmazainak direktszorzatát állítja elő. Az osztálydefiníció, mint halmazmegadás tehát nem konstruktív, hanem csupán a halmazhoz tartozás peremfeltételeit fejezi ki.

Az osztálymegadás a példányok közös jellemzőinek kiemelése, és elvonatkoztatás az egyedi jellemzőktől. Ezt az elvonatkoztatási műveletet az osztály-példány viszonyon túl is folytathatjuk úgy, hogy egyes osztályok bizonyos közös vonásait általánosítjuk, kiemeljük. Ilyen módon létrehozhatjuk az osztályok *ős-* vagy *szülő-osztályát*. Az

ellenkező irányú viszony, vagyis konkrétabb osztályok levezetése esetén a létrejövő osztályokat *részosztályoknak, utód-osztályoknak* vagy *leszármaztatott* osztályoknak nevezzük.

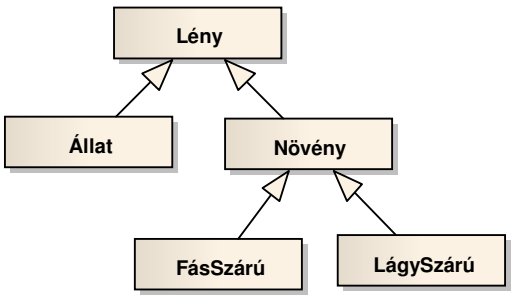
A legáltalánosabb osztályokat, vagyis azokat, amelyeknek nincsen ős-osztályuk, (de vannak részosztályaik), *alaposztálynak* nevezzük. A tényleges szóhasználat azonban ennél kicsit pongyolább: gyakran alaposztálynak neveznek olyan osztályokat is, amelyek „eléggé" alapszintű dolgokat írnak le, de esetleg mégis van további ős-osztályuk is.

Az osztály-leszármaztatást a legkézenfekvőbben *részhalmaz*-viszonnyal írhatjuk le: a részosztályok típus-értékhalmaza részosztálya az ősosztályénak.

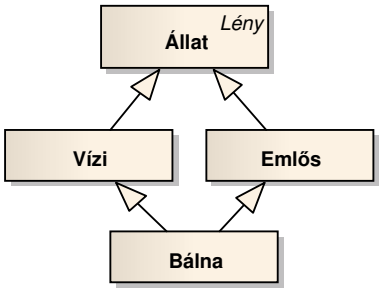
Azokat az osztályokat, amelyek eléggé konkrétak ahhoz, hogy példányaik lehessenek, *konkrét osztálynak* nevezzük. A fogalom ellentéte: a nem konkrét osztályokat, amelyeknek példányai sem hozhatók létre, *absztrakt osztályoknak* nevezzük.

A két fogalom közti megkülönböztetés nem éles: egyes rendszerek csak olyan osztályokat tekintenek konkrétnak (és csak ezekhez tartozó példányok létrehozását engedik meg), amelyeknek további alosztályai már nincsenek. Más rendszerekben ez a megszorítás lazább.

Egy szoftver-rendszer osztályait és azok leszármaztatási viszonyait együttesen *osztályszerkezetnek* (osztályhierarchiának) nevezzük.



10. ábra Fa osztályszerkezet

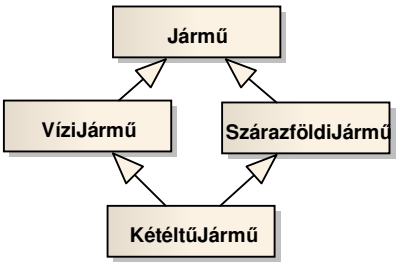


11. ábra Többszörös öröklődés közös ősosztállyal

Egy osztály részosztályaiban csak annak az ős-osztálytól eltérő jellemzőit adjuk meg, mert az osztály örökli az összes ős-osztályában megadott jellemzőit. Ha a részosztály leírásában *új jellemzőket* adunk meg, akkor ezekkel *kiterjesztjük* az eredeti osztályleírást. A részosztályképzésnek lehet célja *újabb megszorítások* bevezetése is, vagyis az eredeti osztály *korlátozása* egy részhalmazára. Ez történhet az ős-osztályban már meglévő jellemzőnek adott *új értékkel*, vagy ilyen *eljárás újbóli megadásával*, de természetesen *újabb OCL megszorításokkal* is.

Ha leszármaztatott osztályban tulajdonságot vagy eljárást ismét megadunk, akkor azt mondjuk, hogy *felülírjuk* (overriding) az ősosztályban rögzítetteket.

Egy adott osztály bizonyos esetekben több ős osztály jellemzőit is örökölheti. Az ilyen szerkezeteket *többszörös öröklődésnek* nevezzük. Többszörös öröklődés esetén előfordulhat, hogy egy osztály ugyanattól az ős-osztályától nemcsak egyetlen, hanem több öröklődési úton, több irányból is örököl. Az ilyen osztályokat *közös ősnek*, vagy *közös alaposztálynak* nevezzük. A fenti példában szereplő (`Jármű-KétéltűJármű`),



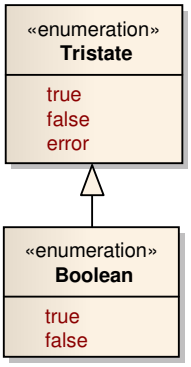
12. ábra Gyémánt formájú többszörös öröklődés

és ahhoz hasonló öröklési szerkezeteket szokásos *gyémánt formájú (diamond) öröklési mintának* is nevezni.

Többszörös öröklődést nem megengedő rendszerekben az osztályszerkezet *topológiailag mindig fa*. Ez lehet egybefüggő, ha a rendszer minden osztálya egyetlen alaposztályból származtatható, vagy lehet széteső erdő akkor, ha „lapos" osztályszerkezetet terveztünk kevés öröklődéssel. Ha többszörös öröklődést is értelmező rendszerünket teljesen kihasználjuk, akkor az osztályszerkezet topológiája *irányított körmentes gráf* (bár a körmentességre vonatkozó ellenőrzést az egyes fordítóprogramok nem feltétlenül tartalmaznak).

2.6.3.3 Öröklődés felsorolástípusok között

13. ábra Felsorolások



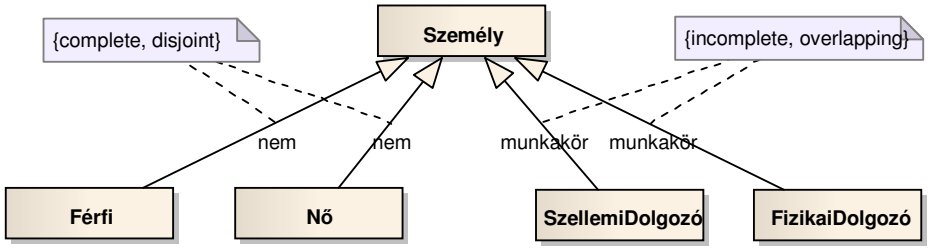
Felsorolások esetében az, hogy a felsorolás elemeit a grafikus megjelenésben osztálytulajdonságokként adjuk meg, logikailag teljesen megalapozatlan. A felsoroláselemek a felsorolás-osztálynak ugyanis nem összetevői, hanem éppen a példányai. Az osztálytulajdonság-szerű ábrázolásmód kizárólag a szemléletesség kedvéért lett választva.

Az UML szabvány nem említi a felsorolástípusok közötti öröklődés lehetőségét, vagyis nem is tiltja, de nem is bátorítja. Az öröklődésnél célszerű továbbra is a részhalmaz-elvet követni, vagyis a több elemű felsorolást választani alaposztálynak, és a kevesebb eleműt alosztálynak úgy, hogy az alosztály elemei az alaposztály elemeinek a részhalmaza legyen. Ez viszont azt is jelenti, hogy a felsorolás-elemekre a szokásos tulajdonság-öröklődési elképzelés nem igaz. Vagyis a felsoroláselemek éppenséggel a konkrétabbaktól az általánosabbak felé öröklődnek. A gyakorlat szempontjából a legszerencsésebb a felsoroláselemek teljes megjelenítése az UML osztálydiagramokon.

2.6.3.4 Leszármaztatás diszkriminátorral

Ha egy ősosztályból újabb osztályokat származtatunk le, az tehát egy részhalmaz viszonyt jelez. Ha nagyobb számú leszármaztatott osztály is van, akkor felmerülhet a leszármaztatott osztályok egymás közötti, illetve az ősosztály és a leszármaztatott osztályok együttesének viszonya. Az is előfordulhat, hogy ugyanazon ősosztályból kiinduló leszármaztatási viszony-kötegek, ha több is van, különböző dimenzió mentén, különböző szempont alapján végeznek leszármaztatást. Ha ilyenkor azon leszármaztatási viszonyokat, amelyek ugyanazon szempont alapján történnek, a szempont nevével megcímkézzük, akkor az *azonos szempontú és címkéjű leszármaztatási viszonyok halmazát diszkriminátornak* nevezzük. A diszkriminátort a címkézés mellett még különleges megszorításokkal is elláthatjuk, amelyek a következők lehetnek:

`complete` *teljes leszármaztatás*. Az alaposztály bármely példánya mindenképpen példánya valamelyik részosztálynak, az alaposztálynak nincs olyan példánya, ami ne lenne példánya valamelyik részosztálynak. Az ábrán a `Férfi` és `Nő` osztályok fedik le teljesen a `Személy` osztályt.



14. ábra Leszármaztatás diszkriminátorral és megszorítással

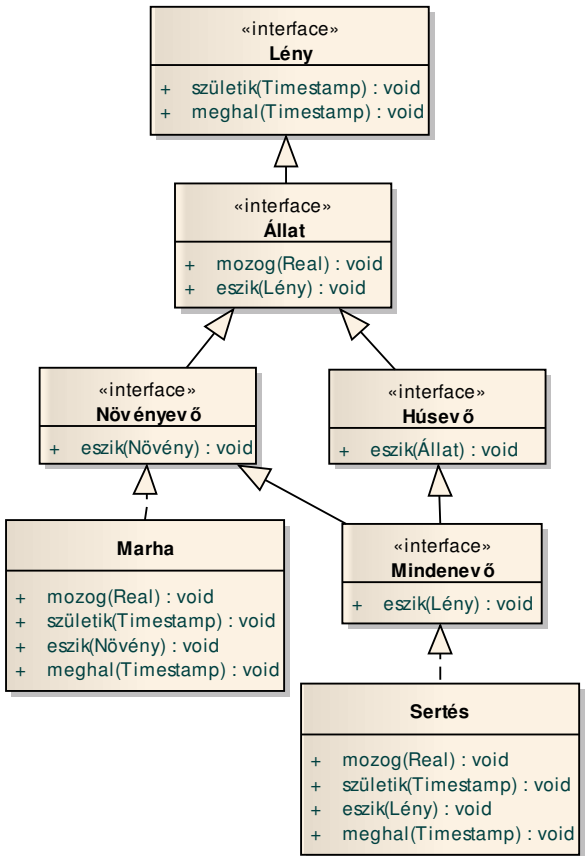
`incomplete` *hiányos leszármaztatás* – a `complete` ellentéte. Azt jelzi, hogy az alaposztálynak lehet olyan példánya, amely egyetlen részosztálynak sem példánya. A fenti ábrán a `SzellemiDolgozó` és a `FizikaiDolgozó` részosztályok ilyenek: a `Személy` osztályt teljesen biztosan nem fedik le teljesen:

`disjoint` *egymást kizáró részosztályok.* A megszorítást közös alaposztállyal rendelkező általánosítási viszonyokra alkalmazhatjuk. Azt fejezi ki, hogy az alaposztály bármely példánya legfeljebb egyetlen részosztálynak lehet eleme. Az ábrán a `Férfi` és `Nő` részosztályok ilyenek: teljesen biztosan nincs olyan `Személy` példány, amely egyszerre `Férfi` és `Nő` is lenne.

`overlapping` átlapolódó részosztályok – a `disjoint` ellentéte. A részosztályok egymást nem zárják ki, lehetséges olyan példány, amely több részosztálynak is eleme. A fenti példában megint a `FizikaiDolgozó` és a `SzellemiDolgozó` jó példa erre: lehetséges olyan `Személy` példány, aki egyszerre mindkét alosztályhoz tartozik.

2.6.3.5 Absztrakt osztályok és felületek

A szoftver tervünkbe felvehetünk olyan osztályokat is, amelyeket nem példányosíthatunk. („`=0`”, tiszta virtuális függvényt tartalmazó osztály C++-ban, illetve az `abstract` módosító a Javában). Az ilyen osztályokat *absztrakt osztályoknak* nevezzük. Az absztrakt osztályok szerepe az, hogy az osztály elemeit (tulajdonságait és eljárásait) egységbe zárva szabványosítsuk: az absztrakt osztályok megvalósítása során egyes eljárásait meg is valósíthatjuk, ezekre vonatkozólag a leszármaztatott osztályokban egységes megvalósítást kötünk ki. Más eljárásoknak azonban csak a *névjegyét (signature)* rögzítjük, azokat a leszármaztatott osztályokban kell megvalósítanunk.



15. ábra Többszörösen is öröklődő felületek megvalósítása

Egy *eljárás névjegye* alatt az eljárás nevét, paramétereinek számát, és a paraméterek típus-vektorát értjük: az objektum-központú programnyelvek megengedik, hogy ugyanolyan nevű, de más névjegyű eljárásokból többet is megadjunk. Ezt a viselkedést az *eljárásnév túlterhelésének (overloading)* is szokás nevezni.

Az absztrakt osztályfogalom még további elvonatkoztatásával kapjuk a *felület (interface)* fogalmát. A felület szintén hasonlatos az absztrakt osztályokhoz, mert nem példányosítható, és az egész specifikációs és szabványosítási célokat szolgál. Viszont a felületek nem tartalmazhatnak sem adattagokat, sem a felületekből kifelé navigálható kapcsolatot, így a felület csupán egy eljárás-csomagot jelent.

A felület tehát egy *eljárás-specifikációs csomag*, amelyet valamilyen szoftver egységek megvalósítanak. Az UML-ben a megvalósító lehet egy osztály, de egy szoftver összetevő is, bár egyes programnyelvek csak osztályokat engednek meg.

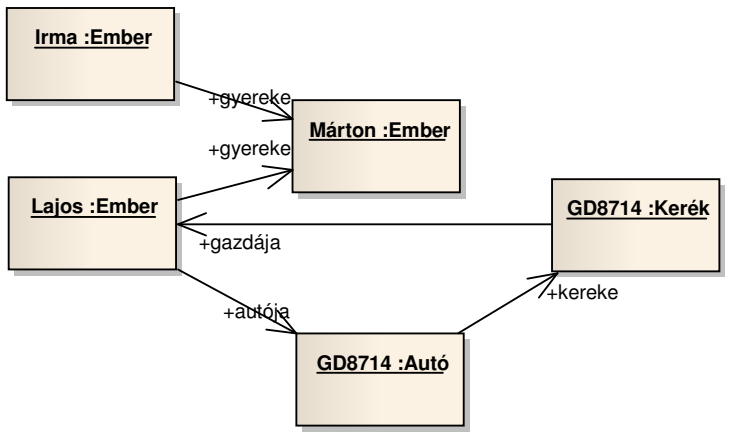
A *felületek öröklődhetnek* is: ha valaki egy leszármaztatott felületet kívánna megvalósítani, akkor a leszármaztatási lánc felületeihez rendelt eljárás-definíciók halmazai egyesülnek. Így a felületek öröklődése a megvalósítás szemszögéből egyenértékű azzal, ha a megvalósító osztály több felületet (az öröklési lánc összes felületét) meg akarná valósítani.

Konkrét osztályok a felületeket *megvalósítják (realization)*, ezt az UML-ben az örökléshez hasonló, de szaggatottan szedett nyíllal jelöljük.

Azokban a rendszerekben és programnyelvekben, amelyek kizárják a többszörös öröklést (pl. Java), ahhoz hasonló működésmódot a felületek alkalmazásával lehet elérni (ld. az ábrán). Az ilyen rendszerek esetleg mégis támogathatják a többszörös öröklést felületek között. Amennyiben egyes rendszerek még ezt sem támogatják, a felületek közötti öröklődést a megvalósítandó osztályban az öröklési lánc minden egyes elemétől való, összegzett örökléssel, illetve megvalósítással helyettesíthetjük.

2.6.4 Objektumok kapcsolatai

2.6.4.1 Objektumhálók



16. ábra Objektumháló

Az objektum-orientált rendszerek által kezelt (egyszerű és összetett) számítógépes objektumokat *objektumhálónak* nevezzük. Az objektumhálók egyedi objektumokból (példányokból) állnak, amelyek egymással is össze lehetnek kapcsolva. Az objektumok és az objektumhálók fogalma rekurzív módon egy halmazt határoz meg, melynek elemei egy-egy adott program adott pillanatában előálló objektumpéldányai, illetve a program teljes objektumhálója, vagy annak egy része. Az egyedi objektumokat egyes helyeken objektumpéldánynak, ott pedig, ahol ez nem érthető félre, egyszerűen objektumnak fogjuk nevezni.

Ezek alapján egyedi objektumok a következők:

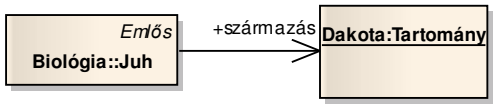
1. *Egyszerű objektumok*, vagyis egész számok, valós számok, szövegek, esetleg dátumok vagy pénzegységek
2. Objektumszerkezeteket felépíthetünk egy szövegesen megcímkézett csomópont és az abból *kifutó élek által megcélzott objektumok* segítségével. A kifutó élek szintén megcímkézhetők a sorszámukkal vagy szöveg címkékkel.
3. Az objektumszerkezetnek lehetnek *összefutó élei* is, vagyis több csomópontból kifutó él is befuthat ugyanabba a csomópontba.
4. Az objektumszerkezet tartalmazhat *körjáratokat* is, vagyis valamely csomópontból indulhat él olyan csomópont felé is, ahonnan az eredeti él közvetlenül vagy közvetve elérhető.
5. A fenti objektumokból összeállított véges halmazok *objektumszerkezetet* képeznek, amelynek felépítését egy program adott lefutásának adott időpontja határozza meg.

A mellékelt példában „Márton” egy olyan Ember, aki két másik emberhez is tartozik, vagyis itt összefutó élekkel találkozunk. A körökre a példa a GD8714:Kerék, amelyből

közvetlenül navigálhatunk magára a tulajdonosra „Lajos:Ember”objektumra is. Ennek a közvetlen navigációnak a megtiltásával körmentes hálózatot kapnánk. Ha az elemi navigációt nem irányítjuk, (nem rajzolunk nyílhegyet a végére), akkor azt mondjuk, hogy a navigáció kétirányú, vagyis mindkét végobjektumból lehetséges a másikra lépni. Az irányítatlan navigációs él egyben a legegyszerűbb kör is: ha a példában a „GD8714:Kerék” és „GD8714:Autó” közötti navigáció irányítását megszüntetjük, akkor egy kételemű körjáratot kapunk. (Az ábrán a navigációs címkéket megelőző „+” jelzés csupán annyit jelent, hogy a navigáció nyilvános, vagyis az adott osztályon, sőt, a csomagon kívül is használható.) A megegyező azonosító „GD8714” használata nem igazán jellemző, de megengedhető olyan osztályok példányai között, amelyek közös elemeket nem tartalmaznak.

Az objektumhálók mellett bevezethetjük a *lekérdezési hálók* fogalmát is. Ez az osztálydiagramok és az objektumhálók közötti átmeneti állapotként nemcsak konkrét példányokat, hanem osztályokat is tartalmazhat, amely ismeretlen osztálypéldányt jelentene. Az effélék egy vagy több konkrét példányra illeszkedhetnének. Az objektumhálókhoz hasonlóan itt is feltehető a kérdés, hogy egy lekérdezési háló egy adott osztálydiagramból származtatható-e, annak példánya-e.

A névadás indítéka: egy későbbi fejezetben tárgyaljuk a lekérdezési hálók alkalmazhatóságát objektumorientált modellek és adatbázisok feletti lekérdezések megfogalmazására.



17. ábra Lekérdezési háló

Mind az objektumhálók, mind a lekérdezési hálók pontos felépítését a metamodelljükkel a legszerencsésebb megadni, amely szintén egy későbbi fejezet tárgya.

2.6.4.2 Kapcsolatok

Különböző objektumok egymással a már említett navigációs viszonyban állhatnak. Ha két objektum efféle kapcsolatban van egymással, akkor azt mondjuk, hogy az osztályaik kapcsolatban vannak egymással. A két konkrét, kapcsolatban levő objektumot együtt a *kapcsolat példányának*, vagy *kapcsolatelemnek (link)* nevezzük.

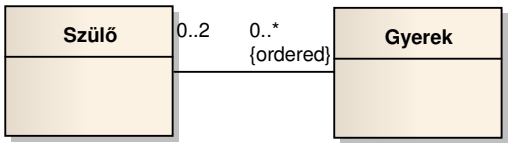
A kapcsolatok által meghatározott halmaz az osztályokhoz igen hasonlóan értelmezhető. Egy kapcsolat nem más, mint a kapcsolatvégek típus-értékhalmazai közötti matematikai reláció: egy kapcsolat tehát a kapcsolatvégek típus-értékhalmazai direkt szorzatának egy részhalmaza. Ha $a(X_1,\ldots,X_n)$-nel jelöljük azt az állításelemet, ami az $X_1,\ldots,X_n$ kapcsolatvég-objektumokra, – illetve kapcsolóosztály esetén annak tulajdonságaira – az „a” kapcsolathalmazhoz tartozásukat fejezi ki, és feltételezzük, hogy az $X_1,\ldots,X_n$ objektumtulajdonságok rendre $t_1,\ldots,t_n$ típusúak, akkor a kapcsolatdefiníció a következő elsőrendű logikai állításnak felel meg:

$$t_1(X_1),\ldots,t_n(X_n) \equiv a(X_1,\ldots,X_n)$$

Ez a definíció még semmi egyebet nem rögzít, mint azt, hogy mit nevezünk matematikai relációnak: a kapcsolatot finomabban a kapcsolat egyes megszorításai és többszörössége segítségével lehet megadni.

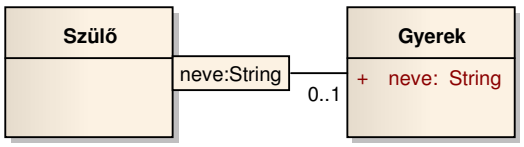
A legáltalánosabb *társítási kapcsolatok (asszociációk)* esetében a kapcsolatot általánosságban, kettő vagy több osztály között határozzuk meg. Az ilyen kapcsolatok a következő további módokon (metatulajdonságokkal) jellemezhetők:

- *navigálási irány*: amelyet a vonalra tett nyíllal jelölünk (ha a nyílhegy hiányzik, akkor mindkét irányban navigálhatunk). Ez azt jelenti, hogy a futó szoftverben csak a nyíl szerinti irányban képes az egyik objektum a másikat elérni: a nyíllal szemben haladva az elérés nem lehetséges.
- a *kapcsolat neve*: amely a teljes kapcsolatot jellemzi,
- illetve a *szerep neve*, vagy *navigációs címke*, amely csak a vonalak egyes végeit jellemzi. Ez adja meg, hogy a másik kapcsolatvégen található objektumpéldányból milyen címkével érhető ez az aktuális kapcsolatvég objektum.



19. ábra Többszörösségek egy társítási kapcsolatban

- A kapcsolat *többszörösségi foka (multiplicity)*: amely egy arra vonatkozó megszorítás, hogy egy objektumpéldány a kapcsolat alapján hány másik objektummal lehet ugyanilyen viszonyban.



18. ábra Minősített társítási kapcsolat

- 1-nél nagyobb többszörösségek esetén használható a rendezettséget megadó `{ordered}` jelzés, vagy a kettőzött példányokat megengedő `{sequence}` ami formailag egy különleges megszorítás (ld. a megszorításoknál). Az `{ordered}` jelentése: a kapcsolt példányok, sorba vannak rendezve, és a sorszámukkal megcímezhetők. A `{sequence}` esetében pedig ugyanaz az objektumpéldány a kapcsolt vektor több helyén, kettőzve vagy megtöbbszörözve is megjelenhet.
- Szintén az 1-nél nagyobb többszörösségek esetén szokásos a *minősítő (qualifier)* használata. Ezt egy eredetileg 1..n-es kapcsolat 1-es oldalához rendeljük, és azt fejezi ki, hogy a minősítés egyértelműsíti azt, hogy a baloldali objektumhoz a több jobboldali közül melyik van rendelve. Ezt futásidőben egy, a minősítő tulajdonsággal indexelhető gyűjteménnyel valósíthatjuk meg.

2.6.4.3 Osztott és kizárólagos birtoklás

A fent leírt általános kapcsolati szemantika mellett még a következő megszorításokat tehetjük:

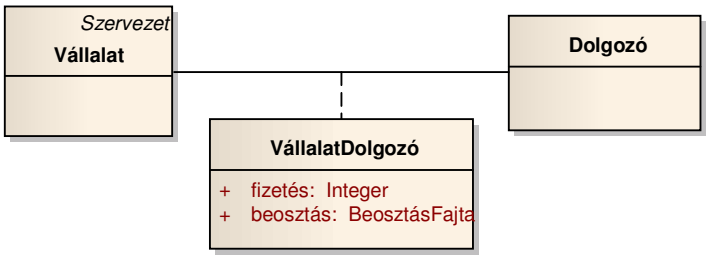
- *rész-egész kapcsolat (aggregáció)*: Ilyen kapcsolat akkor jön létre, ha az egyik objektum fizikailag tartalmazza és birtokolja a másikat, de ez a viszony nem kizárólagos, vagyis a birtok megosztható. A rész-egész kapcsolat mindig

kétoldalú, tranzitív és antiszimmetrikus, vagyis használatának fontos kikötése, hogy a rész nem tartalmazhatja az egészet – senki nem tartalmazhatja sem közvetlenül, sem közvetve önmagát. Vagyis az efféle kapcsolatok alapján felépített *objektumháló nem tartalmazhat kört*. A rész-egész kapcsolatot az UML-ben a kapcsolatot ábrázoló folytonos vonallal, és az egész-oldalon egy üres rombusszal jelöljük.

- Kizárólagos birtoklás, vagy más néven *objektum-összetétel (kompozíció)* esetén a birtok nem lehet másokkal közös. Ilyenkor a birtok mindemellett a birtokostól függ, nélküle nem létezhet, annak létrejöttekor vagy utána jöhet létre, és legkésőbb annak megszűntekor, vagy már korábban megszűnik. Amennyiben *erős összetételről* van szó, akkor ez a birtoklási viszony az objektumok élete során nem is változtatható vagy módosítható. *Gyenge összetétel* esetén a birtok „elkérhető" a birtokostól. A tartalmazási kapcsolat a rész-egész kapcsolat megszorítása azzal, hogy a birtok nem megosztható. Ezért a tartalmazási kapcsolat alapján felépített objektumháló topológiailag mindig körmentes, sőt, összefutó élektől is mentes hálót, vagyis *fát feszít ki*. A szigorú tartalmazási kapcsolatot ugyanúgy jelöljük, mint a gyenge összetételt, de az egész oldalon található rombusz tele.

2.6.4.4 Kapcsolóosztályok

Ha egy kapcsolatelemet – egy kétoldalú kapcsolat, mint párok halmazának egy elemét, egy konkrét párt – a kapcsolat végein, vagyis a pár két tagján kívül más tulajdonságok is jellemzik, akkor ez közönséges kapcsolatként nem modellezhető. Ilyen esetben úgynevezett kapcsolóosztályként modellezzük a kapcsolatot. Ebben éppen a kapcsolatelemekre vonatkozóan vehetünk fel igény szerinti tulajdonságokat.



20. ábra Kapcsolóosztály

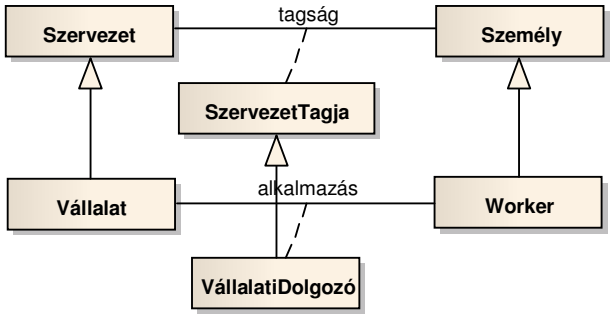
A fenti példa bemutatja mindezt. Egy vállalatnak általában egynél több dolgozója van, és egy dolgozónak éppenséggel egynél több munkahelye is lehet. A dolgozó fizetése és beosztása azonban olyan adat, amely sem a vállalatot, sem a dolgozót önmagában nem jellemzik, hanem csak az adott vállalat-dolgozó párt, azaz a kapcsolatelemet. Az ábrán a kapcsolóosztály a kapcsolattal a szaggatott vonal köti össze.

2.6.4.5 Öröklődés kapcsolatok között

Az UML szabvány kapcsolatok között sem zárja ki az öröklődést, bár ez a lehetőség közvetlenül meg sincs említve. A kapcsolatok közötti öröklődés értelmezése a kapcsolatot megadó halmazok közötti részhalmazviszony alapján, technikailag pedig a kapcsolóosztályaik közötti öröklődés alapján a legkézenfekvőbb. Eszerint minden kapcsolat megfelel egy kapcsolóosztálynak, legfeljebb az nincs külön jelölve. A külön

jelölés hiánya arra utal, hogy a kapcsolóosztálynak a két kapcsolatvégen túl egyetlen tulajdonsága sincs. Egy *leszármaztatott kapcsolat* az ősével akkor és olyan viszonyban áll, mint amilyenben a kapcsolatokhoz rendelhető kapcsolóosztályok, és a logikai viszonya is annak megfelelően alakul, egyetlen kivétellel.

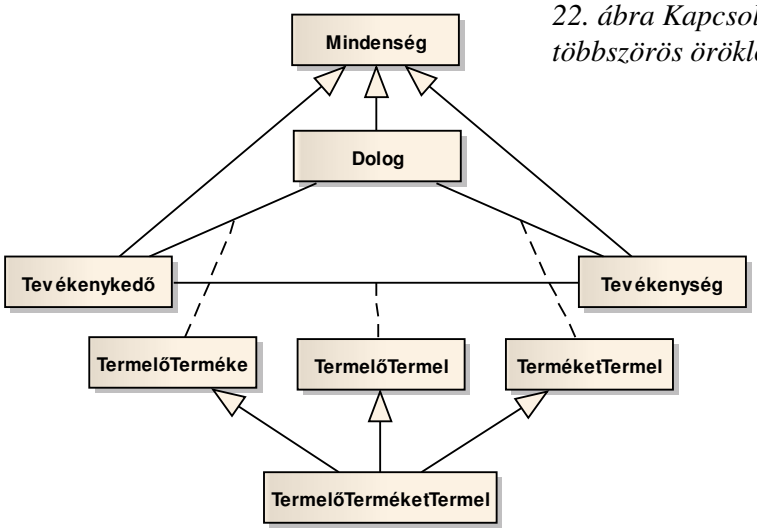
A leszármaztatott kapcsolóosztályhoz nem rendelhetők újabb kapcsolatvégek. Vagyis egy kapcsolatból egyszerű öröklődés útján nem származtatható egy több végű kapcsolat, tehát a kapcsolatba újabb kapcsolatvég nem vonható be.



21. ábra Egyszerű öröklődés kapcsolatok között

Kapcsolatok leszármaztatásakor is igaz mindaz, ami osztályokra is: a leszármaztatott kapcsolat részhalmaza az alapkapcsolatnak, számossága nem nagyobb. Ennek egyik oka az, hogy a leszármaztatott kapcsolatvégek az alapkapcsolat végeinek leszármaztatott osztályaira mutatnak. Másik ok lehet, ha új megszorítás vonatkozik a kapcsolóosztályra, akár tulajdonságérték-megkötés, akár új OCL megszorítás miatt.

Nem tiltja meg azonban a szabvány a kapcsolóosztályok közötti többszörös öröklődést. Vagyis ilyen módon megnövelhető a kapcsolatvégek száma is. Az eredményként kapott kapcsolóosztály példányai – az osztályokhoz hasonlóan – az ős osztályaik metszete, vagyis a számossága nem lehet egyik ős osztályénál sem magasabb.



22. ábra Kapcsolatok közötti többszörös öröklődés

2.7 Szoftver-rendszerek, metamodelljük és meta-metamodelljük

A metamodell[80] fogalmát úgy érthetjük meg a legkönnyebben, ha egy olyan szoftverre gondolunk, amely modelleket kezel (pont ilyenek a CASE eszközök). Az ilyen szoftverek készítését úgy kezdjük, hogy elkészítjük a szoftver által kezelt adatok szakterületi modelljét, vagyis a modellek modelljét, a metamodellt. Lényegtelen részletkérdésnek számít az is, hogy az adatként kezelt modell milyen formalizmust képvisel – ha UML-t, akkor a metamodell az UML egyes elemeit, pl. osztálydiagramok esetében az osztályokat, csomagokat, attribútumokat, műveleteket, és természetesen a többi diagramfajta esetében az ott alkalmazott modellelemeket tartalmazza.

Szintén részletkérdés az is, hogy a szakterületi modell elkészítéséhez milyen formalizmust használunk. Kézenfekvő az UML alkalmazása, de az egyes programnyelvek szintén az UML leképezései (és gazdagításai). Kérdés lehet még az is, hogy az UML milyen elemeit használjuk a modellezéshez - erre az egyértelmű válasz: az osztálydiagramokat.

Az ilyen módon elkészített metamodell tartalmában már független az alkalmazói program elemeitől és főleg annak adataitól. Amint egy program szerkezetének függetlennek kell lennie a konkrét felhasználói adatoktól – és maga az alkalmazói modell sem tartalmaz adat-/példányfüggő dolgokat, ugyanúgy a CASE szoftvernek minden lehetséges modellt tökéletesen kell tudnia kezelni. Lehetséges: itt azt jelenti, hogy minden olyan modellt, amely a metamodell példánya, mivelhogy az általa megszabott korlátozásokat követi.

Metamodellek elkészítése tehát elengedhetetlen akkor, ha a modelleket adatként kezelő szoftvert készítünk. Ezen túl azonban igen hasznos akkor, ha egy rendszert meg akarunk ismerni. Metamodellje minden olyan rendszernek és nyelvnek van, amely maga is modelleket kezel: így adatbázis-kezelőknek, programozási nyelveknek, más adatleképező eszközöknek, szoftvermodellező eszközöknek. Tekintsük az alábbiakban néhány jellegzetes eszköz metamodelljét.

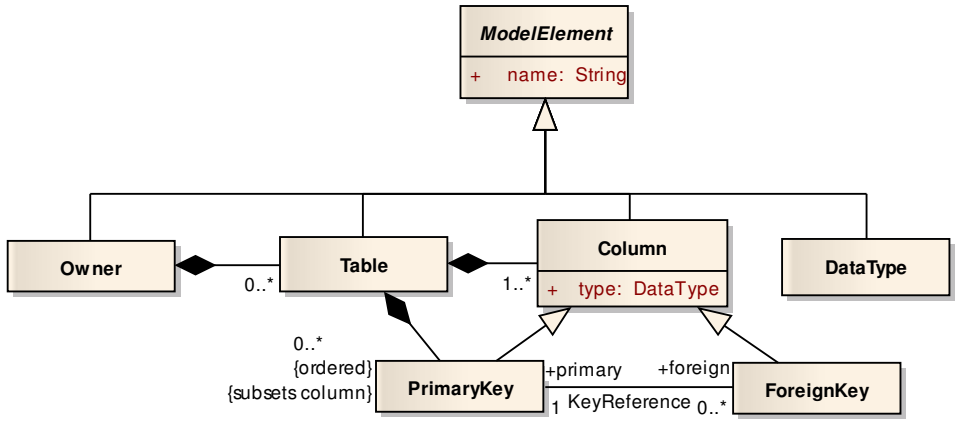
[80] Joint Revised Submission: Meta Object Facility (MOF) Specification [MOF97]

2.7.1 Relációs adatbázis-kezelők metamodellje

A relációs adatkezelő rendszerek nagy sikeréhez nyilvánvalóan hozzájárult a rendszer egyszerű és könnyen áttekinthető szerkezete is. A könnyű áttekinthetőséget a metamodelljén is tetten érhetjük – még ha az alábbiakban egy egyszerűsített metamodellt mutatunk is be.

A metamodell a következő elemeket tartalmazza:

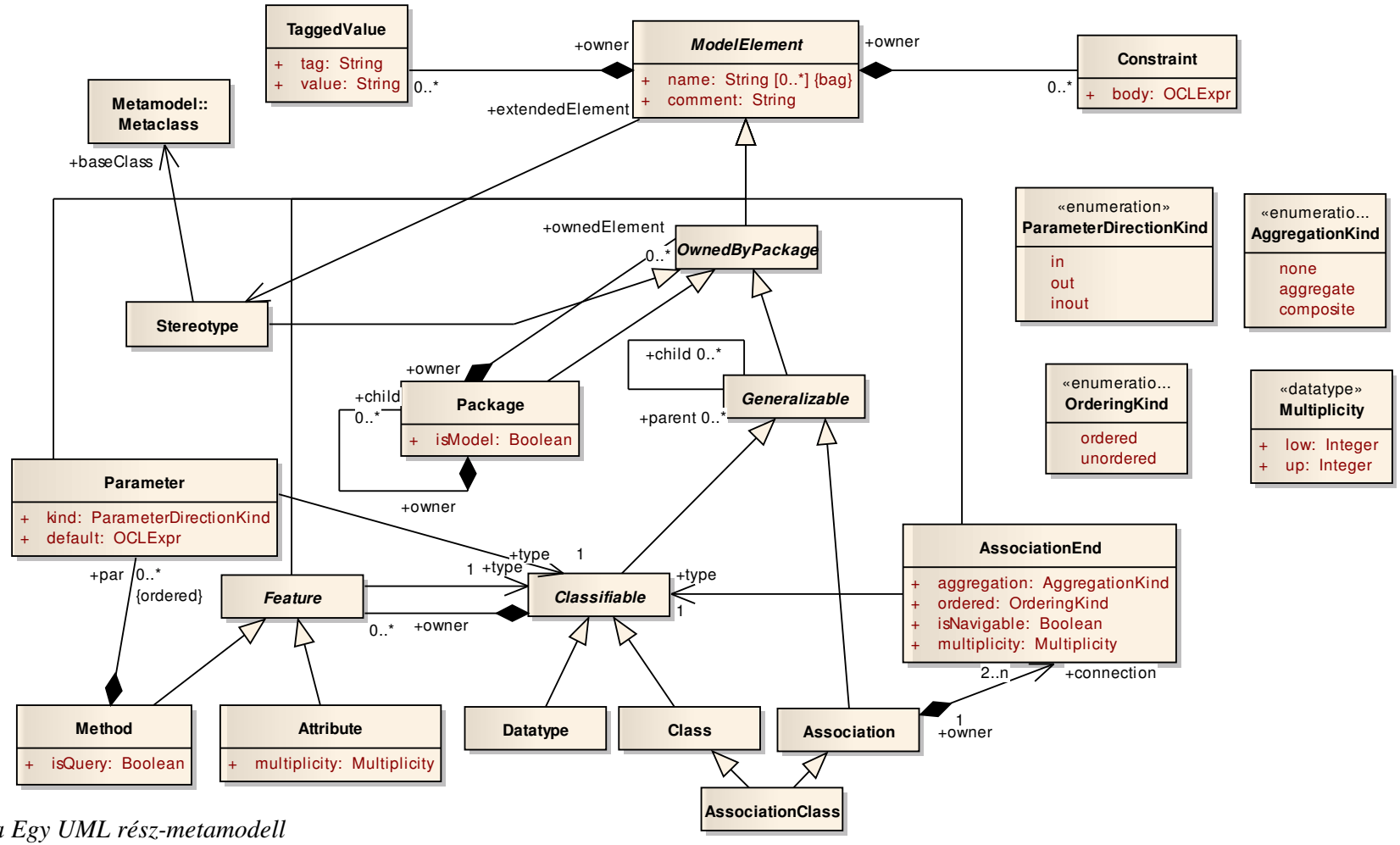
23. ábra Relációs adatbázisok metamodellje



- `ModelElement`: az absztrakt ősosztály tulajdonképpen csak a névparaméter elvonatkoztatására szolgál.
- `Owner`: Névtér vagy tulajdonos: az egyes adatbázis-alkalmazások közötti elhatárolás eszköze.
- `Table`: Relációs tábla. Egy névtérben több tábla is lehet.
- `Column`: Oszlop. Egy tábla legalább egy oszlopot tartalmaz. Az oszlopokat a nevük címzik, egyéb sorrend nincs közöttük nincs megadva.
- `PrimaryKey`: kulcs- vagy azonosító mező. A kulcsmezők az oszlopok részhalmazai, de ők már sorrendbe vannak szedve.
- `ForeignKey`: külső kulcs, vagyis egy táblában egy külső tábla valamilyen kulcsértékének szerepeltetése.
- `DataType`: az egyes elemi adattípusokat (egész, lebegőpontos, string, stb.) példányként tároló osztály.

2.7.2 UML osztálydiagramok rész-metamodellje

A következő lapon – tájkép jellegű tördelésben – az UML osztálydiagramok egy rész-metamodelljét mutatjuk be. Az ábrán a korábbi szakaszokban már részletesen taglalt UML modellelemeket és azok egymással való kapcsolatát mutatjuk be.



24. ábra Egy UML rész-metamodell

2.7.3 A Prolog metamodellje

Az alábbi modell egy általánosított és némiképp egyszerűsített, de a szabványoshoz közel álló Prolog-nyelvjárás metamodellje. Az alábbi leíráshoz be kell vezetnünk a funktor fogalmát. Ez egy predikátumra vagy függvénykifejezésre a nevükből és a paraméterszámukból létrehozott `NÉV/ARG` formátumú kifejezést jelent, és általában a Prolog eljárások *névjegyeként (signature)* szokás használni.

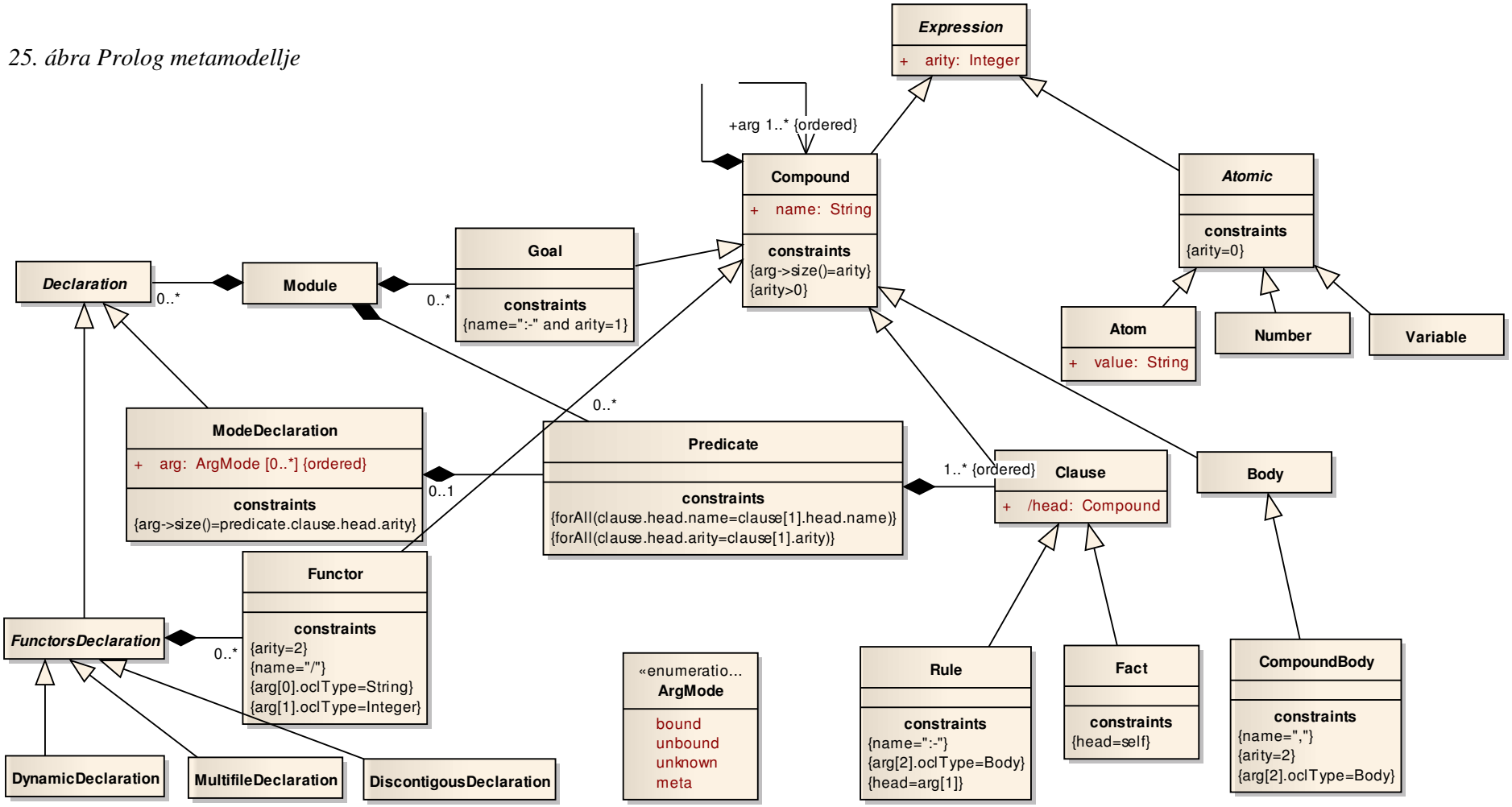
A modell elemei:

- `Expression` (kifejezés): Egy Prolog kifejezés lehet egyszerű és lehet összetett. A kifejezéseknek argumentumszámuk van (`arity`), megállapodás szerint a 0 aritásúak az egyszerű kifejezések.
- `Atomic` (egyszerű kifejezés): Egy egyszerű Prolog kifejezés lehet szövegkonstans (`Atom`), szám (`Number`) vagy logikai változó (`Variable`).
- `Compound` (összetett kifejezés): Az összetett kifejezések egy névből (`name`) és legalább egy argumentum-kifejezésből vannak összetéve.
- `Module` (modul): A Prolog-objektumok között különleges szerepe van a moduloknak. Egy modul predikátumokból, deklarációkból és célállításokból áll.
- `Predicate` (predikátum): Az ugyanolyan nevű és ugyannyi argumentumszámú állítások halmazát együtt predikátumnak nevezzük.
- `Clause` (állítás): Egy állítás lehet egy implikációt kifejező szabály vagy egy tényállítás. Az állítások neve alatt az állításfej nevét értjük.
- `Fact` (tény): Egy tényállítás feltételrész nélküli állítás, amely az adott reláció fennállását rögzíti. A tényállítás maga a saját feje, törzs nélkül.
- `Rule` (szabály): Egy szabály egy feltétel-következmény viszony kifejezése, amelynek a következmény oldalán csak egyetlen literál szerepelhet. A feltétel egy `Body` típusú Prolog kifejezés.
- `Body` (törzs): Egy törzs egy szabály vagy egy célsorozat feltételrészét írja le. A törzs lehet összetett törzs: a nem összetett törzs csupán egyetlen elemi feltételt tartalmaz.
- `CompoundBody` (összetett törzs): Az összetett törzs elemi feltételek konjunkciója, egy olyan szabály, amely a „ ,” kétargumentumú konjunkció operátorral jön létre: az első argumentum az első feltétel, míg a második argumentum törzs típusú.
- `Goal` (célsorozat): A célsorozat egy üres következményű következtetés, formailag egy olyan egyparaméteres Prolog kifejezés, amelyet a „ :-” implikáció jellel és szabálytörzs (`Body`) objektum összekapcsolásával képezünk.
- `Declaration` (deklarációk): A végrehajtásban csak közvetve részt vevő segédinformációk a deklarációk. Ezek közül Prolog modulokban *módlistás* és *funktorlistás* deklarációk szerepelhetnek. Módlistásak a `mode`, a `meta`, és a `block` deklarációk, funktorlistásak a `multifile`, a `dynamic`, a `discontiguous`, a `volatile` és a `public` deklarációk.
- `ModeDeclaration` (mód deklaráció): Egy predikátumhoz vagy tartozik mód deklaráció vagy nem. Egy mód deklaráció tartalmazza a predikátum nevét, és az

argumentumok helyén az úgynevezett *hívási módok*, a paraméterek behelyettesítési állapotainak listáját.

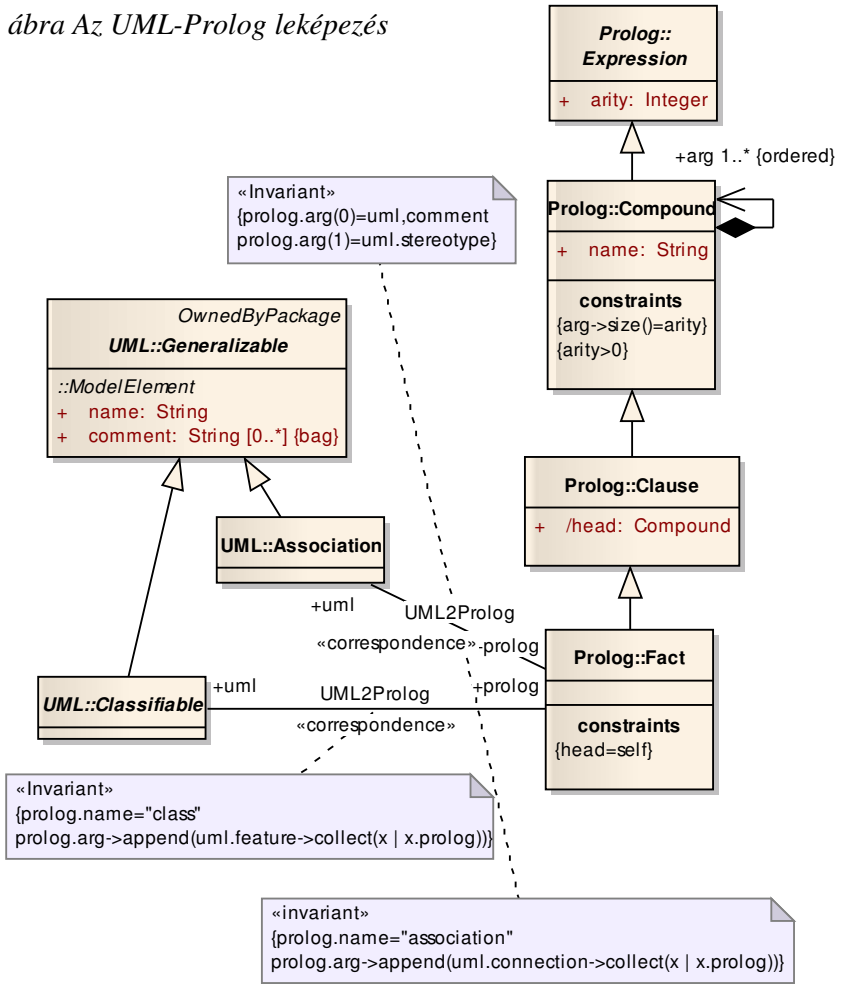
- `ArgMode` (mód): A felsorolásban négyféle módmegadási értéket engedélyezünk:
    - `bound` (+): a változónak a híváskor értékkel kell rendelkeznie
    - `unbound` (-): a változó értékének a híváskor ismeretlennek kell lennie
    - `unknown` (?): a változó lehet behelyettesített és behelyettesítetlen is
    - `meta` (:): a változó metahívást tartalmaz
- `FunctorsDeclaration`: több teljes predikátumra vonatkozó deklaráció, amelyet a predikátumok funktoraival adunk meg. Ilyenek lehetnek: dinamikus deklaráció (`DynamicDeclaration`), több fájlba tartozó deklaráció (`MultifileDeclaration`), szakaszos deklaráció (`DiscontigousDeclaration`)

25. ábra Prolog metamodellje



2.7.3.1 UML-Prolog leképezés

26. ábra Az UML-Prolog leképezés



Különböző programozási paradigmák közötti hozzárendelésre akkor lehet szükség, ha intelligens információ integrációs (I^3) feladatot szeretnénk végezni, vagyis, ha az egyik paradigma keretein belül tárolt adatokat egy másik paradigmába akarjuk átalakítani. Az adat-átalakítást az adatok *modelljei közötti megfeleltetésként* néha nem is lehetséges egységes módon leírni, mivel a külön paradigmák esetleg valami teljesen összeférhetetlen formalizmust használnak. Igény lehet esetleg egy általános, más esetben is használható átalakító algoritmus elkészítésére. Ilyen esetben az egyes paradigmák *metamodellje* között célszerű hozzárendelést létrehozni: a metamodell maga a paradigma, ezek, még ha eltérők is, akkor is leírhatók közös formalizmussal. Ezt a megközelítést illusztráljuk az alábbiakban. Az UML és a Prolog között talán a legkézenfekvőbb hozzárendelés szerint (és a korábban már elemzett UML logikai modell alapján) egy metaosztálynak egy Prolog predikátumot, a metaosztály egy-egy példányának (az alkalmazói modell egyes elemeinek) pedig a predikátum egy klózát feleltetjük meg. A predikátum argumentumaiban a skaláris osztálytulajdonságok értékei, (esetleg a kizárólagos összetevők / composition) típusai találhatók. Ehhez hasonló a hozzárendelés a kapcsolatok és a Prolog predikátumok között: egy kapcsolatnak egy predikátumot, egy példányának pedig a predikátum egy-egy klózát feleltetjük meg. A

predikátum paramétereiben pedig a kapcsolatvégek típusait ábrázoljuk, illetve a tulajdonságait akkor, ha a kapcsolat egy kapcsolóosztály példánya.

