Következtető rendszerek részletes vizsgálata

MEO Projekt

Scriptum Informatika Rt.

2005. szeptember 15.

I Bevezetés

Jelen dolgozatban áttekintést adunk a következtető rendszerekről. Mindenekelőtt azonban ismertetjük terminológiánkat, mivel a különböző forrásokban eltérő az *ontológia*, *ontológia nyelv* és maga a *következtető rendszer* (vagy *gép*) értelmezése.

Ontológia nyelvnek nevezünk bármely olyan formalizmust, mely a megismerhető világról alkotott képünk – azaz *tények* és általános *összefüggések* – leírására alkalmas. Tehát egy ilyen formalizmus egyaránt magába foglalja a *szintaxist*, mely meghatározza, hogy leírásunk milyen alapelemekből épülhet fel és ezeket hogyan kapcsolhatjuk össze; valamint a *szemantikát*, mely pedig megadja, hogy az egyes szintaktikai elemeket és konnektívákat hogyan értelmezzük. Ilyen ontológia nyelv pl. az OWL[1].
Ontológiának nevezünk egy adott ontológia nyelven megfogalmazott leírást. Erre példa a Wine Ontology.
Következtető rendszernek nevezünk egy olyan programot, mely képes beolvasni valamely – általában saját – ontológia nyelven leírt ontológiát, ezek után pedig képes a felhasználó által feltett kérdésekre reagálni. Alapesetben egy kérdés formája egy újabb, az ontológia nyelven megfogalmazott állítás, melynek igazságtartalmáról érdeklődünk. A „reakció”, melyet a gép adhat, ebben az esetben az „igaz”, „hamis”, és „nem tudom” válaszok valamelyike lehet; ez a válasz kiegészülhet azt alátámasztó indoklással. A következtető rendszerek egy speciális csoportját alkotják a **tételbizonyítók**, melyek esetében a feltehető kérdések körét csak az eldöntendő kérdések adják.

Tehát a következtető rendszer használhatóságát felülről korlátozza a használt ontológia nyelv *kifejezőereje*, azaz hogy a nyelven mennyire komplex összefüggések írhatóak le. Azonban minél nagyobb egy nyelv kifejezőereje, annál szűkebb az olyan kérdések köre, melyre (matematikailag) egyáltalán lehetséges a válaszadás. Így találni kell egy egészséges egyensúlyt, melynél már kellően bonyolult összefüggéseket le tudunk írni, ugyanakkor a kérdések megválaszolhatósága még nem reménytelen.

Természetesen – mivel egy-egy ontológia nyelv esetében különböző következtetési mechanizmusok létezhetnek, melyeket egy adott következtető rendszer alkalmazhat – a használhatóság a megvalósított algoritmusok megválasztásától és optimalizált implementációjától is nagy mértékben függ.

Feladatunk az volt, hogy egy egységes vizsgálati szempontrendszert állítsunk fel, mellyel a jelenleg ismert következtető rendszerek képességeit objektíven össze tudjuk mérni. Megjegyezzük, hogy a fenti két tényező különbözik a vizsgálati módszer lehetőségeit tekintve: az ontológia nyelv kifejezőerejének elemzése egy szigorúan *elméleti* meggondolást igénylő folyamat, míg annak „mérése”, hogy a kiválasztott algoritmusok az implementált optimalizálásokat kihasználva mennyire szerencsésen/hatékonyan képesek együttműködni, egy *gyakorlati*, tesztek alkalmazását kívánó kérdéskör.

[1] Vagy más ontológianyelv. Lásd korábbi tanulmányunkat az ontológia nyelvekről.

II A vizsgálat módszertana

Mivel a kifejezőerőt célzó kérdések megválaszolhatóak szigorúan elméleti megfontolások alkalmazásával, így az ezekre adott válaszok megbízhatóak. Ennek tükrében a vizsgált következtető rendszerek mindegyikére elvégeztük ezen vizsgálatokat, mielőtt bármelyiküket is teszteltük volna a gyakorlatban.

Tekintve, hogy az összes vizsgált (nem fuzzy) rendszer nyelve beágyazható az (egyenlőséges) elsőrendű logikába (esetleg egy minimális aritmetikai támogatással), így megpróbáltuk megragadni a következtető motorok nyelvének kifejezőerejét azzal, hogy mekkora részét támogatják az elsőrendű logikának. A tapasztalatok alapján a következő csoportokat sikerült elkülöníteni az engedélyezett formulák szintaxisa alapján:

- azon rendszerek, amik csak Horn-szabályokat támogatnak;
- azon rendszerek, amiknél az implikációk törzsében engedélyezett a negálás;
- azon rendszerek, amik tetszőleges elsőrendű formulát képesek feldolgozni.

Ez a három csoport kifejezőerő szempontjából ilyen sorrendben válik egyre erősebbé.

Továbbá, ettől függetlenül egy rendszer az alábbiakat támogathatja:

Tulajdonság	**Példa**
egyenlőség	2 = succ(1)[2]
függvényszimbólumok	succ(1)
egzisztenciális kvantálás	∃x: Ember(x)
(egész / valós) aritmetika	∀x: Szám(x) → (méter(x)=milliméter(1000*x))
fuzzy értékek	„az asztal 0.2 mértékben poros”
sorted univerzum	van Ember típus, Üveg típus, ...
valószínűségek	„az asztalok 20%-a okkersárga”

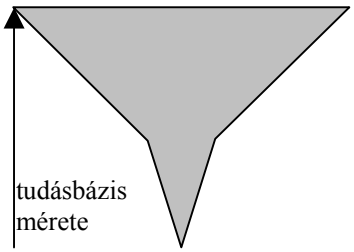
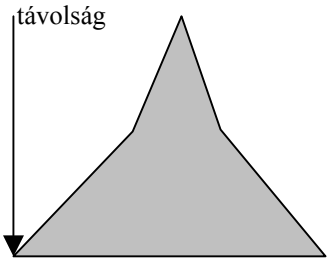
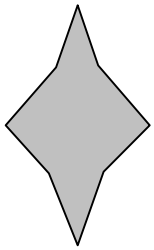
Megjegyezzük, hogy ezen tulajdonságok nem teljesen függetlenek; pl. az egzisztenciális kvantálás „kiváltható” függvényszimbólumokkal – igaz ugyan, hogy ezek bevitele sokkal kényelmetlenebb ekkor.

A következtető rendszerbe a szabályok bevitele csak az első lépés; az adatokkal ellátott rendszertől kérdezni is kell valamilyen lekérdező nyelven (ez a nyelv általában kisebb módosításokkal megegyezik az ontológia nyelvvel – a kettő megkülönböztetése azért lehet fontos, hogy pl. automatikus feldolgozás esetén a rendszer szét tudja választani a kérdéseket és a kijelentéseket). A lekérdező nyelvet az alábbi szempontok szerint osztályozhatjuk:

- Fel lehet-e tenni több független kérdést is egymás után? (Természetesen anélkül, hogy újra betöltenénk a tudásbázist.)
- Csak eldöntendő kérdéseket támogat vagy olyat is, amire egy objektum, esetleg azok valamely halmaza a válasz (elsőrendű logikai megfogalmazásban ground termek)?
- Feltehetőek-e metakérdések (pl. „Mely tulajdonságok különböztetik meg az embert az asztaltól”)?

[2] A succ a rákövetkezés függvényt jelöli. (És persze az 1 után a kettő jön a természetes számok körében. Minkét jel, az 1 és a 2 is konstansok, csak jelölések, nincs különösebb jelentésük.)

A következtetés során a rendszer a bevitt szabályok és tények alapján igyekszik megválaszolni a kérdést. A keresés iránya lehet *előrekövetkeztető* – ekkor a meglévő adatokból a szabályok alapján újabb és újabb tényeket állít elő, míg elő nem áll a válasz, vagy elfogynak az alkalmazható szabályok; *hátrakövetkeztető* – ekkor pedig a kérdésből mint hipotézisből indulva a szabályok és adatok segítségével megpróbálja igazolni azt; vagy *vegyes* rendszerű, amely e fenti két stratégia valamilyen heurisztikával segített kombinációja.

Előre	Hátra	Vegyes
		
Elképzelhető, hogy egy új szabály esetén is jelentős mennyiségű következtetést kell végeznünk, melyek eredményére valószínű, hogy nem leszünk később kíváncsiak.	Lekérdezésnél minél távolabb voltunk eredetileg a válaszhoz szükséges információktól, annál több részcél, következtetés szükséges (és egyúttal annál több érdektelen részkérdést válaszolunk meg).	Ha vegyesen alkalmazzuk az előre és a hátrafelé következtetés szabályait, úgy jóval kevesebb következtetés elegendő lehet a megválaszoláshoz. Persze ezt nem árt, ha az ontológia leírása támogatja (különben esetleg sokkal többet számol).

Ezen típusokat is tovább lehet finomítani az alábbiak szerint:
Előrekövetkeztető, csak implikációt támogató rendszerek esetében fontos attribútum, hogy a rendszer használja-e a Rete algoritmus valamely válfaját.
Hátrakövetkeztető rendszer esetén pedig, hogy valamely (amennyire lehet, optimalizált) tabló módszerrel dolgozik-e a rendszer.
Amennyiben a program megengedi az elsőrendű logikában ismert összes konstruktumot, és futása *vegyes* vagy *hátrakövetkeztetés* alapú, úgy a rezolúció támogatása a kérdés.

Felmerülnek olyan szempontok is, melyek sem a motor sebességét, sem kifejezőerejét nem érintik, ám használhatóságát befolyásolják. Ilyen kérdések a következők:

- Lehet-e fejleszteni a következtető rendszeren („finomra hangolni” a kívánt alkalmazási területhez), és ha igen, mennyire egyszerű ennek folyamata?
- Automatizálható-e tudásbázisának (időszakos vagy eseti) frissítése?
- Támogatja-e valamely szabvány adatcsere-formátumot?
- Létezik-e hozzá szerkesztőfelület, és ha igen, az mennyire felhasználóbarát?

A kapott eredménnyel kapcsolatosan a következő kérdések merülnek fel:

- Csak a választ adja vissza a kérdésre vagy az ahhoz vezető utat (azaz indoklást) is?
- Képes-e felismerni (és tudatni a felhasználóval), ha a válaszhoz szükséges összefüggéseknek nem mindegyike áll rendelkezésére?
- Ha igen, képes-e olyan kérdést feltenni, mely közelebb viszi a válaszhoz?