Az ábrázolt leképezés az UML metamodellt Prolog predikátumokba, a metamodell példányait, vagyis az alkalmazói modell elemeit Prolog állításokba viszi át.

Felmerül a kérdés, hogy hogyan lehetne a fenti metamodell átalakítás alapján az adat-átalakítás szabályát is leírni, esetleg mindezt a modellhez kapcsolt valamilyen OCL megszorítás formájában. Sajnos azonban erre az UML és az OCL önmagában alkalmatlan. Egy UML osztály ábrázolása ugyanis a példányainak halmazát jelenti: nincs eszköz egy metamodell osztály által leírt modellelemek példányaira (egy osztály példányainak példányaira) vonatkozólag bármit is megszabni.

Ha mégis ezt szeretnénk rögzíteni, a következő verbális leírást adhatjuk:

1. A metamodell `Classifiable` példányai az alkalmazói osztályok és kapcsolatok. Ezek példányait egy-egy Prolog tényállításba visszük át.
2. A Prolog tényállítások neve az adott `Classifiable` példányosztály neve lesz.
3. A Prolog tényállítások paraméterei a következők közül azok, amelyek egyáltalán léteznek: 1. kapcsolat-végobjektumok, 2. a tulajdonságok értékei, 3. részobjektumok.
    a. Ezek meghatározása: vesszük a `Classifiable` metaosztály példányait (ez lesz a példány alkalmazói osztálya), majd annak a példányait.
    b. vesszük az alkalmazói osztály tulajdonságait, kapcsolatait és képezzük az objektumok tulajdonság-értékeit, illetve társobjektumait.
    c. Ezen értékeket és/vagy objektumokat Prolog formába alakítjuk. (Ez az alaptípusokra kézenfekvő, részobjektumokat és/vagy adattípusokat közvetlenül átalakítjuk, azonosítható objektumok azonosítóját vesszük).

A leírt lépések közül a 3.a és a 3.b jelent gondot. A 3.a pontban lényegében az OCL rétegszerkezete, illetve a rétegek közötti alapvetően elsőrendű logikai világgal van baj: a szokásoshoz képest itt kétszeres metaszint ugrást kellene megfogalmazni. Erre esetleg az OCL által rögzített `allInstances` függvény kétszeres alkalmazása jelenthet valamiféle megoldást, de ez – úgy érezzük – nagyon feszegeti a tervezők elképzeléseit. A 3.b pontban viszont a jellemző-érték meghatározása nem rögzített, hanem változóban megkapott navigáló kifejezés alapján történne – ami másrészt szintén egyfajta másodrendű logikába tett kilépés. A fenti működésmód – a dinamikus navigálás – az objektumközpontú alapmegoldások között nem található, egyes nyelvek adhatnak rá külön alkalmazói csomagként megoldást (pl. a Java Reflection).

```
class('String',dataType,[]).
class('Integer',dataType,[]).
class('Boolean',dataType,[]).
class('Date',dataType,[]).

class('Állat',abstract,[
       attribute(nem,nem),
       attribute('született','Date')]).
class('Háziállat',abstract,
       [attribute(neve,'String')]).
class(kutya,class,[]).
class(ember,class,[
       attribute(foglalkozas,'String')])
class(nem,enumeration,[]).
```

27. ábra Példányok Prolog átírása

```
kutya(him,2001-szept-25,
       'Vackor').
kutya(nosteny,1999-marc-12,
       'Rozsdás').
kutya(him,2002-dec-3,
       'Bodri').
ember(hím,1958-marc-3,
       'Gábor',matematikus).
ember(hím,1960-maj-8,
       'Imre','mérnök').
gazda(ember(hím,1960-maj-8,
       'Imre','mérnök'),
       kutya(him,2001-szept-25,
       'Vackor')).
gazda(ember(hím,1958-marc-3,
       'Gábor',matematikus),
       kutya(him,2002-dec-3,
       'Bodri')).
```

```
generalization('Állat','Lény').
generalization('Háziállat','Állat').
generalization(kutya,'Háziállat').

association(gazda,
       [associationEnd(gazdi,true,
              unordered,
              multiplicity(1,1,false),
              [],true,ember),
       associationEnd('jószág',false,
              unordered,
              multiplicity(0,0,true),
              [],true,'Háziállat')]).
```

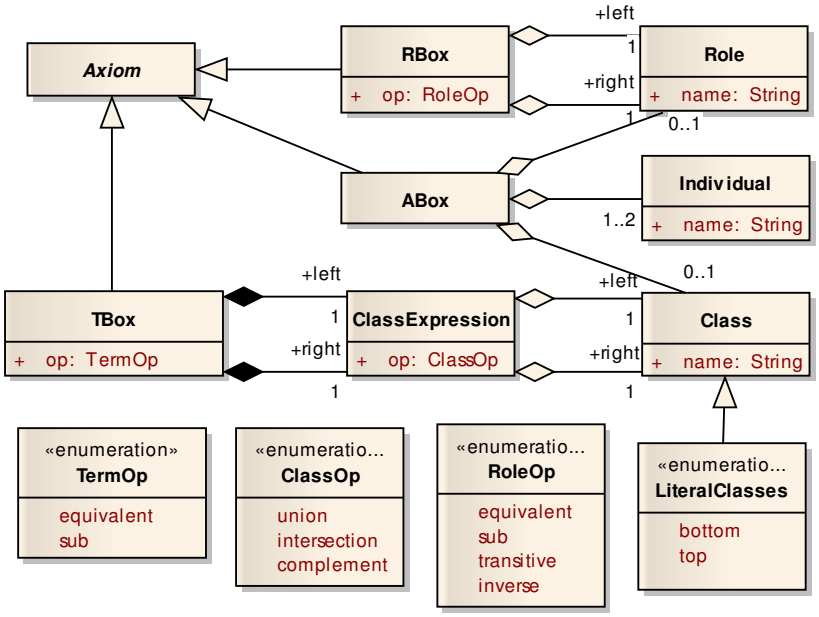
28. ábra Modell Prolog átírása.

A bal oldali ábrán bemutatjuk egy egyszerű alkalmazói modell, a jobb oldalin pedig a példányainak Prolog átírását a fenti megszorítások és megoldások mellett. (A modell a korábbi példamodellekhez hasonlít, és ennek egy példánya a második ábrán már bemutatott objektumháló). Megjegyezzük, hogy a teljes tulajdonságkészlet megismétlése a kapcsolat-példányok esetén (`gazda/2` tényállítások) redundáns. Ezt a redundanciát azonosító, vagy kulcstulajdonság megadásával lehetséges kiküszöbölni, ilyenkor a kapcsolat-példányokban az objektumpéldányokat leíró függvénykifejezés argumentumai között csak ezeket is elegendő lehet megadni.

2.7.4 Az OWL2 metamodellje

Mint azt a leíró logikákkal foglalkozó fejezetben korábban már megírtuk, az OWL2 ontológia-leíró nyelv többféle fokozatban megjelenhet (OWL-Lite, OWL-DL, OWL-Full). Másrészt viszont a nyelvnek többféle megjelenési formája is létezik, amely más és más nyelvi szerkezettel jeleníti meg ugyanazt a háttérbeli logikai szerkezetet. (Létezik XML alapú, de közönséges szövegfájl alapú nyelvtana is, ez utóbbiból több is). Az ilyen, többféle megjelenésű nyelvek esetében nyer igazán jelentőséget a metamodell, ami nem a változó megjelenítésnek, hanem éppen az háttérben rejlő egységes adatszerkezetnek a leírása.

Az alábbiakban az OWL-Lite egy metamodelljét mutatjuk be – a bonyolultabbak ebből már könnyedén származtathatók. [81]



29. ábra Az OWL2-Lite egy metamodellje

A metamodell elemei a következők:

- `Axiom`: a nyelv axiómáit általánosságban magába foglaló, egyébként absztrakt osztály
- `TBox`: a nyelv terminológiai (osztályokra vonatkozó) axiómái. Ezek legfeljebb két osztálykifejezésből (`ClassExpression`) épülnek fel egy `TermOp` műveleti operátor segítségével.
- `RBox`: a nyelv szerepleíró axiómái. Ezek legfeljebb két szerepből (`Role`) épülnek fel egy szerepoperátor (`RoleOp`) segítségével.
- `ABox`: a nyelv adatleíró axiómái. Ezek egy osztályból (`Class`) vagy szerepből (`Role`) egy vagy két egyedi azonosító (`Individual`) segítségével épülnek fel. Osztálypéldányok megadásához egy osztály és egyetlen egyed, szereppéldány megadásához egy szerep és két egyed megadása szükséges.
- `ClassExpression`: osztálykifejezések egy vagy két osztályból egy osztályoperátor (`ClassOp`) segítségével építhetők fel.
- `Class`: Az osztályok az ontológiában megadott osztályokat jelentik, de ide tartoznak az egyetlen elemmel sem rendelkező fenékjel, valamint a minden elemet magába foglaló tetőjel osztályok is.

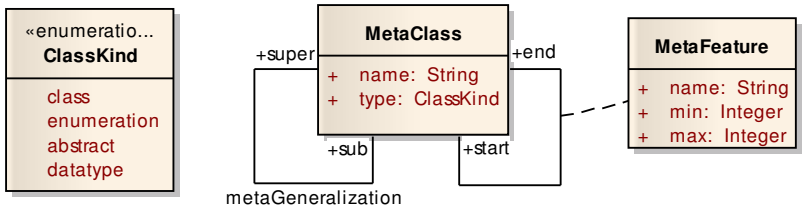
2.7.5 Egy lehetséges meta-metamodell

Az UML négy elvonatkoztatási rétege közül az utolsó – a legelvonatkoztatottabb – a meta-metamodell rétege. Míg a harmadik réteg – a metamodell – egy modellező eszköz,

[81] Object Management Group: Ontology Definition Metamodel Version 1.0 [ODM09]

esetleg egy programnyelv szerkezetének leírását tartalmazza, addig a negyedik réteg, a meta-metamodell olyan fejlesztőeszközökben használatos, amelyek esetében még a program- vagy modellező nyelv sem rögzített. Az ilyen rendszerek tulajdonképpen *keretrendszerek*, amelyeket újabb és újabb metamodellek beprogramozásával újabb és újabb fejlesztőkörnyezetként lehet alkalmazni.

Ilyen például az Eclipse integrált szoftverfejlesztő környezet (IDE)[82], amely szintén keretrendszer. Ez – ha nem is könnyen, de mégiscsak – testre szabható; bármilyen új programnyelv vagy más modellezési nyelv metamodelljének megadásával (és becsatolásával) a keretrendszer alkalmassá tehető az új nyelv kezelésére.



30. ábra Egy lehetséges meta-metamodell UML-ben

A fenti ábra egy lehetséges meta-metamodell UML-ben leírt szerkezetét ábrázolja. A meta-metamodellnek a lehetséges metamodellek jellemzőit kell megragadnia: most is csak osztálydiagramokról, csak adatmodellről, és az UML csomagfogalmának teljes elhanyagolásáról van szó.

A fenti meta-metamodell egyrészt összemossa a tulajdonság és a kapcsolat, az aggregáció és az objektum-összetétel fogalmát, másrészt pedig mindezeket mindegyik lehetséges navigációs irányban meg kell adni. A tulajdonságok egyirányú navigációt jeleznek, a kapcsolatok esetében a navigáció megadásától függően egyirányú vagy kétirányú a kapcsolat.

A meta-metamodell elemei:

- `MetaClass`: a metamodell osztályait magába foglaló meta-metaosztály.
- `metaGeneralization`: a metamodell osztályai felett értelmezhető általánosítási viszony. Ennek a jelentősége az, hogy a metamodellben használhassunk általánosítást, és ne kelljen az öröklést kézzel elvégezni, és az örökölt jellemzőket kézzel összegyűjteni.
- `MetaFeature`: a fentebb már említett metamodell-jellemzőket leíró kapcsolóosztály. Egy példány szigorúan egyirányú navigációt ír le: szimmetrikus navigáció esetében mindkét irányt külön példányként kell felvenni. A kapcsolóosztály jellemzői:
  - `start`: az a metaosztály, amelyikhez a jellemző kapcsolva van (gazdaosztály)
  - `end`: a jellemző típusa
  - `name`: a jellemző neve
  - `min`: a jellemző legkisebb megengedett többszörössége

[82] Az Eclipse Alapítvány honlapja: http:/www.eclipse.org, elérés: 31-Jan-2013.

    - `max`: a jellemző legnagyobb megengedett többszörössége. Megállapodás szerint itt 0 érték megadása a tetszőlegesen nagy többszörösség-értéket jelent.

- `ClassKind`: az osztályok metatípusa. A következő típusjelzők vannak értelmezve:
    - `dataType`: alap adattípus, amelyet nem szükséges tovább részletezni
    - `enumeration`: felsorolástípus. A felsorolások elemeit közvetlenül adjuk meg.
    - `abstract`: az osztálynak nem lehetnek példányai, ilyen osztályokat közös jellemzők kiemelése végett adunk meg.
    - `class`: „normális” osztály, amely semmilyen különleges kezelést nem igényel

A fenti meta-metamodellnek megfelelő metamodell lényegében a három meta-metaosztálynak megfelelő példányok megadását jelenti (ahol a `metaGeneralization` kapcsolatot szintén a metaosztályok között értjük). A modell rögzítésénél csak skaláris tulajdonságokat vettünk fel, ezért az egyes osztályok példánykészletét a legkézenfekvőbben egy relációs (Excel) táblával adhatjuk meg. Tekintsük most példaképpen a `metaFeature` kapcsolóosztály példányait tároló adatbázistábla egy részét, ha éppen a fentebb bemutatott OWL-Lite metamodellt írjuk le vele.

start	**name**	**end**	**min**	**max**
RBox	left	Role	1	1
RBox	right	Role	1	1
RBox	op	RoleOp	1	1
ABox	role	Role	0	1
ABox	individual	Individual	1	2
ABox	class	Class	0	1
Class	name	String	1	1

2.8 Ontológiák és szoftver környezeteik

Az ontológiák fogalma – oly sok máshoz hasonlóan – szintén az ókori görögökkel hozható összefüggésbe. Az eredeti értelmezésben az ontológia lételméletet jelent, amely létünk legfőbb forrásának az alapvető, filozófiai mélységű kérdéseit kutatta.

Hérakleitosz Kr. e. V. századi bölcs alapvetése szerint minden mozog, „Panta Rhei” – mint kamaszkorunk kedvenc – Bartókot és Muszorszkijt és Grieg Peer Gynt musicaljét progresszív rock stílusban játszó – együttesének neve is mondja: semmi sem biztos, minden változik, egy, ami biztos, a mozgás, a változás maga.

Parmenidész, aki ugyanebben a korban élt, úgy gondolta: minden létezőnek a gondolkodás a végső forrása. „Gondolkodni ugyanaz, mint létezni.” A Descartesnek tulajdonított mondás (Cogito ergo sum, Gondolkodom, tehát vagyok) lényegét tekintve tehát jóval öregebb.

Talán egy évszázaddal később, Kr. e. IV. században éltek a nagy idealista filozófusok. Platón és Arisztotelész megítélése szerint a nagyvilág csupán látszat, vetület, és mindaz, amit látunk, az örök érvényű és tökéletes ideák tökéletlen lenyomatai.

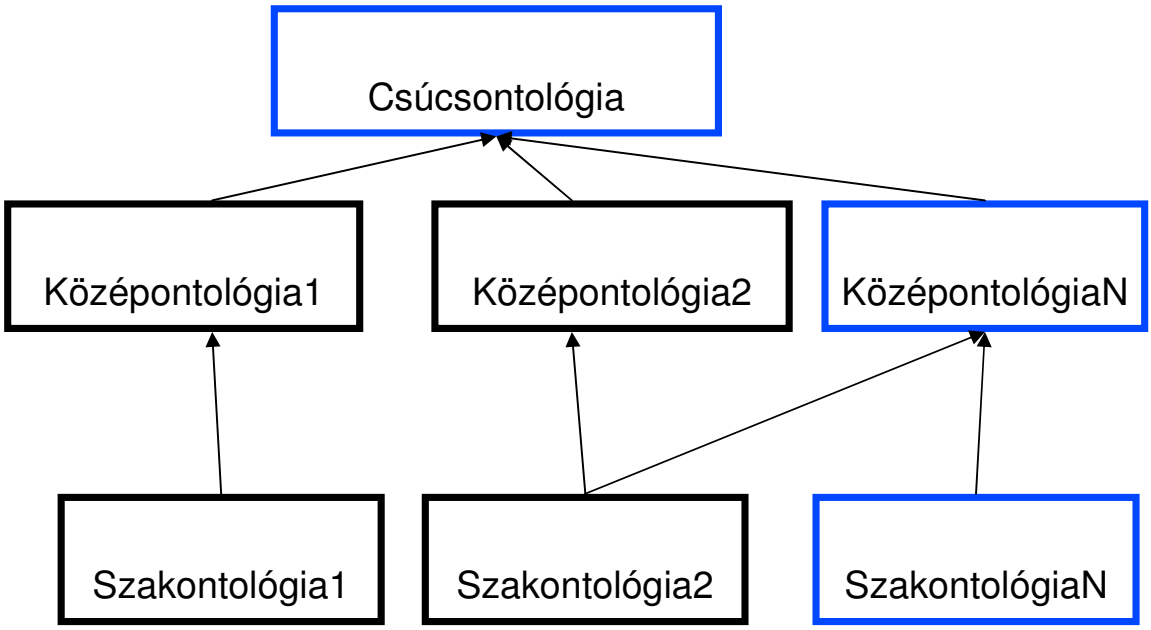
Az ontológiák kérdésköre a Mesterséges Intelligencia, mint tudományág, illetve kutatási terület fejlődésével került ismét a köztudatba. Ahhoz, hogy az embert közelítő intelligenciaszintű gépi rendszereket építhessünk, az emberi tudást magát kell valamilyen módon ábrázolnunk. Az ontológia tehát tudástár, amit régebben esetleg tudásbázisnak, esetleg fogalomtárnak is neveztek. Intelligens rendszerekben a tudástár – a maga filozófiai jelentőségén túl – a következő műszaki szempontokat is teljesíti.

- Ahhoz, hogy az Intelligens Rendszerek egymással is kapcsolódni tudjanak, szükség van rá, hogy közös fogalomkincset, közös szótárat, és a szavaknak azonos értelmezését használják. Ezt a célt szolgálják az ontológiák, amelyeket ezért más rendszerek felé *meg is osztunk*.
- Egyetlen egyéni rendszer sem tartalmazza (és nem is tartalmazhatja) az emberi tudást a maga teljességében. Amikor viszont helyben nem tárolt tudásra van szükség, olyankor más gépeken *megosztott tudásra is hivatkozhatnak*, idegen tudáselemeket is felhasználhatnak a következtetési műveletek során.
- Általánosságban megállapítható, hogy a tudástárat *modularizálhatóvá* kell tenni: vagyis meg kell adni azokat az eszközöket, amelyek a teljes tudás – esetleg nem is egyetlen helyen tárolt – építőköveiből történő felépítését biztosítják.

Az ontológiaépítés [83] során a következő szinteket különítjük el:

- A *csúcsontológiák* (Upper Level Ontology) a lételmélet legelvontabb, legáltalánosabb fogalmait és összefüggéseit tárolják (pl. Előfordulás, Jellemző, Időfüggő, Időtlen, része, jellemzője, stb. Megállapodás szerint az ontológiák fogalmait nagy kezdőbetűvel, az összefüggéseket kis kezdőbetűvel írjuk.)
- A *középontológiák* (Mid-Level Ontology) az úgynevezett „közhelytudás" (Commonsense) már elegendően konkrét, de még általános érvényű elemeit tartalmazzák. (pl. Kontinens, Ország, Év, Teremtmény, lakóhelye, határos, stb.)
- A *szakontológiák* (Domain Ontology) egy-egy konkrét szakterület tudásanyagát ölelik fel.

[83] Mike Bergman: Brown Bag Lunch: An Intrepid Guide to Ontologies [MIKE10]



31. ábra Alkalmazói ontológia felépítése importálással csúcs- és középontológiákból

A fentiek elvontsági fokozatokat jelölnek: a modularizálás természetesen a fentieknél finomabb elemekből is építkezhet. Azt a viszonyt, amikor egy ontológia egy másik ontológia-modul elemeire építkezik, *importálásnak* nevezzük (ezt a fenti ábrán az importált ontológiába mutató nyíllal jelöltük).

Amikor tehát valamiféle konkrét feladathoz készítünk szakontológiát, akkor a legcélszerűbb valamilyen már létező ontológiákra építkeznünk: vagyis megkeresnünk és importálnunk a célnak leginkább megfelelő középontológiát – esetleg többet is. A középontológia feltehetően szintén importál más ontológiákat – csúcsontológiát, esetleg csak egyes csúcsontológiai modulokat. A feladatban használatos teljes ontológiát az *importálási viszony* továbbvitelével, más néven *lezárásával* kapjuk meg.

Ontológiák építésére célszerűen valamiféle *logikai nyelvet* használhatunk. A logikai nyelven következtetési műveleteket von le a *kalkulus*, a *következtetőgép* megvalósítása. A nyelv és a kalkulus egymásnak az ellenpontjai.

A logikai nyelvekhez a feszes matematikai alapok rögzítését nem adhatjuk fel, mert homályos és pontatlan fogalmakkal, ösztönös szoftverkészítői hozzáállással az ontológiaépítés sokszoros elvonatkoztatása nem vihető végig, az ontológiákra épülő alkalmazások pedig nem építhetők fel. Általánosságban itt is igaz: minél egyszerűbb nyelvet választunk, annál könnyebb hozzá megfelelő következtetőgépet létrehozni, de a nyelv annál bőbeszédűbb, redundánsabb lesz: egy nagyon egyszerű nyelv gyakorlatilag alkalmatlan magasan elvonatkoztatott fogalmak kifejezésére. Ezzel szemben viszont sajnos minél inkább emeljük a nyelv elvontságát, annál könnyebben és tömörebben tudunk vele elvont fogalmakat kifejezni, viszont annál kevésbé biztos, hogy létezik alkalmas következtetési módszer, eljárás, számítógépes algoritmus.

A logikai nyelv kiválasztásakor a következő lehetőségeket vehetjük számba:

- Elsőrendű logika vagy résznyelvének alkalmazása. Bár sok problémára gyengének bizonyulhat, az elsőrendű logika nyelve mégis eléggé kifejező. A nyelvre eléggé jó tételbizonyítók léteznek, de teljes tételbizonyító nincsen,

vagyis bármilyen jó tételbizonyítót is találunk, mindig megadható hozzá olyan probléma, amelyet nem képes bebizonyítani (hanem pl. végtelen ciklusba kerül).

- A Horn-klózok nyelve. A Prolog programozási nyelv köré épült kultúra csábító és jól alkalmazható, mégis, a Prologot csak nagyon csapnivaló tételbizonyítónak nevezhetjük. Valószínűleg érdemes lehet a Prologot, mint szoftver környezetet választanunk úgy, hogy a tételbizonyítási mechanizmusát nem elfogadva, annak kiterjesztésére készülünk fel.
- Leíró logikák, OWL. A nyelv tényleges szabványnak számít, többféle szintje van, az egyszerűbbekre létezik tételbizonyító algoritmus. Mégis, az egész önmagában szegényes, inkább csak taxonómiák leírására alkalmas. Az SWRL szabályleíró résznyelvével együtt már hatékonyabb, ez viszont matematikailag nem kristálytiszta. Szerkesztőprogramok, következtetőgépek elérhetők hozzá.
- Modális logikák. A modális logikákat úgy is tekinthetjük, mint egyes másodrendű szerkezetek elsőrendűbe történő reifikációját, amelyben a másodrendű szerkezetek értelmezését maga a következtető motor végzi. Jogi rendszerekhez általánosságban három modalitás együttes kezelésére van szükség egyidejűleg: temporális logika az időmodalitások, az episztemikus logika a bírói helyzetek, illetve a deontikus logika a jogi szerkezetek általános kezeléséhez.

A következtetőgép tekintetében a következő meggondolásokat tehetjük:

- A következtetőgép *nem vonhat le hamis következtetéseket*, ha viszont nem sikerül minden lehetséges következtetést megtalálnia, az elfogadható. Más szóval: a *helyesség alapvető kritérium, a teljesség nem feltétlenül*.
- A levonható következtetések kérdése inkább mennyiségi jellegű, vagyis léteznek jobb és kevésbé jó következtetési módszerek. A következtetési mechanizmust különböző módszerekkel *paraméterezhetővé*, sőt, *programozhatóvá* kell tenni. Ez *meta-következtetési* szabályokon keresztül biztosítaná azt, hogy már a következtetési szabályok pontos alkalmazása is tematikafüggően beállítható, testre szabható, programozható legyen.

2.8.1 Ontológialeíró nyelvek

Az ontológiák leírására használt nyelvek között – sok máshoz, például a MI szoftverek írásához hasonlóan – két markánsan elkülöníthető iskola létezik. Az amerikai iskola az ott általában előnyt élvező LISP programnyelvből kiindulva tervezett meg több nyelvváltozatot. Ezek a nyelvek, még ha erejükben az elsőrendű logikát célozzák is, nyelvtanukban követik a LISP jól ismert sok zárójeles szerkezetét. A másik iskola a Szemantikus Háló projektum OWL nyelvét vagy annak változatait használóké, akik túlnyomórészt európaiak.

2.8.1.1 A Knowledge Interchange Format (KIF) nyelv

A KIF nyelv a tudásleíró nyelvek közötti első szabványosítási törekvés eredménye, amelyet éppen ezért teljesen általános célokra terveztek: úgy a felhasználók és a tudáskezelő szoftver rendszerek közötti, mint az egyes ilyen szoftverek közötti adatcserére.[84] A KIF a Stanford Egyetem berkeiből indult el, de később több amerikai kutatóprojektum épített rá. Talán itt érdemes megemlíteni az Defense Advanced

[84] Michael R. Genesereth-Richard E. Fikes: Knowledge Interchange Format Version 3.0 Reference Manual [GeFi92]

Research Projects Agency (DARPA) Knowledge Sharing Effort 1990-ben indított kezdeményezését, mely a KIF-fel kapcsolatos különféle szoftver csomagokat is közzétett. Így létrejött egy KIF-re épülő, keretalapú tudásábrázolási megközelítést lehetővé tevő programozható felület (API) (természetesen LISP nyelven), egy C++-ban megírt KIF elemző, egy LISP és C felület a KIF-re alapuló Knowledge Base Query and Manipulation Language (KQML) protokoll kezelésére, valamint egy interaktív ontológiaszerkesztő kiszolgáló környezet, amely ontológiák létrehozására, átalakítására és közzétételére alkalmas.[85]

A KIF nyelvet azóta is használják, pl. olyan komoly ontológiák készültek segítségével, mint a SUMO, illetve szabványosított formája a SUO,[86] vagy az olasz Laboratory of Applied Ontology kutatóintézet WonderWeb családja. A család elemei: a DOLCE/Descriptive Ontology for Linguistic and Cognitive Engineering, OCHRE/Core Ontology for Cultural Heritage és a BFO/Basic Formal Ontology.[87] Utóbb a KIF nyelvet továbbfejlesztették: az új ontológia-leíró nyelv neve: Common Logic (CL), amelyet 2007-ben az ISO/IEC 24707 lajstromszámmal szabványként el is fogadott[88].

A KIF nyelv tipikusan az amerikai LISP iskola szülötte: sok zárójeles LISP-szerű nyelvtan, és elsőrendű, feszes logikai alapok, melynek elveit a következőkben lehet összefoglalni:

- Szigorúan kétértékű logika (egy állítás vagy igaz, vagy hamis)
- Entitások ábrázolása, ami magába foglalhat fizikai objektumokat, fogalmakat és tulajdonságokat
- Viszonyok (relációk) ábrázolása, amelyek entitások, illetve entitások és tulajdonság-értékek között állhatnak fenn.
- Kvantorok (egzisztenciális és univerzális) használata
- Explicit negáció kezelése
- Halmazok feletti univerzális kvantifikáció, amit iterációnak neveznek.

A KIF nyelv a háttérben rögzített szerkezet és szemantika felett a következő fajta nyelvi változatokat engedi meg:

- Az EBNF nyelven rögzített LISP-szerű KIF alapnyelvtan
- XCL: egy XML alapú leírónyelv
- Conceptual Graph Interchange Form (CGIF): A John Sowa által 1984-ben bevezetett Conceptual Graph elképzelésnek megfelelő leíró nyelv.[89]

---

[85] Tim Finin-Jay Weber-Gio Wiederhold-Michael Gensereth-Richard Fritzzon-Donald McKay-James McGuire-Richard Pelavin-Stuart Shapiro-Chris Beck: DRAFT Specification of the KQML Agent-Communication Language [KQML93]

[86] Niles, I-Pease, A: Origins of the Standard Upper Merged Ontology: A Proposal for the IEEE Standard Upper Ontology [SUO01]

[87] Claudio Masolo-Stefano Borgo-Aldo Gangemi-Nicola Guarino-Alessandro Oltramari-Luc Schneider: The WonderWeb Library of Foundational Ontologies Preliminary Report [WW03]

[88] Information Technology – Common Logic (CL): a framework for a family of logic-based languages ISO/IEC 24707 szabvány [CL07]

[89] John F. Sowa: Conceptual Structures: Information Processing in Mind and Machine [So84]

Maga a KIF nyelv, illetve a CL szabvány a nyelv és a szemantikus szerkezet feletti következtetési műveletekre nem terjed ki, azokat az egyes utód-projektumok a saját szempontjaiknak megfelelően valósították meg.

2.8.1.2 A CycL ontológia-leíró nyelv

A Cyc projektumot 1984-ben a texasi Microelectronics and Computer Technology Corporation kutatóintézet indította, amelynek célja volt, hogy számítógépes eszközökkel a lehető legteljesebben írjon le és gyűjtsön össze minden emberi tudás darabocskát. A projektum a Doug Lenat vezető kutató által alapított 1986-ban megalapított Cycorp Ltd. cégben élt és él mindmáig tovább. Az időközben létrehozott ontológia ResearchCyc néven összesen milliónyi tudáselemet, közte több százezernyi fogalom-meghatározást tartalmazó nagy tudástár. Az ontológia a CycL nyelven van megírva, de a lefordított alakja Java felületen elérhető, amit a cég 2006-ban szabad felhasználásúvá tett.

A tudástár egy csökkentett változata az OpenCyc, amelynek a 2009-ben közzétett 2.0 változata százezres nagyságrendű fogalom-meghatározást, és milliós nagyságrendű tényhalmazt tartalmaz. Ez azonban a tudásbázisnak elsősorban csak a taxonómia-leíró elemeit jelenti, az érdemi következtetésekre szolgáló szabályok nincsenek a rendszerben. Ezt forrásnyelven is közzétették, sőt, a Szemantikus Web projekt céljaira RDF/OWL alakú változata is létezik.

A CycL nyelv elsőrendű logikai alapokra épül, de (a KIF-hez hasonlóan) LISP-szerű nyelvtannal. Egy teljes tudásbázis *mikroelméletnek* (*microtheory*) nevezett tudás-szegmensekből épül fel. Az ellentmondás-mentesség a teljes tudásbázison belül nem követelmény, de egy mikroelméleten belül igen.[90]

Az elsőrendű logikai alapokat tekintve a CycL a teljes logikát tartalmazza, mindkét fajta kvantorral, sőt az egzisztenciális kvantálás fogalmát kiterjesztették számossági korlátozásokra is. A tudásbázisban azonban már nem találunk egzisztenciális kvantort, mert ezeket a rendszer azonnal Skolem függvényekké, illetve konstansokká alakítja.

Az alábbi példa egy általános következtetési szabály: ha egy objektumpéldány eleme egy halmaznak (gyűjteménynek), és a halmaz részhalmaza egy bővebbnek is, akkor a példány a bővebb halmaznak is eleme.

```
(#$implies
   (#$and
     (#$isa ?OBJ ?SUBSET)
     (#$genls ?SUBSET ?SUPERSET))
   (#$isa ?OBJ ?SUPERSET))
```

32. ábra Példa: 'objektum eleme egy bővebb halmaznak is' CycL nyelven

A Cyc a tudásbázisba felvett állításokat még a következtetés előtt normalizálja. Ez azt jelenti, hogy konjunktív normál formára hozza, és az egzisztenciális kvantorokat Skolem függvényekkel helyettesíti.

Az igazságértékek vonatkozásában egyaránt kezelik a *monoton igaz*, az *általában igaz*, az *ismeretlen*, az *általában hamis* és a *monoton hamis* értékeket. Ez arra utal, hogy a következtetési folyamat a negációt és az aletikus modalitásfogalmat külön folyamatként megvalósítja.

---

[90] TheCycFoundation: Ontological Engineer's Handbook [CycL02]

A következtetés *rezolúciós alapú*, amelynek során egyaránt használják az *előre-* és a *visszafelé haladó stratégiát*, de az általános következtetési mechanizmuson belül saját heurisztikus stratégiavezérlést valósítottak meg, amely tematika és használatfüggő is.

A használatfüggés azt jelenti, hogy az egyes Cyc szabályok, valamint egyes heurisztikus következtetési szabályok is *súlytényezővel* vannak ellátva, ahol az alacsonyabb súly jelenti a valószínűbb alkalmazást. Az alternatív bizonyítási utak (a rezolúciós gráf alternatív útjai) közötti döntés esetén az egyes utakhoz tartozó *súlytényezők összegződnek*, és az *alacsonyabb összegű kerül aktivizálásra*.

Például a kérdések feldolgozásakor a kérdésben megfogalmazott hipotézis bizonyítására egy (a Prologhoz hasonló) visszafelé haladó folyamat indul. Tényállítások felvételekor viszont a tényállításokból előre kikövetkeztethető következményállításokat azonnal létrehozzák.

Az egyenlőségek kezelése lényegileg az egyesítések során történik. A rendszerben a *nevek egyediek* (Unique Names Assumption), ezen csak nyilvánvalóan megadott egyenlőségállítások változtathatnak. Az efféle állítások csak egyesítésidőben vannak kezelve, így maguk önálló következtetési folyamatot sosem indítanak.

A következtetés során az alternatívák kezelése heurisztikus, de például egy mélységi stratégia is működik. A heurisztikus következtetés legfőbb szabályai:

- *Rövidebb klózokat előbb.* A megoldás az egységklóz stratégia egy általánosítása: minél rövidebb a rezolvens klóz, annál hamarabb eljuthatunk a bizonyítás végét jelentő üres klózig.
- *Kevesebb szabad változós klózokat előbb.* A megoldás csökkenti a rezolvens klózokban levő (egzisztenciálisan kötött) változók számát. Extrém esetben a teljesen változómentes (alapliterálokból álló) klózokat általában a legkönnyebb bizonyítani.
- *Óvatosan a negált literálokkal.* A tudásbázis elsősorban pozitív tudást tartalmaz, ezért feltehetőleg könnyebb a pozitív feltételeket bebizonyítani.
- *Részleges előfordulás ellenőrzés* (occur check) letilthatja a rezolúciót olyan irányban, amely az egyesítésben részt vett. Ez megakadályozza a körkörös kifejezések létrejöttét.
- *A tudásbázisból valószínűleg hiányzó elemeket igénylő következtetés tiltása*. A célliterálok között megjelöljük azokat, amelyek olyan állítás-konstans kombinációra hivatkoznak, ami nincs az adatbázisban.

2.8.1.3 A SILK projekt SILan nyelve

A System Integration via Logic and Knowledge (SILK) kutatási projektet a magyar IQSOFT Rt. vezetésével egy négytagú konzorcium futtatta az ezredforduló táján. A projekt céljaiban a későbbi Szemantikus Világháló versenytársaként indult, de később teret vesztett a ma ismert, OWL alapú kezdeményezések előtt.

A SILK végcélja a szemantikus információk intelligens és (legalább fél-) automatikus integrációja volt az UML objektum-orientált szabványra építve, annak logikai megalapozásával, és ezzel összefüggő szoftver eszközök kifejlesztésével. A projekt alapfeltételezése szerint adottak egyes különnemű információforrások, amelyek ennek ellenére az UML nyelvvel vagy valamilyen részhalmazával modellezhetők, illetve a meglevő információforrások pedig visszamodellezhetők (reverse engineering). Ha az információforrások UML modellezése megtörtént, akkor (UML kapcsolatok –

associations) segítségével azok összekapcsolhatók. Az összekapcsolás végcélja: a *mediátor* összetevő a *csatolórétegen* (wrapper) keresztül hozzáfér az említett adatforrásokhoz, és azokon közös, egyesített lekérdezést hajt végre.[91] [92]

A SILK egyik fontos jelentősége és felismerése az UML-ben már használt grafikus nyelv alkalmazása volt. Valóban: a grafikus eszközök és tudásábrázolási megoldások alkalmazása a vizuális emberi képzelet erejére számítva komolyan megnöveli a tervezők hatékonyságát. A projekt az UML osztálydiagramok grafikus nyelvét és erejét a konkrét szoftvertervezés területét kitágítva emberi tudás, ontológiák modellezésére is használta.

A szabványnak tekinthető UML osztálydiagramok mellett a projekt rögzítette a modellek külső, szövegszerű alakját is, ezt nevezték SILan nyelvnek.[93]

A nyelv megtervezésekor fontos szempont volt a már meglevő hasonló célú nyelvek újrahasznosítása. Így a SILan nyelv három résznyelvből áll:

- A CORBA IDL nyelvhez hasonló alapnyelv, amely a csomagok, osztályok, adattulajdonságok, eljárások specifikációját írja le
- Az Object Query Language (OQL) nyelv, amellyel a SILan lekérdezés-objektumai fogalmazhatók meg
- Az UML OCL nyelve, mely egyrészt az osztályokhoz-tulajdonságokhoz hozzárendelhető megszorítások (constraint) nyelve, másrészt pedig az OQL lekérdezésekben a feltételek megadása is ilyen nyelven történik.

A rendszer a háttérben a SILan nyelv szerkezeti alakját tárolta, azon különböző műveleteket hajtott végre. A SILan nyelvet UML alapú ontológia-leíró nyelvnek kell tekintenünk: amely egyrészt grafikus volt, másrészt pedig - az UML eszközeivel - képes volt az egyes ontológiák rögzítésére és tárolására.

A rendszer az ontológiák szerkesztésére, mentésére és betöltésére irányuló alapműveletek mellett az ontológiák és modellek ellenőrzésére is képes volt.

```
model Finance {
  class Employee {
    attribute String firstname, lastname;
    attribute Real salary, tax;
    constraint tax = 0.25*salary and tax >= 10000;
    primary key (firstname,lastname);
  };
};
```

33. ábra Egy SILK osztály ábrázolása SILan nyelven

A SILK projekt talán legfontosabb újdonsága nem pont a SILan nyelv rögzítése, hanem az UML grafikus nyelvi formalizmus használata ontológiaépítési célokra. A grafikus nyelv az emberi képességeket kiterjeszti, a vizuális elemekkel megadható elemek és kapcsolataik pedig könnyebb áttekinthetőséget és közvetve pedig az emberi teljesítmény nagyobb hatásfokát eredményezik. Nem véletlen, hogy az OWL alapú

[91] L. Badea-D. Tilivea-A. Hotaran-Y. Polet-N. Chancevrier-D. Parents-X. Denis: Description of the SILK Mediator tools [SILK01]

[92] L. Badea-D. Tilivea: Query Planning for Intelligent Information Integration using Constraint Handling Rules [Bad00]

[93] SILAN – the SILK language [SILAN00]

ontológiaszerkesztő rendszerek (pl. a Protégé[94]) is tartalmaznak bedugaszolható, UML-hez hasonló megjelenítő felületet.

2.8.1.4 Az OWL nyelv

Mint korábban említettük, az OWL nyelv a DL logikai keretre, valamint az XML formalizmusra épül. Az előző fejezetben megadtuk az OWL UML alapú metamodelljét is. A nyelv végső soron tehát nem más, mint a metamodell egyfajta szöveges ábrázolása (Java terminológiával: *sorosítása*). Az OWL esetében ráadásul több nyelvi változat is elterjedt, amelyet az egyes szerkesztőprogramok különböző felállásban képesek olvasni és írni. Anélkül, hogy most pontosan rögzítenénk az egyes nyelvváltozatok közötti különbséget, álljon itt néhány példa:

Az OWL egyes publikációkban (és a jelen értekezésben is) használatos, (és könnyen olvasható és felfogható) változata pl. semmilyen szabványnak nem felel meg.

```
hasSubject ⊆ topObjectProperty
hasSubProp ⊆ hasSubject
Subject ⊆ Thing
Object = Subject
Subject(Jancsi)
```

Ugyanez a (messze nem teljes) ontológia az OWL XML változatában alább olvasható. Figyeljük meg az ontológiában az XML tagok attribútumait, amelyek vagy az RDF, vagy az XML névterekből származnak. Jelentősen könnyebb az olvasása, (mint a példában is), ha az internetes világ egyértelmű azonosítását szolgáló URI/IRI azonosítókat (Universal/Internationalized Resource Identifier)-megadásokat kicseréljük kicsit egyszerűbb és beszédesebb azonosítókra.

```
<owl:ObjectProperty rdf:about=IRIhasSubject>
    <rdfs:label xml:lang="hu">hasSubject</rdfs:label>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about=IRIhasSubProp>
    <rdfs:label xml:lang="hu">hasSubProp</rdfs:label>
    <rdfs:subPropertyOf IRIhasSubject />
</owl:ObjectProperty>

<owl:Class rdf:about=IRISubject>
    <rdfs:label xml:lang="hu">Subject</rdfs:label>
</owl:Class>

<owl:Class rdf:about=IRIObject>
    <rdfs:label xml:lang="hu">Object</rdfs:label>
    <owl:equivalentClass rdf:resource=IRISubject/>
</owl:Class>
```

[94] http:/protege.stanford.edu, elérés: 28-Aug-2012.

```
<owl:NamedIndividual rdf:about=IRIJancsi>
    <rdf:type rdf:resource=IRIObject/>
    <rdf:type rdf:resource=IRISubject/>
    <rdfs:label xml:lang="hu">Jancsi</rdfs:label>
</owl:NamedIndividual>
```

Az OWL *funkcionális nyelvtanát* azért nevezik így, mert az egyes deklarációs elemek után a paraméterek zárójelbe téve olvashatók. Emiatt ez kicsit hasonlít a Prologra is, és jobban is olvasható, mint az XML változat, de az IRI-k használata még így is eléggé elbonyolítja a dolgot. A példákban az olvashatóság érdekében a többsoros IRI-ket rövid azonosítókkal cseréltük ki.

```
Declaration(ObjectProperty(IRIhasSubject))
AnnotationAssertion(rdfs:label
  IRIhasSubject "hasSubject"@hu)
Declaration(ObjectProperty(IRIhasSubProp))
AnnotationAssertion(rdfs:label
  IRIhasSubProp "hasSubProp"@hu)
SubObjectPropertyOf(IRIhasSubProp IRIhasSubject)
Declaration(Class(IRISubject))
AnnotationAssertion(rdfs:label IRISubject "Subject"@hu)
EquivalentClasses(IRISubject IRIObject)
Declaration(Class(IRIObject))
AnnotationAssertion(rdfs:label IRIObject "Object"@hu)
EquivalentClasses(IRIObject IRISubject)
Declaration(NamedIndividual(IRIJancsi))
AnnotationAssertion(rdfs:label IRIJancsi "Jancsi"@hu)
ClassAssertion(IRISubject IRIJancsi)
ClassAssertion(IRIObject IRIJancsi))
```

A *Manchester nyelvi változat* a nevét a manchesteri egyetemről kapta, ahol a létrehozója dolgozik.[95] A változatot *keretalapúnak (frame-oriented)* vagy *jegyszerkezetesnek (feature structured)* is nevezik. A zárójeleket is eldobja, és helyette az egyes fogalmak meghatározásakor behúzást használ: a behúzással a fogalmat megadó jegyeket jelöli.

```
ObjectProperty: IRIhasSubject
    Annotations: rdfs:label "hasSubject"@hu

ObjectProperty: IRIhasSubProp
    Annotations:
        rdfs:label "hasSubProp"@hu
    SubPropertyOf: IRIhasSubject

Class: IRIObject
    Annotations:
        rdfs:label "Object"@hu
    EquivalentTo: IRISubject
```

[95] http://www.w3.org/2007/OWL/wiki/ManchesterSyntax elérés: 13-Jan-2013.

```
Class: IRISubject
    Annotations:
        rdfs:label "Subject"@hu
    EquivalentTo:
        IRIObject

Individual: IRIJancsi
    Annotations:
        rdfs:label "Jancsi"@hu
    Types: IRIObject, IRISubject
```

A modern OWL szerkesztő környezetek (Protégé, TopBraidComposer[96]) általában mindegyik nyelvtant képesek elolvasni.

2.8.2 Az ontológia-építés alapjai

A csúcsontológiák fogalmai gyakran annyira elvontak, hogy legtöbb fogalmuknak nem is képzelhető el megjelenési formája, példánya. Az efféle fogalmakat szokás *partikuláréknak* is nevezni. A konkrétabb ontológiaépítési réteg fogalmainak létezhet példánya: az efféle fogalmakat *univerzáléknak* nevezik.

Az ontológiákra vonatkozó másik alapvető kérdés az *ontológia dimenziószáma*. Statikus ontológiákat szokás *3 dimenziós*, vagy *3D ontológiáknak* is nevezni. Ezek az idő fogalmát nem tartalmazzák, és feltételezik, hogy minden ábrázolt dolog vagy fogalom időfüggetlen, örök, legalábbis az ontológia és az általa vezérelt szoftver időléptékében. Ezt a véleményt szokás *endurant* – időfüggetlen, v. időtálló álláspontnak is nevezni. Az ezzel szemben álló, úgynevezett *perdurant* – kissé filozofikusan megalapozott – nézet szerint a világon semmi sincs, ami örök lenne, mindent csakis egy időbeli ablakon keresztül szemlélhetünk, és ez az időablak minden ontológiai fogalom rögzített tulajdonsága. Az ezen nézet alapján szerkesztett ontológiákat szokás *4 dimenziós*, vagy *4D ontológiáknak* is nevezni. A 3D fogalmak másik neve: *folyamatos*, vagy *continuant*, míg a 4D fogalmaké *előforduló*, vagy *occurent*.

A két szemlélet közötti egyfajta kiegyezés megkülönbözteti az „örök” fogalmakat, amely főként az elvont, elképzelt, nem fizikailag létező fogalmak, illetve a vizsgált világ szemszögéből állandónak tekinthető fizikai objektumokat jelenti. Ezek esetében az időfüggés fel sem merül. Ezeket 3D-ben, a többit pedig 4D-ben modellezi, és az egész megközelítést emiatt *3.5D ontológiának* nevezik. Ehhez hasonló hozzáállást követ a korábban már említett SUO ontológia.

2.8.2.1 Általános célú csúcsontológia fogalmai

Egy csúcsontológia fogalomrendszerét több megvalósult ontológiában összeállították már úgy, hogy egymás eredményeit átveszik, és legalábbis a csúcsfogalmak szintjén az egyes projektek fogalomrendszere nagyon hasonló. A DOLCE ontológia csúcsfogalmai (táblázatba szervezve és a fogalomszerkezetet behúzással jelölve) a következők:[97]

---

[96] http://www.topquadrant.com/products/TB_Suite.html, elérés: 31-Jan-2013.

[97] Claudio Masolo-Stefano Borgo-Aldo Gangemi-Nicola Guarino-Alessandro Oltramari-Luc Schneider: The WonderWeb Library of Foundational Ontologies Preliminary Report [WW03]

Név	Rövi-dítés	Magyarázat
Entity	ALL	A világ minden objektumát és fogalmát magába foglaló csúcsfogalom
Abstract	AB	Elvont fogalmak
Region	R	Nem önálló objektum, értékkészlet
TemporalRegion	TR	Időmeghatározás: pl. a múlt hétfőn / a keresztre feszítés utáni harmadik napr
TimeInterval	T	Időszakasz
PhysicalRegion	PR	Fizikai mennyiség értékkészlete
SpaceRegion	S	Térbeli kiterjedés meghatározása
NonPhysicalRegion	NPR	Nem fizikai mennyiség értékkészlete
Endurant	ED	Állandó/időfüggetlen fogalmak
Quality	Q	Minőség
TemporalQuality	TQ	Időbeli minőség, pl.: valaminek az időtartama, felfutási ideje, felezési ideje
PhysicalQuality	PQ	Fizikai minőség, pl.: valaminek a színe, az önsúlya
NonPhysicalQuality	NPQ	Nem fizikai jellemző, pl.: részvényárfolyamok
Substantial	SB	Anyagszerű
PhysicalSubstantial	PSB	Anyagszerű fizikai
NonPhysicalSubstantial	NPSB	Anyagszerű nem-fizikai
NonPhysicalObject	NPOB	Nem-fizikai objektum
MentalObject	MOB	Szellemi termék, objektum
SocialObject	SOB	Társadalmi objektum
Agentive-SocialObject	ASO	Öntevékeny társadalmi objektum
SocialAgent	SAG	Társadalmi szereplő
Society	SC	Társaság
NonAgentive-SocialObject	NASO	Nem öntevékeny társadalmi szereplő
Perdurant/ Occurence	PD/ O	Időfüggő/Előfordulás
Event	EV	Esemény
Achievement	ACH	Esetleges eredmény, pl. elérni a vonatot, távozni, meghalni, stb.
Accomplishment	ACC	Teljesítmény, eredmény: pl. konferencia, hegymászás, előadás
Stative	STV	Állapotok és folyamatok
State	ST	Állapot, pl. nyitott, boldog, piros
Process	PRO	Folyamat, pl. mozgás, melegedés, fejlődés
Endurant	ED	Állandó/időfüggetlen fogalmak

2.8.3 Ontológiai tervminták

A tervezési minták fogalmát egy építész és matematikus vezette be, ami elsősorban építészeti fogalmakra és szerkezeti megoldásokra vonatkozott. Christopher Wolfgang Alexander bécsi születésű, de angol neveltetésű. Itt végzi az egyetemet is, de a doktorátust már Amerikában, a Harvard Egyetemen szerzi meg. Alexander klasszikus több-tudományszakos megközelítésű elme és tudós, aki több kutatási projekt befejezése után írja meg „A Pattern Language: Towns, Buildings, Construction" c. könyvét, amely a Chomsky-féle generatív nyelvelmélethez hasonló eszközökkel tárgyalja a településfejlesztési és az építészeti szerkezetek létrehozását.

A Mintanyelv (Pattern Language) fogalmat szintén ő vezette be a tudományos életbe,[98] amit több tudományszakon előszeretettel használnak. Többek között a számítástechnikában is széles körben elterjedt, ahol a fogalmat elsősorban a szoftver tervelemek újrafelhasználására alkalmazták. Erich Gamma, Richard Helm, Ralph Johnson és John Vlissides szerzők a „négyek bandája" néven is ismert. Az általuk összegyűjtött szoftver tervminták könyv formájában[99] már számtalan kiadást és fordítást megértek, másrészt egyes CASE eszközök katalógus-szerűen is tartalmazzák őket, amit a mérnök pár kattintással beemelhet a szoftver tervébe.

A mintanyelvek az ontológiaépítésre vonatkoztatva is elterjedtek, erre irányuló kutatásokat A. Gangemi olasz tudós és munkatársai folytattak a római Institute of Cognitive Science and Technology (ISTC-CNR) kutatóintézet Laboratory of Applied Ontology (LOA) laboratóriumában, aminek eredményeképpen bevezették az Ontológiai Tervminták (Ontology Design Pattern, ODP) fogalmát. Mindezek számára a (http://www.ontologydesignpatterns.org) honlapon egy Web alkalmazást is létrehoztak a tervminták kezelésére és karbantartására, valamint a javasolt új tervminták elfogadására.

Minták leírására Alexander még a [Környezet-Probléma-Erőviszonyok-Megoldás] négyes dimenzióit alkalmazza. A minta egy adott környezetben felmerülő problémára az adott erőviszonyok mellett adott olyan megoldás, amelyet a létrehozók javasolnak, és amely bizonyítottan és kipróbáltan megfelelő és hatékony. Gangemi, Valentina Presuttival közösen írt munkájában az ontológiaépítésre az eredeti négyes módosított és kibővített változatát használja.[100] Egy ontológiaépítési mintát ebben a rendszerben a következő információk írnak le, és egy új mintát is ezekkel az adatokkal kell jellemezni:

- Szándék/cél: A problémakör pár mondatos leírása.
- Szakterület: Milyen szakterületeken alkalmazható a minta?
- Megválaszolt kérdés: A minta milyen kérdésekre szolgálhat válasszal?
- Módszer: A megoldási módszer pár mondatos leírása.
- Következmények: Mennyiben lesz újszerű vagy hasznos az az információs környezet, amit a minta megteremt?

---

[98] Nikos A. Salingaros: The Structure of Pattern Languages [Sa00]

[99] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns Elements of Reusable Object Oriented Software [GHJV94]

[100] A. Gangemi-V. Presutti: Ontology Design Patterns [GaPr09]

- Megoldás: a megoldás programkódja (pl. OWL nyelven), esetleg UML diagramja, illetve a megoldás elemeinek leírása.

A szerzők a cikkükben az ontológiatervezésre használatos mintákat (Ontology Patterns, OP) a következőképpen csoportosítják:

- *Szerkezeti minták (Structural OPs)*, amelyek az alábbiakat foglalják magukba:
    - *Logikai minták (Logical OPs)*. Ezek olyan logikai szerkezeteket jelentenek, amelyek a nyelv kifejezőerejének a szegényességével vannak kapcsolatban, vagyis javaslatokat adnak a kifejezőerő növelésére az adott nyelv lehetőségein belül. Jellemző példa lehet az OWL nyelven egy többoldalú reláció megfogalmazása, miközben az OWL csak kétoldalú relációkat enged meg.
        - *Logikai makrók*. Ezek a leggyakrabban együttesen előforduló logikai kifejezések rövidítésére szolgálnak (pl. az OWL `allValuesFrom` és `someValuesFrom` megszorítását igen gyakran együttesen használják).
        - Az *átalakítási szabályok* a teljes magasabb rendű logikai kifejezések szegényesebb formalizmusba való közvetlen átírását adják meg. Ilyen lehet pl. a már említett többoldalú relációkat OWL-ba átíró szabály.
    - *Felépítmény minták (Architecture OPs).* Az ontológia legnagyvonalúbb szerkezetét határozzák meg. A *belső* felépítmény minták rögzíthetik az alkalmazandó logikai mintákat, esetleg az alkalmazandó OWL változatot. A *külső* felépítmény minták a részontológiák (modulok) egymással való kapcsolatát, az importálást és egy ontológiai hálózat elemeivel való együttműködést rögzítik.
- *Következtetési minták (Reasoning OPs)*. A következtetési minták a választott logikai minták felett végzendő következtetési műveletekre vonatkoznak. A legfontosabb ilyen minták:
    - *Osztályozás (classification)*, azaz a teljes osztályszerkezet megállapítása a megadott megszorítások alapján
    - *Öröklődés (inheritance)* kezelése, az öröklődésből adódó következtetési feladatok elvégzése
    - *Leszármazás (subsumption)*, azaz annak eldöntése, hogy egy osztály egy másik osztály leszármazottja-e
    - *Tárgyiasítás (materialization)*, azaz egyes gyakori vagy kézenfekvő, de számításigényes következtetések eredményének közvetlen tárolása a tudástárban.
    - *Név-hozzárendelés (de-anonymizing).* A következtetési lépések során keletkező névtelen objektumok és eredmények névvel való ellátása…
- *Megfeleltetési minták (Correspondence OPs)* alkalmazásakor valamilyen forrás-modellből ugyanazon vagy másik modellbe képezünk le. A megfeleltetési minták a következőképpen kategorizálhatók tovább:
    - *Visszamodellezési minták (Reengineering OPs)*. Ezek különböző jellegű forrásokból történő, teljesen vagy félig automatikus ontológia-előállításra

alkalmasak. Az egyes forrásoktól függően két alapvető kategóriát különböztethetünk meg:

- *Idegen séma visszamodellezésről* beszélünk akkor, ha másik ontológia-leíró nyelven megfogalmazott ontológiából szeretnénk a célmodell szerinti ontológiát előállítani. Pl. ilyen fordulhat elő, ha a KIF nyelven megírt SUMO ontológia OWL alakra történő átalakításakor.
- *Ontológia-tisztítási (refactoring)* mintákról is beszélhetünk. A szoftver-technológiából ismert eszközök alkalmazása ontológiák szerkesztésére, módosítására és átalakítására.

o *Leképezési minták (Mapping OPs).* A leképezési minták az ontológiai elemek között legalapvetőbb értelmezési viszonyt számítják ki, ezek: *egyenértékűség (equivalence), tartalmazás (containment), átfedés (overlap),* illetve ezek ellentétei.

- *Megjelenítési minták (Presentation OPs).* A megjelenítési minták az ontológia felhasználók általi használhatóságára összepontosít.
  - o *Elnevezési minták (Naming OPs).* Ezek az egyes ontológiai elemek (névterek, fájlok, osztályok, tulajdonságok, stb.) névadására vonatkoznak, céljuk az elnevezések egyöntetűségét és könnyebb olvashatóságát elősegíteni. Erre példa lehet az URI azonosító képzése az internetes URL-ből és egy helyi azonosítóból.
  - o A *jelzetkészítési minták (Annotation OPs)* a jelzet-tulajdonságok (annotation properties) létrehozásának és használatának módjára adnak javaslatot.
- A *lexikai-nyelvi minták (Lexico-Syntactic OPs)* azon leképezési szabályokat tartalmazzák, amelyek egyes természetes nyelvi fordulatok és nyelvtani szerkezetek ontológiai szerkezetekbe történő leképezését írják le.
- *Tartalmi minták (Content OPs, CPs).* A tartalmi minták inkább *fogalmiak*, mint logikaiak. Míg a logikai minták a problémákat az adott ontológia tartalmától függetlenül, elméleti irányból közelítik meg, addig a tartalmi minták egy adott konkrét fogalomrendszerhez viszonyulnak. Tekinthetjük úgy is, hogy a tartalmi minták alkalmazzák a logikaiakat, így egyfajta konkrét építőelemül szolgálhatnak.

A 'www.ontologydesignpatterns.org' honlapon található ontológiaminták igen számosan vannak. Tekintsünk meg ezek közül néhány jellemző példát.

2.8.3.1 Az N-oldalú viszony logikai tervminta[101]

Szándék/cél: A leíró logikák alapvetően kétoldalú (bináris) kapcsolatok leírására használhatók. Sok esetben mégis szükséges az, hogy többoldalú viszonyokat is ábrázoljunk.

Szakterület: Általános logikai tervminta. Minden szakterületen szükséges többoldalú viszonyokat is ábrázolunk.

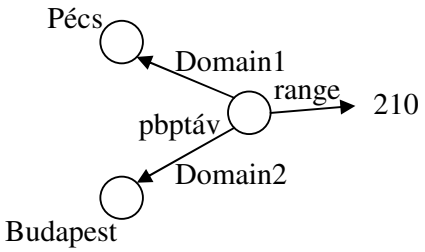
[101]Rinke Hoekstra. Ontology Representation – Design Patterns and Ontologies that Make Sense [RH09]

Kérdés: Mi az értéke egy többváltozós függvénynek? Milyen értékeket vehet fel egy többboldalú reláció, ha az egyes változóit/argumentumait rögzítjük?

Módszer: A többboldalú viszonyt önálló fogalomként tároljuk, melynek egyes példányait elnevezzük (de-anonimizing). Ezt az el is nevezett többboldalú példányt kétoldalú tulajdonságok kötik a reláció egyes argumentumaihoz. A többboldalút felépítő kétoldalú tulajdonságokat `domain1..N` (értelmezési tartomány), illetve `range1..N` (érték) csoportokra osztjuk fel, a felosztás a tervező intuitív szándéka alapján történik. (Az N argumentumú függvények az N+1 argumentumú relációk részfogalmai.) Függvények esetén a tartomány-érték felosztás kézenfekvő.

Következmények: Minden egyes rögzített paraméterszámhoz egy külön tervminta tartozik.

Megoldás: Az alábbiakban OWL nyelven és grafikusan is bemutatjuk a `TernaryRelation` és a `BinaryFunction` összefüggését, és a fentiek alkalmazását egyes települések közötti távolságok megadására.

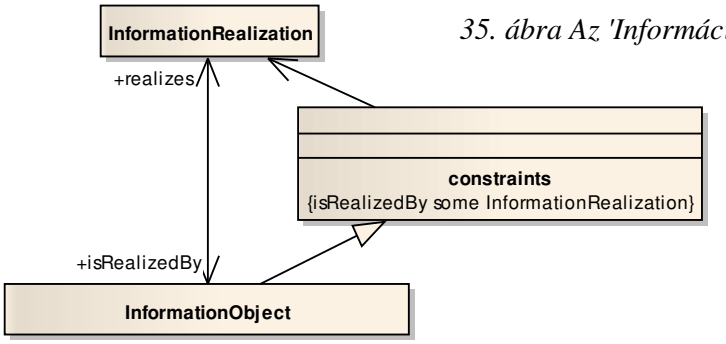



```
TernaryRelation ⊆ Relation
TernaryPredicate ⊆
TernaryRelation
BinaryFunction ⊆ TernaryRelation
TernaryRelation(distance)
distance(pbptáv)
domain1(pbptav,pécs)
domain2(pbptav,budapest)
range(pbptav,210)
```

34. ábra Kétváltozós függvény és egy példánya

2.8.3.2 Az 'Információ hordozó' tartalmi tervminta

Az 'Információ hordozó' (InformationRealization) tervminta egy információegység (pl. regény, dal) és annak rögzítése, tárolása (pl. könyv, hanglemez, mp3, stb.) közötti kapcsolatot mutatja be.



35. ábra Az 'Információ-hordozó' tervminta

Szándék/Cél: egy információ egység és annak fizikai hordozója közötti különbség és összefüggés bemutatása

Szakterület: Tartalomkezelés és -szolgáltatás

Kérdés: Melyik fizikai hordozó tárolja az adott információs egységet? Melyik információegység van egy adott információhordozón tárolva?

Módszer: Az `InformationRealization` és az `InformationObject` fogalmak közötti kétoldalú viszony (`realizes`/`isRealizedBy`) segítségével.

Megoldás: az ábrán látható a tervminta UML modellje, amely az alábbi modellelemeket mutatja be:

- `InformationObject`: az információegység, mint szellemi termék (regény, zenemű, fénykép)
- `InformationRealization`: információhordozó, mint fizikai tárgy (nyomtatott könyvpéldány, fényképmásolat)
- `realizes`: az információhordozó és a hordozott információegység közötti kétoldalú kapcsolat
- `isRealizedBy`: az információegység és hordozója közötti kétoldalú kapcsolat (a `realizes` inverze)

2.8.3.3 Az 'Időfüggő szerepkör' tartalmi tervminta[102]

Szándék/cél: Az időfüggő szerepkör minta a személyek, az általuk játszott szerepek közötti időfüggő, 3 oldalú hozzárendelést valósítja meg. Ilyen pl. a következő állítás: „Gróf Batthyányi Lajos 1848. március 23. és október 2. között Magyarország miniszterelnöke volt."

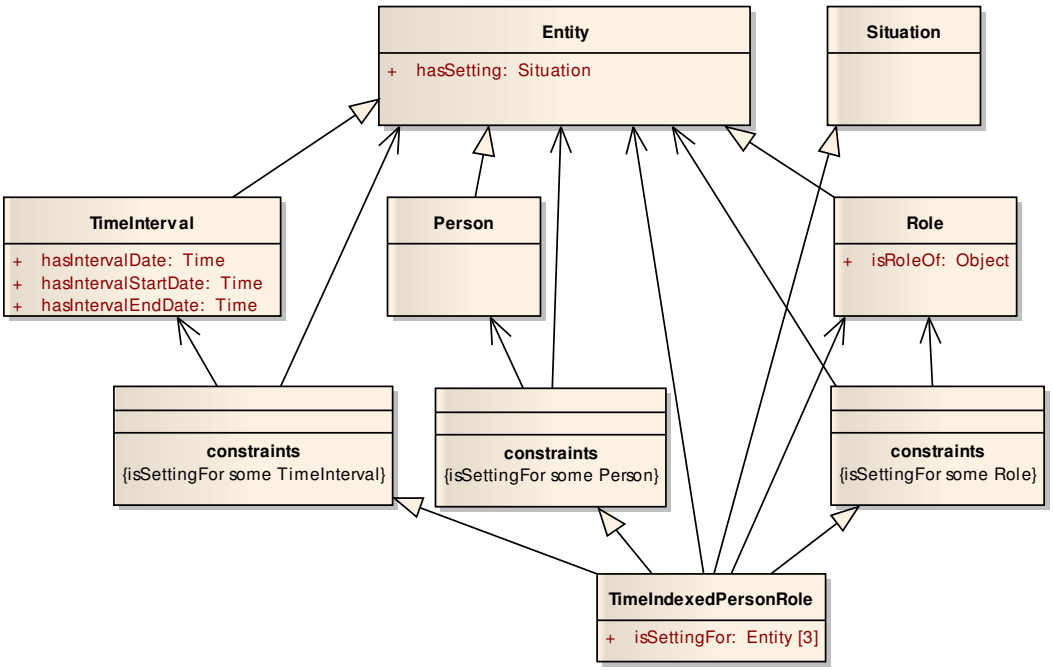
Szakterület: természetes személyekkel összefüggő tudástárak (közigazgatás, személyügyek, művészet).

Kérdések: Ki volt egy adott időszakaszban valamilyen szereplő? Mikor volt egy adott személy valamilyen szereplő?

Módszer: A fogalom maga egy 3 oldalú viszony, amelynek argumentumaira további (exisztenciális) megkötések érvényesek.

[102] Gangemi-Presutti: Ontology Design Patterns [GaPr09]

36. ábra Az időfüggő szerep tartalmi tervminta



Megoldás: Az ábrán látható a tervminta UML modellje, amely a következő modellelemeket mutatja be:

- `TimeIndexedPersonRole`: A 3 oldalú relációt rögzítő szituációs, azaz időfüggő fogalom. A fogalom részhalmaza az argumentumokat leíró névtelen megszorításfogalmaknak, amely fogalmak a paramétertípusokkal vannak kétoldalú kapcsolatban.
  - `isSettingFor: Entity[3]` típusú tulajdonság a 3 argumentum megcímzésére.
- `Entity`: entitás - bármilyen valós, lehetséges vagy csak elképzelt dolog, amiről valamit egyáltalán mondani akarunk. A jelenleg fontos tulajdonságai:
  - `hasSetting`: az `isSettingFor` inverze
- `Person`: személyek, akik lehetnek természetes (emberi) személyek, de jogi személyiségek is
- `Role`: a személy által elvállalt/játszott szerep, amely az illetőt az adott időszakaszban minősíti
- `hasRole`: a személyek és szerepeik közötti kapcsolat
- `isRoleOf`: a szerepek és az entitások közötti kapcsolat (a `hasRole` inverze)

2.8.4 Modell- és ontológiavezérelt alkalmazásépítés

Az ontológiavezérelt alkalmazásépítés lényege a modellvezérelt alkalmazásépítéséhez hasonló.[103] A modellvezérelt felépítményt kétszintű szoftver felépítménynek is nevezhetjük, hiszen a szoftver a hagyományos értelemben vett adatszerkezeteken kívül az adatok modelljét is adatszerűen tartalmazza, és egyes egyöntetűen megvalósítható műveleteket a tárolt modell függvényében, annak vezérlete alatt, azt értelmezve valósít meg.

A hagyományosan készített egyszintű szoftverek olyan korlátokat állítanak a saját fejlődésük elé, melyek egy efféle, kétszintű felépítménnyel feloldhatók:

1. *Nehéz változtathatóság*: a modellszint rögzített, a legkisebb újabb bővítési elképzelés esetén is modellbővítés, a megvalósítás nyelvén újabb programcsomagok megírása, lefordítása és szerkesztése szükséges.
2. A modellek programnyelvekbe történő leképezésének használatos megoldásai esetenként *túlságosan nehézsúlyúak*, de esetenként korlátokat is állítanak (pl. objektum orientált nyelvek esetében a többszörös öröklődés tiltásával). Ennek következményeképpen már a tervezés folyamán ügyelni kell egyes megvalósítási részletekre is, pl. ha a tervből a programkódba átvivő gépi leképezés nem szolgáltat elegendően „pihesúlyú" programobjektumokat, akkor már a modellben sem építhető fel a probléma igényeinek megfelelően aprólékos osztályszerkezet. Vagyis már a tervezés során is figyelembe kell vennünk egyes, a feladat *belső lényegén és összefüggésein túlmutató szempontokat*.
3. *Egyedileg meg kell valósítani* minden olyan eljárást, amelyek az egyes egymással is összefüggő modell- ontológiabeli osztályok elemeire vonatkozóan *adatokat gyűjtenek össze vagy alakítanak át*, noha ezek egyöntetű, magas szintű adatelérő vagy -átalakító nyelven történő megfogalmazása kézenfekvőbb és egyszerűbb lehetne. Ilyen nyelv az Object Query Language (OQL), amelyet az SQL objektumközpontú kiterjesztésének is tekinthetünk.[104] Többek között a már említett SILK projektum is az OQL-t választotta modellvezérelt adatelérő és átalakító nyelveként.[105]
4. Ha mégis a *modellszint tárolásához hasonló adatszerkezetek* ábrázolása szükséges a szoftverben, akkor az is *egyedi megoldással történik*, ami egy egységes megoldáshoz képest lényegesen megnövelheti a fejlesztés befektetésigényét, és növeli a hibakockázatot.

A kétszintű vagy modell-, illetve ontológia vezérelte szoftverek mindezen hátrányok kiküszöbölésére törekszenek. Ezekben a program természetesen ugyanannyira kötött, mint más szoftverekben. A szoftver által kezelt adatok azonban két részre különíthetők el. Az egyik a *modell-, ontológiaszint vagy metaszint,* a másik pedig a tényleges adatok szintje. Ezekre a következők a jellemzők:

- A *modellszinten* tároljuk a világ vizsgált részének ontológiai modelljét, és ez vezérli az adatszintet. Ide tartozhatnak egyedileg beprogramozott függvények (a

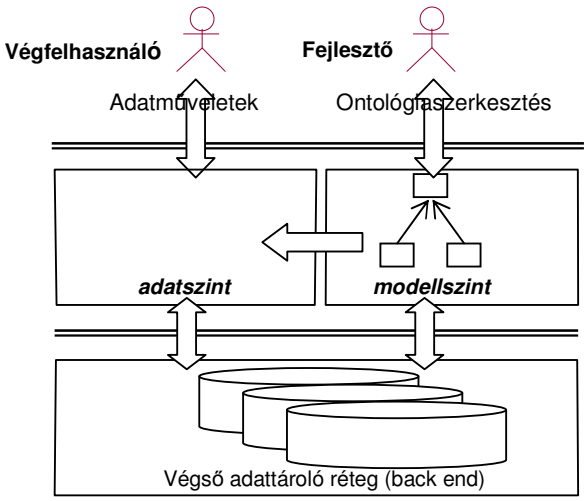
[103] Kilián Imre: Modellvezérelt szoftverek készítése [Kil08]

[104] R.G.G.Cattell-D.Barry és mások: The Object Data Standard: ODMG 3.0 [ODMG99]

[105] L.Badea-D.Tilivea-A.Hotaran-Y.Polet-N.Chancevrier-D.Parents-X.Denis: Description of the SILK Mediator tools [SILK01]

modellek osztályfüggvényei vagy módszerei) is, amelyek az adatszintről önműködően hívódnak meg, de erre a képességre nincs mindig szükség.

- Az *adatszinten* vagy *példányszinten* folynak azok az üzleti modellnek megfelelő adat-átalakítások, amelyek a háttértár rétegét a megjelenítés rétegével összekapcsolják. Ezek a műveletek nincsenek közvetlenül beprogramozva, hanem csak néhány tipikus és általános művelet van megvalósítva. Az adatszint nyitott annyiban, hogy a modellszintről vezérelve objektumpéldányokat hozhatunk létre, és ugyanígy végrehajthatjuk a rajtuk definiált műveleteket és a modellszinten beprogramozott függvényeket is (dinamikus típuslekötés és hívhatóság)
- Kétszintű programfelépítmény létrehozása során technikai jellegű dolog ugyan, de igen fontos említést tenni a programfutást túlélő (perzisztens) objektumok tárolását megvalósító végső adattároló (back end) rétegről is. Itt tároljuk az adatszint objektumait, de a modell-leíró elemeket is.



37. ábra Kétszintű szoftverfelépítmény

A rendszerhez a *végfelhasználó az adatszinten* keresztül kapcsolódik. Itt modellvezérelt általános célú eszközök állnak rendelkezésre, amelyek megjelenítő felületen keresztül működtethetők.

A *modell-/ontológiai szinten a modellező/fejlesztő* elsődlegesen egy *ontológiaépítő és karbantartó eszközön* keresztül kapcsolódik a rendszerhez. Ez az eszköz lehet grafikus jellegű, de mindenképpen lehetővé kell tennie a modellek betöltését, mentését, és osztályfüggvények (módszerek) létrehozását. Vagyis egy olyan nyelvre és integrált fejlesztői környezetre van szükség, amelyik programkód szerkesztésére, majd dinamikus fordításra-betöltésre is alkalmas (Java, Prolog, stb.).

Az adat és a modellszint ilyen kettéválasztásának a legfőbb haszna, hogy az alkalmazás létrehozása során elsődlegesen a célalkalmazás *logikai/formális modelljére, illetve a cél-szakterületet leíró ontológiára összpontosíthatunk*, így a programozási részletkérdésekkel nem kell törődnünk. Az adatszint maga is adhat eszközöket egyéb alkalmazás- és környezetfüggő dolgok (pl. felhasználói felület, tesztkörnyezet) automatikus vagy félautomatikus elkészítésére, de mindez a modellszinttel semmilyen módon nem keverhető össze.

Modellvezérelt rendszerek esetében tehát a célszoftverek adatai vagy azok egy része nem rendelkeznek kötött modell-leírással, hanem a modell maga is a program adatainak része, amely a többi adat szerkezetét írja le. A lehetséges modelleket és ontológiákat leginkább a *metamodelljük* segítségével írhatjuk le, amely immár lehet *rögzített* is, és amely igen erősen meghatározhatja a ténylegesen felismert és értelmezett modellek, valamint az azok alapján feldolgozható adatok és adatműveletek körét.

2.8.4.1 Tartalomvezérelt (szemantikus) szoftver-rendszerek

A szemantikus műveletek és megvalósításaik köre hiába kutatási terület ma is, az első fecskék már kiröppentek, szoftver termékekkel találkozhatunk, amelyek efféle műveleteket végeznek, szabványokat rögzítettek a témában, és a kutatásokat hasznosító különböző (spin-off) cégecskék alakultak.

A körben indított legerősebb projektum a Szemantikus Világháló (Semantic Web), amelyről részletesebben a következő szakaszban szólunk.

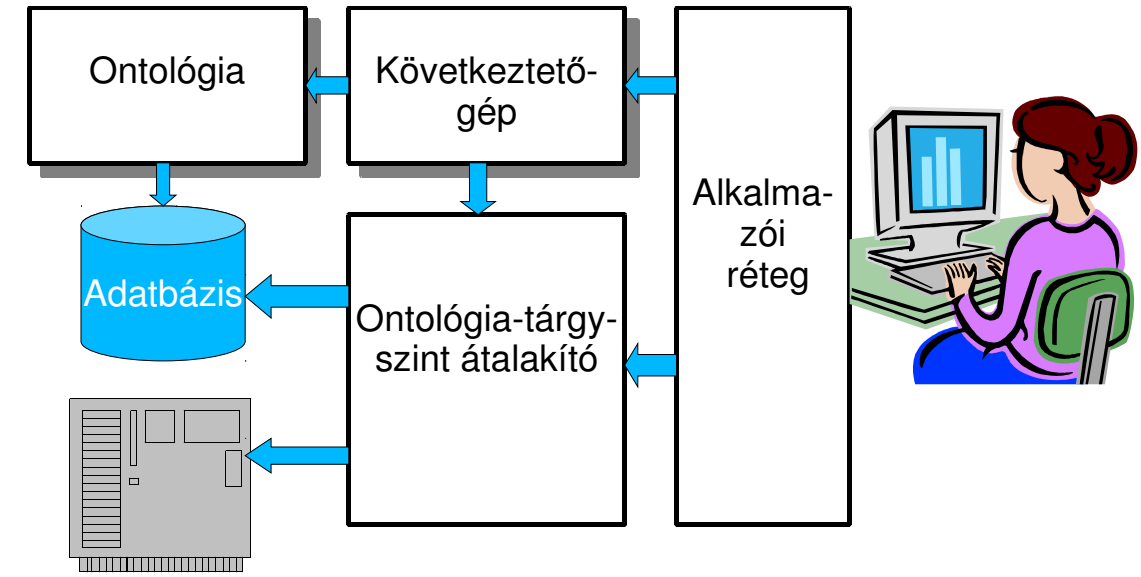
A tartalomvezérelt szoftvereknek a következő jellemzőik vannak, illetve a következő esetekben célszerű őket használni:

- Ha a szoftvernek *gazdag szakterületi fogalomrendszert* kell kezelnie
- Ha a fogalomrendszer *nem rögzített*, hanem a szoftver modell/ontológiai szintjén adatként van kezelve
- Ha a fogalomrendszer esetleg futás közben is *módosulhat*
- A fogalomrendszerre épülve a puszta vázon túl következtetésekre alkalmas *üzleti szabályok* is léteznek, amelyeket alkalmazni kell
- A szoftver *következtetésekre is képes*: a kezelt fogalomrendszerből kiindulva a rendszerben kezdetben nem ábrázolt tények levezetésére is alkalmas.

Egy általános tartalomvezérelt szoftver logikai vázlatát az alábbi ábrán tekinthetjük meg az egyes szoftver-összetevők szerepének és működésének leírásával. A leírásban részletezzük az egyes összetevők skálázási lehetőségeit is, vagyis azt, hogy mely összetevőket milyen ügyfél-kiszolgáló felépítményben célszerű megvalósítani:

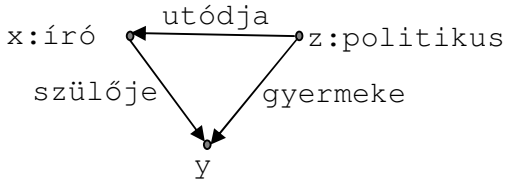
- Az alkalmazói rétegben az *alkalmazói program* található, amely megvalósítja a konkrét feladat kitűzte műveleteket. Ez ráépülhet egy klasszikus réteges szoftver-felépítményre is, esetleg kapcsolódhat már létező szoftverekhez is. Az alkalmazói réteget megvalósíthatjuk vastag-ügyféllel teljes egészében az ügyfél oldalon, de valószínűbb az, hogy az alkalmazás egy része, vagy teljes része a kiszolgáló oldalra kerül - a vékony-ügyfeles megközelítés szerint.
- A háttér rétegben, mint máskor is, elsősorban egy *relációs adatbázis* található. Az adatbázisnak két feladata is lehet:
    - Egyrészt az ontológiához kapcsolódóan tartalmazza az *ontológia szintjén* tárolt *közismert példányadatokat* (pl. Petőfi Sándor születési adatait), amire a közhely-tudáselemek, vagyis a mindenki vagy majdnem mindenki által ismert tudás használatára van szükség.
    - Az ontológia másrészt tartalmazhat *alkalmazói példányadatokat* is. Ezek az adatok csak az adott alkalmazás által kezelt példányadatok. Például egy könyvkiadó tárolni akarja Pityi Pál, egy magyar ponyvaszerző adatait, aki bár elismert kortárs közíró, mégsem mondható annyira közismertnek, mint Petőfi. Tekintve azonban, hogy egy neves kiadó nem csak klasszikus nagyságok műveiből élhet meg igazán, az ilyen szerzők adatait a háttéradatbázisból veszi.

38. ábra Tartalomvezérelt szoftver logikai vázlata



- Az *ontológia* vagy más néven *tudástár* a szoftver működését szabályozó belső logikai összefüggéseket tárolja, melynek jelentősége és szerkezete a jelen értekezés legfőbb témája. Az ontológia közismereteket tartalmaz, ezért olyan kiszolgálói rétegben kell elhelyezkednie, amely efféle ismereteket tárol és kezel. Mindamellett, mint korábban már említettük, az ontológia importálhat is más ontológiákat, így maga a tudástár is megvalósítható elosztott módon, pl. az Internet egyes szakterületekkel foglalkozó kiszolgálóin.
- A *következtetőgépet* (inference engine), mint működő szoftvert elválasztjuk magától a tudástártól, amelynek elsősorban a hatékony tárolásban van szerepe. A szemantikus technológiák viszonylagos fejlettségére utal az, hogy a következtetőgép programfelületére vonatkozólag már tényleges szabványok is léteznek, ilyen területen tevékenykedő cégek pedig következtető szolgáltatást hirdetnek – a fejlettebbek nyilván fizetősek, de éppenséggel ingyeneset is lehet találni. Mindazonáltal a webszolgáltatásként történő megvalósítást inkább csak reklám célokra tudjuk elképzelni. Ha ugyanis egy szoftver gyakoribb következtetési igénnyel lép fel, akkor a hálózaton jelentkező többletköltségek miatt nem lehet célszerű a következtetőgépet az ontológiától szétválasztani, és külön gépre, nagy távolságra elvinni.
- Az *ontológia-tárgyszint átalakító* legfontosabb feladata az, hogy a modellfüggő műveleteket megvalósítsa. Ezért hozzáférése van az ontológiához és a következtetőgéphez is. Az innen vett modell- és ontológiai adatok vezérlik a konkrét adatműveleteket. A modellfüggő műveletekre a legjellemzőbb példa a különböző *lekérdező nyelvek értelmezése*:
    - Ha az ontológia egyfajta objektumközpontú modell (pl. UML vagy részhalmazában megfogalmazott modell), akkor a lekérdezésre az Object Data Management Group *Object Query Language* (OQL) nyelvét

célszerű használni.[106] Megjegyezzük, hogy hasonló megoldást választott a SILK projektum is.



39. ábra Ciklikussá átalakítható lekérdezési háló

- Ha az ontológia az OWL nyelv DL változatában készült, akkor ezzel csereszabatos lekérdező eszközként a SPARQL–DL nyelvváltozatot használhatjuk.[107] A nyelv a relációs adatbázisok SQL nyelvéhez hasonló lehetőségeket biztosít OWL környezetben, vagyis egyszerű ÉS művelettel összekapcsolt lekérdező feltételeket képes ontológiák felett kiértékelni. Bár az OWL az adatbázisokénál lényegesen bonyolultabb szerkezetek modellezésére is képes, a SPARQL-DL nem fedi le az OWL adta lehetőségeket teljesen. Ez azt jelenti, hogy csak a legegyszerűbb következtetési műveleteket (pl. fogalom-példány vizsgálat és felsorolás) képes meghajtani, a bonyolultabbak kiértékelése (pl. ciklikus lekérdezések feloldása kérdés-átírással), már meghaladja a lehetőségeit. Alább olvasható a „ki az az x író, aki z politikus utódja, úgy, az író szülője és a politikus fia ugyanaz az y személy" lekérdezés példánygráfja. Ez – figyelembe véve az értelemszerű `gyermeke=szülője-1` OWL axiómát, – a szülője megfordítása után egy körkörös példánygráfot ad ki. (A kikötést például Esterházy Péter magyar író és nagyapja, Esterházy Móric gróf teljesítik, aki politikusként 1917-ben rövid időre az ország miniszterelnöke is volt)

2.8.4.2 Szemantikus Világháló

A Szemantikus Világháló (Semantic Web) fogalmát ugyanaz a Sir Timothy John Berners-Lee alkotta meg,[108] akinek nevéhez az Internet létrehozása is köthető. Az egyszerű internetes adatok és alkalmazások ugyanis merőben formálisan kezelik az adattartalmakat, vagyis azok formátumának kezelésén és ellenőrzésén túl nem próbálkoznak azok értelmezésével is. Így fordulhat elő az, hogy internetes háttérrel működő alkalmazások vagy csak nagyon szűk területre koncentrálnak, vagy pedig nagyon nagy hibaszázalékkal dolgoznak (a hibára a legjobb példa a keresőprogramok által felhozott rengeteg felesleges találat). A szemantikus technológiák alkalmazásától részben a hibák csökkentését, de a meglevő internetes alkalmazások egy újabb integrációs lehetőségét és infrastruktúráját is várják a projektum életre hívói.

A Szemantikus Világháló tulajdonképpen nem is egy konkrét projektum, hanem inkább egy általános platform, vagyis egymással lazán összefüggő projektumok

---

[106] R.G.G.Cattell-D.Barry és mások: The Object Data Standard: ODMG 3.0 [ODMG99]

[107] Evren Sirin, Bijan Parsia: SPARQL-DL: SPARQL Query for OWL-DL [SiPa07]

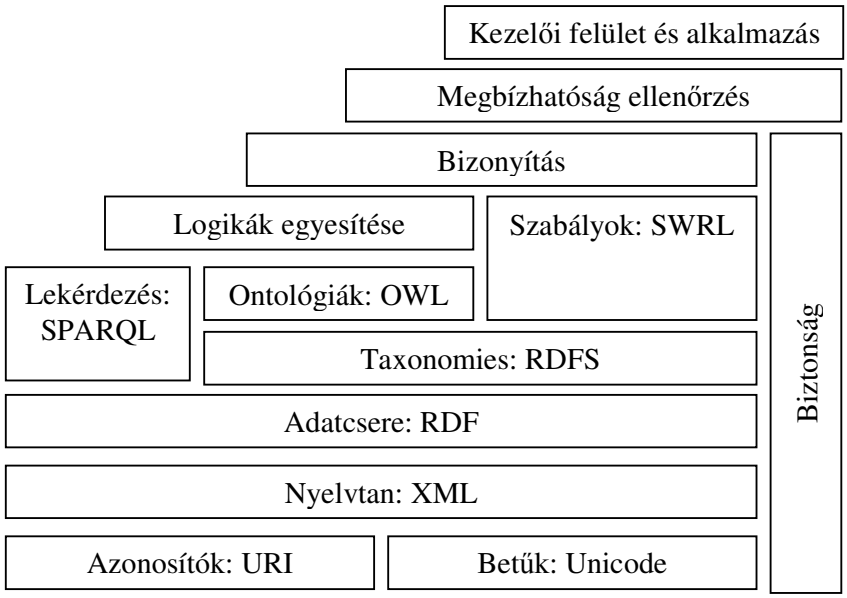
[108] T. Berners-Lee - J. Hendler - O. Lassila The Semantic Web
Scientific American Magazine. 2001. [BLHL01]

együttműködése. Az együttműködés alapja az, hogy internetes technológiai környezetben is használható rész-szabványokat dolgoznak ki, amelyek alkalmazását megállapodás szerint elterjesztik, átveszik és tiszteletben tartják.

A Szemantikus Világháló legalapvetőbb feltételezései:

1. Az Interneten elérhető adatok egyik fontos jellemzője, hogy *ellenőrizetlenek*, vagyis még a szöveges adatokról sem tudjuk, hogy ki, milyen (rész-) nyelven írta őket, és – hacsak nem vetünk be természetes nyelvi elemzési megoldásokat – akkor az adatok tartalmára vonatkozólag semmilyen útmutatásunk nincsen. Ez még akkor is igaz, ha az újabb HTML lapokhoz már megfelelő tagokkal a nyelvre, kulcsszavakra stb. vonatkozó információkat csatolnak. Másrészt az ellenőrizetlenség ellenőrizhetetlenséget is jelent, mihelyt a szöveges információkon túllépve kép-, hang- vagy mozgókép információkat kívánunk kezelni, hiszen az ezek tartalmát felismerő technológiák még mindenképpen kutatási fázisban vannak. Mindezek miatt az ellenőrizetlen tartalmi információk mellett az interneten elérhető adatokhoz ellenőrzött *metainformációkat* kell kapcsolni. Ezek kicsit hasonlatosak a programnyelvek kommentárjaihoz, de valamiféle igen feszes szemantikájú logikai nyelven (RDF, OWL) leírt információrészletek, amelyek ezért további következtetésre is alkalmasak.
2. A Szemantikus Világháló következtetési műveletekre képes. Ezen műveletek alapvetően a Világhálón található *erőforrásokra* (resources), és a rájuk vonatkozó különböző állításokra alapulnak. Az erőforrások olyan, egyedileg rögzített és azonosítható tudáselemek, amelyek az interneten megtalálhatók, azonosíthatók és a környezetükből gépi eszközökkel kiemelhetők (pl. egy személy vagy egy cég adatai).

40. ábra A Szemantikus Világháló rétegszerkezete



3. A következtetés során rögzített *tudásbázisokra (ontológiákra)* támaszkodhatunk. Ezek egyrészt egyfajta *közhelytudást* (commonsense) tárolnak, amelyek akár többlépcsősek is lehetnek, vagyis a teljesen általános közhelyek mellett egyes szakmák jellemző tudását más ontológiarészletek tárolhatják. Másrészt a

rögzített ontológiák alkalmazása egyfajta *közös szótár*, illetve *közös fogalomkészlet* rögzítését is jelenti.

A Szemantikus Világháló felépítését és működési elképzelését a legkönnyebben a fenti *rétegszerkezetből* (stack) érthetjük meg.[109] (A rétegszerkezet értelmezése kicsit pongyola/intuitív: van, ahol konkrét szoftver összetevők rétegzését, máshol protokoll beágyazást jelent.)

- A legalsó réteg dobozai: a Unicode betűkészlet alkalmazása lehetővé teszi egy dokumentumban bármelyik használatos íráskép együttes megjelenítését is. Az azonosítók (Uniform Resource Identifier) az Internet erőforrásait jelzik. Ezek két részből állnak: az ismert felépítésű URL (Uniform Resource Locator) részből, amely a hagyományos Internet térben jelöl meg valamilyen erőforrást, és az ahhoz lokálisan rendelt adatazonosítókból, mely egy egyszerű azonosító névből áll.
- A legalsó réteg betűkészletre és azonosító-szerkezetre vonatkozó szabvány az általános XML nyelvtan rétegében jelenik meg. Az XML (Extended Markup Language) egy ipari szabvány, melynek segítségével faszerkezeteket, bizonyos rásegítéssel (belső azonosítókkal és hivatkozásokkal) pedig körmentes gráfokat lehet leírni. A nyelvet igen széles körben alkalmazzák: többek között szoftverek konfigurálására vagy adatcserére ott, ahol ezt skaláris adatelemekkel vagy ezek sorozatával megoldani már nem kézenfekvő. Az internetes HTML szabvány XML irányába történő kiterjesztésén keresztül az XML a bonyolultabb webalkalmazások legalapvetőbb kommunikációs nyelve is.
- Az XML egy nyitott szabvány, amelynek egy adott alkalmazás egy megszorítását használja, vagyis az XML alapnyelvtant leszűkítő alkalmazói nyelvtannak megfelelő XML mondatokkal történik az adatcsere. A Szemantikus Világháló esetében az adatcserére az RDF (Resource Description Framework) alkalmazói szabványt használják. Ez XML nyelven létrehozott, „alany-állítmány-tárgy” hármasokból álló szerkezetek felépítését teszi lehetővé. A hármasok megnevezése intuitív: az állítmány valamiféle tulajdonságot vagy relációt, míg az alany és a tárgy ennek argumentumait van hivatott jelképezni. Mindazonáltal ezekhez maga az RDF alkalmazói nyelvtan nem fűz semmilyen konkrét jelentést.
- Az RDF egyik alkalmazása az RDFS (RDF Schema) szabvány. Ez az RDF általános nyelvtanához a fogalmi szerkezetekben használatos „részosztálya” és „tulajdonsága” viszonyokat teszi hozzá. RDFS alapon kisebb, egyszerűbb alkalmazói ontológiákat fogalmazhatunk meg, és szabványos ontológiákat rögzíthetünk.
- Az OWL (Web Ontology Language) az RDFS szabványra épülő, az RDFS nyelvén megfogalmazott ontológia-leíró nyelv, amely a leíró logikák (Description Logic) valamelyik kiépítését/kiterjesztését valósítja meg. Az OWL nyelvnek nagyon pontosan meghatározott a szemantikája, és bizonyító algoritmusai is léteznek.
- Az OWL maga elsősorban fogalomrendszerek leírására használatos. Konkrét és programozható következtetési folyamatok céljaira az OWL-ra épülő SWRL

[109] Marek Obitko: Introduction to Ontologies and Semantic Web [Ob07]

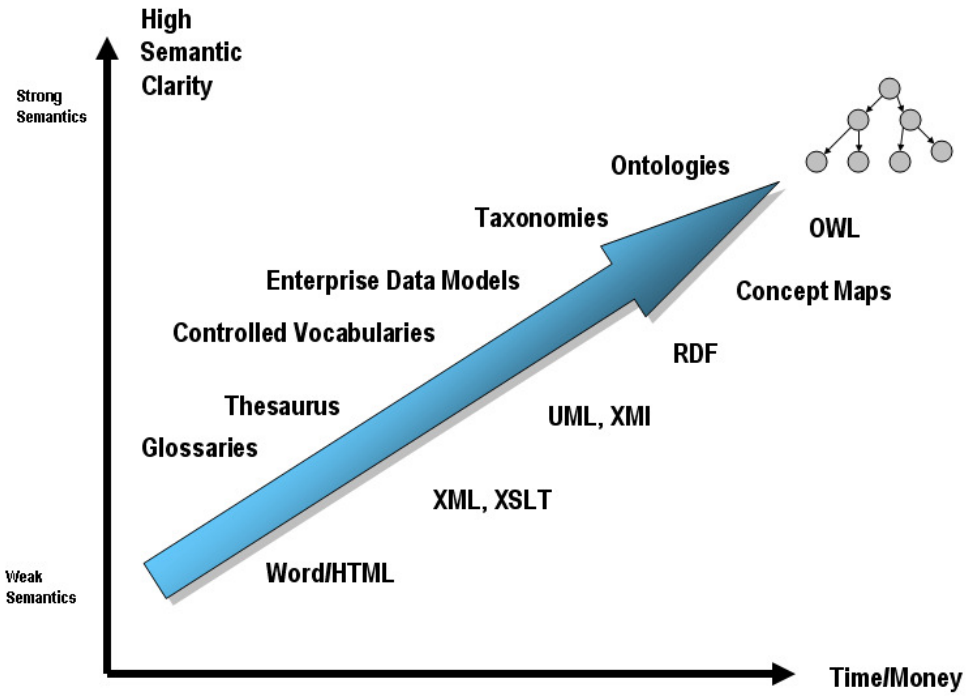
(Semantic Web Rule Language) szabálynyelv használatos. Ez kicsit túlmutat az OWL következtetési képességein: a matematikai pontosság elveszítéséért viszont kárpótol a közvetlen és kézenfekvő módon való programozhatóság képessége.

- A Simple Protocol and RDF Query Language (SPARQL) egy SQL-hez hasonló nyelvtanú lekérdező nyelv, amelyet részben, mint a nevéből is kiolvasható, RDF ontológiákhoz fejlesztettek ki, részben viszont az RDF-re épülő OWL ontológiák is lekérdezhetők vele. A nyelvet elsősorban példányinformációk kinyerésére használhatjuk olyan esetben, amikor bonyolultabb logikai következtetések elvégzésére nincs szükség.
- A 'Logikák egyesítése' doboz műveletei SPARQL lekérdezések és az OWL ontológiából kinyerhető példányadatok egyesítésére szolgálnak.
- Mindezekre épülhet a bizonyító és következtető algoritmus. Ez csak tiszta OWL-ra nézve egy pontosan megadott algoritmus, amelyet kiegészíthetünk a SPARQL lekérdezések és SWRL szabályalkalmazások eredményeinek figyelembe vételével is.
- Adatbiztonsági okokból egészen a bizonyítás és következtetés szintjéig alkalmazhatunk titkosítást.
- A megbízhatóság-ellenőrzési művelet a megbízható eredmények elérése végett már a műveletek bemenő adataira is különböző ellenőrzéseket hajt végre.
- A rétegszerkezet legfelső rétege az alkalmazói szoftver, amelyet (akár fizikailag külön gépre telepítve) a kezelői felület egészít ki.

A Szemantikus Web elképzelés tehát konkrét szoftver kifejlesztése helyett csak kicsit homályosan és általánosan megfogalmazott felhasználói célok megvalósítását lehetővé tevő „alulról-felfelé" stratégiájú eszközrendszer kifejlesztését célozza. A projekt ajánlásait követő szoftvereket szokásos Web 3.0 szoftvereknek is nevezni. Két fontos alkalmazási területe:

- A Facebook-hoz hasonló szociális hálókon működő alkalmazások, amelyek személyeket, ismeretségeket, társaságokat, valamint ezek érdeklődési köreit, tevékenységeit, partnereit tartalmazza – mind természetesen valamiféle ontológia fogalmai mentén rendezve.
- Több olyan élettudományi, biológiai és orvostudományi alkalmazás is létezik, amelyeken keresztül efféle témájú írások, kiadványok, dokumentumok keresőprogramját támogatják meg szemantikus képességekkel.

3 Nyelvtechnológia és szemantikus megoldások a jogi informatikában



41. ábra A szemantikus tisztaság valamint a ráfordítás összefüggése (Mike Bergman blogjából)

A jog területén a létrejövő termékek (törvények, ítéletek, kérvények, stb.) jelenleg még 100 százalékban írásosak, és az adott ország nyelvén íródtak. A jog nem a teljes nemzeti nyelvet használja, hanem annak egy igencsak kötött résznyelvét, amely gyakran nem is teljesen érthető még az anyanyelvüket csupán ösztönösen használók számára sem. Ennek legfőbb oka a jogi fogalomrendszer ismeretlen volta: bár a jogban – nyilván a római jogi alapvetés miatt – ismeretesek a latin kifejezések is, mégis túlnyomórészt magyar kifejezéseket használ. Ezek azonban részben a köznyelvben nem használt szavak és kifejezések, de ha használtak is, gyakran eltérő jogi értelmezéssel bírnak, ezért nem érthetőek.

A számítógépes feldolgozás szempontjából a résznyelv kötöttsége komoly előny: okkal feltételezhetjük, hogy ez igen kevéssé intuitív, kevéssé érzékeny, törekszik a teljes információt szövegesen, kerek, teljes, szabatos és nyelvtanilag is jól formált mondatokban átadni – ellentétben például egy személyes, társasági kommunikációval, ahol a verbális információ a sok-sok metakommunikációs vonás mellett gyakran erősen a háttérbe is szorul.

Az említett jogi írásbeliség miatt azonban alig lehetséges jogi szakterületen számítógépes eszközöket úgy alkalmazni, hogy azok nem tartalmaznak szöveg-feldolgozási, nyelvtechnológiai elemeket. Bizonyos értelemben az alkalmazott nyelvtechnológia szintje jelöli ki a jogi alkalmazások szűk keresztmetszetét – de legalábbis meghatározó tényezőjét. Amint megcélozzuk a nagyfokú szemantikus

tisztaságot, a projektumok ráfordítása, de még a kockázata is erőteljesen növekedik. Ezen megállapításhoz hasonlót Mike Bergman blogjában is olvashatunk,[110] fentebb idézzük a mellékelt tanulságos ábrát is.

Ezzel párhuzamos az az állítás is, hogy a szemantikus tisztaságot célzó módszerek nemigen elérhetők a piacon. Az ilyenek mindig a feldolgozás bonyolultságát is jelentik, és igazából mindmáig kísérleti, kutatási fázisban vannak, részeredmények elérhetők, esetleg nagyon konkrét területre leszabott termék is született már, de érett, általános célú piaci termék mindmáig nem jelent meg.

A következőkben (a fenti ábrán bemutatottakhoz hasonló tárgyalásban) áttekintjük a jogi-számítástechnikai termékekben és kutatási projektekben alkalmazott (vagy alkalmazható) nyelvi technológiákat, és értékeljük aszerint, hogy melyik milyen informatikai technológiák/műveletek megvalósítására alkalmas.

3.1 Szabad szöveges keresőprogramok működése

A szöveges információfeldolgozás talán legelső, és a piacon már évtizedek óta elérhető technológiája a nagy szövegbázisok szöveg szerinti tárolása, indexelése és keresése. Ehhez hasonló megoldást minden, szöveges információt ténylegesen, szöveges formában tároló adatbázisban, szöveges adatbázisokban, fájl-rendszerekben használnak, de alapjaiban ugyanígy működnek az internetes keresőprogramok is. Ezt a megoldást gyakran *szabad szöveges keresésnek* (freetext search) is nevezik.

3.1.1 Szóindexelésen alapuló rendszerek

Az efféle rendszerek középpontjában egy indexelt adatbázis (adatbázis-tábla) áll szavak szerinti indexeléssel, amely tehát az egyes szavak előfordulásait összekapcsolja az előfordulások címével. Az ilyen megoldások egyik gondja a nyelvfüggés, amely több szinten is jelent gondot:

- Korábbi időszakokban komoly gond volt a betűkészlet kérdése. A szabványos ASCII kódtáblába minden kiterjesztés nélkül ugyanis nem tudjuk sem a cirill, sem a görög, sem a japán vagy kínai nyelv betűit belevetíteni, de még a magyar hosszú „ő” és „ű” használatával is gondok vannak. Mindez szerencsére ma már a múlté – a betűkészletek közötti váltást és egységes ábrázolást a Unicode szabvány jól kezeli – és az azóta piacra dobott szoftver eszközök – pl. adatbázis-kezelők, fordítóprogramok, stb. – szintén átvették és megvalósították.
- A táblaépítés és indexelés során nem foglalkozunk egy sor pici, túl gyakran használt szóval, amiket *kivételeknek* (angol kifejezés: *stopword*) nevezünk. (Az angol kivételszótár tartalmazza pl. a névmásokat, „he”, „she”, „it”, stb., a leggyakoribb kötőszavakat: „and”, „or”, „then”, stb.) A kivételszótár legtöbbször könnyen kezelhető, bővíthető, testre szabható, viszont nyilvánvalóan erősen nyelvfüggő. Vagyis minden nyelvre külön kivételszótárt kell megadnunk, így nemzetközi környezetben a nyelvfelismerés, de legalábbis a szövegek létrehozásakor a nyelv kézi megjelölésének feladatát nem tudjuk megkerülni. Hozzá kell tenni: a kivételszótárt projektumonként/alkalmazásonként, és nem felhasználónként adhatjuk meg, vagyis személyes kivételszótár megadás legfeljebb személyi számítógépes egyénileg futtatható alkalmazások esetében lehetséges.