Amikor csak lehetett, megválaszoltuk a fentiekben felvázolt kérdéseket. Azon alternatívákat vizsgáltuk alaposabban, melyek ígéretesnek tűntek, a többi esetében a nemtriviális kérdések eldöntésétől eltekintettünk. Az alábbi „szűrőfeltételeket” állítottuk fel, melyek nem teljesülése esetén az adott motorral nem foglalkoztunk tovább részletesebben:

- Horn-formulákénál nagyobb legyen a kifejezőereje. Áttekintve az általunk választott problémacsoportokat, úgy találtuk, hogy ezek legtöbbjét nem lehet megfogalmazni ezen megszorítás mellett.
- Be lehessen vinni neki az (automatikusan generált) teszteket alkalmas input file-ból – az alkalmazott tudásbázist is ilyen formában fogja kapni.
- Megismerését segítse elegendő dokumentáció.

Természetesen néhány olyan tulajdonságot is feljegyeztünk, mely magát a hatékonyságot és kifejezőerőt explicite nem befolyásolja, ám mégis fontos – ilyen például a licensz, a megvalósítás és a bevitel konkrét nyelve, a fejlesztő cég.
Azon rendszerekre, amelyek ezen attribútumaik alapján versenyképesnek bizonyultak, konkrét teszteket készítettünk. Ezek a tesztek szolgáltak a bevezetőben említett, gyakorlati oldalról vizsgálható tulajdonságok összemérésére. Az alapos vizsgálat nagyszámú próba-probléma bevitelét igényli, így egy-egy rendszer esetében a procedúra több munkanapot vesz igénybe. Az így kétfázísúvá alakult módszertan további előnye abban mutatkozik meg, hogy mivel a gépek nagy részét az első fázisban ki tudtuk zárni, így a második lépésben *jelentősen rövidebb* idő alatt alaposabb vizsgálatokat tudtunk végrehajtani.

A gyakorlati összevetés céljára kifejlesztettünk egy olyan Java alapú frameworköt, mely képes

- tetszőleges elsőrendű egyenlőséges logikai formula eltárolására a `tények`, `szabályok` és `kérdések` listák valamelyikében;
- egy – elég általános – konfigurációs állomány szerinti formátumban kiírni e három lista által reprezentált tudásbázist és a felteendő kérdést (praktikusan egy motorhoz egy konfigurációs állomány tartozik).

Így biztosítani tudtuk, hogy mindegyik rendszer garantáltan ugyanazokat a problémákat kapja meg a saját nyelvére lefordítva.
Továbbá generáltunk teszteket is, melyeket e framework segítségével minden, az első rostán átmenő motornak be tudtunk adni. A tesztek teljes leírása – és hogy miért kerültek be – megtalálható a mellékletben.
Hasonló céllal működik a TPTP Problem Library, mely sok és sokféle problémát rendszerezett. A céljainknak azonban nem felelt meg maradéktalanul, mivel nem ismert minden kezdetben érdekesnek tűnő nyelvet, továbbá szerettünk volna általunk generált teszteket készíteni.

III A lényeges és érdemes eszközök ismertető leírása

Ebben a fejezetben ismertetjük azokat a motorokat, melyek átmentek az első szűrőn és lefuttattuk őket a tesztállományokon. Itt nem térünk ki tulajdonságaik ismertetésére, többnyire a működésük közben szerzett tapasztalatokat foglaljuk össze. Minden terméket ugyanazon a szerveren futtattunk, így az eredmények objektíven összemérhetőek.
Tesztkonfiguráció:
CPU: Intel(R) Pentium(R) 4 CPU 2.80GHz, 1MB cache
Memória: 1,5 GB
OS: RedHat Linux - 2.4.21-4.EL
A C-ben készült, forráskód alakban terjesztett programokat gcc-vel (3.2.3) fordítottuk.

Tesztkörülmények
Minden tesztnél 5 percet adtunk a következtető rendszernek a feladat megoldására. Egyedül az EPILOG esetében nem volt ez megoldható, mivel ott nem tudtunk ilyen feltételt szabni, és sajnos nem sikerült automatikusan sem tesztelnünk.
A teszteléshez használt paramétereket a rendszerek ismertetőjének végén mutatjuk be.
A kérdéseket –amennyiben klóz formába kellett alakítanunk– egy ha-akkor szabállyá alakítottuk és csak az így átalakított kérdést tettük fel, mivel nem voltunk biztosak abban, hogy minden rendszer azonosan kezelte volna a formulákat, illetve az esetleges negálást nem oldhattuk volna meg a végső generálás előtt.

1 Carine (v 0.72)

A teljesítménye jelentősen elmaradt a jó következtetőkétől. Nem támogatja az egyenlőséget. Ugyan az egész elsőrendű logikát támogatja, használatát kényelmetlenné teszi, hogy Skolem-normálformában várja az inputot (bár ehhez léteznek segédeszközök, az adatbázisba bevitt információk ismételt szerkesztése körülményes lehet). Output formátuma mind ember, mind gép számára viszonylag könnyen értelmezhető (amennyiben az ember ért a rezolúcióhoz).
Dokumentációja minimális.
Saját tesztjeinket CNF-ként adtuk át (csak ezt támogatja). TPTP-vel nem teszteltük, mivel azok többsége tartalmazott egyenlőséget.
Paraméterek: `t=300`

2 E (v 0.82)

Gyors, hatékony következtető rendszer. Beviteli szintaxisa a Prolog nyelvére emlékeztet, de mivel támogatja az atomi negálást és a fej oldali diszjunkciót, így annál nagyobb a kifejezőereje (hiszen ez elég az elsőrendű logika egészének megragadására). Támogatja és hatékonyan kezeli az egyenlőséget. Bizonyítási módszere helyes és teljes. Nem túl jól dokumentált; outputja ugyan tartalmaz bizonyítást is, ám nem túl részleteset.
Több rendszer is használja következtetőként, például régebben az E-SETHEO.
Mindkét teszttípusban a CNF-re hozott formát kapta meg.
Megjegyzés: Legújabb (v 0.9) verziója már megengedi az elsőrendű formulák bevitelét is, mely a TPTP-s tesztek szintaxisához hasonló. (A vizsgált változat azonban CNF-et használt.)
Négy TPTP-s teszt esetében hibázott.
Paraméterek: `-xAuto -tAuto --memory-limit=300 -R --cpu-limit=300`

3 EPILOG (v EPILOG-2005-JUN-22)

Egy saját ontológia nyelvvel rendelkező következtető rendszer, mely a sok segéd-eljárás/specialista, valamint a nagyobb kifejezőereje miatt igen lassú lett. Megfelelően megfogalmazott input esetén azonban elképzelhető, hogy versenyképes lehet bizonyos problémák megoldásában. A vizsgált feladatok esetében azonban alig tudott eredményt szolgáltatni.
Következtetési rendszere leginkább a tabló módszerre hasonlít, ám rengeteg szabályt alkalmaz, sokféle heurisztikával. Rendelkezik továbbá többféle specialistával is, melyek kiegészíthetőek sajátokkal. Ezekkel valószínű, hogy bizonyos esetekben hatékonyabb lehet más rendszereknél.
Csak az Allegro Common Lisp segítségével sikerült futtatnunk. (Annak is csak a kipróbálásra szánt változatával próbáltuk.)
(A vizsgálathoz használt LISP implementáció: Trial Edition of Allegro CL 6.2 for Linux or FreeBSD)

Mivel elég körülményes volt tesztelni (másként működött, ha a standard bemenet át volt irányítva, nem úgy mint amikor kézzel írtuk be az utasításokat, kérdéseket), ezért csak a saját tesztjeinket végeztük el rajta, azok közül is kihagytuk az OWL-es tesztek közül a kisebbeket.
Minden tesztet elsőrendű logikai szintaxissal írtunk le.
Jól, részletesen dokumentált.
Megjegyzés: Létezik egy azonos nevű tételbizonyító is, ám azt már nem sikerült meglelnünk a logic.stanford.edu oldalon.
Paraméterek: `(tweak '*qa-iterations* 1000)`

4 Gandalf (2.6 r1)

Egész jól teljesített a tesztekben. Ennek is Skolem-normálformára hozva kell megfogalmazni a bemenetet. Kimenete ember számára nem igazán, gép számára megfelelően értelmezhető.
Dokumentációja minimális.
Minden próbát CNF-re hozva kapott.
A TPTP-s tesztekben –a vizsgált rendszerek közül– ennek az eredményét kellett legtöbbször hibásnak tekintenünk. Ez valószínű, hogy más beállításokkal lehetne csökkenteni, mivel minden olyan teszt, melyet nem vett sikeresen `Theorem` jellegű volt. Némely esetben az egyenlőség reflexivitását állítva a helyes megoldást szolgáltatta.
Paraméterek: `-proof`

5 JTP

A sebessége hasonló, mint a Carine-é, annál valamivel lassabb; legfőbb hiányossága az egyenlőség hiánya. Azonban a fejlesztők elkövettek egy olyan hibát, ami miatt a rendszer nem veszi észre, ha egy frissen készített klóz már le lett vezetve korábban; emiatt a következtetés mélységének függvényében a szükséges idő mindenképp exponenciális lesz. Tapasztalatunk szerint a 8-10 mélységű következtetések ideje már elfogadhatatlan. Ugyan támogatja a halmazlekérdezéseket, a fent említett fejlesztői hiba miatt ugyanazt a megoldást nagyon sokszor előhozza.
Vizsgált változat: Nem számozzák a verziókat, amit használtunk, az 2003. december 20-i.
A teszteket elsőrendű formában kapta meg, TPTP-s próbáknak már nem vetettük alá, mivel a Carine-hoz hasonlóan ez sem támogat egyenlőséget.
Dokumentációja minimális.
Megjegyzés: A rendszerhez kellett írnunk egy kiegészítő osztályt, mely 5 perc futás után megszakítja a kérdéssel kapcsolatos következtetéseket.
Paraméterek: `maxDepth=10`

6 Otter (v 3.3f)

Igen hatékony következtető eszköz. Több rendszer használta fel és fejlesztette tovább az ötleteit. Kiszámítható függvényekkel, predikátumokkal, egész aritmetikával lehet hatékonyabbá tenni a következtetését. A hozzácsomagolt MACE (v 2.0) véges modell keresésére képes, így párhuzamosan futtatva a két programot esetleg hamarabb befejeződhet a keresés, amennyiben kielégíthető a formula-halmaz.
Valószínű, hogy vannak még tartalékai, mivel eléggé sok beállítási lehetősége van. (Képes interaktív módban is dolgozni, ekkor a felhasználó további segítségekkel láthatja el a rendszert.)
A bizonyításból kiolvasható egy válasz is a felsorolást kívánó kérdésre. (Amennyiben helyesen fogalmazta meg kérdését a felhasználó.)
Az Otter támogatja az aritmetikai műveleteket is, így számításokkal kombinált logikai műveletek elvégézésére különösen alkalmas lehet.

Jól, részletesen dokumentált.
Minden teszt elvégzésekor elsőrendű formában kapta az inputot.
A TPTP-s tesztek esetében vétett néhány hibát. Ennél a következtetőnél is –hasonlóan a Gandalfhoz– az egyenlőség reflexivitásának állításával több esetben helyes megoldást szolgáltat.
Paraméterek:
```
set(input_sos_first).
assign(max_mem, 102400).
assign(max_seconds, 300).
```

7 SNePS, SNePSLOG (v 2.6.1)

A SNePS-re épülő nyelv a SNePSLOG, mellyel elsőrendű logikai állításokat formalizálhatunk. Sajnos a lekérdezéshez használható nyelv, valamint a SNePS maga kissé hiányos. (Például nem tudjuk megkérdezni, hogy egy univerzálisan kvantált kérdés igaz-e, valamint a következtetés során a fejben lévő diszjunkciót csak egyszer alkalmazza.)
A SNePS önmagában elég nehezen kezelhető, célszerű a SNePSLOG-ot használni helyette (mi ezt tettük).
Többféle LISP implementációval is sikerült működésre bírnunk.
(A vizsgálathoz használt LISP implementáció: Trial Edition of Allegro CL 6.2 for Linux or FreeBSD)
A 3.0-s változata kissé hiányos dokumentációjú, azonban a vizsgált, 2.6.1-es jól dokumentált.
Csak saját tesztjeinket futtattuk le, melyeket elsőrendű formában kapott meg.
Az alapértelmezett beállításokat használtuk.

8 SPASS (v 2.1)

Létezik hozzá grafikus szerkesztő, ám sok segítséget nem ad a formulák szerkesztéséhez. A következtető rendszere teljes, helyes. Nagyobb méretű tudásbázisok esetén jelentősen lassul a következtetés sebessége; az OWL tesztek esetében már perces nagyságrendűek voltak a legjobb mért idők is. (Ellenben a TPTP-s tesztek esetében néha meglepően jól teljesített.)
Egy érdekes eredményt lehet megfigyelni a saját tesztjeink között. A steamroller esetében kiugróan rosszul teljesített. Ez annak köszönhető, hogy a kizárást és a steamrollert egyszerre teszteltük, és valamiért a kizárás így hatott rá. A szabályait elkezdte alkalmazni és igen hosszú idő után derült ki, hogy így is igaz. A kizárás nélkül futtatott steamroller igen gyorsan végzett. (Emiatt úgy érezzük, hogy egy vegyes, sokszínű ontológiában nem biztos, hogy megfelelő lenne következtető rendszernek.)
Készítettek az alkotók webes és windows-os interface-t is hozzá. Létezik modális logikát támogató kiterjesztése is, az MSPASS.
Jól, részletesen dokumentált.
A saját tesztjeinket konjunktív normál formában kapta, ám képes elsőrendű formulákkal is dolgozni és a TPTP-s tesztekett ebben a formában adtuk át. (Több összehasonlítás a SPASS CNF-re alakító részét – a FLOTTER-t – tartja a legjobbnak. Amennyiben a TPTP-s teszteket vizsgáljuk úgy tűnik, hogy igazuk lehet. Több tekintetben sikeresebb volt, mint bármely rendszer. A saját tesztjeink és a TPTP-s tesztek kontrasztja alapján ezt valószínűleg a hatékony CNF-re alakításnak köszönhető.)
Paraméterek: `-TimeLimit=300 -PProblem=0`

IV Összefüggések feltárása

Azok a rendszerek, melyeknek van fuzzy kiterjesztése, általában nem tartoznak a kellően kifejező nyelvek közé.
A fuzzy rendszerekhez gyakran mellékelnek szerkesztőeszközt is, mivel a szabályok megfogalmazása általában ezekben a legbonyolultabb.
Az OWL tesztek jelentős részében az egyenlőség ismerete szükséges – ez feltehetően az OWL nyílt világszemléletének tesztelésére szolgál. Azok a motorok, amik nem támogatják ezt, legfeljebb a tesztek felén képesek átmenni. (Kivéve, ha a megfelelő szabályokkal kiegészítjük az egyenlőségre vonatkozóan.)
Az elsőrendű formulák CNF-re alakítása egy kritikus pont lehet a következtetések esetében.
A tételbizonyítók hatékonyabbnak bizonyultak az egyenlőséges elsőrendű formulákkal következtetésben, mint azok az ontológianyelvek, melyek szintén támogatják azt.

V Következtető motor elméleti tapasztalatok

Ebben a fejezetben következtető motorok implementációjával kapcsolatos technikákat, elméleteket írunk le, mely leginkább a rezolúcióval és a szuperpozícióval kapcsolatos megvalósításhoz kapcsolódik.

- A klózbázisban érdemes lehet **megkülönböztetni** az előre-, illetve a hátrakövetkeztetés során kapott klózokat; többek között azért, mert az előrekövetkeztetett klózokat el is tárolhatjuk permanensen a tudásbázisban.
- A bizonyítás megjelenítése érdekében egy klózról nyilván kellene tartani annak **eredetét**; ez egy eleve meglevő klóz esetében a formula, melyből származik, levezetett klóz esetében pedig az apa-klózok és a generálás típusa.
- A rezolúció végrehajtása során vannak „érdekes” és „nem érdekes” klózok is. Nem érdekes klózok például azok, melyeknél van szűkebb. A „nem érdekes” klózokat megtartani ugyan meg kell (a későbbi visszakeresések miatt), de lehet használni **két listát**, és a rezolúció lépéseit csak az érdekes klózok listáján belül végrehajtani.
- A Unit Resolution támogatására érdemes lehet a **unit klózokat külön listába** gyűjteni. Ez valamelyest emlékeztet az Otter rendszerbe integrált „hot list strategy”-re.
- Az „érdekes” klózok halmazát úgy célszerű felépíteni, hogy gyorsan lehessen olyan párokat generálni belőle, amik talán rezolválhatók. Így például **minden predikátumra egy külön listába** gyűjteni azokat a klózokat, melyekben a predikátum szerepel (a negáltjukra is egy másik listát használhatunk); ekkor egy rezolúciós lépésben egy $+p$ és egy $-p$-listán kell csak végigmenni és rezolválni.
- Valamilyen módszerrel el kellene érni, hogy két klózt **ne próbáljunk egynél többször** megpróbálni rezolválni, ugyanakkor gyorsan el lehessen dönteni, hogy volt-e már ilyen próbálkozás. Erre egy megoldás lehet az, hogy minden klóznak elkészítése idejében adunk egy sorszámot, és mindegyiküknél megjegyezzük, hogy melyik az a sorszám, ami alatt minden másik klózzal rezolválva volt.
- A levezetett **egyenlőségek** számára ismét csak célszerűnek látszik egy **külön lista** menedzselése – amennyiben nem alakítjuk át olyan formára a tudásbázist, mely csak egyenlőségeket alkalmaz.

VI Értékelés, ajánlás

Az általános célú ontológia nyelvekhez kapcsolódó következtető rendszerek (EPILOG, SNePS) kevésbé voltak sikeresek a válasz megkeresésében. Azonban az EPILOG több kiegészítő funkcióval rendelkezik, magasabb kifejezőerővel bír, mint a többi nyelv. Ezt a rendszert nem biztos, hogy objektíven ítélhetjük meg elsőrendű logikai próbáink alapján (különösen, ha természetes nyelv feldolgozás az ontológia használat célja).

A tételbizonyítók egész jól szerepeltek, sokféle tesztet vettek sikeresen, viszonylag alacsony időkkel. Kiemelkedő volt a vizsgált rendszerek közül az E teljesítménye az OWL-es tesztek esetében. Többnyire ez volt a leggyorsabb. Azonban a SPASS CNF-re alakító részét – a FLOTTER-t – is gyakran dicsérték. Úgy tűnik, hogy mind az általunk készített, mind a többi CNF-re alakító rendszernél jobban teljesít. Érdemes lehet az E-t választani a SPASS FLOTTER részével együtt, bár valószínű, hogy az E is kellően hatékonnyal rendelkezik.

A végső sorrend a következőnek tűnik: E, SPASS, Gandalf, Otter, JTP, Carine, SNePS. Ez a sorrend nem teljesen valós, az első négy hasonló sebességűnek tűnik, bár az Otter kissé lemarad az első háromtól. Az EPILOG nem igazán azonos célú, az támogat modális kijelentéseket, valószínűségeket, időt, valamint a tudás egy jelentős részét hatékonyabban kezeli, ha a specialistái kapják meg, nem pedig elsőrendű formában kapja meg. Az utolsó három pedig az egyenlőség hiánya miatt nem ajánlott (persze ahol nincs szükség egyenlőségre, ott elegendő lehet a Carine, a másik kettő kevésbé megbízható, ám ekkor is érdemes lehet egyenlőséget támogató rendszert használni, mivel a Carine dokumentációja igen szegényes).