[110] http://www.mkbergman.com

- Egyes természetes nyelvek *gazdagon toldalékolnak* – magyar nyelvünk például éppen ilyen. Az ilyen nyelvek esetén értelmetlen és lehetetlen is az említett indextáblában az összes lehetséges szóalak megjelentetése. Ehelyett még az indexelés előtt egy szóalak-elemzést (morfológiai elemzést) futtatunk le, és csupán a szótő alapján indexelünk. Ez a megoldás egyrészt feltételezi, hogy étezik egy használható szóalak-elemző program, másrészt pedig ismételten megköveteli, hogy az indexelendő szöveg nyelve felismerhető legyen, de legalábbis a szöveg maga tartalmazza a saját nyelvének a megjelölését.

Működtetési szempontból a rendszerhez tartozik egy *keresőnyelv*, amely segítségével a keresésre vonatkozó mintát adhatjuk meg. Ez tükrözi a háttérben tárolt, alapvetően szóalapú indexelési megoldást. A keresőnyelv ezért egyes keresőszavak vagy minták különböző – Boole és egyéb kapcsolatainak megadását teszi lehetővé.

Művelet	Jel	[Példa]	Magyarázat
AND		[Petőfi Sándor]	Puszta egymás után írással azokat a dokumentumokat találja meg, amelyben mindkét operandus megtalálható
OR	or	[Petőfi or Arany]	Azokat a dokumentumokat találja meg, amelyben az egyik vagy a másik operandus megjelenik.
NEXT		[„Arany János”]	Azokat a dokumentumokat találja meg, amelyben a két keresőszó egymás mellett (1 távolságban), de bármilyen sorrendben megjelenik
NOT	-	[Arany -László]	Valójában AND NOT, az első operandust tartalmazó, de a másodikat nem tartalmazó dokumentumokat találja meg.

42. ábra A Google keresőnyelvének legfontosabb elemei

Az átlag felhasználó már a fenti táblázatban megadott keresőnyelvi elemeket sem képes tudatosan használni. Ennek ellenére a Google, és más rendszerek is általában még további lehetőségeket is biztosítanak. A keresőnyelv terén az egyik ilyen: a központozás figyelembe vétele, pl. ár jellegű mintamegadási lehetőséggel ([$300]), valamint az előfordulási távolság pontosabb megadása (ilyet a Súgó szerint nem tud a Google).

A szöveges keresés esetén kétféle nagyon tipikus hiba jelentkezhet. Egyfelől hibának számít, ha a rendszer *téves* vagy *felesleges találatokat* is ad. Az efféle hibákat szokás *vaklármának* is nevezni, ami leginkább a *homonimák*, vagyis az azonos alakú, de különböző értelmű szavak miatt jelent felesleges találatokat (pl. ha a „per” kifejezést keressük, akkor a „per”, mint műveleti jelre vonatkozó előfordulásokat is meg fogjuk találni). A másik fajta tipikus hiba a *rejtett találatoké*, azaz, amikor a rendszer nem találja meg a felhasználó kérdésére formálisan nem, de egyébként nyilvánvalóan illeszkedő találatokat. Ilyen leggyakrabban a *szinonimák*, vagyis a különböző alakú, de azonos tartalmú szavak miatt történik, amiket a rendszer nem talál meg (pl. a „kutya” keresőkérdésre az „ebrendelet” kifejezést nem fogja megtalálni).

A szöveges keresés legfontosabb hátulütője a keresés *szemantikus tartalmának* a teljes figyelmen hagyása, amit a piaci keresőalgoritmusok különböző módokon javítottak.

3.1.2 Relevanciaszámítás és keresőrobotok internetes keresőprogramokban

Internetes keresőrendszerek[111] esetében a szabad szavas keresés alapmegoldása mellett még két fontos szempontot meg kell vizsgálnunk.

Szöveges adatbázisok esetén pontosan adott az adatbázis terjedelme, mérete, tárolási módja. Az indexelés ezen alapul, hozzátéve, hogy az adatbázis bármilyen változása (meglévő dokumentum törlése, újabb hozzávétele) esetén az indextáblában is megtörténnek a megfelelő változások. A folyamat az induló adatállomány indexelésével kezdődik, ami meglehetősen hosszadalmas, de ezután már csak a változtatások inkrementális (vagy dekrementális) átvezetése történik meg, ami teljesen elfogadható időkereteken belül lefut.

Internetes esetben az indexelni kívánt adatállomány lényegesen kevésbé pontosan meghatározott, ezért úgynevezett *keresőrobotokat (crawler, spider)* alkalmaznak. A keresőrobotok önállóan működő *ügynökök (agent)*, amelyek időről időre végigpásztázzák az Internetet vagy egy részét, és ismételten elvégzik az indexelést.

A keresőrobotos megoldás nehézségei:

- Az interneten található adatmennyiség *gigantikus méretű*, és *folyamatos robbanásban* van. A www.technet.hu internetes portál szerint kutatói becslések alapján 2008-ban a világ internet forgalma a világ 27 millió üzleti szerverén megközelítette a 10 zettabájtot, vagyis $10*10^{21}$ bájtot! Ebben valószínűleg nem kevés többszörös, redundáns adat is van, a tárolt adatot talán ennek az 1-10%-ára tehetjük. Ekkora mennyiségű adat kezelése és teljes indexelése szinte lehetetlen, különösen vissza-visszatérő stratégiával…
- HTML tagok segítségével *megtilthatjuk* egyes oldalakon a *robotok belépését*. Az ilyeneket aztán nem illik a robotnak lekövetnie – de vajon tényleg be is tartják-e? Hogyan ellenőrizhető, és hogyan kérhető számon mindez?
- Az *adatok ábrázolásmódja* változatos (heterogén), még a szöveges információk is többféle formátumúak (HTML, DOC, PDF, stb.), nem is beszélve az egyéb jellegű, képi vagy hanginformációkról, amelyekre az indexelés technológiája gyakorlatilag nem létezik. Falansztert idéző írások szólnak az egyre biztatóbb gépi arcfelismerési kísérletekről, ezek személy-indexelési eljárássá történő fejlesztése nyilvánvalóan komoly személyiségi jogi kérdéseket vet majd fel.
- Nemcsak a formátum, de a tartalom is *változatos, és ellenőrizetlen*. Biztos, hogy minden kamaszgyerek blogját, vagy esetleg ismétlődő információkat többszörösen indexelnünk kellene? Hányszor lehet vajon megtalálni a magyar Interneten csak Petőfi Nemzeti Dalát?
- *Milyen sűrűn* látogasson meg egy robot egy oldalt, illetve legyen-e megoldás ennek befolyásolására? Legyenek-e időnként mindig felkeresett oldalak, illetve legyen-e műszaki lehetőség komoly tartalmi változtatás jelzésére, és ezért a látogatás mihamarabbi kérelmére? Bizonyos oldalak (pl. hírportálok) esetén a tartalom üzemszerűen és gyakran változik. Mennyire pontosan kövessék az internetes keresőprogramok ezek tartalmi változásait?
- Egy HTML oldalon általában több kimenő ugrás is található, amelyeket a webrobotok lekövetnek. Milyen legyen a *lekövetés stratégiája*? A véges gráfoknál tanulható mélységi és széltében történő keresési stratégia mellett az

[111] Szeredi P. - Lukácsy G. - Benkő T.: A szemantikus világháló elmélete és gyakorlata [SzLB05]

Internetes alkalmazásokra valószínűleg egy harmadik változatot érdemes kidolgozni.

- Az úgynevezett *mély web* figyelembe vétele és lekövetése. Mély web alatt a nem közvetlen adatformátumokat, hanem pl. relációs adatbázisokba, webszolgáltatásokba vagy másba csomagolt szövegeket és egyéb tartalmakat értjük.

További fontos kérdés a talált eredmények felkínálásának sorrendje, illetve ennek stratégiája. Nyilvánvaló, hogy a felhasználók csak az első néhány találatot nézik meg: a felületes felhasználó még az első találati lapot sem böngészi végig teljesen, de még a gondos felhasználó sem megy néhány találati lapnál tovább. Vagyis mondjuk a 100-dik találat már lényegében olyan, mintha ott sem lenne.

A találatok sorrendjének megállapításához minden oldalt egy valós számmal, súlytényezővel látnak el, és a megjelenítéskor a súlytényező alapján állítják sorrendbe a találatokat.

A webrobot bolyongása során megállapítja a súlytényezőket. Ezt a Google Page Rank algoritmusa a következőképpen végzi.

$PR(A) = (1-d) + d*(PR(t_1)/C(t_1) + PR(t_2)/C(t_2)+ \ldots + PR(t_n)/C(t_n))$

Az algoritmus leírásában szereplő jelek magyarázata:

- $PR(t_n)$ a lapra mutató n-dik idegen lap súlya
- $C(t_n)$ a lapra mutató n-dik idegen lap összes kimenő ugrásának száma
- d általános súlyozó tényező, amit 0..1 közé állítanak be, kezdetben általában 0.85 értéket kap.

Az algoritmusból kiolvashatóan egy teljesen vadonatúj lap esetében odamutató ugrás még nincs. Ilyenkor tehát a súlytényezőt az 1-d tag adja. Általánosságban akkor kap egy lap magasabb értéket, ha sokan, és értékes lapok mutatnak rá. A nevezőbeli C tag miatt az algoritmus kevésbé értékeli azokat a rámutató lapokat, amelyek egyidejűleg nagyon sok más lapra mutatnak.

3.2 Tezauruszos kérdéskibővítés

A puszta szabad szöveges keresési megoldás semmilyen módon nem veszi figyelembe a szövegek szemantikus értelmezését. Ezt a hiányosságot utóbb a vezető gyártók (a Google is) a *tezauruszos kérdéskibővítés* nevű megoldással próbálták enyhíteni. A megoldás lényege, hogy egy tezauruszban tárolt szinonima- és egyéb viszonyok alapján az eredeti kérdés találatai mellett még további találatok is megjelennek. Ami az előny, az egyben a hátrány is: miközben a rejtett találatok száma csökkenhet, a megoldás általában növeli a felesleges találatokat.

3.2.1 Tezauruszok

A tezaurusz egy szótárszerű adatbázis, amelyben az egyes bejegyzésekhez (szavakhoz) a bejegyzések között értelmezett alapvető és egyszerű értelmezésbeli viszonyokat is tárolnak. A tezaurusz határozottan nem kétnyelvű szótár, de nem is hagyományos értelmező szótár, hiszen a szavak nem pár mondatos magyarázattal vannak ellátva, hanem az említett viszonyleírással.

A viszonyleírás a viszonyok egy konkrét és rögzített, szűk körére vonatkozik. Tezauruszok általában különböző nyelveken és különböző tematikus szakterületre készülnek, tartalmilag tehát nem, de a viszonyok tekintetében meglehetősen egységesek. Papíralapú tezaurusz már korábban is készült, elektronikus tezauruszokat is eléggé régóta, a múlt század 70-es 80-as évei óta építenek. Példaként tekintsük az Országos Széchenyi Könyvtár (OSZK) által az ezredfordulót követő években felépített Nemzeti Köztezaurusz szerkezetét:[112]

A Köztezaurusz által értelmezett egyes viszonyokat egy-két karakteres betűvel jelölik a következők szerint:

H az adott címszó kitüntetett szinonimája, amelyből csak egy lehet

HV az adott címszó lehetséges szinonimája, amelyből bármennyi lehet

F az adott címszónál általánosabb fogalom (*hipernima*)

A az adott címszónál specifikusabb fogalmak (*hiponima*), ebből is rendszerint nagyon sok létezik

T az adott címszó által lefedett fogalmat, mint részegységet magába foglaló egész egység

P az adott címszó, mint szerelvény részegységei

R az adott címszó, mint fogalom eredményeképpen (okozatként) létrejövő fogalom vagy jelenség

E az adott címszót, mint fogalmat eredményező fogalom, mint ok

	jármű
H	közlekedési eszköz
HV	járműszerkezet
	járműtechnika
	szállítóberendezés
	szállítóeszköz
F	ipari berendezés
A	haszonjármű
	légpárnás jármű
	repülő szerkezet
	szárazföldi jármű
	személyszállító jármű
	teherszállító jármű
	villamos jármű
	vízi jármű
	vontató
	vontatott jármű
T	járműszerelvény
P	járműmotor
R	közlekedés
	szállítmányozás
E	járműjavítás
	járművezetés

43. ábra A Köztezaurusz "jármű" bejegyzése

A tezauruszok legfontosabb használati területei a könyvtárak, és más információtárak. Könyvek katalogizálásakor, szakfolyóiratok cikkeiben, valamint internetes oldalakban is szokás jellemző kulcsszavakat megadni. Ezeket a kulcsszavakat pedig

[112] Ungváry R. szerk: OSZK Köztezaurusz [Un02]

kötelezőszerűen egy megállapodás szerinti tezauruszból választják. Ezért volt hasznos az OSZK kezdeményezése egy egyesített és közös nemzeti tezaurusz létrehozására, amely ezután a magyar információtárakban közösen használható lenne.

3.2.2 A WordNet tezaurusz

A WordNet projektum 1985-ben, a Világháló terjedésével elindított kezdeményezés,[113] melynek célja, hogy kicsit a mai Wikipédiához hasonló közösségi technológiával tezaurusz-szerű, de annál kicsit szélesebb információkört felölelő adatbázis jöjjön létre. Az idők során a WordNet közösségi jellege megszűnt – kiderült, hogy szakemberek igencsak szűk csoportja az, aki az adattárban tárolt nyelvészeti és fogalmi információkhoz érdemben hozzá képes szólni.

A WordNet alapvetően angol nyelven készült, de a későbbiekben sokféle nyelvre, többek között magyarra is átültették.[114] A tezaurusz a korábbi értelmezéstől eltérően nemcsak főneveket és főnévi jelzős szókapcsolatokat (collocation), hanem egyéb fajú szavakat, pl. igéket, jelzőket és határozószókat is tartalmaz. A WordNet által tárolt relációk:

Főnevekre:

- *Gyűjtőfogalom (hiperonima)*: pl. a „juhászkutya" gyűjtőfogalma a „kutya".
- *Részfogalom (hiponima)*: a fenti viszony fordítottja.
- *Kapcsolt fogalmak (coordinate terms)*: két fogalmat akkor tekintünk kapcsoltnak, ha van közös gyűjtőfogalmuk. Például a „juhászkutya" és a „vadászkutya" közös gyűjtőfogalma a „kutya"
- *Fizikai része (holonima)*: pl. „ház" része a „fal".
- *Fizikai szerelvénye (meronima)*: a fizikai rész fogalmának megfordítottja.
- *Ellentétes jelentés (antonimia)*: a jelentések ellentétesek (pl. győzelem, vereség)

Igékre:

- *Gyűjtő és részigék (hiperonimák és troponimák)*, mint a főnevek esetében: pl. „kommunikál" részfogalma „beszélget".
- *Következmény igék (entailment)*: pl. ha valaki horkol, akkor minden bizonnyal alszik is.
- *Kapcsolt igék (coordinate verbs)*: a főnevekhez hasonlóan két igét kapcsolt igének nevezünk akkor, ha van közös gyűjtőigéjük. Pl. a „levelezik" és a „beszél" kapcsolt igék, mert mindkettőnek közös gyűjtőfogalma a „kommunikál".

Jelzőkre:

- jelzőkhöz kapcsolható főnevek
- hasonló jelzők
- szótő ige akkor, ha a jelző igéből képzett

[113] Ch. Fellbaum: WordNet An Electronic Lexical Database [Fe98]

[114] Miháltz M.: Magyar főnévi WordNet ontológia létrehozása automatikus módszerekkel [Mi03]

Határozószókra:

- szótő jelző, amennyiben jelzőből képezték őket

Külön érdekessége a WordNet tezaurusznak a *szinonimacsoport* (*synset*) fogalma, amelyekbe az azonos értelmezéssel bíró szavak tartoznak. Ugyanaz a szó – a szövegkörnyezettől függően – akár több szinonimacsoport tagja is lehet, nyilván csak többjelentésű szavak esetében, ilyenkor mindegyik csoportban más és más jelentésével szerepel.

A WordNet az egyes szavakhoz különböző gyakorisági információkat is tárol: így például szavak előfordulási gyakoriságát nagy korpuszokban (szövegtestekben). A szinonimacsoport fogalmára építve a szavak többértelműségi számát is megkaphatjuk, vagyis azt, hogy egy adott szó hány ilyen csoportban szerepel.

Az eredeti angol WordNet tezaurusz legfrissebb változata a 3.0-s, amelyet 2006-ban bocsátottak ki. Ebben nem kevesebb, mint 150 ezer szó és 207 ezer szókapcsolat található, amelyek összesen 115 ezer szinonimacsoportba vannak rendezve. A rendszer egyébként szabad felhasználású szoftver, a projektum honlapjáról letölthető, a teljes adatbázis tömörítve 12 MB méretű.

Az angolról magyarra átültetett adattárról vagy az ezeket a részeket egységes keretbe foglaló rendszerről egyelőre még nincsen híradás, mint ahogy a magyar adattár honlapját sem sikerült fellelni.

3.3 Szövegelemzésen alapuló megoldások

3.3.1 Sekélyelemzés és szöveganntotálás

A szabad szöveges keresési módszer nyelvi algoritmusokat legfeljebb csak a toldalékelemzés mértékéig alkalmazott, de ilyenre is csak a magyarhoz hasonlóan gazdag toldalékolású nyelvek esetében volt szükség. Már azonban a tezauruszokban sem csak egyszerű címszavak találhatók, hanem – mint a példa is mutatja – nem ritkák a két-három tagból szóösszetétellel, vagy jelzősen felépített szerkezetek szerepeltetése sem. Bár az ilyenek még viszonylag biztonságosan kereshetők szabad szavasan is (ez a megállapítás a magyarra még igaz, de pl. az orosz nyelvben nem, mert itt már a jelzőket is ragozzuk), itt már gyakran alkalmaznak a szóhatárokon túlnyúló elemzési megoldásokat is.

Az ilyen elemzési megoldások általános elnevezése: *nyelvi sekélyelemzés*, vagy *POS-Tagging (Part Of Speech)*, ilyeneket magyar nyelvre is alkalmaznak.[115] Az ilyen megoldások általános célja: *szöveganntotáció*, vagyis a szövegben az elemzési megoldások felismernek egyes nyelvi egységeket (pl. jelzős főnévi kifejezéseket, esetleg teljes főnévi csoportokat), majd ezeket illesztik a kifejezéstár, pl. egy szaktezaurusz elemeihez. Az ilyen módon azonosított előfordulásokból ezután géppel könnyen felismerhető és ellenőrzött tartalmú jelzeteket készítenek (pl. HTML META tagokat állítanak elő a leggyakrabban előforduló kifejezésekből, netán az egész szöveget XML-formába átalakítják, és a megtalált kulcsszavakat/kulcskifejezéseket külön kiemelik.

Az efféle megoldások hatalmas előnye, hogy egyfajta tartalmi ellenőrzést is biztosítanak: a szerzők csak a háttérben levő tezauruszra illesztett kifejezéskészletet

[115] Zsibrita, János-Vincze, Veronika-Farkas, Richárd: Ismeretlen kifejezések és a szófaji egyértelműsítés [ZVF10]

használhatnak, mert az elemzés során azonnal kibukik bármilyen eltérés. A megoldás igen hasznos lehet szakszövegek ellenőrzésére a legkülönbözőbb területen, így jogi szövegekre is.

A nyelvi sekélyelemzések másik iránya, amikor a szöveg tagolását derítik fel gépi eszközökkel. Ez a megoldás különösen jól alkalmazható jogi szövegek esetén: ha az annotálás nem a szókapcsolatok, hanem fejezetek, cikkelyek, bekezdések szintjén működik, az a felhasználó szintjén, a műveletek pontosságát tekintve kétségkívül jól hasznosítható.

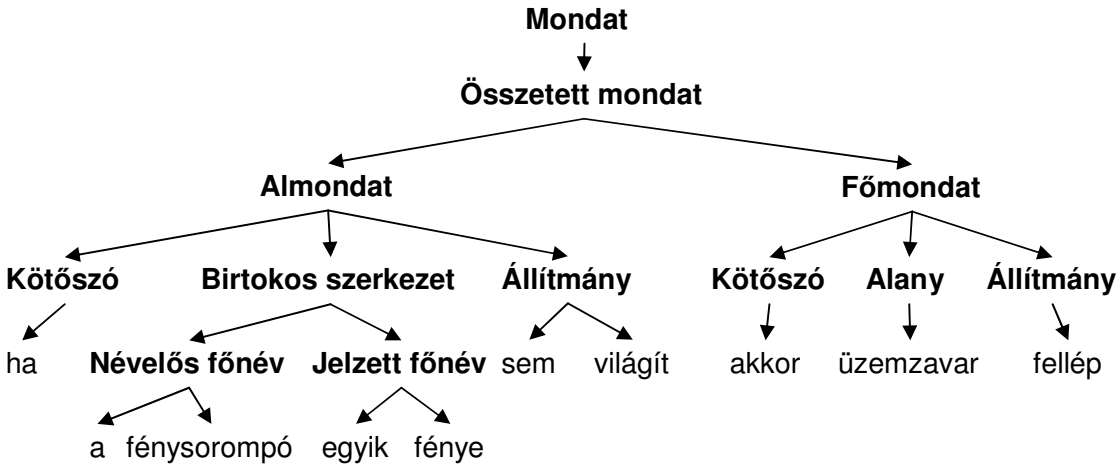
```
<SubPart Útvonalengedélyhez kötött járművek
  <Article 51
    <List
      <SentenceFragment 1 A közúti forgalomban csak az út
      kezelőjének hozzájárulásával (útvonalengedély), az
      abban meghatározott útvonalon és feltételek
      megtartásával szabad részt venni
        <SentenceFragmentSubPart a olyan járművel, amelynek
        megengedett legnagyobb össztömege vagy
        tengelyterhelése meghaladja a külön jogszabályban
        meghatározott mértéket (túlsúlyos jármű, vagy
        tengelytúlsúlyos jármű)
        SentenceFragmentSubPart>
        <SentenceFragmentSubPart b olyan járművel, amelynek
        magassága, szélessége vagy hosszúsága meghaladja a
        külön jogszabályban, illetőleg – rakománnyal együtt
        – a 47. §-ban meghatározott mértéket (túlméretes
        jármű)
        SentenceFragmentSubPart>
        <SentenceFragmentSubPart> c lánctalpas járművel./>
      SentenceFragment>
      <SentenceFragment 2 Ha az útvonalengedély más
      sebességet nem határoz meg, túlsúlyos, illetőleg
      túlméretes járművel legfeljebb 30 km/óra, lánctalpas
      járművel legfeljebb 15 km/óra sebességgel szabad
      közlekedni.
      SentenceFragment>
      <SentenceFragment 3 Az e §-ban foglalt rendelkezések
      nem vonatkoznak a fegyveres erők által üzemben
      tartott járművekre.
      SentenceFragment>
    List>
  Article>
SubPart>
```

44. ábra Példa KRESZ szabály tagolt annotációjára

Jogi területen figyelemre méltó a Metalex/CEN projektum[116], amely jogi szövegek XML annotációjára ad javaslatot. A nemzetközi projektben magyar résztvevők is dolgoztak, a Metalex annotációs szabványnak magyar nyelvű lokalizációja is létezik, sőt, az Interneten az akkori magyar alkotmány egy részének annotált változata is elérhető. Sajnos még ez sem megy le a szófajok szintjéig, csupán egyes formai jellemzők segítségével a szöveg egyes szakaszai, paragrafusai, stb. vannak azonosítva és XML taggal jelezve.

3.3.2 Mondatelemzés és fatárak

Az előzőekben említett sekélyelemzési feladattól alapvetően nem különbözik az, amikor a szövegelemzés határait egészen a mondatig kiterjesztjük. Még mindig csak nyelvtani műveletről van szó, az elemzés szemantikai összefüggéseket egyáltalán nem, vagy csak nagyon korlátozott mértékben vesz figyelembe. Ennek ellenére az ezzel összefüggő, úgynevezett *fatár (treebank)* megoldás a szabad szöveges kereséshez lényeges előrelépést jelent. Ennek lényege az, hogy a szövegtest elveszíti eredeti alakját, és a lemezen szöveges információ mellett, de esetleg helyette is, a szövegesen visszaállítható *nyelvtani elemzési fát* tároljuk, amely az egyes szövegelemekre hivatkozik. Ennek megfelelően, a tárolt szövegtest felett a faszerkezetnek megfelelő keresési nyelv használata is lehetséges. Hazánkban a szegedi nyelvtechnológiai iskola futtatott kiterjedt fatár-projektumot magyar nyelvű szövegekre. [117].



45. ábra KRESZ szöveg elemzési fája

Az elemzési fa levélelemein találhatók a szöveg egyes szavai, a közbülső csomópontok pedig az elemzés alapjául vett nyelvtan egyes kategóriáinak (nyelvtani fogalmainak) felelnek meg. A keresésnél az elemzési fa valamely útjához vagy részútjához kötött mintákból, illetve ezek logikai kapcsolataiból építhetünk fel kereső-kifejezéseket, például a következőképpen:

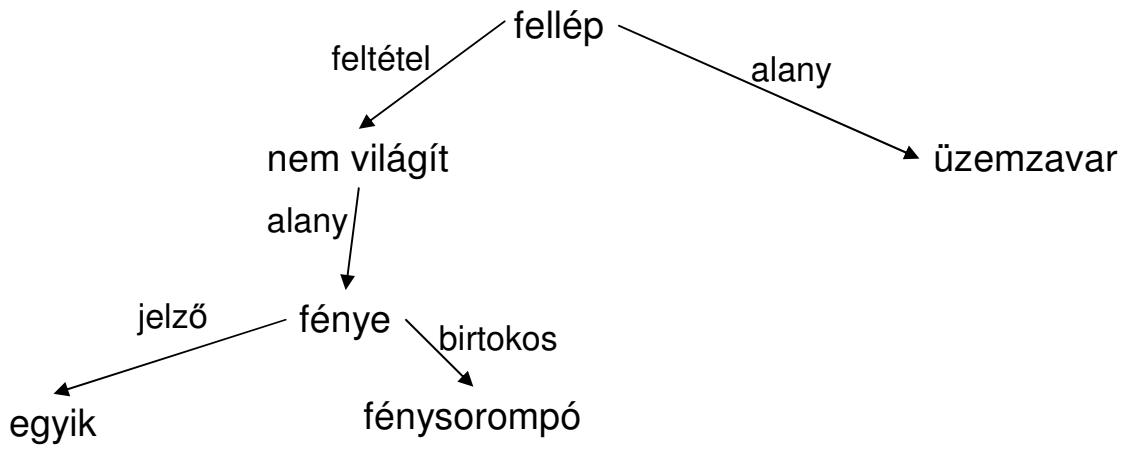
```
?Főmondat(Alany=”üzemzavar”) AND
 Almondat(Állítmány(„sem világít”)).
```

Az elképzelt lekérdezőnyelvi példa azokat a mondatokat keresi a szövegtestben (corpus), amelyek főmondatának alanya „üzemzavar”, mellékmondatának állítmánya

[116] Alexander Boer-Erik Hupkes-Fabio Vitali-Monica Palmirani-Balázs Rátai: CEN Metalex Workshop Proposal [CEN08]

[117] Alexin Z.: A frázisstrukturált Szeged Treebank átalakítása függőségi fa formátumra [Alz07]

pedig a „nem világít”. Nyilván azt várjuk, hogy a rendszer a fenti példamondatot vagy ehhez hasonló jellegű mondatokat talál majd meg.



46. ábra KRESZ mondat függőségi fája

Az elemzési fára épülő fatárakon való keresésnek kellemetlen hátulütője, hogy a hatékony kereséshez jól kell ismerni a nyelvtant, de főleg a nyelvtani kategóriákat, hiszen a kereső-kifejezésekben a fa elemeit a belőlük épített útkifejezésekkel címezzük meg. Ezt a gondot küszöböli ki a *függőségi nyelvtanos* elemzési megoldás, melynek célkitűzése, hogy a fa közbülső csomópontjai is a felhasználó által könnyen felfogható információt hordozzanak. A függőségi fa esetében a fa gyökérpontjában a legfontosabb mondatrész, az állítmány áll, és általában a gyökérhez közelebbi csúcspontokban vannak a fontosabb mondatelemek, a levélhez közelebbi csomópontokban pedig a kevésbé fontos bővítmények. Nyelvtani kategóriát így viszont a függőségi fákban egyáltalán nem ábrázolunk.

A nyelvtani elemzési fa használatához képest a függőségi fát ábrázoló megoldás esetén lényegesen leegyszerűsödik a keresőkérdések megfogalmazása. A fentiekkel megegyező elképzelt keresőkérdés ebben az esetben a következőképpen nézhet ki:

```
?fellép(feltétel=”nem világít”,alany=”üzemzavar”).
```

3.4 Ontológiára épülő módszerek

A tezauruszokra épülő módszerek alapvető jellemzője, hogy a bennük ábrázolt fogalmi viszonyok nem matematikai pontosságúak, ezért egy tezaurusz felett bizonyítási vagy következtetési műveletekről nem is beszélhetünk. Az ontológiák ezzel szemben feszes matematikai szemantikára épülnek, és a konkrét szakterületi fogalmakon túl a közhelytudás, illetve az általános tudás elemeit is tartalmazzák. Az ontológiára épülő módszerek esetében a szokásos, ablakos-menüs kezelői felületek csak nagyon korlátozottan használhatók. Maguk az ontológiák annyira finom módszerekkel ábrázolják az emberi tudást, hogy komoly áttörést csak igényes természetes nyelvű szövegfeldolgozás alapján, nyelvi mélyelemzési technikák bevetésével lehet elérni.

3.4.1 Nyelvi mélyelemzés és diskurzusreprezentáció

A korábban ismertetett nyelvi elemzési megoldások általában alig működnek a mondathatárokon túl. Ez elegendő lehet a mondatok nyelvtani szerkezetének megállapításához, bennük egyes szak- vagy hétköznapi kifejezések megtalálásához, és

ezekre alapozható keresési megoldások kidolgozásához is. Ezen a szinten mindenképpen túlmutat a *nyelvi mélyelemzés*, amely a mondatok, valamint a mondatok közti belső összefüggések ábrázolását is célozza. Úgy gondoljuk, hogy különösen jogi szakterületen nem tekinthetünk el a mélységi szövegkapcsolatok feltárásától, és ezért az effélét célzó kutatási projektumok sikeressége komolyan befolyásolhatja a jogi tudáskezelés eredményességét is. Az ilyen rendszerek egy újabb generációjában a normatívákat és az eseteket már nem szöveges formában, hanem logikai összefüggések formájában kell tárolni, és azon logikai következtetési műveleteket kell végezni. Ehhez képest a szövegek beolvasása és a logikai szerkezet összeállítása, valamint a szövegek létrehozása a logikai alakból, csupán valamiféle export/import műveletként fogható fel.

A mondatok közötti összefüggéseket is kezelő természetes nyelvtechnológiai megoldást *diskurzusreprezentációnak* nevezzük, amely megoldást leggyakrabban egy háttér-ontológiával összekapcsolva célszerű használni. Az efféle megoldások alkalmazása elengedhetetlenül fontos a következő nyelvi jelenségek miatt:

- *Névmások* gyakori alkalmazása. „John has overtaken Julie in a curve yesterday. His maneuver was very dangerous." A névmások általában a megelőző mondatokban bevezetett fogalmakra vagy egyedekre vonatkoznak. A szöveg pontos gépi megértéséhez fel kell deríteni, hogy pontosan melyik egyedről van szó. Az angol szöveg érdekessége, hogy a nemek használata erre pontosabb támpontot ad, mint pl. a magyar nyelvben.
- *Általános köznevek* alkalmazása. Ha az előző példát magyar nyelven képzeljük el, a második mondat valahogy így hangozhatna: „A férfi művelete nagyon veszélyes volt." Sőt: „A művelet nagyon veszélyes volt". Itt a „férfi" és a „művelet" kifejezés nem névmás ugyan, de logikailag ugyanolyan szerepet tölt be: visszautalás (anafora) a megelőző mondatban emlegetett személyre vagy folyamatra.
- Mondatok közötti *retorikai vagy logikai összefüggések* kezelése: „Nőtt az infláció az elmúlt hónapban. Robbantak az energiaárak." Az első mondat a második következménye, még akkor is, ha a szövegben később olvasható.
- *Kurzorok* alkalmazása: „A gyilkos 22.00 órakor érkezett az áldozat házához. Behatolt a lakásba." A második mondat időben az elsőt követi, még akkor is, ha erre semmilyen időhatározó nem utal. A hallgató elméjében ugyanis egy önműködően továbblépő *időkurzor* működik, amely segít az elbeszélés egymás után következő mondatainak az időbeli elhelyezésében. Más esetekben *térkurzorról* vagy *hatáslánc kurzorról* is beszélhetünk.

A következő példák arra szolgálnak, hogy bemutassuk: már a nyelvtani elemzés, de a mondatok közötti összefüggések feltárása sem történhet egy háttérbeli ontológia nélkül:

- „a vörös ukrán terrorista" kifejezés helyes ugyan, "*az ukrán vörös terrorista" viszont nem. (A csillag a nyelvészeti dolgozatokban a helytelen mondatot jelzi.) A főnévi szerkezetek elemzésekor a melléknevek csak megszabott sorrendben következhetnek: a színinformáció meg kell, hogy előzze a nemzetiségre utaló jelzőt. Ezt csak a jelző jelentéstartalmának és szemantikai jegyeinek a feltárásával lehet eldönteni, amit a legcélszerűbben a tudástár lekérdezésével lehet kinyerni.
- „Gabi és Jácint házasságot kötöttek. A rabbi szépen beszélt." Itt többféle következtetési műveletről is szó lehet a mondat számítógépes értelmezése kapcsán. Először is a „Gabi" becenévről önmagában nem dönthető el, hogy a

Gábor férfinév vagy a Gabriella női név becéző alakja-e. Sok beszédértelmező nem lehet tisztában azzal sem, hogy a Jácint férfi- vagy női név-e, de erre egy pontos ontológia választ adhat. A következő mondatban a „rabbi" kifejezés arra utal, hogy az esküvő izraelita vallás szertartása szerint történhetett. Az ortodox izraelita nézetek viszont nem ismerik el az azonos neműek házasságát, tehát ha Jácint volt a férfi, akkor a nő csakis Gabi, azaz anyakönyvileg Gabriella lehetett. A példa egy többlépéses – de teljesen kézenfekvő – logikai következtetéssort mutat be: az emberi elme ilyesféléket is, de esetenként ennél egyszerűbb következtetési lépéseket lépten-nyomon megtesz a szöveg tökéletes megértése érdekében.

- „A feleségemmel Berlinben ismerkedtem meg." A mondat belső ellentmondást tartalmaz: a megismerkedésünkkor még nem lehetett a feleségem. Az efféle belső ellentmondásokat az emberi szövegmegértés áthidalja: felderítése és feloldása (pl. a „későbbi" jelző beszúrásával) ezért szintén az ontológiára épülő következtető programmodul feladata lehet.

3.4.2 Az ESTRELLA jogi szemantikus projektum bemutatása

Az ESTRELLA (European Project for Standardised Transparent Representations in order to Extend Legal Accessibility) projektum egy Európai Uniós kutatási projekt volt, amely figyelemre méltó eredményekkel rendelkezik.[118] A projektum 2006 és 2008 között futott. A felügyelő konzorcium bő tucatnyi tagja között olasz, angol és holland kutatóintézetek és egyetemek, de többek között a Budapesti Corvinus Egyetem és az Adó és Pénzügyi Ellenőrzési Hivatal is megtalálható.

A projektum fő célja a Szemantikus Háló technológiára építve közös és szabványos adat- és tudás-csere platform létrehozása, részben a már meglevő tudás- és ontológiakezelő eszközökkel, részben pedig a projekt keretében létrehozott hasonló eszközökkel. A projekt részletes eredményei a honlapján máig nyilvánosan elérhetők, a jelen szakaszban erősen építünk az onnan elérhető információkra.

A platform a már kifejlesztett vagy még kifejlesztendő egyes összetevők kölcsönös működtethetőségét célozta, amelynek a legfontosabb eszköze az LKIF (Legal Knowledge Interchange Format) jogi tudásleíró résznyelv volt.

Az LKIF javaslat két fontos részből áll:

- A *terminológiai tudás* egy OWL alapú ontológiában van ábrázolva.
- A jogi szabályrendszer számára az OWL SWRL szabálynyelvének egy jogi célokra szolgáló kiterjesztését hozták létre (LKIF Rules).

Az LKIF szabályok az SWRL szabálynyelv elemeit több értelemben is kiterjesztik, illetve alkalmazzák. Egyfelől a nyelvben határozottan megengedik a konstruktív negációs szemantikát. Ez a végrehajtás során a negált literálokon történő következtetést jelenti, amely – tekintve, hogy elsősorban előre haladó végrehajtásról van szó –, csupán a szabályértelmező program megírását bonyolítja kissé el.

Másrészt az LKIF szabálykészlet a modellezendő jogszabályoknak megfelelő alkalmazói elemeket is tartalmaz, pl.:

[118] Az Estrella projektum honlapja: www.estrellaproject.org. Elérés: 31-Jan-2013.

- Szabályazonosító elemeket, amelyek egyrészt megfelelnek a jogi azonosításnak (paragrafusok, cikkelyek), másrészt a szabályok közötti rokonságot, illetve az ezek alapján történő megcímzést is lehetővé teszi.
- A szabályokhoz feltételek kötését (`assuming` ϕ),
- … és a szabályokhoz kivételek megnevezését (`unless` ϕ).

A szabálynyelv egy érdekes vonása, hogy az alapvetően elsőrendű logikai szerkezetekre hasonlító SWRL szabálynyelv nyelvtanát is módosították, és a végső nyelvtant a LISP alapú iskolákat követő, sokzárójeles jellegűre tervezték meg. Az alábbiak szemléltetésére idézzünk egy példát[119]. Ez azt írja le, hogy minden ingóság, ami nem pénz, az árucikk lehet.

```
(rule §-9-105h
  (if (and (movable ?c)
           (unless money ?c)))
      (goods ?c)))
```

3.4.2.1 A CARNEADES szerkesztőprogram

Carneades görög bölcs a Kr.e. 2. században élt, és a Platón által alapított filozófiai iskola továbbvivője és vezetője volt. Az Estrella projektben az ókori bölcs nevét viseli az érvelést támogató szerkesztőprogram, amely érv- és bizonyítási -hálózatok felépítésére, módosítására, megjelenítésére, kiértékelésére, valamint ezek közötti érv-adatcserére alkalmas. Ez utóbbi egyes rész-bizonyítások más érv-hálózatokból történő átvételét teszi lehetővé.

A CARNEADES szerkesztőprogram a Fraunhofer Fokus berlini kutatóintézet terméke, akik éppen ezzel vettek részt az Estrella konzorcium munkájában.

A szerkesztőprogram legalapvetőbb adatszerkezete az „érv-szerkezet", amely egy páros gráf. Vagyis a hálózat csomópontjai érvek, vagy tények lehetnek, amely érvekből újabb tényeket következnek, és amely tények újabb érvek alapjául szolgálhatnak. A következtetés most is tökéletlen (defeasable reasoning), az eszköz tehát a jogi eset hátteréül szolgáló *egyik* (vagy több, alternatív vagy akár ellentmondó) érvszerkezet feltárására alkalmazható. A projektdokumentációban mindezeket alátámasztó jogi esetek megjelenítését találhatjuk, amelyek azonban mind a még könnyen átlátható méret- és bonyolultságkategóriába esnek.

3.4.2.2 A HARNESS elemzőprogram

Az Estrella projekt keretében létrejött másik konkrét szoftver termék a HARNESS, jogi helyzetértékelő rendszer.[120] A rendszert vázlatkészítési, jogi tervező és érvelő segédeszközként kívánták létrehozni, mindezen műveletkörök alapjául a jogi helyzet-értékelés alapvető műveletét valósították meg. A helyzetértékelő művelet egy LKIF tudásbázison dolgozik, és bemenetként egy ilyen nyelvű esetleírást kap. A következtető motor vegyes technológiával működik. Egyrészt az OWL ontológia felett működő szabványos DL (leíró logika) bizonyítóalgoritmusokat használja, úgy, hogy a jogi fogalmakat a lehető legpontosabban leképezi az LKIF résznyelvének számító OWL ontológiára. A jogi fogalmak között a helyzetértékelés, illetve minősítés

[119] Alexander Boer et al.: Specification of the Legal Knowledge Interchange Format: [EST07.1]

[120] Joost Breuker, Saskia van de Ven eds. Abdallah El Ali et al.: Developing HARNESS. Towards a hybrid architecture for LKIF: [EST08]

eredményosztályai is megtalálhatók (`Allowed, Disallowed, Obliged`, stb.). Érdekességképpen megemlítendő még az `Allowed_and_Disallowed` értékelésosztály is, amellyel esetleges normaütközésekre lehet fényt deríteni. A leírásuk alapján az egyes esetleírásokat a szabvány DL következtető a fenti osztályokba sorolja.

A másik működésmód az egyes jogi normatívákat LKIF-Rules nyelven fogalmazza meg. A megfelelő szabályok alkalmazásának végeredménye szintén az eset besorolása a fent említett normatív értékelésosztályokba. A szabályokat vagy - SWRL-be átalakítás után - egy SWRL-t is kezelni képes szabványos, vagy az LKIF-Rules leírás alapján egy saját jogi következtető motorral hajthatjuk meg. A szabványos SWRL megoldás hátránya, hogy az LKIF-Rules formátum kiterjesztései miatt esetlegesen helytelen következtetésekre vezethet.

A kettős (kevert) szabályalkalmazás végén külön algoritmus gondoskodik az eredmények összefésüléséről.

3.4.3 Az LKIF jogi csúcsontológia

Az LKIF egy közhelytudáson alapuló jogi csúcsontológia, amely egyben magába foglalja a már korábban létrehozott ontológiák építése közben szerzett tapasztalatokat is. Az OWL nyelven elkészült csúcsontológia a következő részontológiákat tartalmazza, amelyek egymásra importálással hivatkoznak. [121]

- `lkif-top`: A részontológia a legalapvetőbb fogalmakat és azok összefüggéseit tartalmazza. Pl. (`Physical_Entity, Physical_Object, Mental_Entity, Mental_Object, Abstract_Entity, Occurence`)
- `mereology`: a görög „μεροζ" (part) kifejezésből kiindulva a rész-egész viszonyokat, összetételt leíró részontológia
- `time`: A legalapvetőbb időbeli fogalmakat (`Interval, Moment`), valamint viszonyokat (`before, after, between, overlap`, stb.) rögzítő részontológia.
- `relative-places`: A hellyel kapcsolatos alapvető fogalmakat (`absolute_place, relative_place, location_complex`) és viszonyokat (`cover, partially_coincide, exactly_coincide`) leíró részontológia.
- `process`: Folyamatok időben és térben írhatók le, ezért az ontológia importálja ezeket. A részontológia olyan fogalmakat tartalmaz, mint a változás (`change`), amely valamilyen állapotátmenetet jelent, illetve a folyamat (`process`), amely időben, térben leírható és valamilyen oksági viszony által elindított változást jelent.
- `action`: a Folyamat modult egészíti ki a céltudatos cselekvésre utaló fogalmakkal. Ilyenek pl. a cselekedet (`Action`), amelynek végrehajtója (`actor`) van, aki egy cselekvő (`Agent`) és aki a folyamatban valamilyen szerepet (`Role`) játszik. A cselekvők lehetnek természetes személyek

[121] ESTRELLA (2007): OWL Ontology of Basic Legal Concepts: [EST07-2]

(Person) vagy szervezetek (Organisation). A cselekedetek végrehajtásához gyakran valamilyen szakismeret (Competence) szükséges.

- role: Folyamatok, különösen jogi folyamatok leírásában a szerep (Role) fogalma központi. A modul a következő szereposztályokat rögzíti:
    - Function: Az a cél, amire egy konkrét objektumot valamilyen környezetben használhatnak.
    - Epistemic_Role: A tudás megszerzésével, osztályozásával, következtetéssel, kapcsolatos szerepek. (Assumption, Observation, Reason).
    - Person_Role: Személyes viszonyokat kifejező szerepkörök.
    - Legal_Role: jogi viszonyokat kifejező szerepkörök (ld. alább).
- legal-role: A jogi folyamatokban szokásos szerepkörök meghatározása.
- expression: a modulban az ítélet és a megállapítás (Proposition) fogalma, és egyes személyek valamilyen megállapításhoz fűződő hozzáállása (Propositional_Attitude) van rögzítve. Ez lehet pl. hit (Belief), megállapítás (Assertion), nyilatkozat (Declaration), ígéret (Promise), vágy (Desire), szándék (Intention).
- legal-action: A jogalkalmazással összefüggő egyes jogi lépéseket határozza meg. Ilyen pl.: törvényhozás (Act_of_Law), nyilvános testület számára nyilvános jogkörrel történő felruházás (Assignment), valamilyen feladat vagy jogkör átruházása (Delegation), felhatalmazás valamilyen tevékenység elvégzéséhez (Mandate).
- norm: Egyes jogi fogalmak meghatározása (Legal_Source), illetve az egyes normatíva fajták csoportosítása a következők szerint:
    - engedély: (Permission) amelyben kötelezettségek (Obligation) és tiltások (Prohibition) lehetnek együttesen rögzítve
    - jogok (Right): amelyben megkülönböztetünk felelősségi jogokat (Liability_Right), szabadságjogokat (Liberty_Right), kötelezettségeket (Obligative_Right), illetve valamilyen kötelezettség tagadásából fakadó megengedő jogokat (Permissive_Right).
- time-modification: A részontológia egyes jogi helyzetekkel és állapotokkal kapcsolatos olyan módosításokat tartalmazza,
    - amelyek a tényleges jogi szöveg (szabály) módosítását nem igénylik (SemanticAnnotation). Pl.:
        - Hatáskör módosítás (Modification_of_Scope), pl. a kivétel (Exception), ami megszorítja a normatíva eredeti hatáskörét, vagy kiterjesztés (Extension), ami újabb hatásköri elemeket von be.
        - Értelmezési módosítás (Modification_of_Meaning), pl. újraértelmezéssel (Interpretation), valamilyen kifejezés

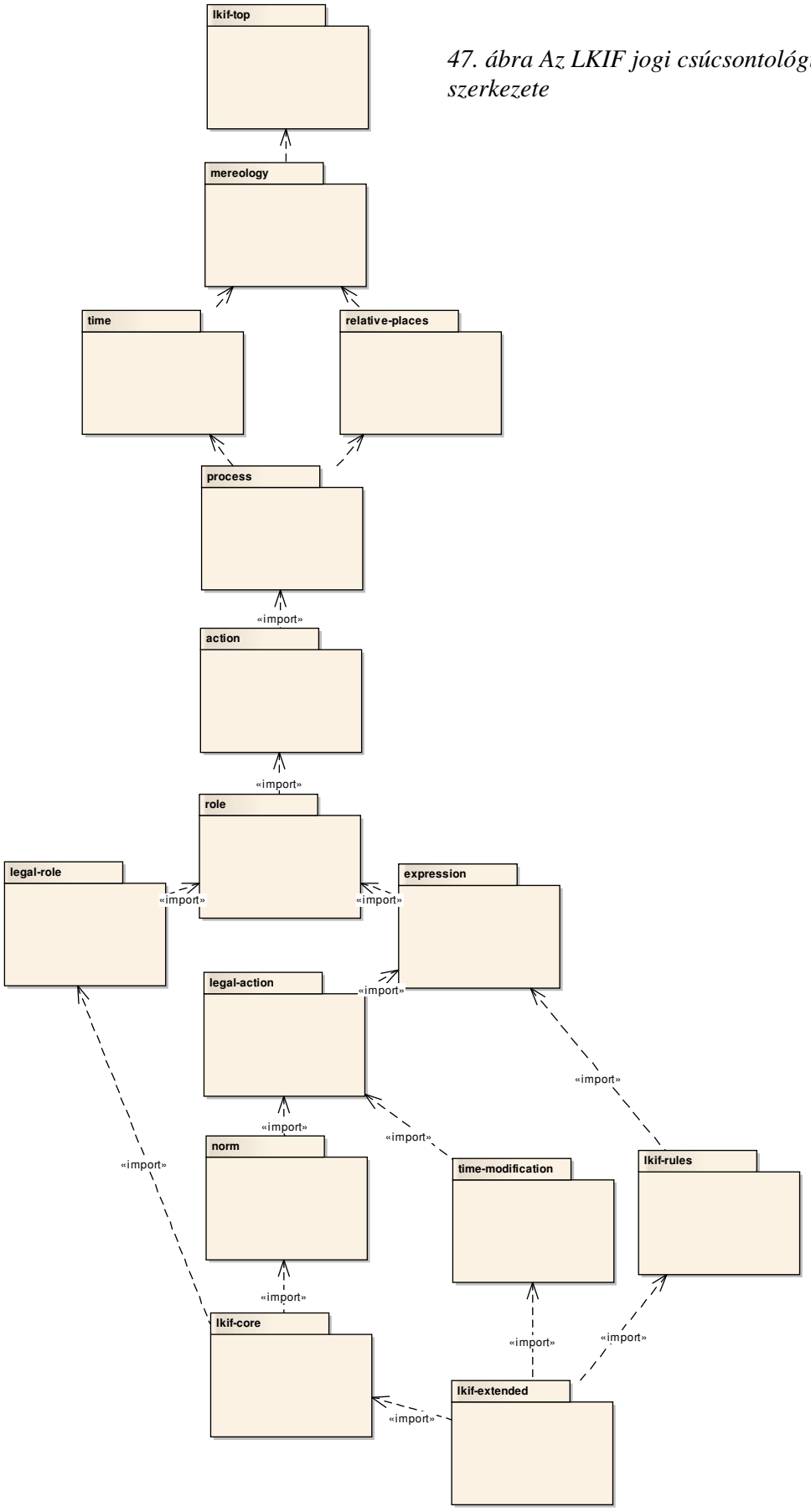
újbóli meghatározásával (Modification_of_Term) vagy körülírásával (Variation).

- Rendszerviszonylatban végzett módosítás (Modification_of_System), amely történhet pl. magasabb szintű normatíva alkalmazásával (Application) egy helyi helyett, valamilyen elsődleges jogforrás másodlagossá sorolásával (Deregulation), valamilyen nemzetközi egyezmény helyileg történő elfogadásával (Ratification), valamilyen normatíva ugyanolyan témában történő teljes átalakításával (Remaking) vagy valamilyen normatíva EU-környezetben történő értelmezésével (Transposition).

- amelyek a tényleges jogi szöveg időbeli hatályára vonatkoznak (Temporal_Modification), illetve
- amelyek szövegszerű módosítást jelentenek (Textual_Modification).

- lkif-rules: A részontológia az LKIF szabálynyelv egyes elemeinek meghatározását tartalmazza. Ezek közül az SWRL nyelven túli elemek a következők:
  - Atom (Atom): egy LKIF szabály tovább nem osztható része
  - Feltételezés (Assumption): olyan feltételek fennállása, amit nem bizonyítunk, hanem a jogi bizonyítás céljára igaznak feltételezünk.
  - Érv (Argument): olyan szabály, amelyet az érvrendszeren alapuló bizonyítás során használunk fel.
  - Kivétel: (Exception): olyan atomi állítás, amely a normatíva hatálya alóli kivételt jelent.
- lkif-core: A részontológia nem rögzít semmilyen fogalom-meghatározást, hanem csupán a hivatkozott ontológiák importját tartalmazza.
- lkif-extended: Szintén csupán import-ontológia, az lkif-core ontológián kívül a time-modification és az lkif-rules ontológiák importját tartalmazza.