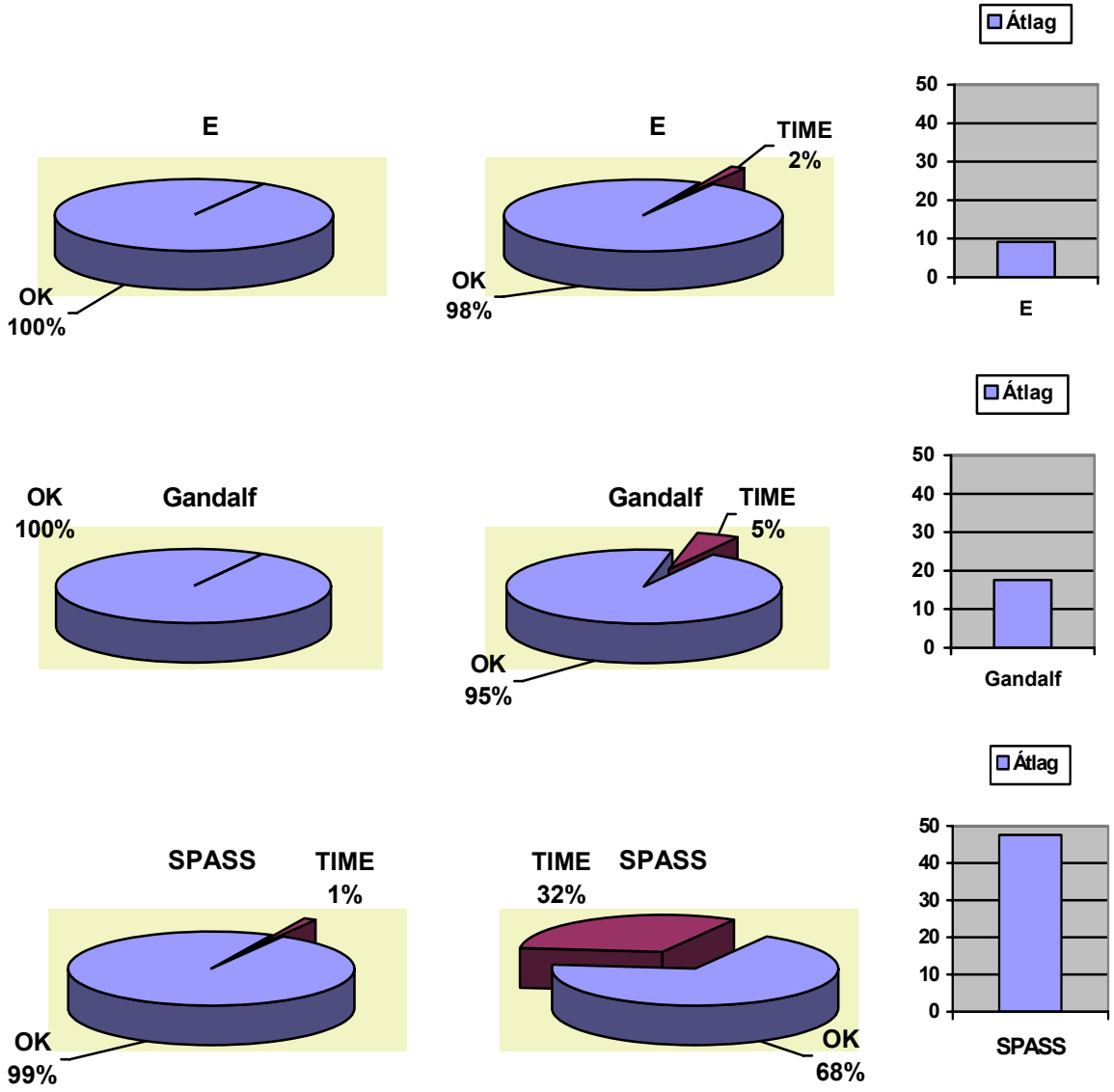
Néhány további (érdekesnek tűnő) következtető rendszert kihagytunk az összehasonlításból az idő hiánya, kevés rendelkezésre álló információ, vagy nagy bonyolultságuk miatt. Ezek a következők voltak: ACL-2 (ez valószínű, hogy nem felelne meg), DCTP, **E-SETHEO**, **HOL**, Isabelle, PTTP, **Paradox**, Vampire, **Waldmeister**.

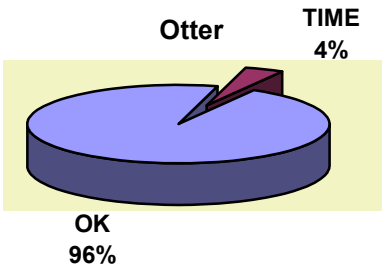
VII Melléklet

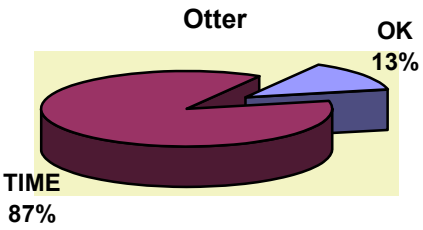
Itt egyrészt összefoglaljuk az általunk használt fogalmakat, helyenként technikai részletekbe menően, másrészt ismertetjük tesztjeinket, ábrázolunk néhány eredményt.

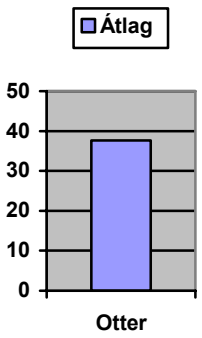
1 Következtető rendszer eredmények

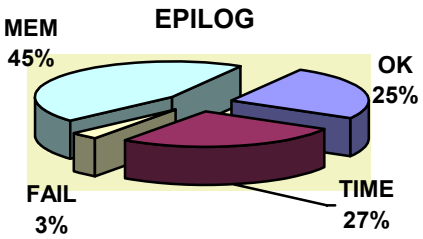
Az első kördiagram a kisméretű teszteken mért eredményeket ábrázolja; a második a nagyobb méretűeken (ezek sok fölösleges információt tartalmaznak) mérteket. A harmadik oszlopban a helyesen megoldott nagyméretű állományokon mért futási idők átlagát ábrázoltuk.

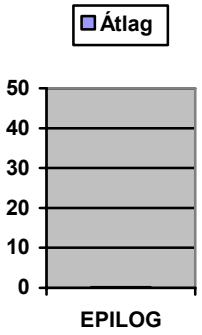


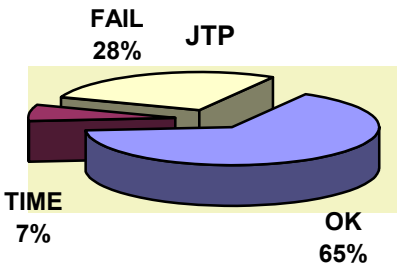

Otter
TIME
4%
OK
96%

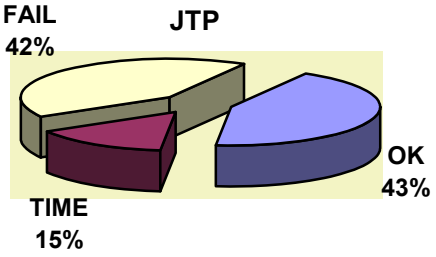

Otter
OK
13%
TIME
87%

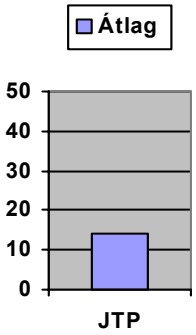

Átlag
50
40
30
20
10
0
Otter

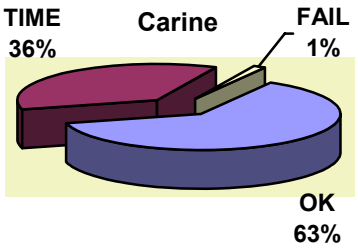

MEM
45%
EPILOG
OK
25%
FAIL
3%
TIME
27%

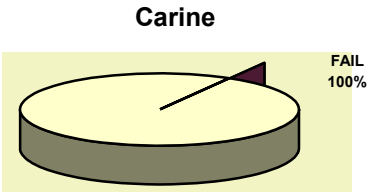

Átlag
50
40
30
20
10
0
EPILOG

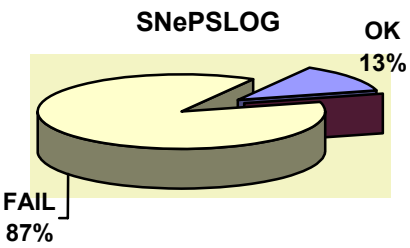

FAIL
28%
JTP
TIME
7%
OK
65%

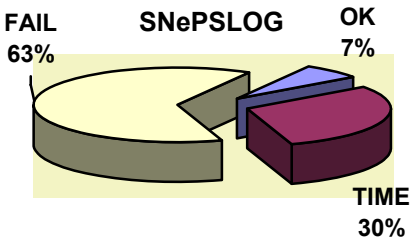

FAIL
42%
JTP
OK
43%
TIME
15%

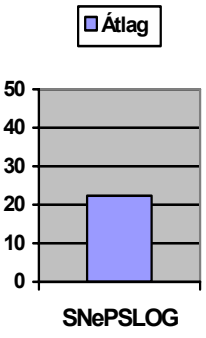

Átlag
50
40
30
20
10
0
JTP


TIME
36%
Carine
FAIL
1%
OK
63%

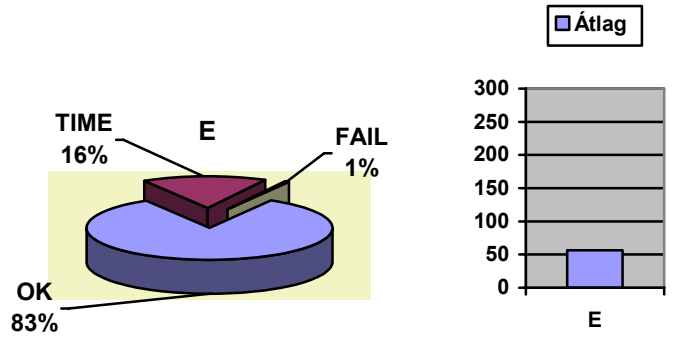

Carine
FAIL
100%

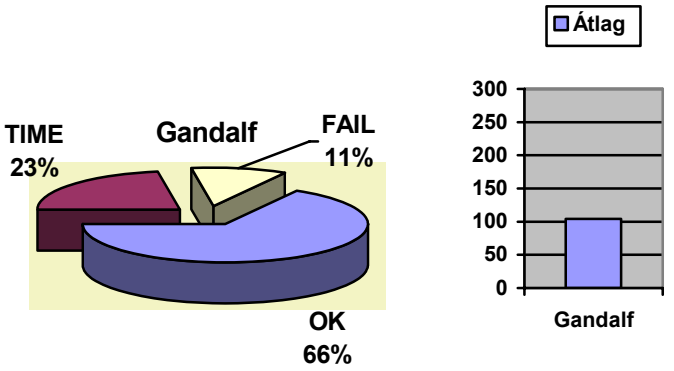

SNePSLOG
OK
13%
FAIL
87%

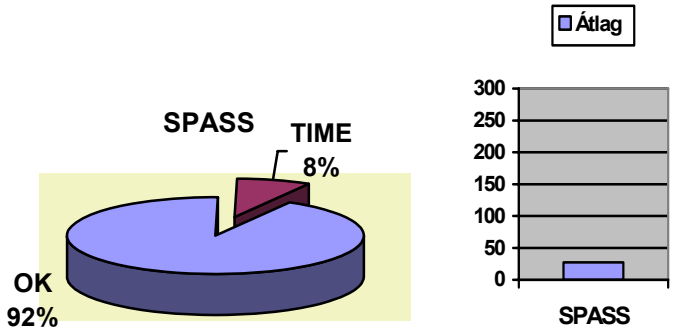

FAIL
63%
SNePSLOG
OK
7%
TIME
30%

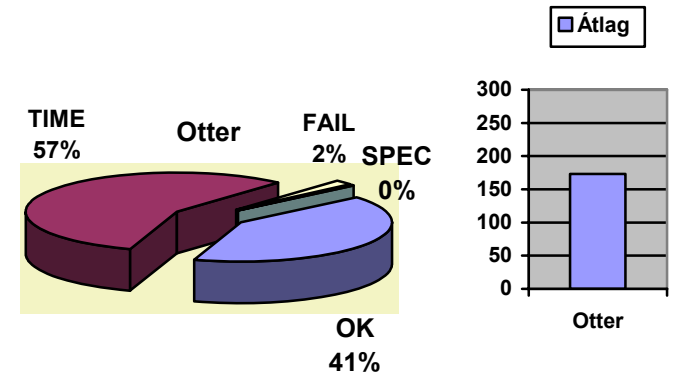

Átlag
50
40
30
20
10
0
SNePSLOG

A következő ábrák a TPTP-s eredményeket mutatják be a négy lejobban teljesítő rendszerre vonatkoztatva:









2 Következtető motorok működésével kapcsolatos fogalmak

2.1 Indefinit válasz (indefinite answer)

Ez akkor fordulhat elő, amikor egy formuláról el tudjuk dönteni, hogy igaz-e, és az igaz, viszont nem tudunk modellt adni a válaszhoz, akkor kapunk indefinit választ. Pl.: A tudásunk: $p(a) \vee p(b)$; a következőt kérdezzük: $\exists x$: $p(x)$? Ekkor a válaszunk mindenképpen igen, azonban nem tudunk modellt állítani, ami ezt igazzá teszi.

2.2 Helyesség

Egy következtetési eljárás *helyes*, ha csak igaz formulákat vezet le. Általában a következtető rendszerektől alapvető elvárás a helyesség. (Az elsőrendű logikához létezik többféle *helyes* következtetési eljárás. Azonban a legtöbb Prolog implementáció **nem** *helyes* eljárást alkalmaz, mivel az egyesítésnél nem végez el minden szükséges ellenőrzést.)

2.3 Teljesség

Egy következtetési eljárás *teljes*, ha tetszőleges igaz formulát képes bebizonyítani. (Az elsőrendű logikához létezik *teljes* következtetési eljárás. A Prologok többsége által megvalósított bizonyítási eljárás **nem** is *teljes*, mivel mélységi keresést alkalmaznak a megoldás keresésekor.)

2.4 Az elsőrendű logika alapfogalmai

Az elsőrendű logikát és a Clausal Logic-ot általában szokás megkülönböztetni, azonban mivel tetszőleges elsőrendű formula CNF-re hozható és viszont, ezt a megkülönböztetést mi nem tesszük meg.

2.4.1 Változó

A változó egy szimbólum, gyakran az x, y, z, … jelek valamelyike; egy speciális term. Az elsőrendű logikában nincs típusuk, azonban a sorted logikákban van. Megkülönböztetjük a *kötött* és a *szabad változó*kat. A *kötött változó*k valamilyen kvantor hatókörében vannak (ahhoz „köthetőek”), míg a *szabad változó*k minden kvantor hatókörén kívül esnek. Egy formulát *mondat*nak nevezünk, ha nem tartalmaz szabad változót.

2.4.2 Függvényszimbólum

Szintén egy szimbólum, az különbözteti meg a predikátumszimbólumoktól, hogy az ezekből képzett termeknek nem tulajdonítunk igazságértéket. Gyakori, hogy az f, g, h, … jelek egyikének választják. (A 0 aritású *függvényszimbólum*ot nevezhetjük *konstans*nak is.)

2.4.3 Predikátumszimbólum

Egy újabb szimbólum, egy *predikátumszimbólum* és termek felhasználásával képezhetünk atomi formulákat, melyeknek csak igazságértékük lehet, azonban nem lehetnek termek argumentumai. Tipikus jelei a p, q, r, ….

2.4.4 Univerzális kvantor

Az *univerzális kvantor*hoz tartozik legalább egy változó, mely(ek)et leköt a hozzá tartozó részformulában. A hozzá tartozó szemantika a következő: a változó(k) tetszőleges

helyettesítésére igaz a részformula. Pl.: a $p(x) \wedge \forall x$: $q(x, y, x)$ formulában az első x változó szabad, a többi kötött, az y változó szintén szabad. Csak a második részformula *univerzálisan kvantált*.

2.4.5 Egzisztenciális kvantor

Az *egzisztenciális kvantor*hoz tartozik legalább egy változó, mely(ek)et leköt a hozzá tartozó részformulában. A hozzá tartozó szemantika a következő: van a változó(k)nak (legalább egy) egy helyettesítése, melyre igaz a részformula. Pl.: $\exists x$: $q(x, x, x)$
Az *egzisztenciális kvantor*t le lehet cserélni a Skolemizációnak nevezett technikával, amely segítségével szükség szerint függvényszimbólumokkal helyettesíthetjük az általa lekötött változókat. Pl.: a $\exists x$: $[q(x, x, x)] \wedge \forall y$: $[\exists z$: $q(y, z, y)]$ formulából a következőt kapjuk: $q(sk0, sk0, sk0) \wedge \forall y$: $[q(y, skf0(y), y)]$, ahol $sk0$, $skf0$ új függvényszimbólumok.

2.4.6 Konjunkció

Két vagy több formulát kapcsol össze, jele: $\wedge$. Jelentése: akkor és csak akkor igaz, ha a formulák mindegyike igaz. Pl.: $(p(a) \vee \neg p(b)) \wedge \forall x$: $q(x)$.

2.4.7 Diszjunkció

Két vagy több formulát kapcsol össze, jele: $\vee$. Jelentése: akkor és csak akkor igaz, ha a formulák valamelyike igaz. Pl.: $p(a) \vee \neg p(b)$.

2.4.8 Negáció

Egy részformulára alkalmazható, jele: $\neg$. Jelentése: akkor és csak akkor igaz, ha a részformula hamis. Pl.: $\neg(p(a) \vee \neg p(b))$.

2.4.9 Implikáció

Két részformulát kapcsol össze, jele: $\rightarrow$. Jelentése: akkor és csak akkor hamis, ha az első részformula igaz, míg a második részformula hamis. Pl.: a $\neg p(a) \rightarrow \forall x$: $p(x)$ formula lehet igaz, amennyiben tudjuk, hogy $\neg p(a)$ hamis, azaz $p(a)$ teljesül. Egy ekvivalens alakja az $A \rightarrow B$ formulának a következő: $\neg A \vee B$.

2.4.10 Term

Változó, vagy függvényszimbólum melynek argumentuma(i) termek (ezek sorrendje számít).

2.4.11 Ground term

Olyan term, ami nem tartalmaz változót (tehát konstansokból és függvényekből van felépítve).

2.4.12 Predikátum, atomi formula

A *predikátum*, vagy *atomi formula* egy predikátumszimbólumból, és az aritásának megfelelő számú termből áll (sorrend számít).

2.4.13 Literál

Az atomi formulákat, melyek esetleg negálva vannak, *literál*nak nevezzük. Ha a *literál*ban a predikátumszimbólum az egyenlőség, úgy *egyenlőséges literál*nak nevezzük.
A p literál *flat literál*, ha a következők valamelyike teljesül:
p **egyenlőséges literál** és az egyenlőség mindkét oldalán legfeljebb 1 magasságú termek állnak;
p nem **egyenlőséges literál** és benne minden term egyetlen változó.

A **flat literál**ok azért fontosak, mert egyesíthetőségük könnyen tesztelhető (azaz könnyű megmondani, hogy a C_1 és C_2 **flat literál**okhoz létezik-e olyan θ_1 és θ_2 helyettesítés, melyre $C_1\theta_1=C_2\theta_2$ fennáll.
Megjegyzés: előfordul, hogy máshogy definiálják a **flat literál**okat; ezt a definíciót épp a fenti jó tulajdonsága miatt választottuk.

2.4.14 Clause, klóz

Literálok egy halmaza. Bármely *C klóz*t tekinthetjük annak a formulának is, mely előáll *C* elemeinek diszjunkciójaként. Általában úgy tekintjük, hogy a *C*-ben szereplő változók univerzálisan kvantáltak.
Speciális klóztípusok a következők:
Az **üres klóz**t a $\perp$ azonosan hamis formulával azonosíthatjuk.
Amennyiben a klóz egyetlen literált tartalmaz, **unit klóz**nak nevezzük – a unit klózok azért fontosak, mert ha egy *C* klózt unit klózzal rezolválunk, akkor az eredményként kapott klóz *C*-nél kisebb (elemszámú) lesz.
A **flat** („lapos") **klóz**ok csak flat literálokat tartalmaznak.

2.4.15 Horn-klózok

Olyan formulák, melyek a következő alakba írhatóak:
$\forall x_1, x_2, \ldots, x_n: p_1(\ldots)\wedge\ldots\wedge p_k(\ldots)\rightarrow q(\ldots)$, ahol $p_1, \ldots, p_k, q$ predikátumok, argumentumaikban csak $x_1, x_2, \ldots, x_n$ változók szerepelhetnek. Ezzel ekvivalens alak a következő: $\forall x_1, x_2, \ldots, x_n: \neg p_1(\ldots)\vee\ldots\vee\neg p_k(\ldots)\vee q\ (\ldots)$, innen látható, hogy miért klóz.

2.4.16 Clausal Normal Form, CNF

Klózok egy halmaza. Bármely *CNF*-et tekinthetjük annak a formulának is, mely előáll a klózhalmaz elemeinek mint formuláknak konjunkciójaként. Bármely formula(halmaz) átalakítható vele s-ekvivalens *CNF*-re. A standard átalakítási módszer a Skolem-normálformát használja fel; ezt – mivel igen nagyméretű *CNF*-eket generálhat – egy körültekintően megírt rendszernek optimalizálnia kell.
Például vegyük a következő formulát:

$$\forall x: \text{animal}(x)\rightarrow$$
$$\{(\forall y: \text{plant}(y)\rightarrow\text{eats}(x, y))\vee$$
$$[\forall u: [\text{animal}(u)\wedge\text{much\_smaller}(u, x)\wedge(\exists z: (\text{plant}(z)\wedge\text{eats}(u, z)))\rightarrow\text{eats}(x, u)]]\}$$

Ebből a következő *CNF*-et kapjuk:

$$\forall x, y, z, u: \neg\text{much\_smaller}(u, x)\vee\text{eats}(x, z)\vee\neg\text{eats}(u, y)\vee\text{eats}(x, u)\vee\ \neg\text{plant}(z)\vee\neg\text{plant}(y)\vee$$
$$\neg\text{animal}(x)\vee\neg\text{animal}(u)$$

2.4.17 Szubsztitúció, helyettesítés

Egy olyan függvény, mely változókhoz termeket rendel. Kiterjesztjük termekre, atomi formulákra, literálokra, klózokra értelemszerűen.