Az LKIF jogi csúcsontológia mindenképpen figyelemre méltó, mert bármilyen alkalmazói jogi ontológia létrehozásához egy meglevő ontológiából érdemes kiindulni. Ha jogi szakontológia létrehozása a cél, akkor azt az LKIF csúcsontológiára alapozva célszerű megtenni.



47. ábra Az LKIF jogi csúcsontológia szerkezete

3.5 A szintaxis és a szemantika egyesített modellje: a ℜeALIS projektum

A Pécsi Egyetemen futó ℜeALIS projektum (**Re**ciprocal **A**nd **L**ifelong **I**nterpretation **S**ystem) természetes nyelvű szövegmegértési modell megvalósítását célozza.[122] A modell legfőbb alapvetései:

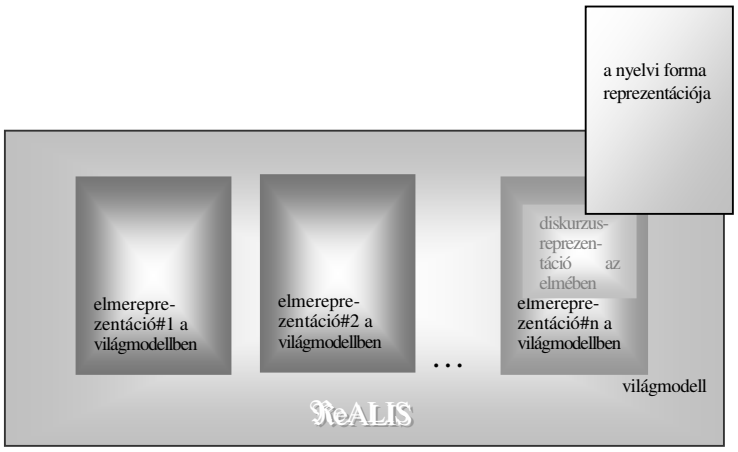
- A napjainkban felmerülő gépi fordítási és jelentéskivonatolási feladatok a szövegek feldolgozását *tartalmilag a lehető legmélyebb szintig* követelik meg. Ez gyakorlatilag azt jelenti, hogy a számítógéppel modellezni kell a körülvevő világot, a szövegértést végző emberi elméket, és a szöveg tartalmát ebben a rendszerben kell elhelyezni.
- A *kölcsönösség* alatt a párbeszédes vagy konferenciahelyzetek *eszmecseréinek kölcsönhatásait* értjük, vagyis azt a helyzetet, amikor az egyik szereplő megnyilvánulása egy *párbeszédes környezetet* nyit meg, és a következő megnyilvánulás már ezen a környezeten belül értendő. A párbeszédes környezetek megnyitása ennél finomabb dolog is lehet. Történhet egy hosszabb eszmecsere alatt, vagy akár egyetlen szereplő nagyobb lélegzetű megnyilvánulásán belül is, és akár többszörösen egymásba is ágyazódhat. Vagyis az elhangzott/leírt mondatok csak egymás viszonylatában, az értelmezés során felépített hierarchikus környezet- dobozrendszeren belül értelmezhetők.
- Az *élethosszíglaniság* az ilyen elvek szerint felépített tudástárakban a tudás folyamatos összegyűlését jelenti.
- *Teljes lexikalizmus*, azaz a nyelvi elemzés szemszögéből minden nyelvi információ kizárólag lexikai (szótári) bejegyzésekben van tárolva.[123] Ennek egyik következményeképpen az egyes mondatelemek kapcsolódásai nem Chomsky-féle vagy ahhoz hasonló általános szabályokként, hanem az egyes szavak szótári alakjaihoz rendelt *kínál-követel rendszerként* vannak tárolva. A szóalakok kapcsolódásában pedig egyfajta *rangparaméteres kötéserősség* játszik szerepet. Emiatt az elemzés során nem a mondatszimbólumból kiinduló szabályrendszert használjuk, és nem is feltétlenül építünk fel afféle mondatszerkezetet, mint amilyet a Chomsky-féle nyelvek, vagy az abból kiinduló, azt továbbfejlesztő nyelvelméleti iskolák elemzései során szokásos. Az elemzés az emberi szövegértelmezés sajátosságához sokkal közelebbi módon történik: egy mondat vagy egy hosszabb szöveg esetén már az első szó vagy szószerkezetek elhangzása is következtetési műveleteket indít el, és egyes szavakat a kínál-követel rendszer és a kötéserősség alapján összekapcsol. A következtetéseknek nem egy Chomsky-féle nyelvi elemzési fa, hanem a mondat tartalmát a lehető legpontosabban tükröző *logikai alak* létrehozása a célja.
- Az elemzések során szakít azzal a Montague által hirdetett elvvel, miszerint a nyelvtani (szintaktikus) és a tartalmi (szemantikus) elemzés elválasztható egymástól. Nem tartható az, hogy a nyelvtani elemzés tartalmi kérdéseket egyáltalán nem vizsgál, és az általa létrehozott nyelvtani elemzési fa egyfajta átalakításával juthatunk a mondat logikai alakjához. A ℜeALIS modell a logikai alak építőköveit szintén a lexikonban tárolja, amelyek a fentebb említett

[122] Alberti Gábor: ℜeALIS. Interpretálók a világban, világok az interpretálóban [AG11]

[123] Kleiber Judit: A totális lexikalizmus elméletétől a kísérleti implementációig [KJ10]

nyelvtani kötések segítségével kapcsolódnak össze, de az *összekapcsolódással egyidejűleg a logikai alak is felépül* a szótárban tárolt elemi építőkövekből.

- A ℜeALIS modell a diskurzusreprezentációs iskolát követi: a kölcsönösségi elvre alapozva egyértelműen a *mondatoknál szélesebb egységeket* elemez. Az almondatok és a mondatok között az elemzés során retorikai viszonyokat épít fel: ezek a retorikai viszonyok egyrészt a részmondatok szerepére (magyarázat, következmény, stb.), másrészt modális viszonyokra utalnak (X tudja, látja, hiszi, stb.). Ezek egyfajta értelmezési környezetet, egy keretet hoznak létre, ezáltal az egész mondat/bekezdés/párbeszéd modellje egy többszörösen egymásba ágyazott dobozrendszert épít fel.
- Az elemzés során a ℜeALIS által létrehozott szerkezet *négy alaprelációt* épít fel, amelyek a következők:
    - Az α (alfa) függvény az egyes *objektumreferensek horgonyzásáért* (összekapcsolásáért) felelős. Ez az, amelyik az egyes részmondatokban bevezetett új referenseket összeköti a későbbi mondatokban (pl. névmások által) ráutaló egyéb referensekkel, valamint az ontológiában esetleg azonosítható példányokkal. Fontos megjegyzés, hogy az α ekvivalenciareláció: az egymással megegyező, de esetleg különböző módokon meghivatkozott objektumpéldányokat köti össze.
    - A λ (lambda) függvény a *részmondat-keretek közötti viszonyokat* írja le. Ezek részben a korábban már említett retorikai relációk, másrészt pedig a részmondatok közötti modális logikai relációk. Mindezek miatt a lambda biztosan nem ekvivalenciareláció.
    - A κ (kappa) függvény az elbeszélés általános *keretéül adódó időpontot, helyszínt, a beszélő személyét* és egyéb pontokat rögzíti. Ezek a korábban már említett kurzor-működésmód kereteit is adják.
    - σ (szigma) eventuális függvény a szövegelemekből a logikai alakokba történő leképezést rögzíti. Az eventuális függvény ilyen módon a lexikai elemekre, de a belőlük felépíthető mondatokra is értelmezhető.



48. ábra A ℜeALIS nyelvi forma- világ- és elmereprezentációjának összefüggése

- Az eventuális függvény által létrehozott logikai alak *egy multimodális, többszereplős (multi-agent) logikai rendszerben* értendő. A multimodalitás elsősorban episztemikus/doxasztikus jellegű, tehát a közbeszédben leginkább előforduló modalitásokat (BDI, belief, desire, intention/tudás, hit, érzékelés, vágy, szándék stb.) írja le. A modalitásnak mindenképpen tartalmaznia kell az időmodalitás valamilyen elfogadható megoldását. Amennyiben a rendszert különleges alkalmazásokba építjük bele, akkor esetleg még ennél többet is: pl. a jelen értekezés tárgyát képező jogi szakértő rendszerben alig kerülhető meg a deontikus modalitásfogalom kezelése.
- Fontos megállapítani, hogy a multimodális episztemikus rendszerben az *interpretálók elmetartalmát* (vagy annak egy részletét) ábrázoljuk, miközben valamelyikük megegyezik a beszélővel magával (Én), egy másikuk pedig a párbeszédes partner lesz (Te). A megértett információtartalmat, illetve az annak megfelelően felépített szerkezetet azonban nem dobhatjuk el: azt valahol, valahogyan tárolnunk kell. A tárolást magát, illetve a tárolás stratégiáját *tudáselhelyezésnek, akkomodációnak* nevezzük. A tárolás pontos helye az episztemikus világocska-rendszer valamelyik *világocskája* lehet. A szövegmegértés folyamatát pragmatikus heurisztikákkal vezérelhetjük: egy hiszékeny szövegmegértő, vagy egy teljesen megbízható beszélő esetén mindent tárolhatunk a megértő saját világában, míg egy kellően kritikus szövegmegértő vagy nem megbízható beszélő esetén csak a megfelelő episztemikus részvilágban. Esetleg, ha már sokan mondták a megértőnek ugyanazt az információdarabot, akkor azt „elhiheti", vagyis az eddig az episztemikus világocskában tárolt információ átkerülhet a saját tudást tároló gyökérvilágba. Hasonlóan: az episztemikus részvilágok láncolata (én tudom azt, hogy te tudod azt, hogy ő tud valamit rólam) pár rekurziós mélység után már lényegileg követhetetlen, ezért a maximális mélységet akár paraméterként is megadhatjuk, és az *episztemikus mélységet* ilyen módon heurisztikusan korlátozhatjuk.
- Végül az egyes predikátumok szemantikus értelmezésekor a felderített igevonzatos szerkezetet illeszteni kell az ontológia megfelelő elemére. Ezt a szokványos vonzatkeretlisták használata helyett az Alberti által leírt *polaritásos hatáslánccsalád-modellben* célszerű ábrázolni.[124]

[124] Alberti, Gábor-Kilián Imre (2010): Vonzatkeretlisták helyett polaritásos hatáslánc-családok – avagy a ℜeALIS σ függvénye [AGKI10]

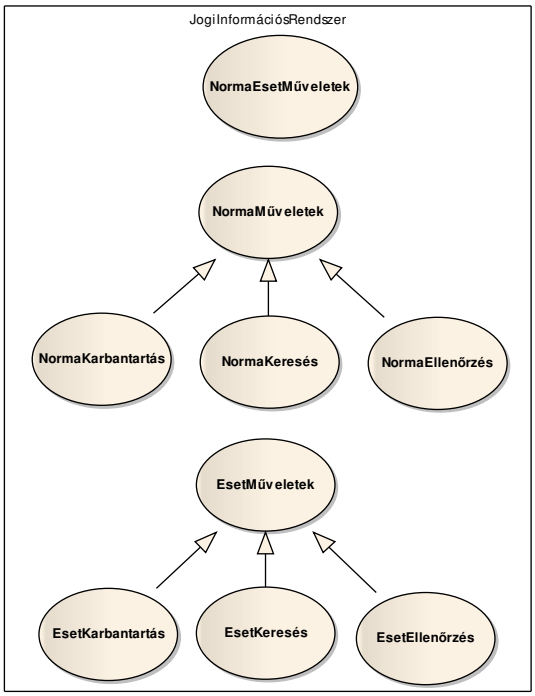
4 Egy jogi szemantikus rendszer terve

4.1 A feladat rögzítése és tervkészítés

A Rational Unified Process (RUP) szoftverfejlesztési módszertan,[125] de a szoftverfejlesztés korábbi módszertanai is a fejlesztés felhasználói igények által vezéreltségét hangsúlyozzák. Ez nem jelent többet, mint annyit: olyat fejlesszünk, amire igény is van. Ennek megfelelően a szoftverfejlesztés legelső fázisa az *igényfelmérés és követelményelemzés*.

A követelményelemzés során még nem a szoftver tényleges fejlesztése történik, hanem csak a fejlesztés peremfeltételeinek a rögzítése. Nem arra törekszünk, hogy a felvetett kérdésekre választ is adjunk, hanem megelégszünk a kérdések megfogalmazásával. A probléma, a felvetett kérdések pontos kifejezésével már *félúton vagyunk* a megoldás felé. A nagyobb probléma az, ha nem is tudjuk, hogy mi a probléma, mert nem tudjuk szabatosan megfogalmazni, leírni, átlátni, netán rendszerezni. A következő szakaszban egy elképzelt jogi szemantikus rendszer használati eset elemzése olvasható. Az alábbi elemzés sok szempontból is elvont: minthogy konkrét felhasználók konkrét igényeit nem volt mód számba venni, egy konkrét terv helyett csupán elképzelt felhasználók elképzelt igényeinek a lehetőségeire és az ezzel összefüggő műveletkörökre mutatunk rá.

4.1.1 Használati eset elemzés



49. ábra Jogi információs rendszer lehetséges műveletcsoportjai

A RUP módszertan az UML eszközeire épít, és több más mellett az úgynevezett *használati eset (use case) diagramokkal* siet a szoftver mérnökök segítségére. Az ezt létrehozó lépés, a használati eset elemzés során leírjuk, hogy milyen műveleteket valósít meg a kifejlesztendő szoftver, (és nem írjuk le azokat, amelyeket nem). Meghúzzuk a rendszer határait, és meghatározzuk a rendszer határain kívül elhelyezkedő azon emberi és gépi szereplőket, akiknek bármilyen közvetlen kapcsolatuk lesz a rendszerrel. A használati eset elemzés során három részmodellt készítünk:

- a *szereplőszerkezet* a kívülről a rendszerhez kapcsolódó emberi és gépi szereplők osztályszerkezetét rajzolja fel az objektumorientált öröklődési elveknek megfelelően.
- az *áttekintő használati eset diagram* a legfontosabb művelet-csoportoknak megfelelő absztrakt

[125] Raffai Mária: Egységesített megoldások a fejlesztésben [Ra01]

használati eset osztályokat és a hozzájuk kapcsolódó szereplőosztályokat mutatja be

- a *részletes használati eset diagram* a használati esetek osztályszerkezetét, illetve azok egymáshoz kapcsolódását, a forgatókönyv-részforgatókönyv viszonyokat mutatja be.

Egy jogi rendszer használati eset elemzéséhez a jogalkalmazás alapmodelljéből lehet kiindulni. Eszerint a jog nem más, mint *normarendszer*, amely a kívánatos viselkedésmódra és tevékenységre vonatkozó korlátokat és szabályokat rögzíti. A normarendszer a világra, az életre, de nem annak az egészére, hanem csupán egyes mozzanataira, az egyes megvalósulásaira vonatkozik. Ezeket a megvalósulásokat *eseteknek* nevezzük.

Modellezési-informatikai szempontból célszerű a normarendszert egyfajta *modellnek* tekintenünk. A modell nem más, mint a neki megfelelő esetek halmaza. A normáknak tényleg megfelelő eseteket *normakövetőnek*, a nem efféléket viszont *normaszegőnek* nevezzük.

Ha a normákat *halmazoknak*, vagy még inkább *modellnek* tekintjük, akkor egy konkrét norma meghatároz egy halmazt: a normát követő esetek halmazát. Ha több, egyenrangú normának együttesen megfelelő esetet keresünk tehát, akkor az az egyes normáknak megfelelő esetek metszete lesz.

A fent vázolt rendszer mellett megemlíthetjük még a *természeti törvények* rendszerét. Ezt tekinthetjük hasonló korlátrendszernek, mint a normarendszert. Alapvető különbség viszont, hogy nem létezik – nem létezhet – a természeti törvényekkel kapcsolatos normaszegő magatartás. A normarendszerrel szemben támasztott fontos elvárás az, hogy nem mondhat ellent a természeti törvényeknek, azaz nem létezhet olyan eset, amely beleesik a normarendszer kötelezettséghalmazába, de nincs benne a természeti törvények megengedte lehetőségek halmazában.

Egy általános jogi információs rendszer esetében a következő műveletekről lehet szó:

4. Normaműveletek. Ez a műveletcsokor a normák kezelésével, szerkesztésével, módosításával kapcsolatos műveletek összegzése. Ez a következő részműveletekből áll:
    - Norma karbantartás. Ez a művelet normák felvitelét, törlését, szerkesztését teszi lehetővé.
    - Norma keresés. Ez a művelet a rögzített normák közötti különféle szempont szerinti kereséseket teszi lehetővé.
    - Norma ellenőrzés. Ez a normarendszerre vonatkozó általános jellemzők ellenőrzésére szolgál. Ilyenek lehetnek pl.
        - Ütközés vizsgálat. Minden normarendszerben előfordulhatnak ütközések (norma-kollíziók). Az ütközések bizonyos esetben feloldhatók: pl. ha az egyik norma egyértelműen magasabb rendű a másiknál (lex superiori), más esetben azonban nem. Az ütközések kiderítése egy általános jogi rendszer alapvető művelete lehet.
        - Átfedés vizsgálat. Normarendszerek egyik jellegzetessége, hogy több norma is ugyanarra vonatkozik. Ha ilyenből túlságosan sok

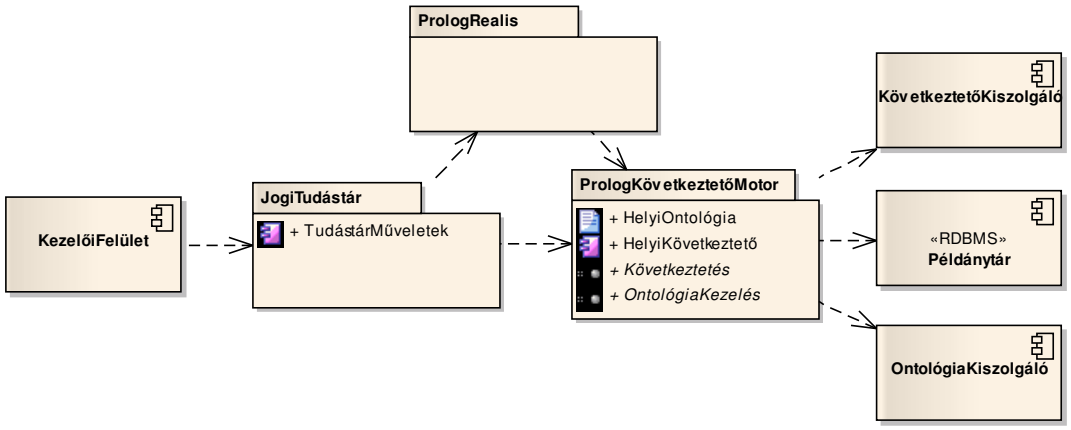
van, az mindenképpen feleslegesen bonyolítja magát a normarendszert.

- Normarendszerek egészére vonatkozó jellemzők kiszámítása. A művelet a legegyszerűbbektől (darabszám, méret), a legbonyolultabbakig (bonyolultság, entrópia) terjedően számíthatja ki a normarendszerek jellemzőit.

5. Esetműveletek. Ez a műveletcsokor a géppel rögzített esetleírások, illetve az esetek dokumentumainak kezelésével, szerkesztésével, módosításával kapcsolatos.
   - Esetek karbantartása. Ez a művelet-esetek felvitelét, törlését, módosítását, esetleg csatolmányok feltöltését vagy törlését jelenti.
   - Eset keresés. A művelet a tárolt esetek közötti több-szempontú keresést valósítja meg.
   - Eset ellenőrzés: A művelet a tárolt esetek egymáshoz képesti viszonyainak felderítésével foglalkozik. Ezek közül a legfontosabbak:
     - Átfedő esetek azonosítása. Két esetet teljesen átfedőnek nevezünk akkor, ha csak a szereplők, a helyszínek és az időpontok különböznek. Esetek lehetnek ezen túl részlegesen átfedőek, vagy teljesen különbözőek. A művelet az átfedés mértékét van hivatva megállapítani.
       - Az átfedés egyik különleges esete, amikor az egyik eset teljesen lefedi a másik eset egy részét. A művelet ennek a tényét, illetve a nem átfedő rész arányát van hivatva megállapítani.
     - Ellentmondó esetek azonosítása. Két esetet akkor mondunk ellentmondónak, ha az esetek egyes állításai – a személyektől, helyszínektől és időpontoktól eltekintve is – különbözőek, ellentmondóak. A művelet ennek a tényét, illetve az ellentmondó részek arányát állapítja meg.
6. Norma-eset műveletek. Ez a normák és esetek egymáshoz való viszonyát megállapító műveleteket tartalmazza. Az eset lehet *normaszegő* vagy *normakövető*. Normaszegés fennállásakor fontos megjelölni azokat a normákat, amelyeket az eset ténylegesen megszegett.

4.1.2 Egy szemantikus jogi keresőrendszer felépítménye

Egy szoftver rendszer felépítménye alatt azokat az alapvető szoftver és hardver eszközöket, összetevőket, valamint azok összekapcsolási megoldásait értjük, amelyek a feladat célkitűzéseit teljesítik. Megjegyezzük, hogy az alapfelépítmény független a jogi szakterülettől: szemantikus- és nyelvtechnológiát egyidejűleg alkalmazó bármilyen szoftver valami efféle felépítményen működhet.



50. ábra Szemantikus jogi rendszer vázlatos felépítménye

A felépítményt áttekintő csomagfüggőségi diagrammal ábrázoljuk, melynek legfontosabb elemei a következők:

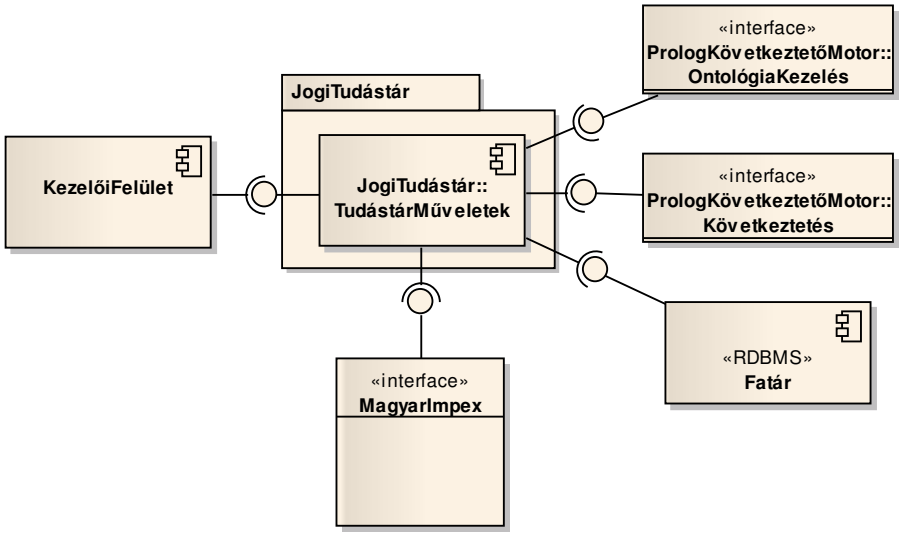
- Az előtérben a *kezelői felület* működik. Ez a legcélszerűbben valami böngészőre alapuló vékonyügyfeles megoldású lehet, de éppenséggel saját (dedikált) ügyfél, netán a tudástár kiszolgálót magába olvasztó vastag ügyfeles megoldás is elfogadható lehet.
- A *Jogi Tudástár* csomag az alkalmazói üzleti logikát valósítja meg, tehát ez áll a rendszer középpontjában. Ezt szokásos középrétegnek is nevezni, bár ez az elnevezés elsősorban a klasszikus háromrétegű felépítményben lehet használatos, míg a mi esetünkben egyes rétegek felhasadhatnak. Szándékosan nem részleteztük a Tudástár Műveletek összetevő szolgáltatásait sem: ha maradunk a keresés mellett, akkor csupán az ezzel összefüggő műveleteket kell itt megvalósítanunk. Ha további műveleteket is tervezünk, akkor ezt az összetevőt kell bővítenünk. A következő szakaszban a csomag működését részletesen elemezzük.
- A *Prolog Realis* csomag a természetes nyelvű szövegelemző és szöveggenerátor programcsomagokat valósítja meg. A névadás azt jelzi, hogy pillanatnyilag egyedül a ℜeALIS természetes nyelv feldolgozó modell lehet képes a szükséges műveletek kellő finomságú megvalósítására. A csomag futhat külön kiszolgálóként is, de ha az alkalmazást is Prolog nyelven írjuk, akkor együttesen (integráltan) is megvalósíthatjuk. A természetes nyelvi csomag működését alább bemutatjuk részletesebben is.
- Mind az Tudástár Műveletek csomag, mind a PrologRealis csomag intenzíven használja a *Prolog Következtető Motor* szolgáltatásait. A rendszer itt is skálázható, a telepítésnél választhatunk: a motort lehetséges külön szoftver összetevőként is megvalósítani, de a Prolog megvalósításokat egyetlen gépre is integrálhatjuk.
- A Prolog következtető motoron keresztül a következő további külső (Interneten elérhető) szolgáltatások kapcsolhatók be választhatóan:
    - *Következtető kiszolgáló*: az Interneten sok következtető szolgáltatás elérhető Web-alkalmazásként. Ezek szabványos programozható felülettel

is rendelkeznek. Ilyen pl. az Apache Jena projekt,[126] illetve a DL Implementation Group/DIG felülete.[127]

- A következtetés alapjául szolgáló ontológia importálással hivatkozhat más ontológiákra is. Ezeket szolgáltatná az *Ontológia Kiszolgáló*. Ez lehet on-line hozzáférésű kiszolgáló is, de a sejtésünk az, hogy a hozzáférést a futás előtt, a program-betöltés idejében, off-line módon célszerű inkább megvalósítani. Ez esetben az Ontológia Kiszolgáló közönséges fájlkiszolgálóként, (vagy http protokollba ágyazott ftp kiszolgálóként) érhető el.
- Már a legszélesebben elérhető közcélú ontológiák is tekintélyes mennyiségű példányinformációt tartalmaznak. Ezek nagy része olyan, amelyek beletartozhatnak a nyugat-európai vagy amerikai közhely-tudásba (pl. egyes amerikai elnökök), de a magyarba nemigen, nem tartalmazzák viszont a magyar közhelytudás elemeit (pl. Petőfi Sándort). A már rögzített tudáselemeket nem lenne célszerű törölni, viszont a használat szerint szegmentálni kell. Csak az adott kultúrában gyakran használt tudáselemeket kell folyamatosan a belső tárban elhelyezni. A ritkábban használt tudáselemek inkább egy külső relációs adatbázisba kerülnek. A *Példánytár* célszerűen szintén külön adatbázis-kiszolgáló gépen található.

4.1.2.1 A Jogi Tudástár alkalmazói csomag felépítése

Tekintsük most az üzleti logikát megvalósító, az előzőekben említett Jogi Tudástár csomag működését részletesebben:



51. ábra Az üzleti logikai csomag részletesebb ábrázolásban

Ha maradunk a keresőrendszer elképzelése mellett, akkor itt a keresőműveletek megvalósítását találjuk, de ebbe a csomagba igény szerint különböző további elemeket is be lehet illeszteni. A működését tekintve a rendszer a PrologRealis csomagban

[126] http://jena.apache.org, elérés: 20-Jan-2013

[127] http://dl.kr.org/dig/, elérés: 20-Jan-2013

megvalósított, és a MagyarImpex felületen keresztül elérhető természetes nyelvű feldolgozó szoftveren keresztül olvas be normatívákat és esetleírásokat, majd azokat tárolja. A tárolás kétféleképpen is történhet:

- Amíg csupán egy *intelligens keresőművelet* megvalósítása a cél, addig az ontológiavezérlés feladata csupán segítségnyújtás a megfelelő keresőkérdés összeállításában, de a tárolt tudás felhasználásával következtetéseket levonni nem szükséges. Ezért az sem szükséges, hogy a jogi adattartalom a következtetőgép tárolási formátumát (lényegében a korábban említett leíró logikákat) kövesse. A *magyar szövegimport* eredménye valamiféle logikai szerkezet, illetve ilyenek valamilyen halmaza, amit közvetlenül is tárolhatunk egy relációs adatbázisban (ld. a Fatár összetevőt). Itt most nem részletezzük, hogy egy logikai szerkezet hogyan tárolható relációsan, és arra sem térünk ki, hogy a Fatár a többi adatbázis összetevővel együtt, vagy külön kiszolgálón van megvalósítva. Mindenesetre ez, a *közvetlen tárolásos* megoldás a logikai szerkezetek közvetlen relációs tárolásának a megoldása alapján egyszerűbb, könnyebb és olcsóbb is lehet, mint az továbbiakban részletezett ontológiai tárolásmód.
- Ha azonban *jogi jellegű következtetési műveletek* megvalósítása is cél, akkor a magyar nyelvű import eredményét előbb logikai alakra kell hozni, és az egészet a háttér-ontológiában kell tárolni. Az *ontológiaszerű tárolás* alapjait a későbbi szakaszokban mutatjuk be.

Az üzleti logikát tartalmazó Jogi Tudástár csomag tervezési és létrehozási kérdéseit nem tárgyaljuk a fentieknél részletesebben, mert a hivatkozott két csomagtól (a magyar import/export műveleteket végző PrologRealis természetes nyelvi csomagtól és a Prolog Következtető Motortól) eltekintve a létrehozás meglehetősen kézenfekvő.

4.2 A ReALIS elemzőprogramjának és logikai keretrendszerének terve

4.2.1 Szoftver felépítmény

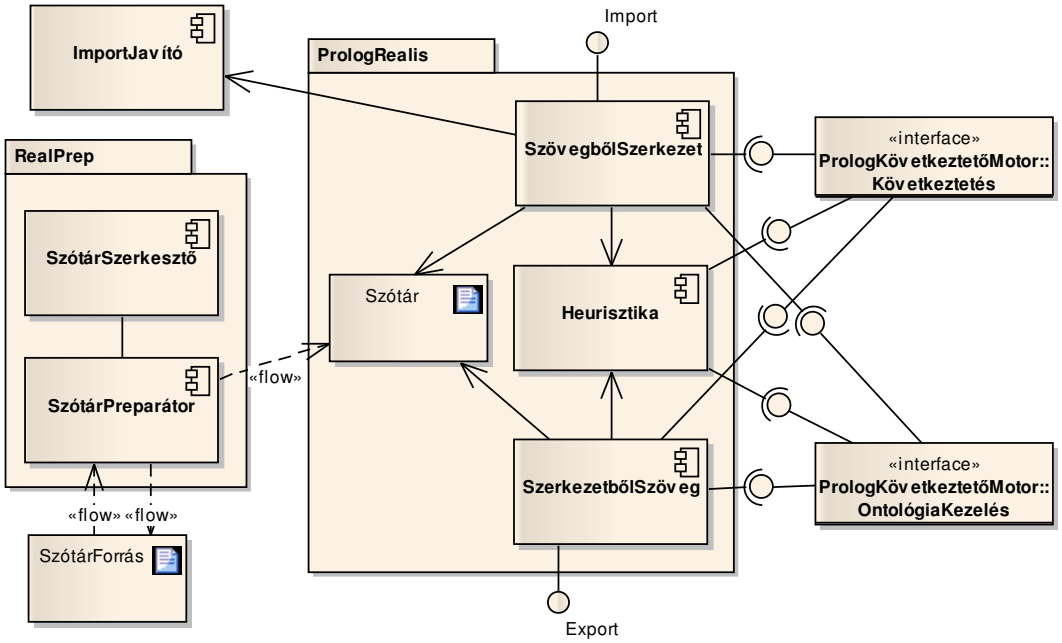
A magyar természetes nyelvű szövegimport műveletét egyfelől különösen igényesen kell megvalósítani (ezért választottuk a célra a ReALIS nyelvi modellt). Másrészt viszont ezen a felületen keresztül nem csak pár mondatos bekezdések vagy egyes dokumentumok tartalmát olvassuk be és értelmezzük, hanem itt kell a *tömeges import* műveletét is megvalósítani. A tömeges import műveletre a már ma is létező szöveges törvényi adatbázisok, valamint esetleírások tömegesen digitalizálása végett van szükség: a szöveges formát be kell olvasni, logikai formára kell hozni, és az ontológiában vagy a háttértáron el kell őket tárolni. Ez a következő részműveletek megvalósítását vonja maga után:

- Feltehetőleg mind a törvények, mind az esetek tartalmaznak – akár csak vétlen elütésekből, esetleg félreértésből, pongyolaságból, pontatlan fogalmazásból fakadó – ellentmondásokat. Az import műveletnek ezért az eredmény utólagos *ellenőrzésére (verifikálására)* is képesnek kell lennie…
- Az esetleges ellentmondások vagy hiányosságok esetén hibajelzésre, és esetlegesen javítási lehetőségre is szükség van. Ezért a magyar szövegimport művelete nem képzelhető el valami kötegelthez hasonló feldolgozásban, amely elindul, feldolgozza a bemenő szövegeket, és létrehozza az eredményt, esetleg csupán egy logikai értékkel jelzi a sikertelenséget. A szövegimport műveletnek *önálló kezelői felülettel* kell rendelkeznie, amely a hibák taglalásán túl esetleges

további *felhasználói párbeszédre, pontatlanságok, kétértelműségek tisztázására* is alkalmas.

A *magyar természetes nyelvű szövegexport* az importtal ellentétes műveletet végzi: a szerkezeti alakból előállítja valamely szövegegység szöveges képét. A művelet szerepe kétféle lehet.

- Amennyiben a végső tárolás szerkezeti vagy logikai szinten történik, akkor a szövegexportnak képesnek kell lennie az egyes szerkezetileg megadott törvények vagy esetleírások sokmondatos szöveggé történő visszaalakítására. Ennek a megoldásnak a hátránya az, hogy esetleg tartalmilag ugyanazt kapjuk vissza, de betűhelyesen nem, mert a szöveggenerátor kicsit másképpen állítja vissza a szöveget a szerkezeti alakból, mint az eredetileg volt. Ha ez a mellékhatás nem elfogadható, akkor a szövegeket a szerkezeti képükön túl redundáns módon, szövegesen is tárolni kell, és a szerkezeti alakban a szövegelemekre mutató hivatkozásokat is el kell helyezni.



52. ábra A ReALIS magyar természetes nyelvi szövegértelmező/-generáló rendszer

- Előfordulhat, hogy egyes következtetési műveletek eredménye nem csupán skaláris érték, melynek megjelenítése kézenfekvő, hanem logikai szerkezet. Az ilyeneket a kezelői felületen szövegesen kell megjeleníteni, amit szintén az export művelettel lehet létrehozni. Megjegyezzük, hogy erre a célra általában kevesebb igényű szöveglétrehozó (export) csomag is elegendő lehet, hiszen ezek az üzenetek általában rövidek, és nem egy kisregényt kell tudni összefüggő formában kibocsátani, hanem legfeljebb néhány magyar mondatot

A fenti rendszertervben található elemek és működésmódjuk:

- A rendszer felhasználja a Prolog Következtető Motor által közzétett felületek szolgáltatásait: úgy az Ontológia Hozzáférést, mint a Következtetést. Mindez a szövegelemzési művelet megvalósításához szükséges: az eközben elért tudástár nagyobbrészt valamiféle közhelytudást jelent, ami nem teljesen azonos a

tudástár által kezelt és tárolt jogi tudással. A kétféle célra használt tudástár éppenséggel külön is megvalósítható, ebben az esetben a közhelytudás vagy egyáltalán nem módosulhat, vagy a módosítást a jogi tudásszegmensben nem tudjuk azonnal átvezetni. Kézenfekvő az a megoldás is, hogy a két működésmódot együtt, egyetlen csomópontban valósítjuk meg. Ilyenkor szükséges lehet a tudásszegmenseket paraméterezhetővé tenni, ezzel lehetővé téve, hogy a kétféle működés ne teljesen ugyanazon a tudásmennyiségen dolgozzon. Ebben az esetben is lehetséges az elemző és generáló algoritmusoknak és a tudáshozzáférésnek az egyetlen Prolog környezetben történő integrálása.

- A következtető modult közvetlenül és közvetve is meg lehet hajtani. Közvetve meghajtás alatt a Heurisztika csomag beiktatásával történő meghajtást értjük. Az alapműködést tekintve a bizonyítás során a teljes ontológiát, sőt, annak teljes szabálykészletét is használjuk. A heurisztika az alapműködését tekintve a teljes tudáskészlet feletti szűrést tesz lehetővé. A szűrés feltételei beállíthatók külső, statikus módon, de beállíthatók egyes szabályok tevékenységrészében is. Ezáltal lehetséges egyes időpontokban és egyes objektumok kezelése során egyes szabályhalmazok aktivizálása és mások letiltása is. Az elképzelhető szűrési feltételek:
    - Az információelemek, szabályok időbeli hatálya alapján. Ez egyfajta „időgép” szerű működésmódot jelent. Míg a rendszer alapértelmezésben az aktuálisan hatályos tudásbázison dolgozik, szükség van arra, hogy bizonyos következtetési feladatokat valamiféle korábbi időpillanatra vonatkozva oldjon meg (Pl. jogi kérdések esetén alapfogalom, hogy minden cselekedet az elkövetésekor hatályos jogi környezetben vizsgálandó).
    - Az információelemek, szabályok tematikus kötődése alapján. Célszerűnek látszhat a jogi tudástárban tematikus belső szerkezetet létrehozni, és egy-egy konkrét kérdés vizsgálatakor csak az adott szakterület, illetve ahhoz képest általánosabb szakterületek közös, alapozó tudásanyagát felhasználni (teljesen felesleges egy közlekedési szakszöveg értelmezésekor pl. az orvosi szakma egyes fogalmait is vizsgálni).
    - Egyes tipikus cselekedetek környezet-, vagy keret- szerűen kiemelhetők, és ilyenkor ezekre vonatkozó különleges információk és szabályok aktivizálhatók. Például a korábban már említett „házasságkötés” példa esetén a rendszer következtetéseket vonhat le a szereplők nemére, vallására, nemzetiségére, a lakodalom jellegére, a nászútra stb. vonatkozólag, még akkor is, ha ezzel kapcsolatosan a szövegben közvetlen információk nem voltak. (Például a következő mondatból: „A banda még hajnalban is húzta a lassú csárdást” következtetni lehet arra, hogy valami parasztlakodalom lehetett, amelyen hagyományos zene szólt, legvalószínűbben magyar kultúrkörnyezetben.)
- A SzövegbőlSzerkezet és a SzerkezetbőlSzöveg csomagok a természetes nyelvű import és export műveleteket valósítják meg. Már korábban említettük, hogy a két irány együttesen (Prolog-szerűen, ugyanazzal a szoftverrel) nemigen valósítható meg, és igényességben sem feltétlenül ugyanaz a két csomaggal szemben támasztott követelmény. Az is elképzelhető, hogy egyes alkalmazói

szoftverek csak az egyik vagy a másik műveletre tartanak igényt. Mindezek indokolják, hogy miért célszerű a két irányt különválasztva, külön szoftver csomagban (Prolog modulkészletben) megvalósítani.

- Mindkét csomag használja viszont az egyetlen és közös Szótár adatszerkezetet. Ennek a konkrét formátuma egy ℜeALIS megvalósítási kérdés.[128] Mindenesetre a Szótár adatszerkezet alatt nem feltétlenül valami tényállításokba besűrített adatszerkezetet értünk, hanem egy olyan Prolog alapú tárgymodellt, amely a legkézenfekvőbben végrehajtható – legcélszerűbben a Prologgal magával. Az ilyen felépítménynek több fontos előnye is van:
  - Számítástechnikus közhely: a programértelmezés (interpretálás) általában egy nagyságrendnyi sebességcsökkenést eredményez. Ez elkerülhető egy lefordított és futóképes szótárállomány használatával
  - A Prolog esetében a tényállításokban tárolt adatszerkezet értelmezése még egy komoly sebességcsökkentő tényezőt is jelent: a nagyméretű, azonos névjegyű predikátumok között az állítások megkeresése lassú lehet. A modern Prolog rendszer ugyan már az állítások indexelésére (gyors visszakeresésére) is kínálnak lehetőséget, ez azonban gyakran korlátozott (pl. esetleg csak a programba forrásszinten beleépített (statikus) állításokra, esetleg csak azok első paraméterére működik).
  - A belső formátumot nem szükséges rögzíteni: ha mégis sebesség vagy más gondok lépnének fel, akkor a formátum módosítható. Ennek egyetlen következménye: a fordítót (SzótárPreparátor) kell alkalmasan módosítani.
- A Szótár Preparátor modul az az imént említett fordítóprogram, amely a szótár forrásnyelvi alakját a futásidejű alakra lefordítja. A kötegelt fordítási művelet mellett a Szótár Preparátor a Szótár Szerkesztőhöz kapcsolódva egyenkénti (inkrementális) fordítási műveletet is lehetővé tesz.
- A Szótár Szerkesztő modul a Preparátor inkrementális fordítási képességére alapozva a szótári bejegyzések vizuális szerkesztési és aktivizálási (betöltési) lehetőséget biztosít.
- A Szótár Preparátor és Szerkesztő modul a szótár forrás alakjából indul ki. Ez a ℜeALIS projektum számára rögzített nyelvfüggetlen nyelvleíró nyelv (ℜeALLan).

4.2.2 A ℜeALIS modális logikai rendszere

A logikai alapokat leíró fejezetben meghatároztuk a többszereplős episztemikus modális logikákat a Kripke-féle világszerkezettel és a rajta alkalmazható kalkulus axiómarendszerével együtt. A világocskák Kripke-féle meghatározása azonban a gyakorlattól kicsit elrugaszkodott több okból is. Egyrészt egyes példarendszereken túl nem könnyű értelmeznünk és meghatároznunk egy valóságos logikai rendszer azon világocskáit vagy állapotait, amelyek fő jellemzője, hogy az egyes logikai állítások kiértékelése más és más lehet az egyes állapotokban. Jó lenne az ösztönös episztemikus hozzáállást a világszerkezetben tisztábban viszontlátnunk: az a tény, hogy valaki tud valamit, csupán egyetlenegy helyen és világban legyen ábrázolva, és ne kelljen az ilyen

[128] Kilián Imre: Tárgymodell változatok a ℜeALIS nyelvi elemzéshez [Kil11.2]

adatot az összes elérhető világot végigjárva összegyűjtenünk. Másrészt a megvalósítás szemszögéből célunk az is, hogy ne kelljen a tudáselemeket minden alternatív világban külön ábrázolnunk, hanem a világok felett értelmezhető legyen valamiféle – az objektumközpontú programkészítésből ismert – *öröklődési viszony*. Az egyes szereplők tudása nagy mértékben átfedhet – legalábbis a hétköznapi tudást tekintve mindenképpen – legyen hát lehetséges a háttérben a közös tudást egységesen és egyetlen példányban ábrázolnunk. Az egyes szereplők esetében csak az ehhez képest értelmezhető *különbségtudást* ábrázoljuk, és a közös tudást öröklődéssel érjük el.

Alberti, Kleiber és Károly közleményeiből[129] [130] [131] ismerhetjük a tudás ábrázolásának modelljét. Eszerint a tudás egy $\mathfrak{R} = \langle W_o, W, Dyn, Tru \rangle$ többszereplős, episztemikusan modális világocska-rendszerben ábrázolható. Ez a világok felett egy faszerkezetű viszonyt feszít ki, amelynek W_o a gyökéreleme, amely a megváltoztathatatlannak és félreérthetetlennek tekinthető külvilágot jelképezi. A világ entitásainak egy részhalmaza a rendszer egyes ismeretszerzésre, -tárolásra és -következtetésre, valamint nyelvek értelmezésére is alkalmas szereplőinek (agents) A halmaza, amelyet első olvasatban a W_o objektív külvilág egy részhalmazaként értelmezhetünk. A rendszer faszerkezetét a W=W[i,t] függvény feszíti ki úgy, hogy egy w_1 világocskától az i cselekvővel és a t időjelölővel, valamint további modális címkékkel (hit, vágy, szándék, érzékelés, álom, stb.) címkézett élek mutatnak a másik, w_2 világocska felé. Az ilyen módon képzett újabb világocskák az episztemikus mélységnek felelnek meg (X /a bíró/ tudja, hogy Y /a tanú/ nem hiszi, hogy Z-nek /a vádlottnak/ szándékában állt volna pl. egy bűncselekmény elkövetése). A tetszőlegesen iterált mélységnek nyilvánvalóan nincs értelme, ezért ezt a mélységet a rendszer paramétereként akár korlátozhatjuk is.

A fenti rendszerben egyfelől értelmezhetünk egy *statikus kiértékelési eljárást*, amely a nyelvi megértés eredményeképpen kapott kifejezések logikai alakját jelenti, amely alak egyben az elemzés során felépülő modális világocska-rendszernek megfelelő geometriai dobozrendszerrel is ábrázolható.

A rendszer *dinamikus kiértékelése* a rendszer hosszútávú élettartamára vonatkozik. Az episztemikus világocskák kezdetben üresek, vagy valamiféle kezdő tudással vannak feltöltve, ezeket azonban az öröklődési mechanizmus is bevetítheti. A rendszer az élete során kapott és kiértékelt információkat az episztemikus világocska-rendszerben *elhelyezi*. Ezt a folyamatot *akkomodációnak* is nevezzük. Az akkomodáció fontos stratégiai kérdés is: hiszékeny felhasználók esetleg megdönthetetlenként helyezik el a hallott információkat, míg – éppen a jogi munkában – a hallott információkat egy pontosan meghatározott, de a vallomástevőhöz kötődő (ő állította) modális világocskába helyezzük el. Innen aztán csak az igazságtartalom többoldalú ellenőrzése után kerülhet egyre megbízhatóbb, a megdönthetetlen globális gyökérvilághoz közelebb eső világocskába.

A fenti világocska-rendszert egyben tekinthetjük a ℜeALIS, mint modális logikai rendszer Kripke-féle keretszerkezetének is, amelyet többszereplős esetben egy c szereplő szerint szegmentálhatunk $F_c=<W,R_c>$. Ez a világhalmaz megegyezik a ℜeALIS világhalmazzal, hiszen a legfőbb ismérve, hogy bennük az egyes elemi

---

[129] Alberti Gábor: ℜeALIS. Interpretálók a világban, világok az interpretálóban [AG11]

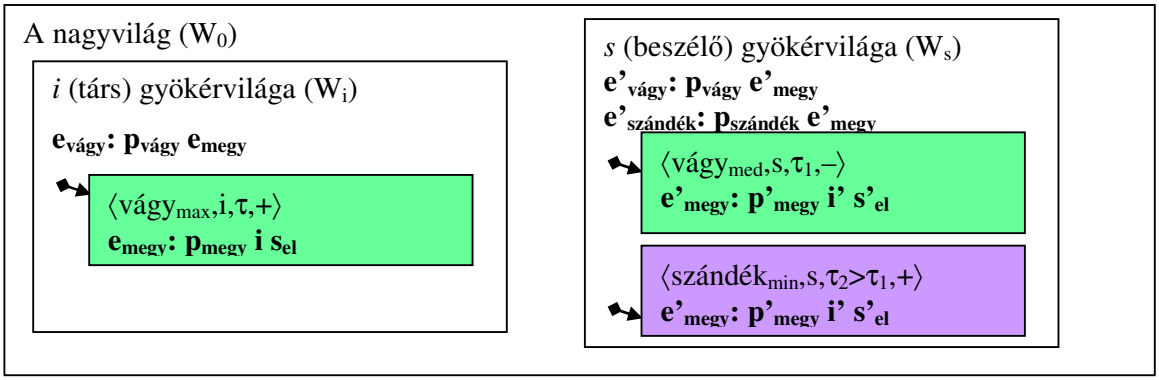
[130] Károly Márton: Interpretáció, intenzionalitás, modalitás – avagy a ℜeALIS λ függvényének interpretációja felé [KM11]

[131] Alberti Gábor-Kleiber Judit: Where are Possible Worlds? (Arguments for ℜeALIS) [AGKJ12]

állítások *különbözőféleképpen is kiértékelődhetnek* (egyes cselekvők más és más személyes ismeretségi körrel rendelkeznek, mások tévesen úgy is gondolhatják, hogy az Anyám tyúkját Petőfi helyett Arany János írta, vagy hogy Magyarország fővárosa Bukarest). Emellett létezhet nyilvánvaló ellentmondás is (pl. II. Nagy Szulimánról a törökök országépítő hősként, mi viszont zsarnok hódítóként emlékezünk meg).

A világok feletti R elérhetőségi reláció megfelel a ℜeALIS W[i,t] függvényének, ami cselekvőfüggő is; egy adott szereplő nemigen tud közvetlen különbséget tenni egy társa által a múltban vagy a jövőben tudott vagy hitt dolgok között, vagy ha igen, akkor az beépül a saját világszerkezetébe. Másrészt egy adott szereplőre nézve a tudott és hitt dolgok, valamint a valóság nemigen állhat ellentmondásban – ha mégis abban állna, akkor bizony valami súlyos elmebetegségről, autizmusról vagy skizofréniáról beszélünk. Mindeközben a tudott és a kimondott dolgok ellentmondásban állhatnak – például ha célunk a hatóság megtévesztése, – az ilyet a világocskák faszerkezetének külön ágaiban tároljuk.

Példaképpen tekintsük először az „Egye fene, elmehetsz." mondat által felépített szereplői elmeállapot ábrázolását![132] (Az ábrázolásmód követi a ℜeALIS



53. ábra Az "Egye fene, elmehetsz!" mondat elmeábrázolása

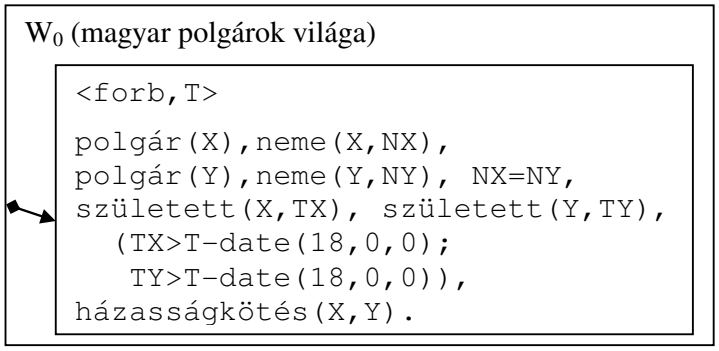
referensjelölési szokásait, és a kiszámítás működési mechanizmusát is.)

Az objektívnak tekintett W_0 nagyvilágon belül ábrázoljuk az s (beszélő) W_s és az i (társa) W_i elmeállapotát. A társ vágyvilágában erős késztetést érez a távozásra, ám az s beszélő ezt nem akarja. Egy későbbi $\tau_2>\tau_1$ időpillanatban azonban megváltoztatja ezt, és mégis gyengén szándékba veszi, vagyis elengedi.

Másik – ezúttal jogi – példaként tekintsük a CsJT 10.§-ban rögzített „Házasságot csak nagykorú férfi és nő köthet" törvény ábrázolását. A törvények nem konkrét cselekvő személyekhez, hanem a nagyvilág egy rész-világához, az egyének egy halmazához kötődnek. Ahhoz a rész-világhoz, amely éppen a törvényalkotás által érintett közösséget képviseli. („Pl. „a keresztények pénteken nem esznek húst" normatíva a nagyvilágon belül csak a keresztény kultúrkörben érvényes.) A törvény alanyait képviselő részvilágon belül a törvény leírta állapot egy további rész-világot ír le: a törvény (vagy más normatíva) által ideálisnak tartott világot. A világocska-szerkezetben csupán a merőleges (**F** és **P**) operátoroknak megfelelő (**forb,** illetve **perm**) rész világokat

[132] Károly Márton: Interpretáció, intenzionalitás, modalitás – avagy a ℜeALIS λ függvényének interpretációja felé [KM11]

ábrázoljuk. (Itt a ℜeALIS terminológiai jelzéseit már elhagytuk, és az egyes világocskákba leképezett állításokat elsőrendű logikai nyelven ábrázoltuk.)



54. ábra Törvény ábrázolása a modális világocska-rendszerben

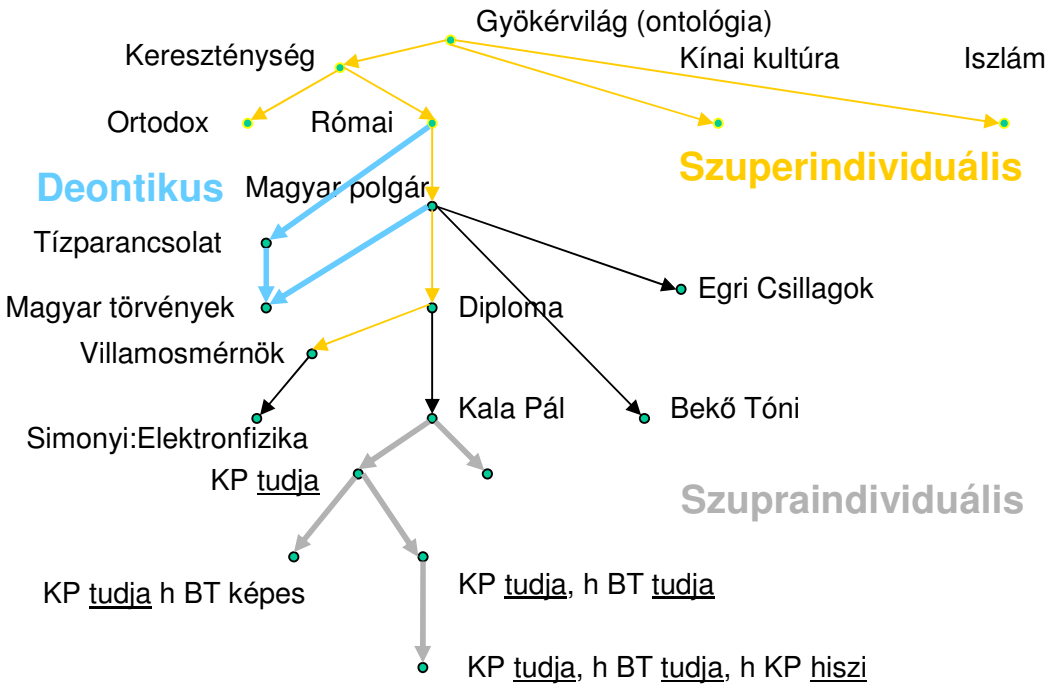
Az elemzés során a 'lehet' modális módosítót először a tényleges törvényalkotói szándéknak megfelelő 'csak így lehet' módosítóvá alakítottuk. Ezután a feltétel negálásával a módosítót 'tilossá' változtattuk, és az egész törvényszöveg logikai alakját a tiltott világocskába helyeztük.

Az iménti tárgyalásban átsiklottunk a W_o gyökérvilág és az A cselekvőhalmaz közötti réteg tárgyalásán, és ezt a fenti példában sem részleteztük. Ez a kérdés alkalmat ad a R relációhalmaz két részre, a RS és a RA halmazra történő szétválasztására. Az RS halmaz az *egyéniségek feletti (szuperindividuális) vagy csoport szintet* jelenti, és a csomópontjai az önálló tudásmennyiséggel rendelkező csoportokat jelzik. A RS relációhalmaz így szereplőfüggetlen, és egyfajta (az objektumorientáláshoz hasonló) öröklődési viszonyt értelmezhetünk rajta, azaz egy w_1 világocska nem ábrázolja közvetlenül, de igaznak tekinti az örökölt tudást is, a W_o gyökérvilág pedig a minden szereplő számára kézenfekvő és vitathatatlan ismereteket tárolja.

A csoporttudások szintjén célszerűnek látszik enyhíteni az R relációhalmaz faszerkezetére vonatkozó megkötést: egy csoport, egy szellemi iskola általában több másik szellemi iskola eredményeire és tudására épít, egy szereplő pedig több csoport tudását is összegezheti. Ezért az RS részrelációra vonatkozóan az összefutó éleket is meg kell engednünk, vagyis a részreláció nem fát, hanem körmentes irányított gráfot (KIG) feszít ki.

Az RA részreláció az *egyéni alatti (szupraindividuális)* szint, amely a szereplők hitére, meggyőződésére stb. vonatkozó információkat tárolja. Ez az R és az RS relációk különbsége, topológiailag egy erdő, melynek gyökérpontjai az egyes egyedi szereplőket (individuumokat) jelölik. Ezek általában csupán az őscsoportjaik tudását összegzik, és nincs konkrétan ábrázolt adatuk.

A csoporttudás fenti bevezetésének legfőbb célja: amennyiben a tudástárat fizikailag is egyetlen gépen tároljuk, akkor ezzel csökkentjük az ábrázolás redundanciáját és tárigényét, hiszen a közösen ismert dolgok csak egyetlenegy ponton vannak tárolva. A csoporttudások feletti öröklés a megvalósítás szempontjából nem más, mint egy *dinamikus import művelet*: technikailag az import megvalósítható *közvetlen másolással* éppúgy, mint a közös tudástárhoz történő hozzáféréssel. Ha másolással valósítjuk meg, akkor az egyes szereplők gyökérvilágában halmozódik fel a közös tudás is. Ennek talán inkább akkor van jelentősége, ha a szereplők nem egyetlen gépen, vannak ábrázolva, hanem ténylegesen elosztott rendszerként, külön-külön gépre kerülnek.



55. ábra Egy elképzelt világszerkezet a ReALIS rendszerben

A fentebb részletezetteken túl célszerű mindenféle zárt tárgyalási környezet számára szintén önálló világocskákat létrehozni. Így önálló világba kerülnek egyes *irodalmi művek*, de egyes *bírósági tárgyalások* is. Az efféle világocskák szintén valamiféle vonatkoztatási környezethez kapcsolódnak (az ábrán a néhai Simonyi Károly professzor írta Elektronfizika egyetemi tankönyv a villamosmérnöki diplomával rendelkezők számára, az Egri Csillagok – legalábbis a modellben – a magyar olvasók számára érthető).

Az említett modális faszerkezet síkjára merőleges a *deontikus világrendszer* síkja. Ez ideális világokat ír le (amiben senki sem lop, csal, stb.). Az egyes deontikus világok egyrészt a törvények hatáskörét jelképező társadalmi csoporthoz kapcsolódnak (pl. a Tízparancsolat a keresztény kultúrkörhöz, míg a magyar törvények a magyarokhoz). A törvények világa ezen kívül egymáshoz is kapcsolódik, hisz a magyar Alkotmány kapcsolódik, – de legalábbis nincs ellentmondásban – az európai vagy a nemzetközi törvényekkel, vagy éppen a Tízparancsolattal. Másrészt a különböző szintű helyi törvényerejű rendelkezések viszont kapcsolódnak, de nem állhatnak ellentmondásban az országos törvényekkel, vagy az Alkotmánnyal.

Az öröklés megvalósítása során elméleti és gyakorlati részkérdés is az, hogy az öröklés *monoton-e*? Mindkét szempontból komoly könnyítés az, ha az öröklés teljes, felejtést nem kívánunk beépíteni, és az egyes csoportok és egyéniségek legfeljebb csupán újabb tudáselemeket vehetnek hozzá az örökölt tudáshoz. A továbbiakban mindenütt ezzel a feltételezéssel élünk.

4.2.2.1 Modális operátorok

A ℜeALIS teljesen gyakorlati meggondolásokból létrehozott modális rendszer, amelyben az elméleti meggondolások eredményei nem mindig használhatók

közvetlenül. Ennek ellenére egy közvetett alkalmazás érdekében fontos áttekinteni, és esetlegesen a gyakorlatba is átemelni őket.

A Kripke-féle rendszer modális operátoraival szemben Alberti Gábor által rögzített rendszer a vegytiszta Kripke-rendszernél lényegesen bonyolultabb (és kifejezőbb) modális operátorkészletet ad meg: a rendszer ezáltal többszörösen is multimodális lesz. A modális operátorokat az alábbi négyesekből álló (csak részlegesen rögzített) halmaz írja le:

M={<L={bel,des,int,…},GR={min,med,max},A,T,P={+,0,-}>}, ahol:

- L (believe, desire, intent,…): a multimodalitásért is felelős *modális címke*. Ez az igényeknek és a megelőző nyelvtani elemzésnek megfelelően tovább bővíthető. Például a **supp**, illetve **cons** címke feltétel- és következmény-világocskát ír le, az egyes érzékelésfajtákért pedig a **hear, smell, taste, touch** címkék a felelősek. Ha a ℜeALIS-t normatívák elemzésére is fel kívánjuk használni, akkor ezt a halmazt deontikus modalitásokat kezelő elemekkel is ki kell bővíteni. A tényleges tudástárban (ontológiában) ezeket nem az alábbi gazdagságban ábrázoljuk, hanem normáljuk úgy, hogy csupán egymásra merőleges két fogalom maradjon. (pl. a **perm**, illetve a **forb**, esetleg **lib**).
  - **perm**: engedélyezettség kifejezésére („A gyermekelhelyezés megváltoztatását akkor lehet kérni, ha korábbi bírósági döntés körülményei lényegesen megváltoztak.”)
  - **obl**: kötelezettség kifejezésére („A szülő köteles megosztani kiskorú gyermekével azt, ami az eltartásukra rendelkezésre áll.”)
  - **forb**: tiltás kifejezésére („A vágányokon átjárni tilos.”)
  - **lib**: a szabadság kifejezésére („Magyarország elismeri a sajtó szabadságát.”)
- GR (minimal, medium, maximal): a *modalitás fokozata*. A <bel,min,…> jegy jelzi a hagyományos **B** (believe / gyenge episztemikus) operátort. A **K** (know) operátort a <bel,max,…> párral adhatjuk meg.
- T *időjelölő*, ami jelenthet egyszerű időpontot, időintervallumot, vagy bármilyen bonyolultabb időmeghatározást. Az időjelölőkön végezhető műveleteket közvetlenül Prolog szinten valósítjuk meg.
- A *szereplők halmaza*. A szereplők megfelelnek a háttér-ontológia egy osztályának. Az RS relációban résztvevők közbülső elemei aktív társadalmi csoportoknak (Society), a levélelemek, valamint az RA relációban résztvevők gyökérelemei aktív egyéneknek (Agent / HumanAgent) felelnek meg.
- P={+,-} pozitív vagy negatív *modális polaritás*: amely a negatív modális operátor leírására szolgál, a gyakorlatban a *konstruktív tagadás* számára használjuk.

A ℜeALIS episztemikus rendszerében az erős és a gyenge modalitásra vonatkozó univerzális és egzisztenciális feltételeket elvetjük, hiszen a világocska-rendszerben a modalitás-címkékkel explicit módon adjuk meg az egyes modalitásfajtákat. Ezért a világocska rendszer feletti meta-elérhetőségi relációk (reflexív-e, tranzitív-e, stb.) vizsgálata okafogyottá válik. Az elemzésnek az egyes modális rendszerek axiómarendszerének és a belőlük levezethető ℜeALIS axiómák létrehozására kell korlátozódnia.

A fenti, háromfokozatú modellt feltételezve a szokásos modális operátorok a következőképpen értelmezhetők, mindezt az időjelölőjük metszetintervallumára vonatkoztatva:

- A <bel,max,i,τ,P> operátor megfelel a $\mathbf{K_i}$ tudásoperátornak.
- A <bel,min,i,τ,P> operátor megfelel a $\mathbf{B_i}$ hit- vagy sejtés- operátornak
- A <bel,med,i,τ,P> operátort valamiféle erősebb hit, de a konkrét tudásnál gyengébb tudatosság – pl. meggyőződés-fogalom – formalizálásaképpen foghatjuk fel
- A <ML,GR,i,τ,P> operátor-kifejezésben ML a modális címke, GR a modális fokozat, i a szereplőazonosító.
- A τ időjelölő a modális időlogikák operátorainak (**P**, illetve **H** a múltban, **F**, illetve **G** pedig a jövőben) felel meg. A ℜeALIS **τ=before** időjelölőjének a jelentése: a *múltban valamikor* megtörtént, vagy igaz volt, míg a **τ=after** időjelölőé: a *jövőben valamikor* meg fog történni, vagy igaz lesz.
- A fenti, gyenge időmodalitást kifejező időjelölő mellett szükséges bevezetni az erős időmodalitás jelölőit. Ezt a következő két jelölővel érjük el: a **beforeA** (past always) jelentse azt, hogy valami a *múltban mindig igaz volt*, míg jövőbeli párja, az **afterA** (future always) pedig azt, hogy a *jövőben mindig igaz lesz.*

A továbbiakban azokat az axiómákat, amelyekben a modális operátorok megfelelnek az egyszerű, konvencionális $\mathbf{K_i}$, illetve $\mathbf{B_i}$ operátoroknak, konvencionális jelöléssel írjuk fel. Ettől eltérő esetben a fentebbi, részletes jelölésmódot alkalmazzuk. A részletes jelölésmód egyes elemeinek elhagyása – amennyiben még egyértelmű marad – az elhagyott elemek tetszőleges értékét jelentik.

4.2.2.2 Modális axiómák és Prolog megvalósításuk

Grosof cikkében részletesen leírja az OWL ontológiák elsőrendű logikába és Prologba történő leképezését.[133] Ohlbach cikkében[134] viszont javaslatot ad a modális operátorok kiküszöbölésére és első rendbe történő reifikációjára úgy, hogy az operátorokat kifejezés-argumentumokká képezzük le, miután minden további nélkül futtatható pl. a rezolúciós algoritmus. A Prolog ábrázolásmód nemcsak a megvalósítás konkrét szempontjából érdekes: az Ohlbach-féle javaslatokat követve, és kiterjesztve a ℜeALIS multimodális rendszerére a következő megkötéseket tehetjük:

Egy elsőrendű logikában felírt `p` literális kifejezést első paraméterként kibővítünk egy MOD modális kifejezéssel, vagyis

`p(X1,X2,…,Xn)` → `p(MOD,X1,X2,…,Xn)`

A `MOD` modális kifejezés a következő formákat öltheti:

`root` megfelel a gyökérvilágnak

[133] Benjamin N. Grosof-Ian Horrocks-Raphael Volz-Stefan Decker: Description Logic Programs: Combining Logic Programs with Description Logic [Gro03]

[134] Hans Jürgen Ohlbach: A Resolution Calculus for Modal Logic [Ohl88]

`ID`	ahol `ID` egy Prolog azonosító (logikai konstans), egyes csoportok azonosítóit jelöli, amelyek megfelelnek a háttér-ontológia „társadalmi csoport” fogalmának…
`ID`	…illetve `ID` egy Prolog azonosító (logikai konstans), amely megfelel az egyes egyéni szereplők azonosítóinak. A modális címke maga az adott szereplő gyökérvilágát jelöli.
`bel(GR,I,T,P,F)`	
`int(GR,I,T,P,F)`	
…	az egyes modalitásfajtákhoz külön modális címkeazonosító (Prolog függvényszimbólum) tartozik, ahol `GR` (grade) a modalitás fokozata, `I` a szereplőazonosító, `T` az időjelölő, `F` pedig a szülő világocska modális kifejezése
`GR`	A modalitás fokozatot célszerű lehet egész számokká leképezni (pl. 0,1,2), ahhoz, hogy esetlegesen számtani műveleteket, (pl. összehasonlítást) tudjunk rajta végezni.

A világok feletti relációt a `world/2` Prolog állítás adja meg. Ez két részből áll. A szuperindividuális szinten az egyes csoportok azonosítóira hivatkozik, amelyet az `sWorld/2` reláció tényállításonként tárol. A reláció tartalma esetlegesen egyes ontológiabéli relációkkal is kifejezhető (melyik szellemi iskola milyen másik csoporttudásra épít). A szupraindividuális szinten reláció a modális címkékből kifejezhető, pl. az alábbi, vagy hasonló Prolog kód segítségével:

```
iWorld(SUP,bel(GR,I,T,P,SUP)).
```

A korábbi fejezetben tárgyalt modális axiómák a fentiek alapján könnyen kifejezhetők Prolog nyelven. Az axiómák megvalósítását illetően az alábbi példák a *lefordított/ kompilált* megvalósítást mutatják be. Ez minden tudásrészletre és az azokból létrejövő Prolog állításra újabb állítások létrehozását (lefordítást) jelenti. Lefordított megvalósítást azonban csak a gyakran használt axiómákra célszerű csinálni. A másik hozzáállás az *értelmezett (interpretált)* axiómáké. Ez a Prolog alapfokú kurzusokból ismert metaértelmező (Prolog programot egy másik Prolog program végrehajt) egyfajta modális kibővítésével érhető el.

- **MG:** A modális általánosítás ismert axiómájának egyfajta általánosítása az alábbi. A javasolt rendszerben a *világocskahálózat feletti öröklődés* alapvető axiómaként fogalmazható meg (ϕ-vel és φ-vel a továbbiakban tetszőleges logikai kifejezéseket jelölünk):

- $\phi_{w1} \rightarrow \phi_{w2}$, ha w_1 R w_2

  A megvalósításhoz az alábbi örökítő állítást minden predikátumhoz létre kell hozni. A megoldás sajnos csak akkor alkalmazható hatékonyan, ha a csoportszerkezet topológiája szigorúan fa.

  ```
  p(SUB,X1,X2,…,Xn):- world(SUP,SUB), p(SUP,X1,X2,…,Xn).
  ```

  Az általánosabb (és valószínűbb) esetben a topológia körmentes gráf. Ilyenkor az ősöket előbb összegyűjtjük a következőképpen:

```
p(ID,X1,X2,…,Xn):-
  sAncestors(ANC,ID), member(SUP,ANC),
  p(SUP,X1,X2,…,Xn).
```

- **D/ser:** A deontikus rendszerben D-vel is jelölt *szerialitási* (ser) *axióma* szerint, ha valami kötelező (O), akkor meg is van engedve (P).

  $O\,\phi \rightarrow P\,\phi$

  Ehhez hasonló axióma a doxasztikus rendszerben azt mondja ki, hogy ha valaki biztosan tud (K) valamit, akkor azt hiszi (B) is.

  $K_c\,\phi \rightarrow B_c\,\phi$

  A ℜeALIS modális rendszerében a tudás többfokozatú. A fenti axióma általánosítása ezért az *episztemikus fokozatöröklés* axiómája:

  <bel,GR1,i,t,+> $\phi \rightarrow$ <bel, GR2,i,t,+> ϕ, ha GR2<=GR1.

  Az axiómamegvalósításban a CLPQ/R rendszert alkalmazzuk annak érdekében, hogy az összehasonlítás végrehajtása független legyen a logikai változók lekötöttségétől. (Az összehasonlítás kiértékelését addig késlelteti, amíg mindkét változó értéket nem kap.)

```
p(bel(GR1,I,T,P,F),X1,X2,…,Xn):-
  p(bel(GR2,I,T,P,F),X1,X2,…,Xn), {G1<G2}.
```

- **K**: $K_c(\phi \rightarrow \varphi) \rightarrow (K_c\,\phi \rightarrow K_c\,\varphi)$

  A Kripke-féle K axiómának is nevezett axióma elfogadása és megvalósítása (vagy elvetése) egy alapvető kérdést vet fel. Igaz-e, hogy ha ismerjük a következtetéseket, és ismerjük a nekik megfelelő feltételeket, akkor képesek vagyunk a következmény létrehozására is? Igaz-e az axióma tranzitív lezártjára is, igaz-e véges, de hosszú következtetési láncokon is? Vajon egy közismert híresség, még ha különösen intelligens is, tudja-e, hogy végtelen sok prímszám van, csak azért, mert tanult számtant az iskolában? Ezt a kérdést a *mindentudás (omniscience) kérdésének* is nevezik.

- **T:** $K_cA \rightarrow A$ Még kevésbé látjuk használhatónak az *igazolhatóság (veridicality) axiómáját*. Mérő László klasszikusában olvashatjuk, hogy egy ausztrál törzs tagjai biztosan tudják, hogy a gyermekáldást valamelyik istenség adja, és ezt különböző módon bizonyítottnak is gondolják. Másrészt gyermekeink biztosak lehetnek bizonyos angyalok közreműködésében is a karácsonyi ajándékok előkészítése terén… Az egyes modális világok ellent is mondhatnak egymásnak: ha tehát a szigorú modalitásból magára az objektív világra is következtethetnénk, akkor az is ellentmondáshoz vezetne. Az axióma Prolog megfogalmazását alább bemutatjuk, de az axiómát a fentebb már említettek miatt nem alkalmazzuk.

```
p(root,X1,X2,…,Xn):- p(bel(max,I,T,+,F),X1,X2,…,Xn).
```

- **4:** $K_cA \rightarrow K_c\,K_c\,A$ Az axióma a KT4 logikai rendszer névadója, és a *pozitív önismeret (positive introspection)* filozófiai kérdését veti fel. Tényleg tudjuk-e magunkról, hogy mit is tudunk. Igaz-e ez a tudásunk minden elemére, beleértve az ösztönös cselekvéseinket is? A ℜeALIS rendszerében az episztemikus tudás a beágyazott világocskák tudását jelenti, amelyeket ott részben explicit módon ábrázolunk, részben viszont a fenti MG axióma miatt a szülő-világocskából öröklünk. Emiatt az axióma értelemszerűen teljesül. Mindennek korlátja a

beágyazhatóság mértékére vonatkozó korlát. Javasoljuk ugyanis, hogy az episztemikus világocskákra vonatkozóan egy – esetleg paraméterezhető – mélységi korlát legyen. Ez természetesen megegyezik az episztemikus operátorok beágyazhatóságának a mélységi korlátjával is.

- **5**: $\sim K_c A \rightarrow K_c \sim K_c A$ A 4. axióma párja, az 5. axióma a KT5 logikai rendszer névadója, és a *negatív önismeret (negative introspection)* kérdését veti fel. Tudjuk-e magunkról, ha valamihez nem értünk? Ha igen, akkor csak az ismeretlen tudáscsomagok valamiféle hivatkozását, jelölőjét tudjuk ide értelmesen elképzelni, ez viszont a tiszta logikában nem ismert fogalom. Az axióma közvetlen megvalósíthatósága a logikailag tiszta negáció kérdését is felveti. Ez a Prolog kudarcalapú negációs (Negation As Failure, NAF) rendszerében egyáltalán nem kézenfekvő.

A ℜeALIS rendszer az idő modalitását a <L,GR,i,**before**>, illetve a <L,GR,i,**after**> modális címkéken és társaikon keresztül ragadja meg. Ennek azonban különféle korlátai vannak. Egyrészt a javasolt ábrázolásmód az időt csupán címkézi, a megoldás nem jelent következetes operátorhasználatot. Emiatt a rekurzív operátorhasználat megoldása egyáltalán nem kézenfekvő, azt célszerű inkább elkerülni. Az efféle időmodális rendszerekről (TL0, TL1, TL2, illetve TL5), illetve azok rekurzív axiómáiról (**antisym**, **tra**, **tri**, esetleg **den**) ha szót is ejtünk, azok pontos megvalósítását egyelőre nem tervezzük. Az időfogalom kezelésére a szigorú axiomatikus kezelésmód helyett egy önálló Prolog modult vezetünk be (`module(time,[…])`), amely az időfüggéssel kapcsolatos problémákat a megvalósítás szintjére helyezi át.

A következő időmodális axiómák megvalósítása került megfontolásra:

- Időbeli dualitás (ha valami valaha fennállt, akkor az lehetetlen, hogy az ellentéte mindig fennállt volna, illetve a jövőbeli tükörképe): Az axióma megvalósítása a kétszeres negáció, illetve annak Prolog-béli nehézségei miatt egyelőre nincs tervben.
- Időbeli általánosítás: Az axióma az alábbi két, szimmetrikus Prolog szabállyal valósítható meg:

  ```
  p(before(T),X1,…,Xn):- p(_,X1,…,Xn).
  p(after(T),X1,…,Xn):- p(_,X1,…,Xn).
  ```

  A szabályt közvetlenül nem szükséges megvalósítani, hiszen a kötetlen modális címkéjű állítás bármely konkretizáltja a Prolog egyesítési algoritmus miatt igaz lesz.
- A TL0 időmodális rendszer antiszimmetria axiómája szerint, ha valami időfüggetlenül igaz, akkor mindig igaz lesz az, hogy valaha igaz volt. (Ennek a tüköraxiómája is igaz). Ez a következő két Prolog szabállyal valósítható meg:

  ```
  p(after(beforeA(T)),X1,…,Xn):- p(X1,…,Xn).
  p(before(afterA(T)),X1,…,Xn):- p(X1,…,Xn).
  ```
- A TL0 időmodális rendszer tranzitivitási axiómája szerint, ha valami mindig igaz lesz, akkor mindig igaz lesz az is, hogy az mindig igaz is marad. (Ennek a múltra vonatkozó tüköraxiómája is igaz). Ez a következő két Prolog szabállyal valósítható meg:

  ```
  p(afterA(afterA(T)),X1,…,Xn):- p(afterA(T),X1,…,Xn).
  p(beforeA(beforeA(T)),X1,…,Xn):- p(beforeA(T),X1,…,Xn).
  ```

- A TL2 trichotómia axiómája (**tri**) a következmény-oldali diszjunkció miatt Prologban nem valósítható meg kézenfekvő módon.
- A TL5 sűrűségaxiómája (**den**) szerint, ha valami a jövőben valamikor igaz lesz, akkor mindig lesz olyan jövőbeli pillanat, amihez képest szintén a jövőben lesz az eredeti dolog igaz. Ennek a múltra vonatkozó tükörképe is igaz. Ez a következőképpen valósítható meg Prologban:

```
p(after(after(T)),X1,…,Xn):- p(after(T),X1,…,Xn).
p(before(before(T)),X1,…,Xn):- p(before(T),X1,…,Xn).
```

4.2.2.3 Ontológiák bekapcsolása és a megvalósítás

A javasolt rendszerben mindenképpen cél a már meglevő ontológiák újrafelhasználása, méghozzá olyan modell megalkotása kívánatos, amelybe tetszés szerint bekapcsolható bármelyik elérhető ontológia (csúcsontológia vagy szakontológiák).

A Prolog környezetben a végrehajtás miatt olyan eszközt kerestünk, amely valamelyik szabványnak tekinthető ontológia-leíró nyelvet Prolog kóddá alakít. Így esett a választás az SWI-Prolog nyílt forráskódú környezetre és a környezetben Vangelis Vassiliadis és Chris Mungall által létrehozott Thea ontológiakezelő szoftverre. Ez – több más működésmód mellett – OWL ontológiák beolvasását, kezelését és különböző célformátumokban, például magában Prologban történő mentését végzi, valamint következtetőgépekhez történő kapcsolódást biztosít. A Thea szoftver egyik előnyös tulajdonsága, hogy a betöltési és mentési formátumok dugaszfelépítmény-szerűen kapcsolódnak a szoftver-maghoz, így a mellékelt formátumok könnyen testre szabhatók, módosíthatók.

A ℜeALIS tudáskezelő szoftvernek alapvetően háromféle *programozási felülete* van:

- Ontológiákat be is tölthetünk az ontológiai felületen keresztül, de ezeket kiterjesztve létrehozhatjuk a magunk szakontológiáját is. Az ontológiáknak leíró logikai alapúaknak kell lenniük, lehet OWL nyelvű, tartalmazhat SWRL szabályokat, de közvetlen a Prolog tárgymodell szintjén is betölthetjük őket. OWL ontológiák szerkesztésre többféle szoftver is kínálkozhat:
  - A Protégé ontológiaszerkesztő program. Ez szabad felhasználású program, több változata is közkézen forog, sőt, internetes változat is elérhető.
  - Az SWI-Prologhoz írt Triple nevű szerkesztő program. Ez az OWL alatt húzódó RDF réteg szerkesztésére alkalmas, de mivel az OWL is erre épül, annak kezelésére is képes.
  - A Top Quadrant Inc. TopBraidComposer szoftverével. Valószínűleg ez lehet a legstabilabb és kiforrottabb szerkesztőprogram, sajnos azonban fizetős.
- A betöltött ontológiák felett a ℜeALM nyelven esetleírásokat fogalmazhatunk meg, és azokon következtetési műveleteket végezhetünk. Ez a felület gyakorlatilag az elemzett természetes nyelvű mondatok és diskurzusok eredményét ábrázolja, és ezen a tudáson végez műveleteket.
- A rendszernek Prolog nyelvű programozható felülete (ℜePI) is létezik.

Nézzünk mindezekre egy példát! A példában egy ontológiából csupán néhány elemet használunk ki:

- A `motion⊆process` és a `walking⊆motion` OWL fogalommeghatározásoknak (T-box) a Grosof és Ohlbach-féle átírásban a következő Prolog állítások felelnek meg:

```
process(MOD,PR):-
        motion(MOD,PR).

motion(MOD,PR):-
        walking(MOD,PR).
```

- A tényleírás az alábbi, OWL A-Box állításokból épül fel. Ez az „egye fene, elmehetsz” mondat korábban grafikusan is ábrázolt szerkezetének Prolog tényállításokból álló alakja:

```
walking(des(max,s,present,+,root(s)),w1).
agent(w1,s).
from(w1,here).
walking(des(med,s,past,-,root(i)),w2).
agent(w2,s).
from(w2,here).
walking(int(min,s,present,+,root(i)),w3).
agent(w2,s).
from(w2,here).
```

- A fentiek alapján készült Prolog program tesztelésének eredményeképpen a következő érdekesebb fajta lekérdezéseket lehetséges megválaszolni:

```
motion(des(max,a1,TIME,+,root(a1)),W).
```

Szándékában áll-e ’a1’ szereplőnek elmozdulni, és mikor?

```
process(des(max,a1,TIME,+,root(a1)),W),
  earlier(TIME,present).
```

Mit akart ’a1’ szereplő a múltban? (Erősen vágyott-e ’a1’ szereplő a múltban bármilyen folyamatban részt venni, és mikor?)

```
process(des(_,WHO1,TIME,-,root(WHO)),W1),
  earlier(TIME,present),
  process(int(_,WHO1,present,+,root(WHO)),W2).
```

Ki az a szereplő, aki a korábbi negatív vágyát megváltoztatta (valamiféle pozitív szándékra)?

4.2.3 Az elemzőprogram megvalósítási kérdései

4.2.3.1 A ℜeALLAN nyelvleíró nyelv

A teljes lexikalizmus jegyében minden nyelvtani információ a lexikon bejegyzéseihez kötődik: elsősorban a keres-kínál működésmód elemei, másodsorban a kötéserősség meghatározása alapján. Ez azt jelenti, hogy az egyes lexikai elemek – ha a mondatban/szövegben megjelennek, akkor valamilyen elvont információszerkezetet (akár benne ismeretlen elemekkel is) felkínálnak, és másokat pedig, akár többet is igényelnek. Az igénylés minden esetben kötéserősség rögzítését is jelenti. A kínál-

követel működés alapján egyes elemek összekapcsolódhatnak, sőt, esetleg több ilyen összekapcsolódás is lehetséges. A többszörös, nemdeterminisztikus eredmények közötti további szűrésre alkalmas a kötéserősség: az egyes elemek a tartozékaikat adott erősséggel keresik, a gyengébben kötő elemeknek távolabb kell a mondatban elhelyezkedniük. Az összekapcsolás a nyelvészetben ismert kétirányú régens-vonzat viszony, az egyirányú adjunktum-alaptag viszony, valamint az anafora-előzmény viszony felderítését jelenti. (A régens-vonzat viszony olyan mondatrészek, pl. ige és vonzatai között lép fel, amelyek kölcsönösen vonzzák egymást. Az adjunktum-alaptag viszony esetében az adjunktum választható vonzatot jelent, amely keresi az alaptagját, de ez fordítva nincs így. Az anafora-előzmény viszony viszont a névmások és határozott névelők esetében keresnek afféle tagokat, amelyek a használatukat bevezetik.)

A megfelelő lexikai elemek leírása mindenképpen nyelvészi feladat: ehhez azonban a számítástechnikusnak egy megfelelő eszközt kell adnia, egy számítógéppel is felismerhető, feldolgozható leíró nyelvet kell létrehoznia.

Ez lenne a ℜeALLan nyelvészeti leíró nyelv. Ennek tervezése Alberti Gábor[135] és a szerző más írásaiban olvasott példák alapján készült – azokat csupán kismértékben módosítva, olyan irányban, hogy a választott programozási környezettel (Prolog) könnyen feldolgozható legyen. A ℜeALLan jellemzői a következőkben foglalhatók össze:

- A Prolog környezetben könnyű elemezhetőség végett a nyelv teljes egészében követi a Prolog kifejezések alapnyelvtanát.
- Az alapnyelvtanhoz képesti megszorítások a korábban már említett ProType típusleíró résznyelvvel lettek rögzítve
- A ℜeALLan alapvetően jegyszerkezetes leíró nyelv, de egyes elemeiben lehetőséget kell adni tömörített, Prolog függvényszerkezetű, sőt, közbeékelődő operátoros ábrázolásra is.
- A nyelv gyökértípusa a „realclause”. Ez írja le egy lexikonbejegyzés formátumát, vagyis a ℜeALLan nyelv minden egyes mondatának ezen megkötéseknek kell megfelelnie.

A nyelv pontos és részletes megadása helyett nézzünk egy példát:

```
'hasonlít'--->
[sigma:(EDES=
      pred3(desire(TIME,RDES1,RDES2),TIME,RDES1,RDES2)),
+alpha:(RDES1=[cat(0,verb(nom,sub)),agr(+2,sg-3)]),
-alpha:(RDES1:=[argn(ord(-7,nei),cat(+2,noun),
                case(+2,nom)),argd(cat(+7,gqd))]),
-alpha:(RDES2:=[argn(ord(+7,nei),cat(+2,noun),
                case(+2,sub)),argd(cat(+2,gqd))])
]
```

A szógyököt és a lexikonbejegyzéstől a `--->/2` operátor választja el. A lexikonbejegyzés jegyszerkezetében a `sigma` eventuális referens, valamint az `+alpha` és `-alpha` kereslet-kínálatleíró jegyek olvashatók. A keresletek mindegyike egy Prolog változót köt le a megtalált elemek eventuális referensével (`RDES1`, illetve

[135] Alberti Gábor: ℜeALIS. Interpretálók a világban, világok az interpretálóban [AG11]

RDES2). Mind a kínálatot, mind a keresleteket szintén jegyszerkezettel adjuk meg. A kínálat: a 'hasonlít' ige egy alanyi és egy szublativuszi (-ra, -re ragos) esetű ige, amely egyes szám harmadik személyben áll (agr(…,sg-3)). Az első kereslet: a 'hasonlít' ige első argumentumként (alanyi mondatrész szerepében) egy őt szósorrendben (ord) megelőző, sőt lehetőleg szomszédos (nei), alanyesetű (case(…,nom)) főnevet (cat(…,noun)) keres. Az argd jegy megadása ezen túl még az úgynevezett *általános kvantordeterminánsra* (gqd) jelenlétét is megköveteli: ez névelőt vagy egyéb determinánsi pillért ír elő az ige alanya számára. A második vonzatnak (az ige tárgyának) mint főnévnek (cat(…,noun)) az igét lehetőleg szomszédosan kell követnie (argn(ord(+7,nei)), szublativuszi raggal (case(…,sub)), és szintén szükséges az általános kvantordeterminánsra megléte.

4.2.3.2 A ℜeALIS tárgymodellje

Prolog programokat általában az úgynevezett *relációs tárgymodellben* szokás megírni. Ez azt jelenti, hogy a programot végső soron egyetlen Prolog relációként felfogva az a bemenő és kimenő adatok közötti relációt számítja ki. A relációs szemlélet efféle alkalmazásának kicsit ellentmond a beépített eljárások használatából fakadó mód-megszorítások terjedése, de némi odafigyeléssel és ügyeskedéssel mégiscsak lehetséges a relációs modell alapján oda- és vissza is működő Prolog programot írni.

A relációs tárgymodell alkalmazásakor tehát egy program bemenet/kimenet relációját egy adott Prolog szabály számítja ki. Ha egy reláció több részrelációból van összetéve, akkor azokat a szabály feltételrészében nevezzük meg úgy, hogy a be- és kimenő paraméterek egymáshoz láncszerűen kapcsolódnak. Az ilyen, be- és kimenő szerepű változópárokat a Prolog szakirodalom *akkumulátorpárnak* nevezi.

```
reláció(BE,KI):-
  rész1(BE,TMP1),rész2(TMP1,TMP2),…,részN(TMPN-1,KI).
```

A Definite Clause Grammar (DCG) formalizmus relációs tárgymodell szerinti nyelvtani elemzésekor a <bemenet, még elemzetlen szöveg> párt használjuk akkumulátorként, a tetszőleges argumentumszerkezethez az akkumulátorpárt pedig a DCG előfordító maga hozza létre az alábbiakhoz hasonlóan.

```
nonterm(…)→nonterm1(…),nonterm2(…),…,nontermN(…).

nonterm(…,BE,KI):-
  nonterm1(…,BE,TMP1),nonterm2(…,TMP1,TMP2),…,
  nontermN(…,TMPN-1,KI).
```

A megoldás egyik hátránya: a *relációk nemdeterminisztikus kiértékelése* miatt az eredményreláció számossága legrosszabb esetben az egyes részrelációk számosságának a szorzata is lehet. Ha viszont a szorzatban az első részreláció számossága legnagyobb, akkor a nemdeterminizmus visszalépéses kezelése miatt egészen az első relációig tartó, *mély visszalépés* történik.

A ℜeALIS relációs tárgymodell szerinti megvalósításában a bemenő paraméter az elemzendő szöveg, a kimenő pedig a szövegnek megfelelő logikai kifejezés-szerkezet. Értelmes rész-relációk lehetnek: szóalaktani, nyelvtani-szemantikai elemzés vagy pragmatika. Ilyen értelmezés mellett ugyanazt a szabályt használhatjuk elemzésre, (ha híváskor TEXT adott, LOGEXPR viszont változó), illetve szöveggenerálásra is (ha híváskor TEXT változó, de LOGEXPR adott).

```
text2logic(TEXT,LOGEXPR):-
  morphology(TEXT,MORPHLIST),
  syntaxSemantics(MORPHLIST,PUREEXPR),
  pragmatics(PUREEXPR,LOGEXPR).
```

Sajnos a relációs tárgymodell és az ezzel összefüggő Prolog DCG formalizmus a mi céljainkra nemigen alkalmas. A ℜeALIS környezeti feltételei (pl. vonzatok bizonyos távolságban) csak úgy lennének elemezhetők, ha azokat a bemenő szövegben előre-hátra mozgással ellenőriznénk. Ennek a megvalósítása is körülményes, és komoly hatékonysági aggályokat is felvet.

A ℜeALIS megvalósítás célkitűzése a szöveg és a diskurzusreprezentáció közötti reláció kiszámítása. Ez (Prolog-szerű értelmezésben) mindkét irányú kapcsolatot jelenti. Ha a szöveg adott, akkor a program azt a reprezentációs kifejezést számítja ki, amely az adott logikai rendszerben és az interpretáló belső tudatállapotát leíró tudásbázisban (ontológiában) kiértékelhető, bizonyítható, vagy hozzávehető a tudásbázishoz. Az ellenkező irányban: ha a tudáskezelő összetevő által (pl. egy kérdésre adott válaszként) egy logikai kifejezést kapunk, akkor a reláció a szöveg képét állítja elő.

A megoldás másik hátránya, hogy a szöveg legalább egy bekezdésnyi, de esetleg akár több oldalnyi hosszú is lehet. Ez egyrészt a feldolgozás időigényét behatárolja, másrészt a hosszú bemenő adatokon az igen mély visszalépések csökkenthetik az elemzés hatékonyságát. Harmadrészt a szélsőségesen összetett adatszerkezetek sok esetben a Prolog megvalósítás fizikai határait is feszegethetik (pl. verem túlcsordulást okozhatnak).

A relációs tárgymodellel szemben a ℜeALIS esetében a *következtetéses tárgymodell* alkalmazását javasoljuk. Ilyen esetben a bemenő szöveget nem listaparaméterként, hanem *tényállításokként* ábrázoljuk. Például: „A vádlott hasonlít arra az ismert vörös ukrán szeszcsempészre” mondat az alábbi tényállítás-sorozattal ábrázolható. (Feltételezzük, hogy a szóalaktani elemzés már megtörtént, és már csak a nyelvtani-szemantikai elemzés van hátra.)

```
word(0,art(def,cons)).
word(1,noun('vádlott',proper,nom,sing-3)).
word(2,verb('hasonlít',[], decl, pres, sing-3)).
word(3,noun('az',pro(point),sub,sing-3)).
word(4,art(def,cons)).
word(5,adj('vörös')).
word(6,adj('ukrán')).
word(7,adj('szesz')).
word(7,noun('csempész',common,sub,sing-3)).
```

A ℜeALLan szabályok követel-kínál mechanizmusa szinte kínálja magát arra, hogy Horn-klózokká képezzük le őket, hiszen emlékezzünk vissza: egyetlen kínálati eleme, és tetszőleges követelési eleme lehet. Az alábbi klóz pl. a ’hasonlít’ ige és kötelező vonzatai közötti kapcsolatot írja le.

```
regArg2(ID,S,XV,verb('hasonlít',[],MODE,VTIME,AGR),
              XS,noun(SUBJ,SKIND,nom,AGR),-7,
              XO,noun(OBJ,OKIND,sub,OAGR),7):-
  verb(ID,S,XV,'hasonlít',[],MODE,VTIME,AGR),
  gqdet(ID,S,XS,SUBJ,SKIND,nom,AGR), order(XV,XS,-7,nei),
  gqdet(ID,S,XO,OBJ,OKIND,sub,OAGR), order(XV,XO,7,nei).
```

Szintén Horn-klózok írják le a ℜeALIS σ (sigma) függvényének megfelelő eventuális kifejezések részkifejezésekből történő felépítését is.

```
sigma3(ID,S,XV,TIME,SUB,OB,CLAUSE):-
  regArg2(ID,S,XV,verb('hasonlít',[],_MODE,VTIME,_AGR),
       XS,SUBJ,_PRS,XO,OBJ,_PRO), {TIME =.. [VTIME,_]},
  sigma3(ID,S,XS,TIME,SUB,CLAUSE,
          (similar(TIME,SUB,OB):-CONS)),
  sigma3(ID,S,XO,TIME,OB,CONS).
```

A fenti állítás eredményképpen a mondat logikai alakjaként a következőket kapjuk. (A kettős implikáció egy egyszerű normáló program segítségével átalakítható feltételek konjunciójává.)

```
CLAUSE=((similar(pres(T),SUB,OB):-
           ukrain(T, OB), red(T, OB),
           moonshiner(T, OB)):- defendant(T,SUB))
```

A fenti szabályok futtatásának a legkézenfekvőbb módja az, ha a futtatásra a *visszafelé haladó* stratégia alapján magát a Prolog értelmezőt használjuk úgy, hogy az általános következtetéses tárgymodellt vesszük alapul. Ebben a megközelítésben az elemzést a logikai alakra változóként hivatkozó *célállítás* hívásával indítjuk. Ha visszavezethető a célállítás a szöveget rögzítő tényállításokra, akkor a mondat elemezhető volt, és a közben elvégzett változóhelyettesítésekből kiadódik a célállításban szereplő logikai alak is.

A megközelítés egyik hátránya, hogy a bizonyításhoz *hipotézist* kell felállítani, ez gyakorlatilag a célállítás. A bizonyítás időpontjában már minden ténynek ismertnek kell lennie – a rendszer nem alkalmas csővezeték- (pipe) -szerű feldolgozásra.

Egy további hátrány az, hogy a visszafelé bizonyítás logikája szerint a Prolog az ismétlődő rész-bizonyításokat újra és újra többszörösen is elvégzi, ezzel komolyan rontva a hatékonyságát.

A fentebb vázolt tárgymodell alapvetően *deduktívan, felismerőként* használható, mégis kicsi módosítással *abduktív, szöveggenerátor* célú használatra is alkalmas. Ha a célállítást a logikai alak megadásával, de hiányzó szövegkép-tényállításokkal indítjuk, akkor a visszafelé bizonyítás során előbb-utóbb a tényállítások szintjéig ér. Ha az ismeretlen tényállításokat *visszaléptethető állításfelvétellel* (`assert`) valósítjuk meg, akkor a program végeredményben abduktív bizonyítást fog végezni.

```
word(ID,S,X,WORD):-
  (assert(w(ID,S,X,WORD));
  retract(w(ID,S,X,WORD)), fail).
```

A hátrányok kiküszöbölésére a korábban már bemutatott Contralog csomagot használjuk, amely a Prolog előrehaladó kiterjesztésének nevezhető. A csomag rugalmasan integrálható tetszés szerinti Prolog programba, a működése jól ellenőrizhető és korlátok között tartható. A Contralog alkalmazásának hátránya viszont, hogy az imént említett, szöveg-előállításra (generálásra) szolgáló deduktív következtetési modell nem vihető át rá mélyebb átalakítás nélkül. Ez azt is jelenti, hogy a szöveg-előállító programcsomagot mindenképpen a felismerőtől külön kell megvalósítani. Ezt a hátrányt enyhítheti az, ha a kevésbé igényes szövegelőállító-megvalósítással is megelégednénk.

4.2.3.3 A polaritásos hatáslánc modellje.

A nyelvtani elemzés eredményeképpen előáll a mondat vagy egyes részeinek szemantikáját tükröző szerkezeti alak, a mondat logikai képe. Nem szabad megjegyzés nélkül hagyni azonban azt a tényt, hogy közös igék köré az értelmezési változatoknak egy népes családja épül fel. Ezeket korábban úgynevezett *vonzatkeretlistákban* foglalták össze, amelyek az egyes ige és vonzattípusok modelljéül szolgáltak, de önmagukban semmi támpontot nem adtak a szemantikus értelmezésre. Alberti cikkében az igék leírásával kapcsolatban absztrakt tematikus szerepekre hivatkozik[136] (Cselekvő (Agent), Szenvedő (Patient), Ok, Eszköz, Tér, stb.). Megjegyezzük, hogy az alább részletezett információs szerep (a topik-fókusz szerepek) fogalmának gyakorlati jelentősége az, hogy a szöveggenerálás esetében még a hangsúlyozásra és az intonációra is útmutatást ad. A rendszer 6 paramétert nevez meg, amelyek segítségével mintegy táblázatszerűen leírható az igék pontos szemantikája. A 6 paraméter a következő:

- *Hatóköri sorrend*: ez egy természetes szám, amely leírja, hogy az adott, akár több szóból, jelzős szerkezetből felépített kifejezés a mondatban sorrendben hányadikként szerepel.
- Hatásláncbeli *tematikus szerep*, a fenti szerepek egyike (*Cselekvő, Szenvedő, Ok, Eszköz, Tér*), a felsorolás igény szerint bővíthető.
- Az *Esetprominencia* háromféle értéket vehet fel: a *centrális* megjelölés alatt értjük az alanyi vagy tárgyi szerepkört. *Nem centrálisnak* nevezzük a megkövetelt vonzatot, míg *szabad határozónak* az egyéb bővítményeket nevezzük.
- *Információs szerep*: a topik-fókusz elméletnek megfelelő szerepjelzés (T/topik, K/kontrasztív topik, F/fókusz, Q/kvantor, M/igemódosító, C/komplementum)
- A *Hivatkozási fokozat* (referencialitási fokozat) az adott bővítmény nyelvi határozottságára utaló jelzés, amely a következők egyike lehet: *határozott* (+hat), *határozatlan* (-hat), *hivatkozással rendelkező* (+ref), vagy *-nem rendelkező* (-ref), illetve ∅, amely alapértelmezés szerinti hivatkozási fokozatot jelent. A jelzések közül (az egymást kizáróktól eltekintve) egyszerre több is szerepelhet, és a lista is bővíthető.
- *Beszédaktusbeli részvétel*: az adott elem szám és személy szerinti egyeztetése.

A fenti paraméterek nyelvészeti megalapozottságának tárgyalása a jelen értekezés keretein mindenképpen túlmutat. Alberti Gábor polarizált hatáslánc-modell-elméletének ellenőrzésére készített Prolog tanulmányprogram igazolta az elvárásokat. A program a várt elemzési eredményeket és logikai modelleket állította elő, esetenként azonban feleslegesnek tűnő nemdeterminisztikus eredmények léptek fel. A mélyebb elemzés azonban megmutatta, hogy az elburjánzás megállítható egy egyszerű tartalmi/ontológiai ellenőrzéssel. (Pl. „A vádlott kiszúratta az abroncsot *egy szöggel/egy barátjával*” mondatpár esetében a szög eszköz, illetve a barát tevékeny (ágens) jellege a puszta nyelvtani szerkezetből még nem eldönthető. A két eset elkülönítéséhez az egyes szavak tartalmi összefüggéseit is meg kell vizsgálni.)

[136] Alberti, Gábor-Kilián Imre (2010): Vonzatkeretlisták helyett polaritásos hatáslánc-családok – avagy a ℜeALIS σ függvénye [AGKI10]

4.2.3.4 ℜeALM: a ℜeALIS elemzés eredménye

A ℜeALIS elemzés egyik eredménye a mondat/bekezdés/párbeszéd logikai alakja, amely a σ eventuális függvény kiértékelésének az eredménye. Az eredmény egy modális logikai nyelven megfogalmazott rész-elmélet, amely a háttér-ontológia környezetében érthető. A logikai nyelv az egész párbeszéd – esetleg az egész dokumentum vagy irodalmi mű ábrázolására alkalmas, amelyet az egyes szövegértelmezők a saját elmeállapotuk háttér-ontológiája szerint értelmeznek. Előfordulhat az is, hogy egyes művek egyes értelmezők által értelmezhetetlenek maradnak – abban az esetben, ha a mű nyelvezetében az értelmezők által ismeretlen relációkat, osztályokat, egyedeket neveznek meg.

Az elemzés végeredményéül kapott nyelvet nevezzük ℜeALM nyelvnek (ℜeALIS Logical Modal Language)

Bár a ℜeALM logikai nyelv, mégis úgy tervezzük meg, hogy lehetőleg a nyelvhasználatba rejtett szemantikus finomságokat a lehető legtökéletesebben ki lehessen vele fejezni, vagyis az átalakítással ne veszítsünk információt. Azt várjuk tőle, hogy a nyelv a későbbiekben – legalábbis szakszövegek esetén – dolgozatok, dokumentumok, publikációk, törvények, jegyzőkönyvek végső ábrázolási és tárolási formája legyen. Ennek egyik következménye az is, hogy a nyelv szöveges formára visszaalakítható legyen, és lehetőleg(!) kapjuk vissza az eredeti szöveges formát, de legalábbis egy, az eredetivel egyenértékű formát. Vagyis a ℜeALM nyelvnek a logikai kifejezések mellett tartalmaznia kell a ℜeALIS logikai modell fejezetében leírt modális logikai operátorokat, és retorikai relációkat is.

A gyakorlati feldolgozhatóság érdekében a ℜeALM-nek is Prolog résznyelvnek kell lennie, vagyis a nyelv rögzítését szintén a ProType típusleíró nyelven végezzük.

Az alábbiakban a ℜeALM nyelv pontos meghatározása helyett néhány tervezési alapelvet foglalunk össze, és pár példát mutatunk be a nyelv használatára. Rámutatunk azokra a pontokra is, amelyek egy későbbi bővítés vagy módosítás tárgyai lehetnek.

- A nyelv modális logikai nyelv. A nyelv egy állítása tehát egy modális és egy klasszikus logikai leíró részből áll az alábbi meghatározás szerint:

  ```
  type(realmClause=modalOps:folClause).
  ```

  A kifejezést a ':' közbeékelődő operátor meghatározásától (és prioritásától) függően esetleg zárójelbe kell tenni.