2.4.18 Unifikáció, egyesítés

Két term (*u* és *v*) *egyesíthető*, ha létezik olyan θ helyettesítés, melyre $u\theta=v\theta$. Az előző ponthoz hasonlóan ez is kiterjeszthető literálokra, klózokra.

2.4.19 Átírási szabály

Olyan következtetési lépés, mely úgy következik a tudásbázis bizonyos elemeiből, hogy azok közül legalább egy törölhetővé is válik (vagyis bármire, amire használni tudnánk a

későbbiekben a törölt ősöket, használni tudjuk az új elemet is, és legalább olyan jó eredményt kapunk).

2.4.20 Fölösleges klóz

Egy CNF-ben egy C *klóz fölösleges*, ha mint formula következik egy (szintén a CNF-ben szereplő) D klózból. Ez akkor áll fenn, ha $D\theta$ részhalmaza C-nek valamely θ helyettesítésre.
A *fölösleges klóz*ok kiszűrésének hatékony elvégzése komoly technikákat igényel; például már annak eldöntése sem egyszerű kérdés, hogy a C és a D klózok megkaphatók-e egymásból változó-átnevezéssel. Továbbá bizonyos esetekben célszerű megtartani a *fölösleges klóz*okat is, például ha C factoringgal kapható D-ből, és unit klóz – ugyanis a unit klózokat sok heurisztika előnyben részesíti.

2.5 Az elsőrendű logika kiterjesztései

2.5.1 Many-sorted logikák

A many-sorted logikák esetében az univerzumot típusokra (ezek a *sort*-ok) osztjuk, és mind a predikátum-, mind a függvényszimbólumok esetében megadunk értelmezési tartományt, hogy melyik koordinátára mely sort elemeit lehet behelyettesíteni (és függvény esetében, hogy mi az eredmény sortja). A kvantorok esetében – mivel a változók is típusosak – „minden X-re” helyett „minden S sort-beli X-re” változik a szemantika, az egzisztenciális kvantor kezelése analóg módon zajlik. Általában megkövetelik, hogy a sortok az univerzum diszjunkt partícióját adják – ennek oka az, hogy ekkor még megmarad jó néhány eldönthetőségi eredmény.

2.6 Stratégiák a propozicionális logikában

2.6.1 A Davis-Putnam-Logemann-Loveland (DPLL) Procedure

Lényege: egy formulának úgy ellenőrzi a kielégíthetőségét, hogy felépít egy fát, melynek csúcsaiban egy-egy formula van (kezdetben a fának egyetlen csúcsa van a kérdéses formulával), majd minden lépésben választ egy levelet és abban egy változót; ennek a levélnek ad két fiút, az egyikben az adott változó igaz, a másikban hamis értékkel kerül kiértékelésre, eldobva a formulának így triviálissá vált részeit.

2.7 Elterjedtebb következtetési módszerek

2.7.1 Rezolúció

Egy CNF-eken működő, általános következtetési módszer. Működése a következőképp történik: keres egy C_1 és egy C_2 klózt a CNF-ben, valamint egy θ_1 és egy θ_2 szubsztitúciót, melyekre $C_1\theta_1$ tartalmaz egy p^+ literált, $C_2\theta_2$ pedig annak p^- ellentettjét. Ekkor a rezolúciós lépés eredményeként a CNF-be felvesszük a $C_1\theta_1 \cup C_2\theta_2 \setminus \{ p^+, p^- \}$ klózt. Egy rezolúciós lépés (mely **nem** átírási szabály) mindig olyan klózt hoz be a halmazba, mely C_1-nek és C_2-nek logikai következménye, így a rezolúció egy helyes kalkulus. Amennyiben rezolúciós lépések sorával megkapjuk az $\perp$ üres klózt, igazoljuk, hogy az eredeti CNF kielégíthetetlen volt.
Kérdések eldöntésére a következőképp szokás alkalmazni: ha adott a tudásbázist reprezentáló CNF és egy kérdés, úgy felvesszük a kérdés negáltját is a CNF-be. Ha az így kapott CNF-en a *rezolúció* végrehajtása során megkapjuk az $\perp$ klózt, úgy a kérdésre a válasz „igen”.
A módszer sikeressége erősen múlik azon, hogy milyen stratégia szerint keressük ill. választjuk ki a fenti C_1 és C_2 klózokat, és határozzuk meg a θ_1 és θ_2 szubsztitúciót (ez utóbbira az ún. unifikációs algoritmus alkalmas, melyhez implementációs trükkök képzelhetők el). További fontos kritérium, hogy a CNF-ben szereplő klózok ne vagy csak kevés fölösleges

klózt tartalmazzanak; ennek biztosítására hatékony indexelési és hash-elési stratégiák kifejlesztése szükséges.
A *rezolúció* egy helyes és teljes kalkulus az *egyenlőség nélküli* predikátumkalkulusban, azaz ha $\perp$ levezethető, meg is kapjuk. Egyenlőség használata esetén ki kell egészíteni valamely módszerrel, mert abban a kalkulusban nem teljes.

2.7.2 Rendezett rezolúció

Bevezetünk feltételeket, amik arra irányulnak, hogy a rezolúció futása során „kevesebb választási lehetőséget adjunk magunknak". Ehhez kell egy > rendezés, ami ground termeken teljes (lineáris) és helyettesítésre stabil.
Ezt kiterjesztjük literálokra úgy, hogy argumentumaikat hasonlítjuk össze lexikografikusan; ha egyenlők, akkor előjelüket (negatív>pozitív).
Ezt kiterjesztjük klózokra, mint > literálokon vett értelmének multihalmazra kiterjesztését (azaz két klóz között akkor áll fenn $C>D$, ha D minden olyan literáljánál, ami nincs C-ben benne, van olyan nagyobb literál D-ben, ami C-ben nincs).
A *rendezett rezolúció* feltétele: csak akkor próbálhatunk rezolválni két klózt, ha a két literál, amik mentén rezolváljuk őket, maximális a saját klózukon belül. Az így megszorított rezolúció (egyenlőség nélküli esetben) továbbra is teljes marad.

2.7.3 Szuperpozíciós kalkulus

Olyan CNF-kalkulus, mely csak az egyenlőséget engedi meg predikátumként. Ez nem megszorítás, hiszen a predikátumokat függvényként kódolva és "[nem] egyenlő true-val"-ként leírva egy ekvivalens alakot kapunk. Következtetési szabályai a pozitív és a negatív szuperpozíció, az equality resolution és az equality factoring. Rendezési feltételeket is bevezetünk; ha > egy olyan redukciós rendezés, ami teljes (lienáris) a ground termeken, akkor a pozitív $s=t$ literálokhoz az $\{s,t\}$ multihalmazt, a negatív $s\neq t$ literálokhoz az $\{s,s,t,t\}$ multihalmazt rendeljük. Klózokhoz a bennük szereplő literálokhoz rendelt halmazok multihalmazát rendeljük. Két literált vagy klózt a megfelelő multihalmaz-rendezéssel hasonlítunk össze.
Csak akkor végzünk el egy következtetési lépést, ha az érintett literálok maximálisak a saját klózukon belül, továbbá a bal oldalukon álló term nem kisebb, mint a jobb oldalukon álló. A *szuperpozíciós* esetekben azt is megköveteljük, hogy az egyesítés elvégzése után is fennálljon ez a sorrend.
A *szuperpozíciós kalkulus* helyes és teljes (az elsőrendű egyenlőséges logikában). Mindazonáltal hatékony term-indexelő stratégiák megléte szükséges a valóban gyors működéshez.

2.7.4 Tabló módszerek

A *tabló (tableaux) módszer* többféle logikán alkalmazható, itt az elsőrendű logikához kapcsolódó algoritmust mutatjuk be.
A *tabló* formulák halmaza.
Skolem normálformára hozva a tudásbázist, azon végzünk különböző átalakításokat.
Azokat a formulákat, melyek konjunkcióval ($\wedge$) vannak összekötve (Pl.: $A\wedge B$) szétválaszthatjuk és a két részformulával bővíthetjük a tablót, az eredetit elhagyva. (Tehát ha a tablóban szerepelt $A\wedge B$, akkor a következő tablóban már nincs jelen, viszont az már tartalmazza A-t és B-t is.)
Azokat a formulákat, melyek diszjunkcióval ($\vee$) vannak összekötve (Pl.: $A\vee B$) szétválaszthatjuk és két új tablót készíthetünk belőle. Az új tablók nem tartalmazzák az eredeti formulát, azonban a részformulák közül pontosan egyet tartalmaznak (és különbözőt).

(Tehát ha a tablóban szerepelt $A \vee B$, akkor a következő tablóban már nincs jelen, viszont az már tartalmazza A-t és B-t is.)
Az olyan formulák esetében, melyek univerzális kvantorral ($\forall$) vannak lekötve (Pl.: $\forall x: p(x)$), a változó minden előfordulását helyettesítjük a tablóban előforduló összes termmel. Ezekkel a helyettesítésekkel kapott formulákkal lecseréljük az eredeti formulát.
Egy formulából kiindulva először negáljuk, majd erről igyekszünk belátni, hogy kielégíthetetlen. Ellentmondásos/ütközéses (clash) egy tabló, ha egy literál és annak ellentettje is szerepel benne. Amennyiben minden ágon van ellentmondás, úgy a formula kielégíthetetlen.

2.8 Következtetési szabályok CNF alapú rendszerekben

2.8.1 Factoring (reflection)

Olyan következtetési lépés, mely választ egy C klózt a CNF-ből, melyhez létezik olyan θ helyettesítés, amire $C\theta$ kevesebb literált tartalmaz, mint C. Ekkor felvesszük a CNF-be $C\theta$-t is. Ez is egy helyes következtetési lépés; a legtöbb rezolúciós heurisztika szerint annál „jobb" egy klóz, minél kevesebb literált tartalmaz. A *factoring* új klózként veszi fel $C\theta$-t (vagyis **nem** átírási szabály).

2.8.2 Subsumption

Ha a CNF-ünk tartalmaz olyan C és D klózokat, melyekre $D=C\theta$ valamely θ helyettesítésre, viszont fordítva nincs ilyen helyettesítés, akkor D-t törölhetjük. Ezt a lépést bizonyos esetekben (pl. ha D factoringgal kapható C-ből, azaz kevesebb literált tartalmaz) nem célszerű elvégezni.

2.8.3 Pozitív szuperpozíció

Menete: ha a C klózban szerepel egy $t=t'$ egyenlőség, a D klózban pedig egy olyan $s[u]=s'$ egyenlőség, ahol u nem változó és egyesíthető t-vel a θ helyettesítés alkalmazásával, akkor vegyük fel a $(C \cup D \setminus \{t=t', s[u]=s'\} \cup \{s[t']=s'\})\theta$ klózt.