- A modularitást leíró `modularities` kifejezés egyetlen modális operátor lehet, vagy ilyenek láncolata Prolog listába fűzve

  ```
  type(modalOps=slistof(modalOp)).
  ```

- A ℜeALM modális operátorai lényegileg tükrözik a ℜeALIS elemzési eredményeit, és a világszerkezetre egyértelműen leképezhetők. Az episztemikus operátorok az első paraméterükben minden esetben azokat a cselekvőket jelölik, akikre vonatkoznak, deontikus operátorok esetében pedig azt a szereplőcsoportot, akikre az általuk leírt normatíva vonatkozik.

  ```
  type(modalOp=(temporalOp, epistemicOp;
                epiTempOp; deonticOp)).
  ```

```
type(temporalOp= before; after; beforeA, afterA,
     before(expr); after(expr);
     beforeA(expr), afterA(expr)).

type(deonticOp=(perm(expr); obl(expr); forb(expr))).

type(epistemicOp=(bel(expr,mgrad,polarity);
     des(expr,mgrad,polarity);
     int(expr,mgrad,polarity)).

type(epiTempOp=(bel(expr,mgrad,polarity,temporalOp);
     des(expr,mgrad,polarity);
     int(expr,mgrad,polarity)).

type(mgrad=(min;med;max)).

type(polarity=(-;0;+)).
```

- A modularitásjelzők hatáskörében szereplő `folClause` típusjelző az alaptípusokkal kibővített elsőrendű logika nyelvén leírt kifejezéseket jelez. Ezt részletesen nem elemezzük tovább.
- A fentebb megadott `epiTempOp` csupán egy nyelvtani könnyítés a temporális és az episztemikus operátorok egymás utáni alkalmazása esetén. A `temporalOp` definíció rögzíti a két gyenge időmodális operátort (`after`, `before` a múltban/a jövőben valamikor), valamint a két erős időmodális operátort is (`afterA`, `beforeA` a múltban/a jövőben mindig). Az operátorok paraméterében egy vonatkoztatási időcímkét is megadhatunk. A gyakorlatban a fentieknél lényegesen gazdagabb időcímkéket használunk, ez tehát egy későbbi bővítési pont.

A fentiek érzékeltetésére tekintsünk egy bonyolultabb, jogi példát. Tegyük fel, hogy a Bíró udvarias érdeklődésére Vádlott az alibijét akként adja elő, miszerint a bűntény időpontjában Fradi-Újpest meccsen volt, a helyszíntől 100 km távolságban. Az elemzés eredményeképpen előáll a Bíró világocska-rendszerébe belehelyezett, a Vádlottal címkézett részvilágocskában egy logikai állítás. Az egész a ℜeALM Prolog alapú ábrázolásában a következőképpen nézhet ki:

```
[bel(bíró,max,+),tell(vádlott)]:
     (bűntény(X,LOC,TIME), tartózkodik(vádlott, M),
     futballmeccs(M, fradi, újpest,L,T),
     TIME⊆T, táv(LOC,L,100km)).
```

A fenti a Bíró elemzésének eredménye volt, ezért a Bíró által ismert világba tartozónak jeleztük a fenti példában. Ennek ellenére Bíró feltehetőleg objektívnak tekinti az elmondást magát, és saját magát kihagyva a dologból a társadalmi környezetbe, de legalábbis a bírói esetek környezetébe helyezi az egészet. (Erre utal, hogy jegyzőkönyvbe veszik a vallomást, amely a bírósági esetek tárában ezután a bírótól függetlenül megtalálható és elolvasható.) Ezek után az ítélethozatal érdekében feltehetőleg a társadalmi környezet közhelytudásai és más tanúk vallomásai alapján ellenőrzi az állítás helyességét. Pl. a sporteseményekről adatbázisából megtudhatja, hogy L helyszínen a T időpontban volt-e Fradi Újpest meccs egyáltalán, és ha mégsem, akkor megdőlt az alibi.

4.3 Jogi ontológia építése

4.3.1 Jogi következtető rendszer felépítménye

A fent ábrázolt felépítmény egyik alapvető feltételezése a Prolog környezet választása – természetesen más megvalósítási nyelv is működhet. Úgy véljük azonban, hogy a Prolog használata a nyelv magasabb kifejező ereje miatt egy sor kényelmetlenségtől megvéd: a nyelv elsajátítása után a hagyományos nyelveknél lényegesen könnyebben és gyorsabban lehet benne nagybonyolultságú és elvont programokat létrehozni.

A következtető motor felületeinek rögzítésénél célszerű figyelembe venni a Description Logic Implementation Group (DIG) nyereségmentes szervezet szabványjavaslatait,[137] amelyek mind az ontológia elérésre (OWL API), mind a következtetésre vonatkoznak.[138] Bár több következtetőgép megvalósítás is létezik, mégis célszerű az Apache Jena projektet figyelembe venni,[139] amely egy nyílt forráskódú szemantikus-világhálós infrastruktúrát ajánl, a felületeiben pedig DIG csereszabatosságot ígér. A következőkben felsorolt szabadon elérhető vagy fizetős következtetőgépek részletes elemzését és összehasonlítását Förhécz András műegyetemi szakdolgozata mutatja be:[140]

- FacT, FacT++: Ian Horrocks PhD dolgozataként elkészült leíró logikai következtetőgép[141]
- A RACER Volker Haarslev és Ralf Möller következtetőgépe, amely egyetemi környezetből indult ki, és szabad szoftver volt. A továbbfejlesztés eredményeképpen elkészült a RacerPro szoftver a Racer Systems GmbH terméke,[142] és fizetős szoftver, de internetes szolgáltatásként is elérhető.
- A Pellet következtetőgép a Clark&Parsia LLC terméke,[143] amely mégis szabad szoftver, többek között a Protégé szabad szerkesztő környezetbe is ez a csomag van beépítve.
- A Hermit szintén egy szabad következtetőgép, amelyet az Oxford Egyetemen, Angliában fejlesztett ki egy lelkes csapat.[144]

A javasolt felépítmény főbb elemei a következők:

- Ontológia Kiszolgáló – a hivatkozott szabványos vagy egyéb nyílt forráskódú ontológiák kiszolgálója, esetleg egy egyszerű fájlkiszolgáló vagy Interneten működő FTP kiszolgáló.
- Példánytár: Az OWL ontológiák példányinformációi közül a ritkábban használtakat relációs adatbázisban helyezzük el, és csak konkrét igény esetén

---

[137] http://owl.man.ac.uk Elérés: 21Aug-2012.

[138] Matthew Horridge, Sean Bechhofer. The OWL API: A Java API for OWL Ontologies [HoBe11]

[139] http://jena.apache.org Elérés: 23-Aug-2012.

[140] Förhécz András: Ontológia kezelő modul tervezése szöveges információt kezelő informatikai rendszer számára [Fö04]

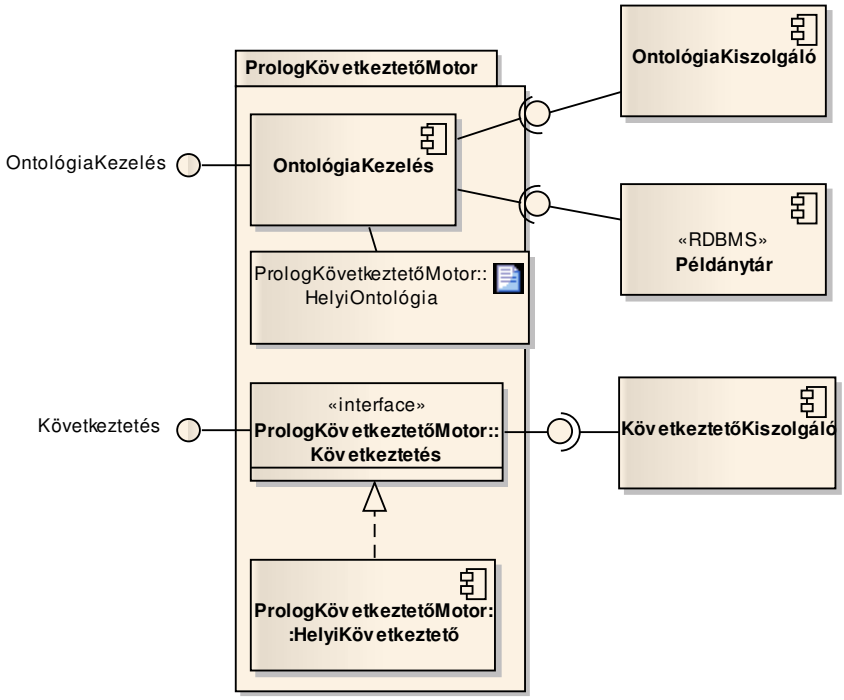
[141] http://www.cs.man.ac.uk/~horrocks/FaCT Elérés: 21-Aug-2012.

[142] www.racer-systems.com Elérés: 21-Aug-2012.

[143] www.clarkparsia.com Elérés: 23-Aug-2012.

[144] http://hermit-reasoner.com Elérés: 24-Aug-2012.

töltjük be a belső tárba. Az adatbázis-kezelő célszerűen külön kiszolgáló gépen futhat.



56. ábra A Prolog Következtető Motor felépítménye

- Következtető Kiszolgáló. A következtetésnek képesnek kell lennie szabványos DL következtetések végrehajtására, de ugyanígy képesnek kell lennie SWRL szabályok futtatására is. Minderre egy lehetőség az, hogy szabványos elérési felülettel rendelkező bizonyító-kiszolgálókat használjunk – nyilván ezek (többé-kevésbé) zökkenőmentesen működnek. Hátrány, hogy a távoli következtetéshez előbb a tudásbázist át kell küldeni, ami komoly alkalmazásoknál gondot jelenthet. Ugyancsak hátrány, hogy nyúlfarknyi, technikai következtetési műveletek sorozatát, pláne ugyanazon tudásbázis felett a sok hálózati tranzakció miatt nemigen lehet hatékony végrehajtani. Megjegyezzük, hogy léteznek nyílt forráskódú következtető csomagok is, amit hozzászerkeszthetünk a helyi programhoz, és amely ezért esetleg „csak” paradigmaváltást követel meg (pl. Prologból át kell lépni Java környezetbe).
- Helyi Következtető: amely ugyanazokra képes, mint a Következtető Kiszolgáló. A megoldás egyik előnye az, hogy a helyi programmodul hálózati forgalom nélkül, közvetlenül férhet hozzá a tudástár elemeihez, így a tömeges, egyszerű következtetési feladatokat feltehetően hatékonyabban oldja meg. Másik előnye az, hogy a szabályalkalmazás során felmerülő esetleges bővítési igények is beépíthetők, míg ugyanez egy polcról levehető eszköz esetében legalábbis nyűgösebb, lassabb feladat, de nem is biztos, hogy lehetséges. Hátránya természetesen az erőforrás igényessége – egy hatékony következtető csomag elkészítése nem magától értetődő feladat.
- Következtetés felület, amely maga alá beilleszti mind a távoli, mind a helyi következtető modult. A következtetésre szabványos felületek is léteznek, a

Prolog alapú felületet is célszerű erre való tekintettel megtervezni. Jelenleg a következő felület-meghatározásokat érdemes számba venni:

- Ontológia Kezelés: Erre a célra az SWI-Prolog szabad felhasználású Prolog rendszert és az alatta futó Thea csomagot javasoljuk. Ez az OWL ontológiákat RDF formában betölti a belső tárba. A Thea egyfelől megvalósítja az ontológiakezelési felületet, másrészt gyenge képességű Prolog alapú következtetőket is ad, harmadrészt pedig szabványos felülettel rendelkező külső következtetők bekapcsolását is lehetővé teszi.
- Helyi Ontológia: A Thea a használat előtt betölti a használni kívánt ontológiát (az esetleges importokkal együtt) a belső tárba. Ehhez létezik RDF szintű hozzáférő művelet is, de a szabványos OWL szintű felületet is biztosítja a rendszer.

4.3.2 A deontikus modalitás korlátai

Jogi ontológia építésének legalapvetőbb problémája a deontikus logika és a jogi felfogás ütközése.

Az első kétely a Kripke-féle értelmezéssel függ össze. Eszerint az aktuális világunkat összeköthetjük egy alternatív deontikus világhalmazzal, amelyek intuitíven jobbak az aktuálisnál. $\boldsymbol{O}\ \alpha$ (kötelező) jelentése: az α állítás ezekben az alternatív világokban teljesül, függetlenül az aktuális világbéli teljesüléstől.

Egy gyakorlatban használható rendszer számára igen nehéz az említett világszerkezet értelmezése és azonosítása. Hol vannak a lehetséges világok, mihez kötődnek, és egyáltalán: mit fejeznek ki? Melyek azok a környezetek, amelyekben az egyes állításelemek különböző módon értékelődhetnek ki? Episztemikus modalitás esetében erre még könnyebben lehetséges válaszolni: az egyes világok az egyes szereplők által tudott/hitt/feltételezett dolgok,[145] de a jogi normatívák értelmezésénél az „ideális világok" fogalma kicsit nehezen megfoghatónak és körülhatárolhatónak tűnik.

Mindazonáltal egy $\boldsymbol{O}\ \alpha$ állítás semmit nem mond arról, hogy ki a felelős α teljesüléséért vagy nem teljesüléséért, de arról sem, hogy hogyan, milyen lépések során keresztül lehetne α-t teljesíteni, vagy egyáltalán lehetne-e, és α teljesítése nem követel-e lehetetlen lépéseket. Nem állít az operátor arról sem semmit, hogy kicsit, nagyon vagy mennyire kötelező α teljesítése. Csupán egyet állít: az α-val jelölt állítások bekövetkezését valamiért másokénál *előbbre helyezzük, valamilyen prioritással látjuk el* – azért, mert az a jogszerű, és a jog nem más, mint a társadalmilag kialakult elvárások rögzített része.

A deontikus normafogalom nem feltétlenül logikai értékkel rendelkező állításokra vonatkozik. Ez azt jelenti, hogy az említett fogalom nem egy idealizált cselekvéssort, hanem egy idealizált állapotot ír le, amely nem írja le az oda vezető utakat és lépéseket. Ha az egyes cselekvéseket nem tudjuk logikai értékkel rendelkező állításokként felfogni, akkor az egész deontikus logikafogalom tévút. Másrészt viszont egyes normatívák között tényleg létezik egy következtetési viszony, ami a klasszikus logikához hasonló. Ezt a meggondolást *Jörgensen dilemmájának* is nevezzük.[146]

---

[145] Alberti Gábor-Kleiber Judit: Where are Possible Worlds? (Arguments for ℜeALIS) [AGKJ12]

[146] McNamara, Paul: Deontic Logic [Na10]

Mindezekből következően a deontikus modalitás nem adhat semmilyen tanácsot valamiféle jogszerű vagy akár erkölcsös tevékenység- vagy műveletsorra, csupán efféle cselekmények eredményeképpen kapott állapotok jogszerűségét képes megítélni.

Az, hogy egy jogi szemantikus rendszer felépítésekor a deontikus modalitásfogalom közvetlen alkalmazhatóságának komoly korlátai vannak, nem vonatkozik a közvetett alkalmazásokra is. A vegytiszta deontikus modalitás egyes részeredményeit át lehet és kell venni, a lefektetett elvekre pedig, mint kiindulópontra lehet tekinteni. A logikai rendszer elemzése után pl. egyes axiómákat célszerű a jogi következtető rendszerünkbe átemelni, és azokat szoftver eszközökkel megvalósítani.

Nem csak a deontikus modalitás, de a kristálytiszta matematikai logika eszközrendszerének közvetlen alkalmazhatóságát is cáfolja az a vélemény, hogy a jogi logikában és érvelésben a következtetési lépések nem tökéletesek. A *tökéletlen következtetés (defeasible reasoning)* a jogi következtetések egyik alapfogalma. A fogalom a középkori angol szerződésjogból származik, amelyek minden esetben tartalmaztak a szerződés érvénytelenségére utaló feltételeket is. A jogi következtetési lépések több szempontból is lehetnek tökéletlenek:

- *Korrekció vagy revízió*: a következtetési lépéseket követően, de akár már azok során is felmerülhetnek olyan újabb információelemek, amelyek a következtetés alapjait kétségessé teszik. Ilyenkor a következtetések ismételt megfontolására és esetleg megváltoztatására van szükség. Az efféle jelenségek a nem monoton következtetési rendszerekhez hasonló *konzisztencia megőrző módszerek (truth maintenance)* alkalmazását teszik szükségessé.
- *Kivételek megnevezése*: a pontos feltételrendszer megfogalmazása helyett. A törvényalkotó számára sokszor könnyebbséget jelent egy törvény logikailag pontos feltételrendszere helyett csupán egyszerűbb kivételeket megfogalmazni.
- *Heurisztikus módszerek a következtetésben.* Az efféle módszerek alapjellemzője, hogy logikai értelemben nem teljesek, vagyis nem képesek az összes következmény létrehozására. Az alkalmazásuk azért szerencsés mégiscsak, mert egy sor gyakorlati esetet lefedhetnek, és gyakran egy általános algoritmusnál lényegesen egyszerűbb vagy gyorsabb algoritmust igényelnek csak.
- *Nem biztos premisszák.* Egy jogi bizonyítási folyamatban nem mindig van lehetőség 100%-ban biztos tényekre támaszkodni (a legmegalapozottabb bizonyítékokról is kiderülhet, hogy hamisítvány). Ezt a gondot nem valószínűségi jellegű következtetési folyamattal kezeljük, hanem a megfelelően valószínű, de mégis tökéletlen tényekre alapozva egy tökéletlen következtetési folyamatot viszünk végbe. Ha a biztosnak tűnő tények és bizonyítékok megalapozottsága megkérdőjeleződik, akkor a következtetési folyamatokat ismételten végre kell hajtani. Másrészt a logikai következtetéseket akkor is végre lehet hajtani, ha tudatában vagyunk, hogy egyes feltételezések nem bizonyítottak („tegyük fel, hogy”… jellegű mondatok esetén).

4.3.3 Normák ábrázolása és következtetés

Normák ábrázolása céljából az alapvető deontikus operátorokból is kiindulhatunk. A normák általában valamilyen tulajdonság fennállására, vagy tevékenység elvégzésére vonatkoznak, ebből kiindulva a következő deontikus modális operátorok alkalmazása célszerű:

O szabály kötelező (obligatory) szabály: amelyek valamit megkövetelnek. (pl. „Házasság felbontásánál a kiskorú gyermek érdekét figyelembe kell venni.")

P szabály lehetségességi (possibility) szabály: amely valamit határozottan lehetővé tesz. (pl. „A jegyző a 30 napos határidő letöltése alól felmentést adhat.")

F szabály tiltó (forbidden) szabály, amely valamit egyértelműen tilt. (pl. „Fűre lépni tilos!")

L szabály szabadság (liberty) rögzítő szabály, olyan alapjogok esetében, mint pl. a szólásszabadság vagy a vallásszabadság. Szigorú logikai értelemben az operátor megegyezik a P lehetségességi operátorral, ezért visszavezethető arra. Megkülönböztetni akkor célszerű, ha a következtetési folyamat során az L alapjogok esetén valamilyen különleges kezelést, elsőbbséget, vagy egyszerűen csak másféle magyarázatot szeretnénk adni.

A fentiek alapján a deontikus modalitás kezelésére az ESTRELLA projektum által is rögzített elképzelést javasoljuk. Eszerint a következtető rendszer csupán egy esetleírás minősítését végzi. A minősítés a következő fokozatokból állhat:

- *tiltott* (F, forbidden), ha az esetleírás legalább egy elemében kimeríti legalább egy tiltó (F) szabály feltételeit, vagy legalább egyetlen eleme ellentmond a legalább egy kötelező (O) szabály feltételeinek.
- *engedélyezett* (P, permitted), ha az esetleírásnak sem tiltott eleme, sem a kötelezettségeket megsértő eleme nincs.
- *javasolt* (PP, proposed), ha az esetleírás lehetséges (P possible) elemeket is tartalmaz.

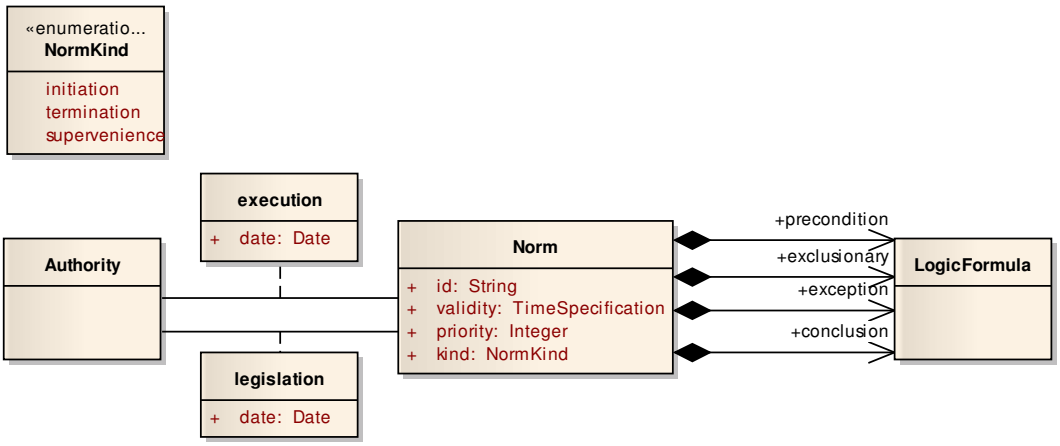
A minősítés tehát háromelemű, a sorrend pedig: tiltott<engedélyezett<javasolt, tehát magasabb prioritást adunk azoknak az eseteknek, amelyeket a törvények nyilvánvalóan engedélyezettnek minősítenek. Megjegyezzük, hogy ez az utóbbi kérdés már finomságnak számít, és teljes értékű gépi bizonyító rendszer hozható létre akkor is, ha csak kétértékű modalitást tételezünk fel: tiltott<engedélyezett. Vagyis egy ilyen rendszerben csupán a tiltott helyzetek kikövetkeztetését programozzuk be: minden egyéb viselkedést engedélyezettnek és törvényesnek minősítünk.

Jogi ontológiák építésekor a jogi normatívákat (szabályokat) modellezzük logikai eszközökkel. Ezek a szabályok általános érvényűek, amelyben az általánosságot különböző paraméterek használatával ragadjuk meg (pl.: az Eladó vagy a Vevő személye, vagy a Hatálybalépés dátuma).

A jogi szabályok mindazonáltal nem kivétel nélküliek: egyes szabályok más szabályokhoz képest kivételeket fogalmazhatnak meg, és nem is feltétlenül ellentmondásmentesek. Az ellentmondások feloldására bizonyos esetben általános meggondolásokat követünk. Ilyen meggondolások a következők:

- *„lex superior* derogat legi inferiori": a magasabb joghatóság normái előbbvalók az alacsonyabbaknál
- *„lex posteriori* derogat legi priori": a később keletkezett (újabb) normák előbbvalók a régebbieknél
- *„lex specialis* derogat legi generali": a konkrétabban meghatározott esetekre vonatkozó szabályok előbbvalók az általánosabbaknál

Normatívák ábrázolása az alábbi UML modell alapján történhet:



57. ábra Normatívák ábrázolásának UML modellje

Az ábrázolás során nem fejtettük ki a `LogicFormula` és az `Authority` osztályok értelmezését. Az előbbi egy modális logikai formulát jelent, amely a korábban már említett ℜeALM logikai nyelvet követi. Az `Authority` osztály az ontológiában található azon osztály (hatóság) egy példányát, illetve rá történő hivatkozást jelenti, akiknek a normatíva rögzítésére, végrehajtására vagy számonkérésére felhatalmazásuk van.

A normatíváknak a következő elemeik vannak:

- *előfeltétel* (precondition): azon feltételek logikai leírása, amelyek fennállására a normatíva vonatkozik.
- *kizáró feltétel* (exclusionary condition): a használati feltételek inverze. Azon feltételeket írja le, amelyek mellett a norma semmiképpen sem alkalmazható
- *kivétel* (exception) amelyek a norma hatálya alóli kivételeket jelentenek (pl. „Behajtani tilos. Kivéve itt lakók")
- *következmény* (conclusion): az előfeltételek bekövetkezése esetén – eltekintve a kizáró feltételektől és a kivételektől – a következmény bekövetkezése igaz. Ez gyakran csak valamiféle modális kifejezést jelent, pl. „A vezetővel menetközben beszélgetni tilos" mondatból a „beszélgetni tilos" a következményrész.
- *jogalkotás* (legislation): a normatíva megalkotásának (elfogadásának) dátuma és az elfogadó intézmény.
- *végrehajtás* (execution): a normatívát végrehajtó, illetve a végrehajtásért felelős intézmény (hatóság) és a hatáskörbe vétel kezdete.
- *hatályba lépés vagy a hatályvesztés dátuma* (validity), illetve az érvényesség egyéb időbeli kritériumai (pl. hegyi utakon, *télen* csak hólánccal szabad közlekedni).
- *használati elsőbbség, prioritás*: bár ez alapvető elvekből kikövetkeztethető lehetne, célszerű lehet egy egyszerű egész-szám típusú tulajdonság rögzítése is.

- *normafajta* (kind), amely megmondja, hogy valamilyen helyzetet létrehozó, felfüggesztő vagy megalapozó normáról van szó. Ez utóbbi fogalmat az ESTRELLA projektummal megegyezően[147] a következőképpen adhatjuk meg:
    - *kezdő szabályok*: amelyek valamilyen feltétel fennállása esetén egy normatív ítélet, mint állapotleírás kezdetét rögzítik. (pl. „Házasság akkor jön létre, ha a jelenlevő házasulók az anyakönyvvezető előtt erre vonatkozó egybehangzó nyilatkozatot tesznek.”). Fontos megjegyezni, hogy a feltétel nem a következmény folyamatos *fennállásának*, hanem csupán a *bekövetkezésének* feltétele.
    - *befejező szabályok*: amelyek egy normatív ítélet megszűnését rögzítik bizonyos feltételek fennállása esetén. (pl. „A házasságot a bíróság felbontja akkor, ha az véglegesen és helyrehozhatatlanul megromlott.”).
    - *fennálló szabályok*: amelyek a normatív ítélet fennállását kötik bizonyos feltételekhez. Amíg a feltétel fennáll, addig a normatív ítélet is.(pl. „A házastársak osztatlan közös tulajdona mindaz, amit a házassági életközösség alatt szereztek.”)

Mindezek fényében a normaleíró formátum a következő lehet. Az egyes nagybetűs elemek közül a `CONCL, PRECOND, COND1,` illetve `COND2` a már említett modális logikai nyelven megfogalmazott kifejezések. A szabályazonosító az UML tervet kibővítve tetszőleges Prolog kifejezés is lehet. A `TIME` kifejezés egy időreferens, amelynek a pontos formátumát már korábban leírtuk.

Az alkalmazott deontikus modalitásfajta külön szabálytulajdonságban nincs kiemelve, azt a következményrészben alkalmazott modális operátorok határozzák meg.

```
CONCL ← PRECOND : [excl:COND,
                   except:COND,
                   id: EXPR,
                   validity:TIME,
                   priority:INTEGER,
                   legislation(AUTH,TIME),
                   execution(AUTH,TIME)].
```

A leírtak bemutatására nézzünk egy példát. A CsJT 2.§ tartalmát, mely kimondja: „Házasság akkor jön létre, ha az együttesen jelenlevő házasulók az anyakönyvvezető előtt személyesen kijelentik, hogy egymással házasságot kötnek.” Más helyeken rögzítve van, hogy a házasulók csak különneműek lehetnek. A törvény tiszta (modális) logikai alakja pl. a következő lehet:

```
házasulók(FN,T):- emberpár(FN,T),
int(FN,T):házasság(FN,afterA(T)).
```

Vagyis házasulók egy adott pillanatban azok, akik emberpárt alkotnak, és szándékuk, hogy később házasok legyenek. A meghatározásban az `emberpár` egy ontológiából vett fogalom lehet. Ezt a meghatározást figyelembe véve a házasságkötés eseményét a következőképpen lehet rögzíteni:

[147] Rossella Rubino, Antonino Rotolo, Giovanni Sartor: An OWL Ontology of Norms and Normative Judgements [Rub07]

```
házasságkötés(E,T):- esemény(E,T), helyszine(E,T,H),
  házasulók(FN,T), jelenlét(FN,H,T),
  anyakönyvvezető(A,T), jelenlét(A,H,T),
  polgár(T1), jelenlét(T1,H,T), szerepe(T1,E,tanú,T),
  polgár(T2), jelenlét(T2,H,T), szerepe(T2,E,tanú,T),
  [tell(FN,T),int(FN,T)]:házasság(FN,later(T)).
```

Vagyis a házasságkötés feltétele, hogy a házassági esemény (`E`) időpontjában (`T`) és helyszínén (`H`) a házasulók, az anyakönyvvezető (`A`) és két tanú (`T1, T2`) együttesen jelen legyen, és a házasulók kimondják (`tell` modális operátor), hogy házasságot kívánnak (`int` modális operátor) kötni. Ezek után már csak a házassági viszony kezdetét kell meghatároznunk:

```
házasság(E,afterA(T)):- házasságkötés(E,T0).
```

A törvény a fent meghatározott alakban viszont a következőképpen nézhet ki:

```
házasság(E,later(T))←
  esemény(E,T), helyszine(E,T,H),
  házasulók(FN,T), jelenlét(FN,H,T),
  anyakönyvvezető(A,T), jelenlét(A,H,T),
  polgár(T1), jelenlét(T1,H,T), szerepe(T1,E,tanú,T),
  polgár(T2), jelenlét(T2,H,T), szerepe(T2,E,tanú,T),
  [tell(FN,T),int(FN,T)]:házasság(FN,later(T)):
       [id:CsJT-2/1,
       validity:later(1952,jan,1)].
```

A törvény fenti kódjában nem alkalmaztuk a `priority`, a `legislation` és az `execution` elemeket, sem az `except` és `excl` elemeket. Az előbbiek egyszerű tulajdonságok: amennyiben a megfelelő információelemek rendelkezésre állnak, úgy a törvény számítógépes logikai kódjába bevezethetők. Érdekesebb az `except`, illetve az `excl` elemek használata. Egy lehetőség, hogy a kivételek közé felvehetjük a házasság érvénytelenségével összefüggő összes feltételt (rokonok, nem nagykorúak vagy a gyámhatóság nem engedélyezte, stb.)

4.3.4 Megjelenítési javaslatok

Bonyolultabb, nem lineáris adatszerkezetek esetében általában komolyan felvetődik a megjelenítés kérdése. Fák esetében a behúzásos megjelenítés még teljesen közismert és elfogadott (pl. a Windows Explorer is ilyet csinál). A behúzásos megjelenítés alkalmazható esetleg körmentes gráfokra, ilyenkor az összefutó élekből tovaterjedő részgráfot esetleg megtöbbszörözik, ami redundancia és helypazarlás, vagy esetleg valahogyan képszerűen megjelenítik az élek összefutását. Ha még ennél is szabadabb gráfokat szeretnénk megjeleníteni, akkor a törekvés egyre komolyabb akadályokba ütközik. Egyes részosztályokra még lehetséges viszonylag áttekinthető megjelenítést alkalmazni (síkgráfok, hálók), de teljesen általános gráfokra úgy érezzük, már nem is lehetséges. Akár lehetséges, akár nem, a kérdés még messze nem eldöntött, és sok kutatási jelentés, szakdolgozat és más munka születik évente újabb és tetszetősebb gráf megjelenítő algoritmusokat nyilvánosságra hozva.

A kérdésfelvetés hasonlóan aktuális a szemantikus technológiák világában is.

Úgy érezzük: a hagyományosnak mondható grafikus kezelői felületek komoly előrelépést jelenthettek a parancssoros párbeszédhez képest: Ez a 3-4. generációs szoftverek esetében, amikor viszonylag rögzített rekordszerkezetekkel, és alapvetően

lineáris adatszerkezetekkel kellett dolgozni, még tökéletesen megfelelt. Talán éppen a tartalomközpontú technológiák, a hatalmas hálózatok megjelenítési igényei az első olyan felmerülő esetek, amikor ezek a korábbi megoldások közvetlenül már nem alkalmazhatók, legalábbis az ergonómiai hatékonyság komoly csökkenése nélkül nem.

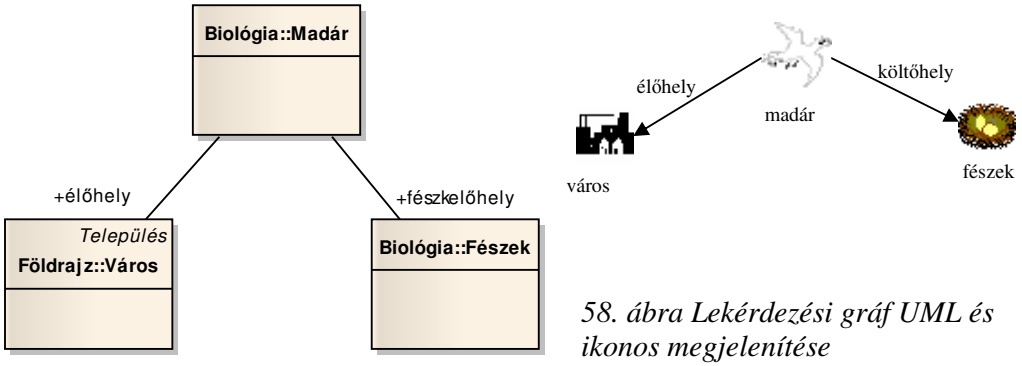
Az alapprobléma kettős:

- A hagyományos űrlapos párbeszédes modell elemei rögzítettek, és a megjelenítés is azt sugallja, hogy egymással rekord-szerűen összetartozó skaláris adatokról van szó. A technológia túlságosan statikus: ha lekérdezéshez használjuk, akkor csupán rögzített lekérdezések paraméterezhető végrehajtása lehetséges, egy teljesen új lekérdezés (egy teljesen új rekordszerkezet) futás közbeni összeállítására nincs megoldás, legalábbis olyan nincs, amilyet a végfelhasználó is képes lenne működtetni.
- Jó lenne a háttérbeli bonyolult szemantikus szerkezetről valamiféle áttekintő képet kapni. Ez azonban lehetetlen, hiszen tíz- és százezer csomópontból álló gráfot megjeleníteni sem lehet, pláne nem úgy, hogy az áttekinthető legyen.

Rugalmas és nagy adattömegre alkalmazható megjelenítési megoldások vizsgálata és efféle megoldások beépítése egy jogi ontológia esetében is elengedhetetlen. Az alábbiakban három olyan megoldást mutatunk be, amelyek némi átalakítás után egy jogi rendszerben is használhatók lennének.

4.3.4.1 A FACES megjelenítő modulja

A FACES (Frame Oriented Concise Shell for Expert Systems) a szerző által a 90-es évek elején végzett kutatási-fejlesztési munka eredménye volt. A munka két dologra koncentrált: egyrészt egy olyan *tudástár* (akkoriban az ontológia kifejezést még nem használták) létrehozására, amely a hétköznapi emberi tudás tárolásán keresztül a mai ontológiák igényével lépett fel. Másrészt pedig egy olyan ontológiavezérelt párbeszédes lekérdező felület létrehozására, amely a végfelhasználók számára is lehetőséget ad tetszőleges szerkezetű lekérdezés végrehajtására. A két célkitűzés közül az elsőt a leíró logikán alapuló ontológiák megjelenése elavulttá tette. A lekérdező felület azonban mindmáig hiányzik a palettáról, a húsz éves eredmények leporolása, megvalósítása, netán egy erre alapuló termékfejlesztés még mindmáig aktuális.



58. ábra Lekérdezési gráf UML és ikonos megjelenítése

A FACES lekérdező modulja kicsit a Sowa-féle fogalmi hálók elméletéből,[148] kicsit a szerző által az objektumorientálásról és UML-ről szóló fejezetben leírt objektumhálók és lekérdezési hálók fogalmából építkezik. Eszerint a lekérdezési hálók:

[148] John F. Sowa: Conceptual Structures: Information Processing in Mind and Machine [So84]

- csomópontjai *ismeretlen, de meghatározott osztályba tartozó* objektumpéldányoknak felelnek meg. A képszerű hatás érdekében az osztályoknak ikonok felelnek meg, és a hálón ezeket az ikonokat jelenítjük meg.
- a csomópontok közötti *élek irányítottak*, az elemek közötti *relációknak* /UML kapcsolatoknak, asszociációknak/, /OWL réseknek/ felelnek meg. Az éleknek nincs külön ikonos megjelenítésük.
- *összefutó élek* is lehetségesek akkor, ha két kapcsolatvéghez ugyanolyan osztályba tartozó példányokat ír elő a tudástár. Az összefutással a két példány azonosságát írjuk elő.
- A Prologhoz (és SQL-hez, OQL-hez) hasonlóan megengedhetünk *halmaztulajdonságokat* (aggregált függvényeket). Ezek a gazdaobjektum összes lehetséges előfordulásához rendelnek konkrét objektum vagy skaláris értéket (pl. a legalacsonyabb tanuló az osztályban).
- A lekérdezés esetleg *nemdeterminisztikus megoldásokat* hozhat. A lekérdezés a hálókat példányosítja, azaz az ismeretlen példányokat meghatározza. Egy megoldás egy példánygráf, amelyre a lekérdezési gráf által rögzített megkötések teljesülnek.

A leírt lekérdezés megszerkesztéséhez egy grafikus felület szükséges. A felületen kezdetben mindig megjelenik egy aktuális lekérdezési gráf. Ez kezdetben az egyetlen 'Világegyetem' csomópontból áll. A lekérdezési gráfon az alábbi grafikus műveleteket lehet végrehajtani:

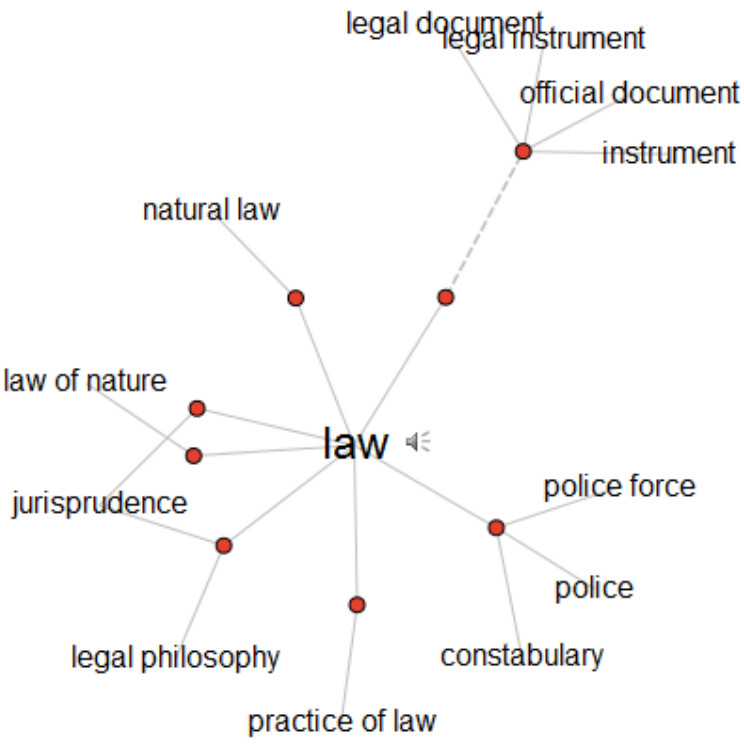
- Egy adott csomópont osztályát menüs/böngészős megoldással *konkretizálhatjuk*
- Egy adott csomópont példányát menüs/böngészős/beírásos megoldással *rögzíthetjük*. Az objektumok esetében a tudásbázis által ismert példányokból válogathatunk, skaláris adattípus tulajdonságok értékét közvetlenül beírhatjuk, esetleg értéktartományt adhatunk meg.
- Egy adott csomóponthoz a rá vonatkozó relációk közül egyet *hozzárendelhetünk*. Az érték a relációérték alaptípusa lesz, amit később konkretizálhatunk.
- Egy adott csomópontot egy másikkal *egyesíthetünk*. Az egyesítés csak akkor működhet, ha a két csomópont egymással kompatibilis, azaz a mindkét csomóponthoz kapcsolódó vagy azokból kiinduló relációs megkötések, valamint a csomópontok megkötéseinek van közös részhalmaza.

Az ilyen módon megadott lekérdezést kétféleképpen lehet futtatni:

- Közvetlenül a *tudástár (ontológia) felett*: a lekérdezés a tudástárban tárolt osztály és példányadatok alapján végzi el a következtetéseket. Megjegyezzük, hogy ez a ritkább eset: a példányadatok a közhelytudást tárolják, és gyakran a kérdező ennél lényegesen szélesebb körű adatokat kíván lekérdezni.
- Egy háttérbeli – pl. internetes – *nagy adattár-rendszer felett*. A rendszer különböző heurisztikus és jogosultságalapú beállítások alapján hozzáférhet egyes internetes adattárakhoz. Ezekhez a grafikus kérdés alapján létrejön egy saját lekérdezőnyelvükön megírt kérdés, ami futtatásra kerül.

A leírt lekérdezés szerkesztési megoldást ontológiavezérelt programokban lehet használni. Olyan esetekben lehet igazán hasznos, amikor a háttérben valami igazán *széles fogalmi rendszert* használó alkalmazás áll, vagy ha esetleg több alkalmazás, több információforrás integrációját kívánjuk elvégezni.

A lekérdezések szerkesztésén túl a megfelelő alkalmazások megcímzését, valamint azt, hogy a grafikusan összeállított lekérdező kifejezést hogyan alakítjuk át az adott alkalmazások saját lekérdező nyelvébe, az értekezés keretein belül nem tárgyaljuk.



59. ábra Egy Visual Thesaurus keresés eredménye

4.3.4.2 A Thinkmap Inc Visual Thesaurusa

A másik nagy kérdéskör a hatalmas tudástár, mint bonyolult gráfszerű adatszerkezet áttekintő megjelenítését célozza. Erre kettő, egymással is rokon megoldást találhatunk.

Mindkettő valamiféleképpen a szemantikus távolság fogalmára épít. A szemantikus távolság egyes fogalmak vagy objektumpéldányok közötti értelmezett természetes szám értékű függvény. A meghatározás alapfeltételezése, hogy a tudásunk fogalmai és példányai hálószerű elrendezésben vannak tárolva. A szemantikus távolság ezek után a két fogalomnak vagy példánynak megfelelő hálócsomópontok közötti távolsággal (élek számával) egyezik meg.

A fogalom először a szemantikus hálós tudástár modell kialakulásával lett bevezetve az informatikai kultúra kialakulása táján, de könnyű általánosítani a modern tudástárak (pl. OWL alapú) elemeire is. Mindazonáltal a szemantikus távolságot, mint függvényértéket pontosan meghatározni nem lehet, mert az érték erősen függ nemcsak a tudásábrázolási módtól, hanem a konkrét tudástártól is. Vagyis nem nehéz még ugyanazon

tudásábrázolási iskola alapján sem olyan példákat találni, amelyben két fogalom távolsága erősen különbözik az egyik, vagy a másik konkrét tudástár felett mérve.

A VisualThesaurus[149] egy olyan tezaurusz, amely a keresett fogalmat középre helyezve az azzal kapcsolatos egyéb fogalmakat és a fogalmak szóképét körben fa-szerűen ábrázolja. Ezzel lényegében csak a saját tudástárában 1 szemantikus távolságra fekvő fogalmak, (illetve a 2 távolságban levő szóalakok) lesznek megjelenítve, de a sejtésünk az, hogy az infrastruktúra alkalmas lehet mélyebb távolságú szókörnyezetek megjelenítésére is, nyilván a környezet méretének növekedése nem folytatható tetszőlegesen, hiszen talán még a 2 távolságot ítélhetnénk áttekinthetőnek, az ennél terjedelmesebbeket már aligha. Az ábrán a Visual Thesaurus internetes változatának eredményét láthatjuk a 'law' keresőkérdésre.

4.3.4.3 VisualLaw, IBM SPSS

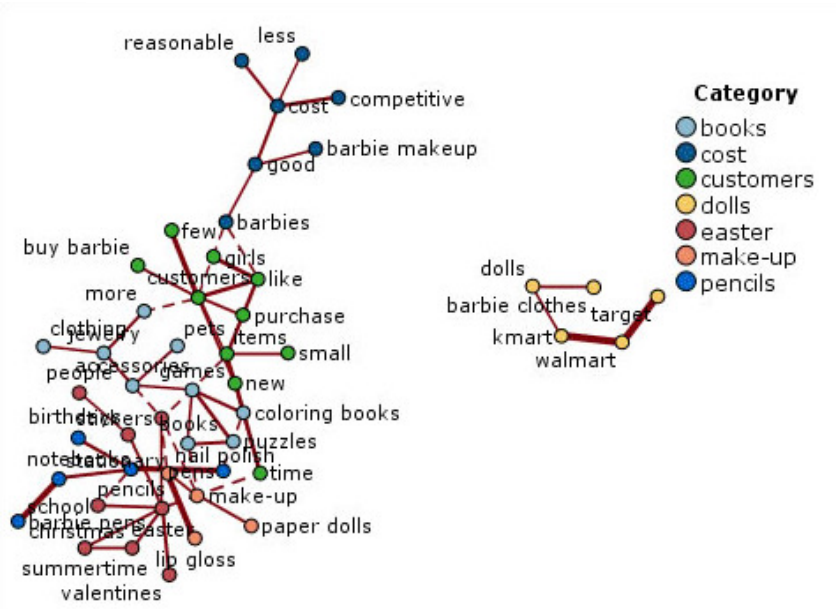
60. ábra A VisualLaw által megjelenített információelemek ábrája

Hazai (sőt, pécsi egyetemi) berkeinkben született egyik, projektkezdemény Ludányi Zoltán VisualLaw alkalmazása.[150] A rendszer a Facebook közösségi alkalmazásra épül, és a rendszeren belüli jogi (nagyrészt adatvédelmi és személyiségi jogi) figyelmeztetéseket elemzi ontológiavezérelt szövegbányászati módszerekkel, majd egy megjelenítő csomag segítségével, a Visual Thesaurushoz némileg hasonló módon, a szemantikus távolságok szerint koncentrikus körökben jeleníti meg a további részadatokat. (A színezés a „javasolt" és „nem javasolt" tartományokat jelenti: a háttérbeli döntéstámogató rendszer a piros színezésűekhez tartozó adatvédelmi szabályzatokat aggályosnak ítélte.)

[149] Using the Visual Thesaurus, Thinkmap Inc. New York, USA [VT]

[150] Mártonffy Attila: Interaktív adatvédelem [MA01]

61. ábra Az IBM SPSS modellező szoftver fogalmi térképe



Anélkül, hogy a működés mélyebb elemzésébe belemennénk érdemes egy pillantást vetni az IBM SPSS modellező és szövegbányászati szoftverének egyik megjelenítő lapjára. Az SPSS üzleti folyamatok szövegelemzésen és ontológiaelérésen alapuló modellezésére alkalmas.[151] A szoftver egyik megjelenítési lehetősége az általa kezelt modell fogalmait szintén gráfszerű formában ábrázolja – bár nem éppen a szemantikus távolság a rendező szempont, de ez nyilvánvalóan az ábrázolásmód egyik határoló tényezője kell, hogy legyen.

4.3.5 Közlekedésjogi szakontológia szerkesztése

Az értekezésben lefektetett alapelvek alkalmazhatóságát egyetemi hallgatók közreműködésével egy közlekedési jogi szakontológia-modellben ellenőriztük.

A közlekedési jog (KRESZ) a jog többi területénél sokkal kedvezőbbnek látszik a szemantikus modellezésre, mert a fogalmai sokkal kevésbé elvontak, konkrét objektumok, járművek mozgását modellezzük, amelyeknek bizonyos korlátfeltételeket kell betartaniuk.

A felépített közlekedési szakértő rendszer tudásbázisa alapvetően a következő elemekből áll.

Háttér-ontológia: ez egy OWL terminológiai tudásbázis (T-box), amely a közlekedésben használt statikus fogalmakat (utak, járművek, gyalogosok, egyéb műtárgyak), valamint a közlekedés egyes mozzanatait (haladás, kanyarodás, előzés, parkolás, megállás, stb.), és azok egymással való összefüggéseit határozza meg. A terminológiai tudás általános, tehát nem függ konkrét helyzettől, és alapvetően időfüggetlen. A háttér-ontológia egy csúcsontológiára építhető, amely a már elemzett ontológiák valamelyike, vagy – célszerűen – az LKIF jogi csúcsontológia lehet.

Nézzünk pár térképészeti és közlekedési fogalmat leíró OWL példát a háttér-ontológiából:

[151] http://www-01.ibm.com/software/analytics/spss, elérés: 21-Aug-2012.

- `Road ⊆ Section ∧ >=1section.Section`

  Az út (utca) egy olyan útszakasz, amelynek további részletei (útszakaszai) vannak.

- `Section ⊆ Location_Complex`
  `∧ 1kind.SectionKind ∧ 1rank.integer`
  `∧ 1begin.Place ∧ 1end.Place`
  `∧ ∀next.Section ∧ ∀prev.Section`

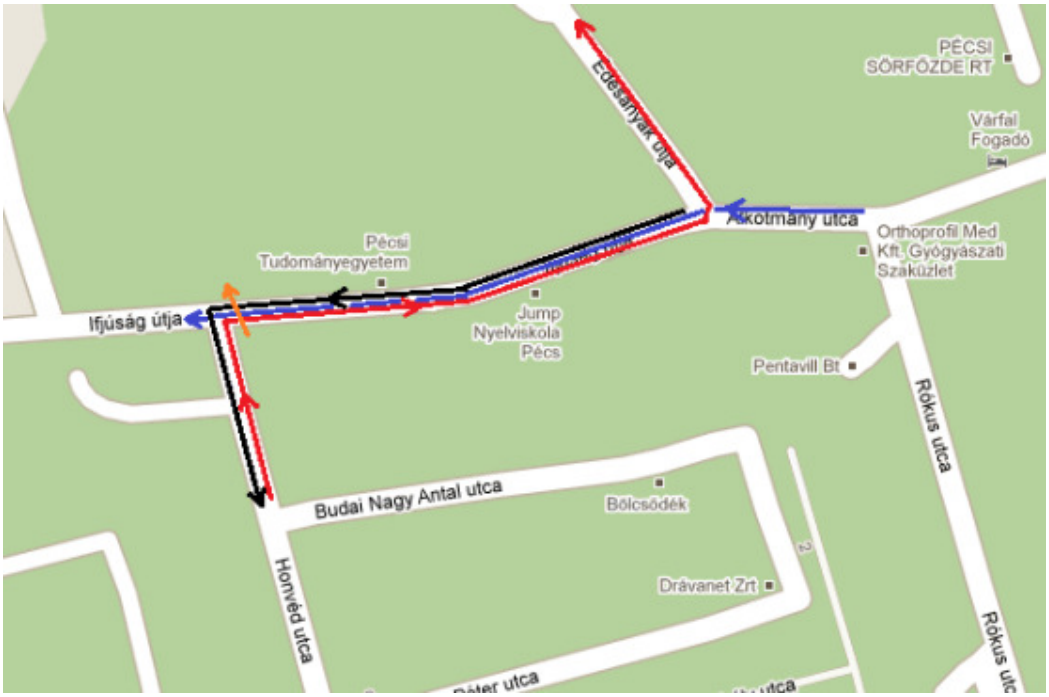
  Egy útszakasznak van pontosan egy fajtája, valamint pontosan egy kezdő és végpontja, valamint lehet egy követő és egy megelőző útszakasza is. A követő és megelőző útszakaszok egymás inverzei. Az út rangja egy egészként kódolt felsorolás, amely az elsőbbségadásban játszik szerepet (a főútvonal rangja magasabb a mellékutakénál).

- ```
  InhabitatedStreetKind={mainStreet,street,pavement,
                         crossing,zebraCrossing}
  UninhabitatedRoadKind{highway,mainRoad,accessRoad}
  TrafficArtifactKind={trainCrossing,bridge,tunnel}
  SectionKind=InhabitatedStreetKind
              ∨ UninhabitatedRoadKind
              ∨ TrafficArtifactKind
  ```

  A rendszerben a fenti útszakasz-fajták értelmezhetők: bekötő út, autópálya, országút, főútvonal, mellékútvonal, utca, járda, kereszteződés, gyalogátkelőhely, vasúti átjáró, híd, alagút; ezek feloszthatók lakott területen kívüli és belüli útszakaszokra, valamint közlekedési műtárgyakra.