2.8.4 Negatív szuperpozíció

Menete: ha a C klózban szerepel egy $t=t'$ egyenlőség, a D klózban pedig egy olyan $s[u] \neq s'$ egyenlőség, ahol u nem változó és egyesíthető t-vel a θ helyettesítés alkalmazásával, akkor vegyük fel a $(C \cup D \setminus \{t=t', s[u] \neq s'\} \cup \{s[t'] \neq s'\})\theta$ klózt.

2.8.5 Equality resolution

Menete: ha a C klózban szerepel egy $s \neq s'$ literál, ahol s és s' egyesíthető θ-val, akkor vegyük fel a $(C \setminus \{ s \neq s'\})\theta$ klózt.

2.8.6 Equality factoring

Menete: ha $C=C' \cup \{s=t, s'=t'\}$, ahol s és s' egyesíthető θ-val, vegyük fel a $(C' \cup \{t \neq t', s=t'\})\theta$ klózt.

2.8.7 Paramodulation

Menete: ha van egy $C=C' \cup \{s=t\}$ klózunk és egy $D=D' \cup \{p[r]\}$ klózunk, ahol s és r egyesíthető θ-val, akkor ezekből *paramodulációval* kapható a $(C' \cup D' \cup \{p[t]\})\theta$ klóz.

2.8.8 Demodulation

A paramoduláció azon speciális esete, amikor is C' az üres halmaz.

2.8.9 Hyperresolution

Következtetési szabály. Amennyiben van egy C klózunk, ami tartalmaz legalább egy negatív literált, továbbá van $D_1,D_2,...,D_n$ klózunk, melyek csak pozitív literált tartalmaznak, és egy θ szubsztitúció, mely bármely i esetén alkalmas C és D_i rezolválására (azaz C egy negatív literálját párosítja D_i valamely pozitív literáljával), akkor eredményképpen kapjuk a $(C \cup D_1 \cup D_2 \cup ... \cup D_n \setminus X)$ θ klózt, ahol X a párosított literálok halmaza.
Könnyen meggondolható, hogy a *hiperrezolúció* eredménye mindig megkapható az egyszerű rezolúció néhányszori ismétlésével.

2.8.10 Unit Resolution

Következtetési szabály. A rezolúció azon sepciális esete, mikor az egyik résztvevő klóz unit klóz. Ha ekkor az egyesítő alkalmazása nem változtatja meg a másik klózt, akkor átírási szabályként is alkalmazható, ekkor *Unit Deletion*-ról beszélünk.

2.9 Az egyenlőség kezelésére irányuló fogalmak

Az egyenlőség jelenléte annyira megbonyolítja a következtetési mechanizmusokat, hogy szinte mindenhol külön módszereket fejlesztettek ki kezelésére. Ezek nagy része az egyenlő termek egymásra átírásával dolgozik.

2.9.1 Az egyenlőség naiv megközelítése

Ha az egyenlőség támogatását a lehető legkevesebb munkával akarjuk elvégezni, akkor úgy kezeljük, mint bármelyik másik predikátumszimbólumot, és kimondjuk rá a reflexivitás, tranzitivitás, szimmetria, függvény- és relációkompatibilitás tulajdonságokat. Ekkor máris egy teljes és helyes következtetési rendszert kapunk (pl. rezolúcióval vagy tabló módszerrel) – a (nagy) hátrány az, hogy (főleg a tranzitivitás és a kompatibilitás miatt) a rendszerünk messze nem hatékony ebben az esetben.

2.9.2 Redukciós rendezés

Termeken értelmezett olyan > jólrendezés, mely megőrzi a szubsztitúciót és kompatibilis a függvényképzéssel. Azaz nincs végtelen $t_1>t_2>t_3>...$ lánc; tetszőleges θ szubsztitúcióra és t_1, t_2 termre ha $t_1>t_2$, akkor $t_1\theta>t_2\theta$; és ha f egy n-változós függvényszimbólum, $t_1>t_1',...,t_n>t_n'$ termek, akkor $f(t_1,...,t_n)>f(t_1',...,t_n')$.
Ennek speciális esete az *egyszerűsítő rendezés*, mely *redukciós rendezés* és amennyiben a < rendezésre még az is teljesül, hogy tetszőleges t_1, t_2 termekre ha t_1 résztermje t_2-nek, akkor $t_1<t_2$ is fennáll.

2.9.3 Átírási szabályok konvergenciája

Termátírási szabályok egy halmaza *konvergens*, ha konfluens és terminális; ez azt jelenti, hogy tetszőleges termből kiindulva a szabályok alkalmazási sorrendjétől függetlenül ugyanahhoz a normálformához – azaz egy olyan alakhoz, amit nem lehet átírni a szabályok segítségével – jutok véges sok lépésben.

2.9.4 Lexicographic Path Ordering, LPO

Egy egyszerűsítő rendezéscsalád, amit az egyenlőség kezelésekor fel lehet használni. Paraméterezhető egy > részbenrendezéssel a függvényszimbólumok fölött. A > által indukált *LPO* szerint az s term nagyobb a t termnél (jelben $s >_{lpo} t$), ha az alábbiak valamelyike teljesül:

- t egy olyan változó, ami előfordul s-ben (de nem s maga);
- s valamelyik s' résztermjére $s' \geq_{lpo} t$;
- s szimbóluma nagyobb t szimbólumánál és $s>_{lpo}t'$ a t minden közvetlen résztermjére;

- s és t szimbóluma megegyezik, és közvetlen résztermjeikre fennáll $(s_1,s_2,...,s_n) >_{lpolex} (t_1,t_2,...,t_n)$ – a lexikografikus rendezés szerint.

A rendezés jó tulajdonsága, hogy ha > teljes (lineáris) rendezés a függvényszimbólumokon, akkor $>_{lpo}$ a ground termeken teljes (lineáris).

2.9.5 Knuth-Bendix Ordering, KBO

Egy másik egyszerűsítő rendezéscsalád, szintén gyakran használatos az egyenlőség kezelésére. Ezt is egy > részbenrendezéssel paraméterezzük, ami a függvényszimbólumok fölött értelmezett. Továbbá egy w súlyfüggvényt is definiálnunk kell, ami minden változóhoz és függvényszimbólumhoz hozzárendel egy nemnegatív valós súlyt úgy, hogy az alábbiak fennállnak:

- minden változó ugyanazt a súlyt kapja, mely súlynál bármelyik konstans nagyobb súlyt kap;
- egyváltozós függvény csak akkor kaphatja a 0 súlyt, ha > szerint ő a legnagyobb elem.

Ezt a súlyfüggvényt kiterjesztjük termekre egy egyszerű összegzésként: a term súlya a benne szereplő szimbólumok súlyainak összege, multiplicitással.
A > részbenrendezés és w súlyfüggvény által indulált $>_{KBO}$ rendezés a termek fölött a következő: s pontosan akkor nagyobb t-nél $>_{KBO}$ szerint, ha az alábbiak valamelyike fennáll:

- s súlya nagyobb, mint t-é és benne minden változó legalább annyiszor fordul elő, mint t-ben;
- s súlya megegyezik t-évek, s-ben minden változó legalább annyiszor fordul elő, mint t-ben és
- vagy $s = f(f(f(....(f(t))....)))$ (ekkor f a 0 súlyú egyváltozós függvény);
- vagy s szimbóluma a nagyobb > szerint, mint t-é;
- vagy s és t szimbóluma megegyezik, és közvetlen résztermjeikre fennáll $(s_1,s_2,...,s_n) >_{kbolex} (t_1,t_2,...,t_n)$ – a lexikografikus rendezés szerint.

A Knuth-Bendix rendezés is egyszerűsítő rendezés.

2.9.6 Knuth-Bendix Completion

Egy módszer arra, hogy termegyenlőségek egy E halmazát termátírási szabályok konvergens halmazává alakítsuk. Ennek a haszna az, hogy innen kezdve két tetszőleges term egyenlősége eldönthető úgy, hogy mindkettőt (a konvergencia miatt létező és meghatározható) normálformára hozzuk és pontosan akkor egyenlőek E szerint, ha a normálformájuk ugyanaz. Ehhez szükségünk van egy > redukciós rendezésre (pl. a LPO vagy a KBO megfelelő).
Az algoritmus vázlatosan:

- Olyan termegyenlőségeket, amik között > fennáll, a kisebb felé irányítunk;
- Triviális egyenlőségeket eldobunk;
- A beérkező kritikus párokat mindig felvesszük új egyenlőségként;
- (esetleg) E-ben előforduló átírható egyenlőségeket átírunk (így egyszerűsítve azt);
- (esetleg) átírási szabály átírható jobboldalát átírjuk.

A *Knuth-Bendix Completion* kimenete többféle lehet – elképzelhető, hogy az algoritmus nem áll meg, sikeresen áll meg (E kiürül és nincs új kritikus pár a szabályrendszerben), sikertelenül áll meg (E nem üres, de nincs több lépés – ekkor pl. egy másik > rendezéssel lehet próbálkozni).
"Unfailing Completion"-nak nevezzük, ha mindig tudunk irányítani vagy törölni egyenlőséget. Ezt biztosíthatjuk azzal, hogy egy olyan > redukciós rendezést veszünk, ami a ground termeken teljes (lineáris), és ha két term közt nem áll fenn >, irányítsuk mindkét irányba.

2.10 Egyéb CNF-technikák

2.10.1 Flattening

Tetszőleges klóz (egyenlőséget is tartalmazó) flat klózzá alakítható úgy, hogy a nem-flat literálokban a rossz helyen álló termeket (ahol a flat literál definíciója szerint változónak kellene állni, mégis egy magasabb term áll) egy új változóval helyettesítjük, és ezt az új változót egyenlővé tesszük a helyettesített termben. Ezt a helyettesítést szükség esetén többször ismételhetjük, eredményképpen egy, az eredetivel ekvivalens flat klózt kapunk (mely általában ugyan flat, viszont hosszabb, mint az eredeti). Olyan motorokban használatos, mely csak flat klózokon dolgoznak; ekkor átírási szabályként alkalmazzák.
Bizonyos esetekben csak a ground literálokat írják át.

2.10.2 Súlyozás, weighting

A módszer lényege, hogy olyan súlyfüggvényt definiáljunk a klózokon, ami segít annak kiválasztásában, hogy melyik klóz(oka)t válasszuk ki a következő lépés inputjaként. Egy ilyen súlyozás lehet például a Knuth-Bendix rendezés definíciójában szereplő súlyozás, értelemszerűen kiterjesztve klózokra.
A súlyozás másik alkalmazása, hogy beállítva egy maximális súlyt, nem engedünk generálni a tudásbázisba olyan klózt, aminek súlya nagyobb ennél a limitnél.