- `Journey ⊆ Motion ∧ >=1motion.Motion`
  `Motion ⊆ Interval ∧ Section`
  `∧ 1starts.Moment ∧ 1finishes.Moment`

  Egy utazás egy mozgásokból (mozgások legalább egy-elemű láncolatából) áll. A mozgás egy útszakasz és egy időintervallum közös részhalmaza, amelynek az útszakasztól örökölteken túl pontosan egy kezdő és vég-időpontja is van.
  ```



62. ábra Pécs térképrészlete esetleírás-jelzésekkel

A *térkép leírása* OWL példányok (A-box) formájában történik, Ez a formátum eltér a szokványos földrajzi információs rendszerek (pl. navigációs szoftverek) formátumától, de azokhoz képest nagyrészt csupán formai módosításra és csupán kismértékű adatbővítésre van szükség. A térképi adatbázisok ugyanis nemigen tárolják az összes lehetséges közlekedési jelzést vagy műtárgyat. A navigációs adatbázisok ismerhetik az utcák egyirányúságát, az egyes utak rangját (főútvonal, autópálya, stb.), a kanyarodási szabályokat, de teljesen biztosan nem tárolnak adatokat a közlekedési jelzőtáblákról, parkolási szabályokról és egyéb műtárgyakról sem (pl. hidak, gyalogos átkelőhelyek, stb.). A KRESZ szakértő rendszer céljaira a meglevő adatbázisokat érdemes használni, kiegészítve azokat a szükséges pótlólagos információkkal.
Példaképpen tekintsük az alábbi pécsi résztérkép OWL-Prolog átírását, illetve annak egy részletét:

```
section(ifjusag1_5).
sectionKind(ifjusag1_5,mainStreet).
begin(ifjusag1_5,46.076543-18.209966).
end(ifjusag1_5,46.075962-18.207301).

section(ifjusagHonved).
sectionKind(ifjusagHonved,crossing).
begin(ifjusagHonved,46.075962-18.207301).
end(ifjusagHonved,46.075962-18.207301).

section(honved1_7).
sectionKind(honved1_7,street).
begin(honved1_7,46.075962-18.207301).
end(honved1_7,46.075022-18.207583).
```

A *közlekedési eset leírása* szintén OWL példányokat (A-box) jelent. Ezeket a modellben kézzel vittük fel a rendszerbe, de egy valóságos helyzetben a járművekre szerelt valamiféle fekete-doboz szerű szerkezet GPS adatok kiértékelésével állíthatná elő az adatokat. Alább bemutatjuk a fenti ábrán fekete nyíllal jelölt mozgás Prolog ábrázolását, illetve annak egy részletét:

```
journey(black).
motion(black,ifjusag1_5).
motion(ifjusag1_5).
starts(ifjusag1_5,10:10:10).
finishes(ifjusag1_5,10:10:45).
next(ifjusag1_5,ifjusagHonved).

motion(black,ifjusagHonved).
motion(ifjusagHonved).
starts(ifjusagHonved,10:10:45).
finishes(ifjusagHonved,10:10:45).
next(ifjusagHonved,honved1_7).

motion(black,honved1_7).
motion(honved1_7).
starts(honved1_7,10:10:45).
finishes(honved1_7,10:11:10).
```

A közlekedési *szabályrendszer leírása* az OWL ontológia-leíró nyelv SWRL szabályleíró résznyelvével tehető meg. Egy KRESZ szabály egy közlekedési mozzanatot tiltottnak, kötelezőnek vagy engedélyezettnek nyilvánít. A szabályok a szabályalkalmazási fázisban a feltételektől függően tüzelnek, és az adott esetleírást a fentieknek megfelelően minősítik. A minősítést a későbbiek során pedig egy egyedi szoftverösszetevő értékelheti ki (esetlegesen összeszámolva a kiszabható közlekedési bírság mértékét). A KRESZ szabályok átírását tekintve az előzőekben lefektetetteket követtük: de csak kétértékű, szabályos-szabálytalan osztályozást készítettünk. Ezért a szabályok csupán a szabálytalan eseteket sorolják fel: minden egyéb eset feltételezhetően szabályos. Hajszálpontos következtetést nem lehetséges csinálni, hiszen még a KRESZ is tartalmaz pontatlan megfogalmazásokat (relatív gyorshajtás, stb.) Lássunk néhány jellemző szabályra példát (a szabályok Prologra vagy Contralogra átírt SWRL-ben olvashatók).

- ```
  'Motion'(M), kind(S,K), 'InhabitatedSectionKind'(K),
    begin(M,B), end(M,E), {wgs:diff(B,E,DIST)},
    starts(S,ST), finishes(M,F),
    {time:diff(F,ST,DIFF), time:toSecs(DIFF,SECS)},
    {SPEED is DIST/TIME*3.6, SPEED>50}
  → strictlyDisallowed(M).
  ```

  A szabály egy olyan mozgásról szól, amelynek kezdő és végidőpontja, valamint időtartama is van (`TIME`). Ugyanígy kezdő és végpontja, valamint szakasztávolsága is van (`DIST`), az útszakasz viszont olyan kicsi, aminek a hosszát lineárisan, a sebességét pedig egyszerű átlagszámítással közelíthetjük. Amennyiben ez az útszakasz lakott területen belülre esik, akkor 50 km/h-nál gyorsabb sebességgel közlekedni tilos.
  ```

- ```
  'Motion'(M1), next(M1,M11), kind(M11,crossing),
    'Motion'(M2), next(M2,M21), kind(M22,crossing),
    section(M11,CROSS), finishes(M11,F1),
    section(M22,CROSS), finishes(M22,F2),
    rank(M1,R1), rank(M2,R2), {R1<R2, F1<F2},
  → strictlyDisallowed(M1)
  ```

  A szabály ugyanahhoz a kereszteződéshez (`CROSS`) egy időben (`ARRIVAL`) érkező két járművet keres. Amennyiben ilyenkor az alacsonyabb prioritású úton érkező jármű hagyná el a kereszteződést korábban, az egy tiltott mozgást jelent.
  ```

5 Összefoglalás és további lehetőségek

A jelen értekezés nem jogi, hanem informatikai tárgyú. Informatikáról lehet általánosságban beszélni: szakterület megemlítése nélkül, vagy azzal együtt. Bár a dolgozat nem jogi, hanem informatikai tárgyú, de az informatika tárgya mégiscsak a jog. Az értekezésben az informatika eszközeinek jogi szakterületre való alkalmazhatóságát vizsgáltuk meg. Ez lehet a lezárása egy informatikai tanulási és kutatási folyamat egy szakaszának – a mesterséges intelligencia szakterületén –, de messze nem a lezárása a kutatási és tanulási folyamat egészének – sem az informatika, sem a természetes nyelvfeldolgozás, sem a számítástechnika területén. A bölcsek kövét nem találtuk fel, a dolgozat ezt ezért be sem mutathatja. Egy tudományos értekezésről van szó, amely nem egy konkrét szoftverfejlesztés eredményének bemutatása. Sem egy többé-kevésbé teljes jogi ontológiát, sem ezen konzisztensen működő, kerek, egész következtető rendszert nem hoztunk létre. A dolgozat a hipotéziseinek alátámasztására bemutat viszont egy sor olyan eszközt, módszert, technikát – olyanokat, amelyeket mások hoztak létre, és nem keveset a dolgozat tárgyaként önállóan kidolgozottakból is – amelyekből, mint építőkövekből egy jogi szakértő rendszer felépíthető. Bemutat mindezek illusztrációjaként egy olyan miniatűr szakértő rendszer modellt, amely – a közlekedési jog szakterületén – egy közlekedési eset leírásából kielemzi annak jogszerűségét, és – legalábbis a modellezett terület korlátai mellett véleményt, ítéletet alkot.

A példák általában erősen, és nem ritkán nevetségesen leegyszerűsített esetek. Az iskolapéldák túlegyszerűsített volta és a valóságos esetek vagy normatívák közötti szakadék – mint a közleményekben és tankönyvekben szokásos méretű, leírható és megérhető bonyolultságú és a valós élet felmutatta bonyolultságok közötti szakadék esetében is – tehát e szakadék áthidalása most sem sikerült igazán tökéletesre.

A dolgozatban ennek ellenére beszámolhattunk egy sor figyelemre méltó eredmény megszületéséről; olyanokról, amelyek hihető módon alátámaszthatják az alaptézist. Nevezetesen azt a tézis, hogy komoly áttörést a mai számítógéppel segített jogkezelés témájában elsősorban az alkalmazott mesterséges intelligencia és a nyelvfeldolgozó technikákon keresztül lehetséges elérni.

A mesterséges intelligencia technológiák terén számos kutatási projektum indult már a jogi szakterület modellezésére, és ilyen célú szakértő, tanácsadó rendszerek elkészítésére. A kutatások eredményesek voltak, az elkészített prototípus alkalmazások bizonyították a technológia alkalmazhatóságát, hasonlóan a szerző által elkészített közlekedési szakontológiához. Mindegyik esetben érzékelhető azonban egy komoly alkalmazási akadály: nevezetesen a szakterület annyira összetett fogalmakkal dolgozik, hogy a már-már hagyományosnak mondható menüs, ablakos grafikus kezelői felület a párbeszédes információcsere-igényt kielégíteni már nemigen tudja. Ez egy fontos alkalmazási szűk keresztmetszet, amin esetleg újszerű grafikus információcsere és megjelenítési módszerekkel lehet segíteni, de a végső megoldást csak a mélyelemzésen alapuló nyelvtechnológiák alkalmazása jelentheti.

Sajnos ezen technológiák sincsenek még a polcról leemelhető állapotban. A dolgozat bemutatja a ℜeALIS nyelvészeti projektet, amelybe a szerző maga is aktívan bekapcsolódott. A projektet az alapelképzelései kétségkívül alkalmassá tehetik egy jogi szakértői rendszerhez kapcsolható természetes nyelvi modul szerepére, de a munka sajnos még csak tervezés, illetve kísérleti megvalósítási fázisban van.

Ütőképes megvalósításra akkor számíthatunk, ha a két irány összetalálkozik. Nevezetesen, ha a mesterséges intelligencia oldaláról jogi ontológiák, és rajtuk következtető szoftverek épülnek, és ha eközben a természetes nyelvi mélyelemzési megoldások is elérik a közvetlen alkalmazhatóság szintjét, és képesek lesznek jogi szövegek hatékony feldolgozására is. Úgy gondoljuk, és a dolgozatban ennek az alátámasztását is megadtuk, hogy ez az időpont önmagában már szinte karnyújtásnyi közelségben van. Az egész azonban semmiképpen sem fedhető le egy középkori szerzetes gyertyalángos és lúdtollas kódexíró munkájával vagy a magányos hosszútávfutó erőfeszítésével, hanem csakis csoportosan, több, több tudományszakos szakember együttes munkája árán hozható létre. Feltéve legalábbis, hogy az emberi, az anyagi és a műszaki feltételek is kedveznek.

A dolgozat kapcsán mindenesetre a *téma legmarkánsabb akadályozó tényezőit* is meg kell említeni.

Az egyik ilyen a jogi normatívákban nem ritkán alkalmazott *képlékeny megfogalmazások* témája. Nem könnyű matematikusan leírni, hogy mit jelent a bérlemények adásvételi szerződéseiben alkalmazott: „a jó gazda gondosságával köteles kezelni" fordulat. Vajon mit jelent a rendvédelmi szervek esetében, pl. a gyülekezési jog nem engedélyezett gyakorlásakor alkalmazható „arányos beavatkozás" elve? Hogyan lehet számszerűsíteni a közlekedési jogban a „látási és útviszonyoknak megfelelő sebességválasztás" elvét? Netán a „jó erkölcs", a „kellő körültekintés" vagy az „elvárható gondosság" elvét?

A kérdésekre hálistennek létezik válasz, bár az mindenképpen kívül esik a dolgozat témakörén. Hasonló kérdésekkel foglalkozik a matematikai logika *elmosódó (fuzzy)* logikai értékekkel foglalkozó ága. Ha tehát csak a dolgozatban leírt elveket követjük, akkor az ehhez hasonló megfogalmazások és normatívák kezelése kimarad a számítógépes megoldásból. Hozzávenni őket gyakorlatilag az alkalmazott megoldások integrációjával lehetséges. Vagyis, kezdve a modális logikákkal, az ontológiákkal, és az azokon következtetési műveleteket végző algoritmusokkal, mindezeket képessé kell tenni a fuzzy logikai állítások és megoldások kezelésére is.

Létezik egy másik fontos akadályozó tényező is: az *emberi megérzés (intuíció)* kérdése. Arról van szó, amikor a bíró esetleg már az első kérdés feltevése előtt is pontosan tudja, hogy ki a tettes. Hogyan? Ránézésre. Megérzésre. A századik és ezredik hasonló eset alapján. A vádlott viselkedése, metakommunikációja alapján. Arról, hogy izzad, remeg a hangja, esetleg nyugtalanul izeg-mozog.

Sajnos az efféle megérzések gépi helyettesítésére egyelőre nem tudunk használható megoldást javasolni, még elméletben sem. Ilyen működésmód ezért nem illeszthető be a rendszerbe – se most, se később.

A következtetésekben megfogalmazottak szinte ki is jelölik a továbblépés lehetséges irányát:

1. Célszerű lenne – valamilyen nagyobb szabású kutatási együttműködés keretén belül az értekezésben leírtakat megvalósítani. Úgy véljük, hogy egy általános célú, de természetesen jogi területen bevethető szemantikus tudásmenedzsment platform és/vagy eszközrendszer igen sokrétű konkrét alkalmazásra találhat
2. Hasznos lenne – az értekezés tárgyán túlmutató – fuzzy megoldások kutatási szintű vizsgálata, valamint ezek integrálása a kutatás tárgyát képező szoftver rendszerjavaslatokba

3. A közlekedési alkalmazás egy kész, megvalósításra érett dolog, amelyben kockázatot jelentő kutatási feladat minimális, vagy egyáltalán nincs is. Közvetlenül megbecsülhető fejlesztési erőforrásokat igényel, az eredmény alkalmazhatósága nem kétséges. Meg kell találni a fejlesztési erőforrásokat, a piaci modellt, szakjogászi munkával el kell oszlatni az esetleges kételyeket, ki kell fejleszteni és alkalmazásba kell venni az eszközt. Gyakorlati szemszögből az eszköz esetlegesen piaci GPS alapú térképi rendszerekbe is könnyen beépíthető lehet.

6 Köszönetnyilvánítás

A dolgozat megírásában igen sokan, igen sokféle módon nyújtottak segítséget, amit így, általánosságban is köszönök. Külön ki szeretném azonban fejezni a köszönetemet a következő személyeknek és csoportoknak:

- A rendszerváltást megelőző évtizedben a néhai Számítástechnikai Koordinációs Intézet Elméleti Laborjában dolgozó munkatársaimnak, hogy a Dömölki Bálint, Szeredi Péter és Köves Péter vezette néhai MPROLOG csapat tagjaként a mesterséges intelligencia feladatok terén az alapműveltségemet és gyakorlatomat is megszerezhettem.
- A ℜeALIS projektbéli munkatársaimnak, Dr. Alberti Gábor nyelvész-matematikusnak, valamint Dr. Kleiber Juditnak és Károly Mártonnak. Az ő gondolataik – az egész ℜeALIS koncepción keresztül – leírt vagy csupán személyes közlés révén átadott formában – a jelen dolgozatnak igen komoly és kihagyhatatlan háttéranyagát képezték.
- Dr. Balogh Zsolt György témavezetőmnek, aki a jogi tapasztalatlanságomat egy-két jól irányzott megjegyzéssel mindig tudta kezelni és mederbe terelni.

Kilián Imre doktorjelölt

Kelt Gyűrűfűn, 2013. április hó 17-én.

7 Summary

Juristics had already separated from people's common sense a long time ago. It is continuously developing, multiplying and getting more and more complex in a quantitative and in a qualitative sense as well. It is changing in such an enormous speed, that it can't be expected from an average citizen to be aware with legal borders any more. One consequence of its exploding nature – an explosion never makes it differently – is a mess, a chaos. The more new normatives are arising on each level of the administration, the more mistakes, incompatibilities and inconsistencies are becoming inside and among them. Inconsistency means: inaccuracies, clashes (normatic collisions), uncovered gaps and more fold covered topics.

To overcome these inaccuracies there can be two approaches to the solution. The more difficult (and more expensive!) option is: if the law is an impenetrable bush wood, then let's take our machetes and cut a clearing! If the law is too complex, then let's take motivated lawyers, who would make it simpler.

The easier option is: to have computers cope with law. Although the author is convinced that the more difficult option is also the more efficient, he has much more to say about modelling complex domains semantically. This is the special expertise of the author: and this is the main topic of the present thesis.

Regarding computer aided juristics there are already clearly perceivable results, and there is an obviously recognizable trend that concerns the development of the underlying technology. These can be characterized the most apparently by the interaction of two technological directions: the one of lingual technologies and the other of artificial intelligence technologies.

The pioneering technological stand – that is still on the market today – is that of freetext search, being eventually enhanced by morphological analysis of words. (The latter is necessary of course only in case of agglutinative languages, like Hungarian, that are rich in systems of morphological affixes and endings.) This technological level has been later improved by the technology of thesaurus-based query enhancing, which is also the state-of-commerce today.

As an improvement, the second technological generation applied already lingual shallow-analysis (Part Of Speech/ POS-Tagging), and its result was assembled in XML-databases. In the background it has still used thesauri, but this slowly transitioned into the application of ontologies – in the beginning mainly with poor semantic links, only for terminology classification purposes. Concerning such technologies there are successful and closed research projects, but nobody speaks about market success stories.

The third generation of computer aided juristic technologies has a close relationship with so called deep natural linguistic analysis, and with the strong application of semantically rich ontologies: the detailed and accurate application of them, and inference operations over them. These are still much in state-of-the-art today. Could we reach a strong breakthrough concerning these, we can also cause a breakthrough in its applicational world. That is, we could market a quite new generation of juristic software products. And this is the main topic of the current thesis: the results of the author, mainly from the third generation of juristic software.

Concerning the technological development mentioned above: the higher the technological niveau evolves, the more complex the software will be, and also the higher its research and development costs will increase.

From the viewpoint of artificial intelligence and lingual technologies choosing the legal domain is a very advantageous solution, because the alliance can be mutually fertilizing for all parties. It makes the job of AI expert significantly easier, if at least the semantics of the target domain – along with the relevant language – is accurate, at least intentionally. Without any doubt, legal texts at least are trying to be clear, exact and accurate. Beyond this, it would also become a juristic result, if the technology can reveal the legal utterance mistakes – that is if the technology could significantly help to discover legal inaccuracies, gaps, overlappings and collisions.

The main hypothesis of this dissertation is that there is still an enormous amount of unused potential in computer aided juristics. These are mainly connected with the effective application of lingual and artificial intelligence technologies. The user demands are still mainly hidden, because the commercial technological niveau is still unable to perform certain groups of emerging everyday tasks. We expect from the current research, that when integrating advanced technology successfully, it would make completely new, still unapproached classes of tasks solvable for the first time.

To solve the problems sketched above, we need interdisciplinary – or rather multidisciplinary – approach. The solution lays somewhere on the triple or quadruple border of computer aided linguistics, mathematical logics, informatics and juristics. To support (and/or prove) this hypothesis the following operations are made.

- The dissertation discusses the evolution of computerized lingual technologies: it is important to differentiate between the state-of-commerce and the state-of-art technology. We are pointing out those topics and questions (bottlenecks) which could mean significant development or even a breakthrough.
- We perform similar examinations concerning also artificial intelligence technologies.
- We are reviewing similar research projects or commercial products from the viewpoint of results to be taken over. First of all we mean here certain standards or proposals, and/or products that can manage, verify or control the quality of standards mentioned above.
- We are developing study-software in cases when theoretical investigations are not enough, too difficult, or unable to serve an inevitable evidence for the hypothesis. It is also advised to develop certain key-software elements, where they open new perspectives to develop other new software (e.g. certain compilers). It is by no means feasible to expect the development of a full fledged juristic inference software, as an objective of the present research, not even in a prototypical state.
- The research, described above, will determine a circle of operations, and also a software architecture. When evaluating the results, we are listing the operations being implemented, and pointing out certain tasks that the sketched architecture cannot comply.

The aimed results and technologies can be reached and implemented by the following steps:

- To model and map information, and even knowledge, to perform inference over the knowledge, is possible only on a strong logical background. To model legal

knowledge – normatives and cases – however, the means of classical logic are not enough any more – we need higher logical formalism for this, the framework of modal logic at least. Concerning modal logics, we need multimodality and polimodality at the same time, that is we need more than two modal operators, and we may also need operators with more than one argument.

i. We must use temporal modal logics to describe time relations.

ii. We need multi-agent epistemic and doxastic logics when we want to describe legal processes and cases.

iii. We must use deontic logical framework to describe normatives, laws and rules.

- We must be able to perform inference steps over the selected logical framework. Already existing inference methods (e.g. basic/binary resolution) in many real-life cases are not efficient enough. This is especially true for concrete domains (e.g. natural/real numbers) when the domains have special purpose solver algorithms. Concerning these, the author has defined the concept of embedded resolution: which allows invoking special solver packages in the general process of resolution.
- As far as applied software tools are concerned, we think that the complexity of software to be realized can be reached only by the paradigm of logic programming. In a narrower sense, this denotes Prolog as a programming language; in a broader sense, all the software design and realization methods that are based on mathematical logics. Prolog is itself a theorem prover of mathematical logics, but its capabilities are presumably not enough. But the Prolog family of connected tools is rich, and it also enables to easily create further extensions.

    a. Prolog is basically typeless, but it is easy to extend by type concept. As a side result of the research, the author has defined and implemented ProType, a type-declarational package for Prolog, and developed the relevant type checking algorithms.

    b. The author has developed Contralog, a forward-chaining inference mechanism. This extends the native backward chaining inference mechanism of Prolog by a forward-chaining mechanism, and the PC-Log (Pro-Contra-Log) that programmatically connects them.

    c. For different tasks and domains, constraint-logical solver packages (CLP) are available. It may also be necessary to apply such packages for intelligent legal software.

2. It is only the paradigm of broadly mentioned logic programming that enables to build knowledge-banks based on the ground of mathematically precise formalism and tool set. For this purpose we can see the following solutions:

    a. Unified Modelling Language (UML) is a software modelling and designing toolset. We could utilize its class-diagrams especially well, because they are depicting concepts, classes and relations very precisely and effectively. A common counter-argument is its semantic inaccuracy. The author of this thesis has elaborated and described the semantic grounding of UML class diagrams that connects UML and logic programming paradigm. Nevertheless, we have rather used UML only as a perfect visual representation language in this dissertation.

b. As the basic architectural model for general-purpose software handling knowledge bases, the author has introduced the concept of model- and ontology-driven software. Another term for this is: two-level software, because their main feature is to handle software models or ontologies also as data. This level of the software is called model-level. Here a model of information is stored and managed, along with their relevant operations (loading, saving, editing, verification, etc.). Users of this level are always educated IT professionals. The other is the data-level that conforms to the model described on the model-level. Operations on the data-level are general, and they are governed by the model itself. There may be an application software layer connected to the data level, whereby any end-user may access and operate it.

c. Concerning model- and ontology-driven applications, the author has implemented an UML-model verifier package that has been applied in the SILK research project. The verifier also applied the embedded resolution principle. However, to apply the principles to OWL knowledge bases is not necessary, because there are already used algorithms for this purpose (the family of so-called tableaux-algorithms).

d. Similar technologies have already been applied in the Semantic Web project initiative. It applied the Description Logic (DL) framework, and has designed the OWL (i.e. Ontology Description Language). The DL framework is also interesting, because it is theoretically possible to implement verifier and theorem prover (the tableaux algorithm) without facing problems of theoretical undecidability and unsolvability in higher orders of logic. In the same time, OWL is a de-facto standard and there is already a number of different level ontologies (upper-level, middle-level and target-level ontologies) for different domains available; among those, legal ontologies also. There are further different toolkits to edit, manage and verify OWL ontologies. For our legal ontologies it is inevitable to respect some standards, and OWL is the most suitable standard for this. We also propose to take over existing experience in ontology design, and also existing ontology products.

e. The most important existing ontologies, to regard as a candidate to take them over, are: SUO, the American common purpose standard upper ontology, DOLCE the Italian ontology (first of all for lingual processing purposes), or LKIF, the legal upper level ontology of the Estrella international research project.

f. OWL ontologies are based primarily on classical logical frameworks (DL is a subset of First Order Logic). However, to build intelligent legal advisory systems, these frameworks are not satisfiable any more. We need an enhancement of OWL in the modal direction.

3. Our language-technological principles demand a deep-analysis solution. Deep lingual analysis means rather to discover the meaning of sentences, and the internal conceptual connections instead of pure syntax. Because of this, deep analysis goes far beyond the borders of a sentence. This technology is also called discourse-representation. The author has been involved in the ℜeALIS linguistic research project at the University of Technology, Faculty of Humanities. In the course of this collaboration he has reached the following results.

a. Based on the earlier mentioned ProType extension package that implements type concept in Prolog, the author has specified the ℜeALLan language that is suitable to define lingual information for the ℜeALIS project. The language is an application of Prolog's operator grammar that is syntactically a subset of Prolog. Its well-formedness can be checked by the ProType package.

b. By the help of ProType the author has also specified the ℜeALM modal logical language, which could be a final result of the ℜeALIS lingual deep-analysis. The language mirrors back the bases of human brain-representation, as the project sets it forward, and includes the logical means of epistemic, doxastic, and for legal applications, also deontic modality.

c. The author has defined a target model for ℜeALIS lingual analysis. This means the Prolog code-pattern that can be received from ℜeALLAN, after a translation step. This code-pattern is the basis of the code that itself performs lingual analysis. In the course of this, the author has defined the concept of relational and implicational target models, and applied the last one for the basic model of ℜeALIS lingual analysis. The experimental lingual analysis program has been run on Prolog; first under pure Prolog, exploiting its backward-chaining inference strategy; then under Contralog, to experiment with forward chaining strategy. The results have justified the expectations.

d. The author has created an experimental Prolog program to implement Alberti's model of bipolar influence-chain for semantic representation of verbs. Results were satisifiable here also: certain queries caused a number of nondeterministic results. Those can be filtered out later by a query to the ontology package.

The principles, laid down in the previous sections, and the tools and solutions, being partly own-implemented, partly available in the public domain, served for the author as a sufficient background to create a general design of semantically based legal knowledge-management software.

1. In the use case analysis the author has sketched up the use cases of a general purpose legal software. The analysis could not go too deep, because there were no concrete end-user demands. We have tried to define the possible use cases by multiplication, on a greatest common divisor principle, that is when speaking about concrete demands and concrete use cases, the uninteresting ones can simply be left out.

2. The author has sketched up the architectural design of a general purpose legal software. The architecture reflects mostly the applied technologies which serve as a ground for the actual operations, but at the same time the architecture itself is highly independent from them. It was an important factor to ensure a high level of reusability of the designed architecture. Because of this, the designed software components are general enough to serve different client requests. Another important criterion was the scalability of the system: that is the software components may run in a single machine, but in response to the ever rising demands, the functions could be detached to separate components, and could be

deployed on different servers. The design has identified three big groups of components:

a. The best is to deploy the application software on a separate server. This is the component where all the use cases are implemented. The circle of actual operations can be increased or decreased freely: depending on the actual end-user requirements.

b. The natural language processing package must store the lexicon that controls its operations, and it must use the services of an ontological database. The lexicon is the target format of a compiler that compiles from the source format (the ReALLan language), and its result is linked into the target program. The ontology services can be shared with the general purpose ontology component of the system, but it is also feasible to store the lingual information separately, possibly together with the natural language processing component. Anyhow, if we want to use a common, shared ontology server, they must apply the same logical background, and also the same logical and physical interfaces. Another option may be to share some common ontology segments (e.g. a common upper level ontology) in their source form. In this case, the shared ontology pieces can be stored on a common server, but for sharing the ontology files, a pretty simple file service is completely enough.

c. The knowledge base itself stores all the information necessary for the legal inference operations, that is: 1. it stores legal norms 2. stores legal cases and 3. stores legal procedures and all supporting documents (evidences, decrees, etc.). The storing format should follow the principles of ontology creation, as discussed above.

As an experimental implementation, with the supervision of the author, students have created a model ontology for the traffic rules. This application does not use lingual technology: all the traffic norms have been transformed into ontological format by hand. The results have proven the expectations: if we presume the availability of a digital map as a background, which would also contain traffic information, and if we presume a case description that is entered for test purposes by hand, the system can inference if the case has violated traffic rules. Such a device could be installed successfully either similarly to the black box of aircrafts, or as an extension to commercial GPS systems.

As a summary, we can state: the most important hypothesis has been justified. Having applied artificial intelligence technologies and natural lingual deep-analysis techniques, the efficiency of computerized legal applications can be significantly improved. A serious problem, opposing this, is that these technological elements are still in research phase and they cannot be applied immediately. Because of this, even to set up only an introductory application could not be seamless; neither cheap, nor risk-free.

1. At the same time, we must also recognize the bottlenecks of legal reasoning processes. First of all: in the legal domain, there are a great number of ambiguous expressions as mistakes, or intentionally flexible definitions, which make the task of automated reasoning significantly harder. Such are: “the necessary carefulness” in case of rented properties, “proportional interference” for example in case of breaching the right of civil assembly, or even “exceeding the appropriate speed” in case of bad road or sight conditions (fog or ice).

Managing similar expressions is out of the scope of the dissertation, but it is still possible by the help of fuzzy logic frameworks.

2. In many cases, lawyers work by intuitions. (For example an experienced judge very likely can state, only from metacommunication, whether the defendant is guilty or not). Such intuitions, of course, can not be replaced by automated reasoning devices.
3. Applying computerized legal tools is, in many cases, self-evident and obvious. Yet there might be a number of circumstances that raises further questions, for example ethical ones or those of concerning civil rights. (For example: can we install "black boxes" in private vehicles to record their movements? Could a machine judge a situation, could it make decisions over the fate of human beings?) Without doubt, to set these questions, to examine or answer them is also beyond the scope of the present dissertation. There is only one point we can add, as a final word: computer aided solutions don't take anything over from human beings. Neither task, nor responsibility. They may help in getting administrative tasks done more quickly, they may search and collect information more effectively, they may also sketch up certain argumentation models, but this is only an advice. The final word is always told by a human being who makes a decision and signs the appropriate documents.

8 Irodalomjegyzék

Logikai és mesterséges intelligencia alapozó művek

[BI01] Bach Iván: Formális nyelvek.
Typotex Kiadó, Budapest, 2001.

[ChLee83] Ч. Чень-Р. Ли: Математическая логика и автоматическое доказательство теорем.
Наука, Москва 1983.

[Csir93] Csirmaz László: Matematikai Logika
ELTE Egyetemi Jegyzet. 1993, Budapest

[DePa99] Demetrovics-Denev-Pavlov: A számítástudomány matematikai alapjai
Nemzeti Tankönyvkiadó, 1999.

[DPLL00] Peter Baumgartner: FDPLL – A First-Order Davis-Putnam-Logeman-Loveland Procedure
Institut für Informatik, Universität Koblenz-Landau 2000.
(http://www.uni-koblenz.de/~peter, letöltve: 21-Aug-2006.)

[GeNi87] Michael R. Genesereth-Nils J. Nilsson: Logical Foundations of Artificial Intelligence
Stanford University-Morgan Kaufmann Publishers Inc. 1987.

[Lov04] Lovász László: Computational Complexity Lecture Notes
(http://research.microsoft.com/users/lovasz/notes.htm, elérés: 2006. már nem elérhető 2013-Ápr-15-én)

[Mé89] Mérő László: Észjárások
Akadémiai Kiadó-Optimum Kiadó, Budapest 1989.

[MPB] Maria Paola Bonacina: A taxonomy of theorem-proving strategies
Dept of Computer Sciences, The University of Iowa, Iowa City, USA

[Per97] Jaroslav Peregrin: What Does One Need When She Needs „Higher Order Logic”? Proceedings of LOGICA'96, FILOSOFIA, Praha, 1997.

[Zoh81] Zohar Manna: Programozáselmélet
Műszaki Könyvkiadó 1981.

Logikai programozás

[Apt93] Krzysztof Apt: Reasoning about Termination of Pure Prolog Programs 1993.

[Bar99] Roman Barták: Constraint Programming: in Pursuit of the Holy Gral
Charles University. Fac.of Mathematics and Physics, 1998.

[CHR94] Thom Frühwirth: Theory and Practice of Constraint Handling Rules
Journal of Logic Programming 1994:19 20:1-679

[CloMe] W. F. Clockshin-C. S. Mellish: Programming in Prolog
Springer Verlag, New York 1987.

[DPR00] Agostino Dovier, Carla Piazza, Gianfranco Rossi: A uniform approach to constraint-solving for lists, multisets, compact lists and sets
Rapporto di Ricerca, Dipartimento di Matematica, Università di Parma, 2000
(http://www.math.unipr.it/~gianfr/papers.html, letöltve: 21-Aug-2006.)

[DPR98] Agostino Dovier-Alberto Policriti-Gianfranco Rossi: A uniform axiomatic view of lists, multisets and sets, and the relevant unification algorithms
Fundamenta Informaticae 36 (2/3) 1998. (pp.201-235)

[Ger97] C. Gervet: Interval Propagation to Reason about Sets: Definition and Implementation of a Practical Language
IC-Parc, Imperial College, William Penney Laboratory, London, UK.
c.gervet@doc.ic.a.uk
(http://www.icparc.ic.ak.uk/~cg6/conjunto.html, letöltve: 21-Aug-2006.)
Appeared in Constraints, An International Journal, I. pp 191-246
Kluwer Academic Publishers, Boston, 1997

[GCLA92] Per Kreuger: GCLA II. A Definitional Approach to Control
Swedish Institute of Computer Science April 1992.

[RKee90] The Craft of Prolog
The MIT Press, Cambridge, Massachusetts London, England 1990.

[Ro00] Gianfranco Rossi: {log} User's Manual
Universita di Parma, Dip. Di Matematica, Italy, Sept. 2000.
(http://www.math.unipr.it/~gianfr/setlog.Home.html, letöltve: 21-Aug-2006.)

[SICS02] M. Carlsson-J. Widen: SICStus Prolog 3.9. Users Manual
Swedish Institute of Computer Science 2002.

[SzPBT02] Szeredi Péter-Benkő Tamás: Nagyhatékonyságú logikai programozás
Egyetemi jegyzet.
Budapesti Műszaki Egyetem, Számítástudományi és Információelméleti Tanszék
2002. szeptember

[Wal89] Clifford Walinsky: CLP(Sigma)
ICLP Proceedings 1989. pp.181-190

Objektum-orientált technológia

[Ang97] Angster Erzsébet: Az objektumorientált tervezés és programozás alapjai
4Kör Kft, 1997 Budapest

[Bad00] Liviu Badea, Doina Tilivea: Query Planning for Intelligent Information Integration using Constraint Handling Rules

National Institute for Research and Development in Informatics
Bucharest, Romania 2001.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Design Patterns Elements of Reusable Object Oriented Software
Addison-Wesley 1994.

[ICOM] ICOM – Intelligent Conceptual Modelling Tool
Version 1.1. Manual, 2000
(http://www.cs.man.ac.uk/~franconi/icom Letöltve: 21-Aug-2005.)

[MDA01] Joaquin Miller-Jishnu Mukerji: Model Driven Architecture
OMG Document July-2001.

[MDAJS-01] Jon Siegel: Developing in OMG's Model-Driven Architecture
OMG White Paper, November 2001.

[MDA03] Joaquin Miller-Jishnu Mukerji: MDA Guide Version 1.0
OMG Document July-2003.

[MOF97] Joint Revised Submission: Meta Object Facility (MOF) Specification
OMG Document 1997.

[ODMG99] R. G. G. Cattell-D. Barry és mások: The Object Data Standard: ODMG 3.0
Morgan Kaufmann publishers San Francisco, USA 1999.

[Ra01] Raffai Mária: Egységesített megoldások a fejlesztésben
Novadat kiadó, 2001.

[Sa00] Nikos A. Salingaros: The Structure of Pattern Languages
Division of Mathematics, University of Texas, San Antonio, Texas, USA
Architectural Research Quarterly Vol. 4. pp.149-161.
(http://www.math.utsa.edu/ftp/salingar.old/StructurePattern.html, letöltve: 29-Máj-2012.)

[SILAN00] SILAN – the SILK language
IQSOFT, Budapest Hungary 2000

[SILK01] L. Badea-D. Tilivea-A. Hotaran-Y. Polet-N. Chancevrier-D. Parents-X. Denis: Description of the SILK Mediator tools
SILK Consortium, 2001.

[UML99] Rumbaugh-Jacobson-Booch: Unified Modelling Language Reference Manual
Addison-Wesley-Longman Inc. 1999.

Ontológiák, Szemantikus Világháló

[BCG03] F. Baader - D. Calvanese – D. McGuinness et al. eds: The Description Logic Handbook. Theory, Implementation and Applications
Cambridge University Press, 2003.

[BaSa00] F. Baader – U. Sattler: An Overview of Tableau Algorithms for Description Logics
Theoretical Computer Science, RWTH Aachen University, Germany 2000.

(http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.29.3832&rep=rep1&type=pdf, elérés: 2012-Ápr-17.)

[BLHL01] T. Berners-Lee - J. Hendler - O. Lassila The Semantic Web *Scientific American Magazine*. 2001. (http://www.sciam.com/article.cfm?id=the-semantic-web&print=true, letöltve: 21-Már-2012.)

[CHMP08] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler: OWL 2: The next step for OWL. Journal of Web Semantics, 6(4):309-322, Nov 2008.

[CL07] Information Technology – Common Logic (CL): a framework for a family of logic-based languages
ISO/IEC 24707 szabvány
(http://standards.iso.org/ittf/PubliclyAvailableStandards, letöltve: 25-Ápr-2012.)

[CycL02] TheCycFoundation: Ontological Engineer's Handbook
(http://www.cyc.com/doc/handbook/oe, letöltve: 13-Ápr-2012.)

[GaPr09] A. Gangemi-V. Presutti: Ontology Design Patterns
in Handbook on Ontologies, pp.221-243.
Springer Verlag, Heidelberg, 2009.

[GeFi92] Michael R. Genesereth-Richard E. Fikes: Knowledge Interchange Format Version 3.0 Reference Manual
Computer Science Department, Stanford University
Standford, California, 1992.

[Gro03] Benjamin N. Grosof-Ian Horrocks-Raphael Volz-Stefan Decker: Description Logic Programs: Combining Logic Programs with Description Logic
Proceedings of the Twelfth International World Wide Web Conference (WWW 2003), pages 48-57. ACM, 2003.

[HoBe11] Matthew Horridge, Sean Bechhofer. The OWL API: A Java API for OWL Ontologies. Semantic Web Journal 2(1), Special Issue on Semantic Web Tools and Systems, pp. 11-21, 2011.

[KQML93] Tim Finin-Jay Weber-Gio Wiederhold-Michael Gensereth-Richard Fritzzon-Donald McKay-James McGuire-Richard Pelavin-Stuart Shapiro-Chris Beck: DRAFT Specification of the KQML Agent-Communication Language
DARPA Knowledge Sharing Effort, 1993.
(http://www.cs.umbc.edu/KQML/kqmlspec.ps, letöltve: 27-Ápr-2012.)

[Knu04] Knublauch, Holger - Musen, Mark A. - Rector, Alan L. (2004): Editing Description Logic Ontologies with the Protégé OWL Plugin. In: Haarslev, Volker-Möller, Ralf eds, *International Workshop on Formal Biomedical Knowledge Representation*, Whistler BC, Canada. pp. 70-79.

[KrMá85] Krauth Péter-Márkus András:
OPS5: egy sikeres eszköz szakértői rendszerek készítésére
Információ Elektronika 1985/3 pp.129-136, 1985/4 pp.211-221

[MA11] Mártonffy Attila: Interaktív adatvédelem, ITBusiness folyóirat
IT-Business Publishing Kft. Budapest, 6-Szept-2011.

[MIKE10] Mike Bergman: Brown Bag Lunch: An Intrepid Guide to Ontologies
(http://www.mkbergman.com, letöltve: 28-Jan-2012.)

[Nagy04] Nagy Zsolt: Leíró logikán alapuló tudáskezelő rendszer
BMGE-VIK TDK dolgozat, Budapest 2004.

[Ob07] Marek Obitko: Introduction to Ontologies and Semantic Web
Faculty of Electrical Engineering, Czech Technical University, Prague, 2007.
(Elektronikus jegyzet, http://obitko.com/tutorials/ontologies-semantic-web/ letöltve 16-Aug-2012.)

[OCL02] Damien Pollet-Didier Vojtisek-Jean-Marc Jezequel:
OCL as a Core UML Transformation Language
INRIA/IRISA Campus de Beaulieu, April 2002.

[ODM09] Object Management Group: Ontology Definition Metamodel Version 1.0
Object Management Group, 2009 május.
(http://www.omg.hu, letöltve 17-Jún-2012.)

[Ohl88] Hans Jürgen Ohlbach: A Resolution Calculus for Modal Logic
FB Informatik, University of Kaiserslautern, Germany, 1988.
(http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.5003, letöltve: 25-Jun-2012.)

[OWL209] Motik, Bruno - Patel, Peter F. - Schneider-Bijan, Parsia (eds, 2009): OWL2 Web Ontology Language Structural Specification and Functional Style Syntax, W3C Recommendation

[Pa05] Bijan Parsia, Evren Sirin, Bernardo Cuenca Grau, Edna Ruckhaus, Daniel Hewlett: Cautiously approaching SWRL
(http://www.mindswap.org/papers/cautiousSWRL.pdf, letöltve: 3-Aug-2012.)

[PS04] Peter F. Patel-Schneider: A Proposal for a SWRL Extension towards First-Order Logic
Bell Labs Research, Lucent Technologies, W3C Member Submission 2005.
(http://www.w3.org/Submission//2005/SUBM-SWRL-FOL-20050411, letöltve: 28-Jul-2012.)

[SiPa07] Evren Sirin, Bijan Parsia: SPARQL-DL: SPARQL Query for OWL-DL In
Proceedings of OWLED Third International Workshop, 2007.

[So84] John F. Sowa: Conceptual Structures: Information Processing in Mind and Machine
IBM Systems Research Institute – Addison Wesley Publishing Company 1984.

[SzLB05] Szeredi P. - Lukácsy G. - Benkő T.: A szemantikus világháló elmélete és gyakorlata
Typotex, Budapest 2005.

[SWRL04] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, Mike Dean: SWRL: A Semantic Web Rule Language Combining OWL and RuleML
National Research Council of Canada, Network Inference, Stanford University, W3C Member Submission, 2004
(http://www.w3.org/Submission/2004/SUBM-SWRL-20040521, letöltve: 28-Jul-2012)

[SUO01] Niles, I - Pease, A: Origins of the Standard Upper Merged Ontology: A Proposal for the IEEE Standard Upper Ontology
Proceedings of Measuring Intelligence and Performance of Intelligent Systems Conference, 2001.

[VT] Using the Visual Thesaurus
Thinkmap Inc. New York, USA
(http://www.visualthesaurus.com, letöltve: 21-Aug-2012.)

[WW03] Claudio Masolo-Stefano Borgo-Aldo Gangemi-Nicola Guarino-Alessandro Oltramari-Luc Schneider: The WonderWeb Library of Foundational Ontologies Preliminary Report
ISIB-CNR, Padova, Italy, 2003

Nyelvészet, nyelvtechnológia

[AGKJ09] Alberti G. - Kleiber J.: Grammar of ℜeALIS and the implementation of its dynamic interpretation
Kézirat, PTE-BTK, Nyelvészeti Tanszék, 2009.

[AG11] Alberti Gábor: ℜeALIS. Interpretálók a világban, világok az interpretálóban.
Akadémiai Kiadó, Budapest, 2011.

[AGKJ12] Alberti Gábor - Kleiber Judit: Where are Possible Worlds? (Arguments for ℜeALIS) In: Acta Linguistica Hungarica, Vol. 59 (1–2), pp. 3–26 (2012)

[Alz07] Alexin Z.: A frázisstrukturált Szeged Treebank átalakítása függőségi fa formátumra.
Tanács A. - Csendes D. (szerk.): V. Magyar Magyar Számítógépes Nyelvészeti Konferencia
Szegedi Tudományegyetem, Szeged 2007

[ASC] Almási A. - Vincze V. - Sulyok M. - Csirik J.: Adó- és jövedéki jogi wordnet
SZTE Alkotmányjogi Tsz-Informatikai tanszékcsoport
(http://www.maszeker.hu/download/publication/ado_es_jovedeki.pdf, letöltve: 3-Máj-2009.)

[Fe98] Ch. Fellbaum: WordNet An Electronic Lexical Database
The MIT Press 1998. ISBN-10:0-262-06197-X
ISBN-13:978-0-262-06197-1

[KJ10] Kleiber Judit: A totális lexikalizmus elméletétől a kísérleti implementációig
PhD értekezés. Pécsi Egyetem, Bölcsészettudomány Kar 2010.

[KM11] Károly Márton: Interpretáció, intenzionalitás, modalitás – avagy a ℜeALIS λ függvényének interpretációja felé
VIII. Magyar Számítógépes Nyelvészeti Konferencia, Szeged, 2011.

[Mi03] Miháltz M.: Magyar főnévi WordNet ontológia létrehozása automatikus módszerekkel
Morphologic Kft. 2003.
(http://www.morphologic.hu/downloads/publications/mm/mszny_2003_mm.pdf, letöltve: 3-Máj-2009.)

[Un02] Ungváry R. szerk: OSZK Köztezaurusz
OSZK, Budapest, 2002.

[ZVF10] Zsibrita, János - Vincze, Veronika - Farkas, Richárd: Ismeretlen kifejezések és a szófaji egyértelműsítés. In: Tanács Attila, Vincze Veronika (eds.): VII. Magyar Számítógépes Nyelvészeti Konferencia pp. 275-283.
Szegedi Tudományegyetem, Szeged 2010

Modális logikák

[Ga08] Galton, Antony: Temporal Logic, *The Stanford Encyclopedia of Philosophy (Fall 2008 Edition)*, Edward N. Zalta (ed.) (http://plato.stanford.edu/archives/fall2008/entries/logic-temporal, letöltve: 21-Már-2012.)

[HeSy09] Hendricks, Vincent - Symons, John: Epistemic Logic, *The Stanford Encyclopedia of Philosophy (Spring 2009 Edition)*, Edward N. Zalta (ed.), (http://plato.stanford.edu/archives/spr2009/entries/logic-epistemic, letöltve: 21-Már-2012.)

[Lo08] Lokhorst, Gert - Jan: Mally's Deontic Logic, *The Stanford Encyclopedia of Philosophy (Winter 2008 Edition)*, Edward N. Zalta (ed.), (http://plato.stanford.edu/archives/win2008/entries/mally-deontic, letöltve: 21-Már-2012.)

[Na10] McNamara, Paul: Deontic Logic, *The Stanford Encyclopedia of Philosophy (Fall 2010 Edition)*, Edward N. Zalta (ed.), (http://plato.stanford.edu/archives/fall2010/entries/logic-deontic, letöltve: 21-Már-2012.)

[Ru84] Ruzsa Imre: Klasszikus, modális és intenzionális logika
Akadémiai Kiadó, 1984.

Jogi logika, jogi alkalmazások

[BF04] Bognár László-Forrai Gábor: Esszéírás és Informális Logika
Miskolci Egyetem, Bölcsészettudomány Kar, 2004. elektronikus jegyzet
(http://www.uni-miskolc.hu/~bolantro/informalis/, letöltve: 20-Jan-2013)

[CEN08] Alexander Boer-Erik Hupkes-Fabio Vitali-Monica Palmirani-Balázs Rátai: CEN Metalex Workshop Proposal

Metalex/CEN projekt, 2008.
(http://www.metalex.eu, letöltve 22-Ápr-2012)

[EST07.1] Alexander Boer et al. (2007): Specification of the Legal Knowledge Interchange Format: ESTRELLA Project Deliverable 1.1, University of Amsterdam.
(www.estrellaproject.org, letöltve: 21-Dec-2011)

[EST07-2] ESTRELLA (2007): OWL Ontology of Basic Legal Concepts: ESTRELLA Project Deliverable 1.4, University of Amsterdam.
(www.estrellaproject.org, letöltve: 21-Dec-2011)

[EST08] Joost Breuker, Saskia van de Ven eds. Abdallah El Ali et al.: Developing HARNESS. Towards a hybrid architecture for LKIF ESTRELLA Project Deliverable 4.6, University of Amsterdam
(www.estrellaproject.org, letöltve: 20-Jan-2013)

[Fö04] Förhécz András: Ontológia kezelő modul tervezése szöveges információt kezelő informatikai rendszer számára
BMGE VIK Diplomaterv, 2004.
(http://home.mit.bme.hu/~fandrew/diplomaterv_hu.html, letöltve: 21-Aug-2012.)

[FöSt09] Förhécz András és Strausz György. *Legal Assessment Using Conjunctive Queries*. Third Workshop on Legal Ontologies and Artificial Intelligence Techniques, Barcelona, 2009. június

[Ga08] A. Gangemi: Design patterns for legal ontology construction Laboratory for Applied Ontology, ISTC-CNR, Roma, Italy

[Ku-03] Kutrovátz Gábor: Bevezetés a logikába és az érveléselméletbe (Nemhivatalos jegyzet)
2003. ELTE Tudománytörténet és Tudományfilozófia Tanszék
(http://hps.elte.hu/~kutrovatz/logjegyz.htm letöltve: 13-Ápr-2012.)

[LMPV07] A. Lenci-S. Montemagni-V. Pirrelli-G. Venturi: NLP-based ontology learning from legal texts. A case study.
Department of Linguistic, University of Pisa, Italy, 2007.

[PaT09] Papp T.: Jogi szakértő rendszer fejlesztése. (önálló laboratóriumi beszámoló, konzulens Dr. Strausz György)
BMGE-VIK, Budapest 2009.

[Pérez11] Gonzalo Pérez: Automated Reasoning on Traffic Rules
PTE-TTK, Erasmus Dolgozat, Pécs 2011.

[RH09] Rinke Hoekstra. Ontology Representation – Design Patterns and Ontologies that Make Sense, volume 197 of Frontiers of Artificial Intelligence and Applications. IOS Press, Amsterdam, June 2009.

[Rub07] Rossella Rubino, Antonino Rotolo, Giovanni Sartor: An OWL Ontology of Norms and Normative Judgements In: Carlo Biagioli, Enrico Francesconi, Giovanni Sartor eds. *Proceedings of the V. Legislative XML Workshop* pp. 173-187. European Press Academic Publishing.

[SoKo96] Solt Kornél: Jogi logika I-II.
MTA Állam és Jogtudományi Intézete, SENECA kiadó, 1996.

Önhivatkozások

[Kil07] Kilián Imre: Mixed strategy reasoning
An approach for resolution based verification of OCL constraints in UML models
Pollack Periodica, Volume 2 Supplement Akadémiai Kiadó, Budapest 2007.

[Kil08] Kilián Imre: Modellvezérelt szoftverek készítése I-II
Alkalmazott Matematikai Lapok MTA, Budapest 2008.

[AGKI10] Alberti, Gábor - Kilián Imre (2010): Vonzatkeretlisták helyett polaritásos hatáslánc-családok – avagy a ℜeALIS σ függvénye, In: Tanács Attila-Vincze Veronika szerk. *VII. Magyar Számítógépes Nyelvészeti Konferencia*, University of Szeged, Informatics Department Group, Szeged. 113-127.

[Kil11.1] Kilián Imre: Contralog: egy előre haladó Prolog motor és alkalmazása ℜeALIS nyelvi elemzésre
Erdélyi Magyar Műszaki Tudományos Társaság, SzámOkt 2011. konferencia kiadványa,
Kolozsvár, 2011, pp. 199-205.

[Kil11.2] Kilián Imre: Tárgymodell változatok a ℜeALIS nyelvi elemzéshez
VIII. Magyar Számítógépes Nyelvészeti Konferencia, Tanács Attila-Vincze Veronika szerk.
Szegedi Tudományegyetem, Informatikai Tanszékcsoport, Szeged, 2011.

[Kil12.1] Kilián Imre: Semantic projection of verbal argument structures In: Kleiber Judit szerk. *Vonzásban és változásban*, Pécsi Egyetem, Bölcsészettudományi Kar, Általános Nyelvészeti Tanszék, Pécs 2012.