2.10.3 Lineáris rezolúció

A rezolúció egy megszorítása: a levezetés aktuális lépésében az előző lépésben kapott klózt fel kell használjuk. Meglepő módon (backtracking lehetőséggel) ezzel a stratégiával is teljes marad a rezolúció, az egyenlőség nélküli logikában. Ez a tárigény csökkentését teszi lehetővé (ha pedig meg tudjuk jegyezni a literálpárokra, hogy mikor utoljára teszteltük, egyesíthetőek voltak-e, az időigény sem feltétlenül nő akkorát).
Ennek egy további finomítása a *model eliminációs eljárás* (*model elimination* - *ME*). Utóbbit használja a PTTP (Prolog Technology Theorem Prover), valamint a logic.standord.edu címen lévő EPILOG is. A model eliminációs eljárás előnye, hogy jól párhuzamosítható, így nagy elosztott számítási teljesítményt hatékonyan képes kihasználni.

2.10.4 Set-of-support (SOS) stratégia

A módszer használatával két részre osztjuk a klózhalmazt. Az egyik halmazba, *T*-be olyan klózok kerülnek, melyek még nem ellentmondóak. A másik részbe, *S*-be az esetleg ellentmondást okozó klózok (azaz, az egyszerű rezolúció esetében *T* üres). A rezolúció menetét úgy kontrolláljuk, hogy minden lépésben legalább az egyik rezolválandó klóz *S*-beli kell legyen (és az eredmény is oda kerül). Ez egy széles körben használt, hatékonynak bizonyult stratégia.

2.11 Nem-CNF alapú rendszerek következtetései

2.11.1 Rete algoritmus

Olyan rendszerek esetében alkalmazható, amik csak implikációkat támogatnak a szabályaikban, ott az előrekövetkeztetés során. A módszer lényege, hogy a szabályok törzsében összegyűjtjük a különböző mintákat, és ezekből egy Petri-háló szerű mintaillesztő hálózatot generálunk. Így ha két szabálynak van közös része a bal oldalukon, nem kell mindkétszer kiértékelni azt a részt.
Ez a módszer láthatóan nem támogatja a visszavonást, így csak előrekövetkeztetés során alkalmazható; a feltételek-akciók felosztás pedig implikatív alakot követel meg.

A *Rete algoritmus*t továbbfejlesztve megszületett a Rete-II és Rete-III algoritmus is, ezek azonban nem publikusak.

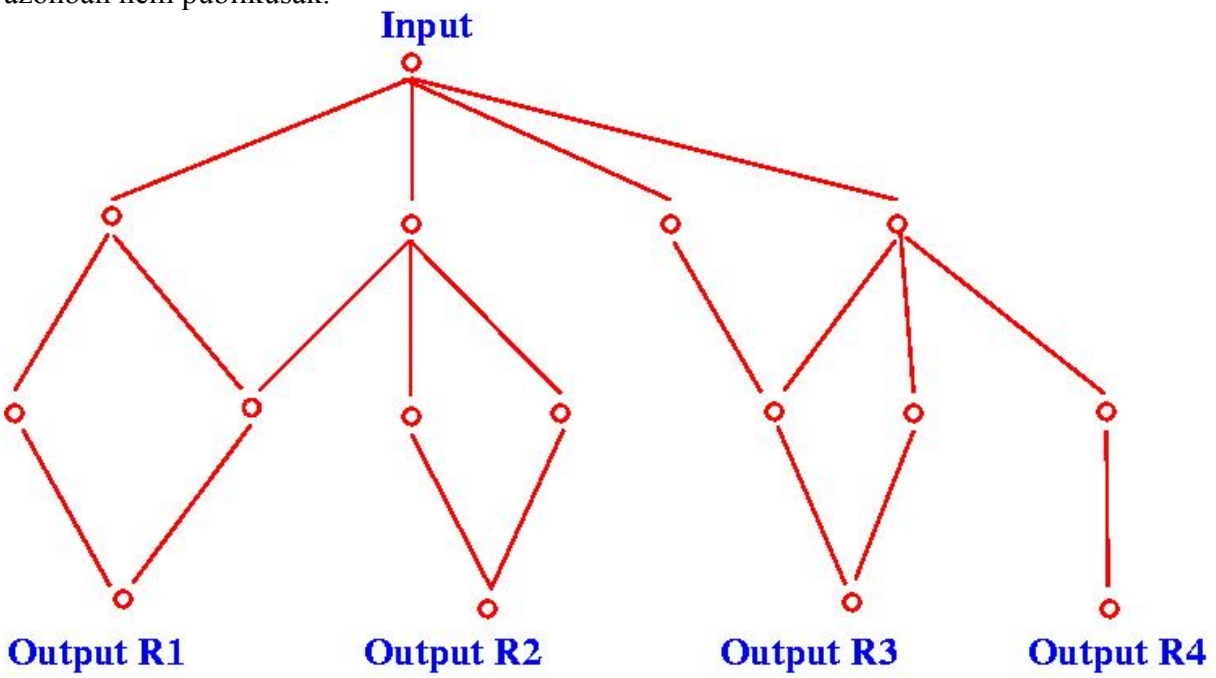


Figure 1 Egy Rete háló. Forrás: http://www.cs.bham.ac.uk/~mmk/Teaching/AI/l2.html

2.11.2 Redukciók

A nem-CNF alapú, ám az elsőrendű logikát teljes egészében támogató rendszerek többségükben a tabló módszereket vagy valamely mátrix alapú módszert használnak. Ezen esetekben kulcsfontosságú az elágazási faktor csökkentése, azaz a formulák olyan ekvivalens átalakítása, ami a keresés során kevesebb elágazást eredményez. Erre például egy lehetőség a többször ismétlődő részformulák helyett azokkal ekvivalensnek definiált literálok bevezetése vagy csak a prímimplikánsok használata. Az ezzel kapcsolatos irodalomban csak a propozicionális logikában használt esetet fejtették ki a szerzők, és említették, hogy módszerük „kiterjeszthető az elsőrendű logika esetére". Továbbá, a – tapasztalatunk szerint – egyik legnagyobb problémát a CNF-re alakítással kapcsolatosan nem oldják meg. A probléma, hogy egymásba ágyazott ekvivalenciák esetén a formulák száma exponenciálisan növekszik; a szerzők ezzel annyira nem foglalkoztak, hogy egyszerűen nem engedték meg az ekvivalencia használatát.

2.12 Fuzzy rendszerekkel kapcsolatos fogalmak

2.12.1 Halmazhoz tartozás függvénye, fuzzy halmaz

A fuzzy rendszereknél egy halmazt nem csak az elemeivel, hanem a lehetséges elemei és egy *halmazhoz tartozási függvény* segítségével reprezentálnak. Ez utóbbi tekinthető a halmaz karakterisztikus függvényének általánosításának. A *halmazhoz tartozás függvény* azt az igazságértéket reprezentája, hogy egy adott elem mennyire tartozik a hivatkozott halmazhoz. Az értékkészlete: [0, 1], értelmezési tartománya pedig egy halmaz. Például a **fiatal** fuzzy halmazt értelmezhetjük az embereken (egy adott időpontban) és valószínű, hogy az újszülöttek esetén 1 lesz az értéke, míg a 120 évesek esetében ez már 0 értékű. Természetesen közte is tetszőlegesen változhat, azonban itt sejthető, hogy ez monoton csökkenő függvény lesz. A **középkorú** halmaz azonban valamely kor környékén emelkedik 1 értékűre és az alacsony és a magas kor környékén 0-vá válik. (A *halmazhoz tartozás függvény*nek nem feltétlenül kell felvennie sem a 0-t, sem az 1-et az értelmezési tartományán.)

2.12.2 Fuzzy érték
Egy halmaz eleméhez (például egy emeberhez) rendelt valamilyen fuzzy halmaz és halmazhoz tartozási függvény értékének párja. Például: x 0,7 mértékben magas.

2.12.3 Fuzzy változó
Egy olyan változó, mely egy fuzzy halmazból kaphat (fuzzy) értékeket.

2.12.4 Fuzzy szabályok
Azon rendszerek, melyeket vizsgáltunk, csak Horn formulákat képes kezelni. Minden halmazhoz meghatározható, hogy mely változót hogyan állítsa be. Mivel minden szabályt végrehajt, mely feltételére illeszkedik az ismert adat, így nincs értelme „különben” ágaknak. Ezek a szabályok a nulladrendű (propozícionális) logika kiterjesztésével kaphatóak. (Tehát nincsenek függvényszimbólumok, kvantorok.)

2.13 Az elsőrendű logikába beágyazható funkcionalitások

2.13.1 Aritmetika
Az aritmetikai műveleteket egész számokon a Peano-axiómák beépítésével tudjuk ugyan emulálni az elsőrendű logika segítségével, ám ez elfogadhatatlan mértékben lelassítja a számokkal kapcsolatos következtetéseket. Az egyetlen járható útnak many-sorted logika alkalmazása tűnik, melynél megengedjük bizonyos függvényszimbólumok esetén azok kiszámításának programkóddal való megadását is (így például az integer sort-ra az alapműveletekét). Bizonyos esetekben az eredmény és egyes paraméterek ismeretéből a többi paraméter meghatározása is értelmes elvárás lehet (pl. ha van egy `3+X=5` termünk, abból a rendszer képes legyen kikövetkeztetni, hogy `X=2`). Ennek részletei még kidolgozandók, néhány rendszer azonban már támogatja.

2.13.2 Időkezelés
Az idő- ill. szituációkezelésre a bevett módszer az ún. szituációkalkulus (situation calculus) alkalmazása: itt (szintén many-sorted logika alkalmazása mellett) bevezetünk egy új sortot, a szituációk típusát, továbbá minden predikátumot kiegészítünk még egy paraméterrel, mely megfelel a szituációnak, amikor a predikátum fennáll. Így például a `fekszik(kutya)` predikátumból a `fekszik(kutya,t0)` predikátum születhet. A szituációk közti összefüggések kezelésére az akciók sortjának bevezetése, valamint egy rákövetkezés függvényszimbólum nyújt lehetőséget; például `rákövetkezés(t0,kisütanap)` az a szituáció, mely a `t0` után következik annak a „`kisütanap`” akció hatására.

2.13.3 Vélekedések
A „Béla úgy gondolja, hogy a kutya fekszik”-típusú mondatok, melyek egy-egy személy vélekedését írják le, szintén beágyazhatók az elsőrendű logikába egy szintaktikai változtatással. Ez a változtatás hasonló a szuperpozíció alkalmazásakor történő átalakításhoz: az összes predikátumszimbólumból függvényszimbólumot képzünk (és felveszünk egy `hiszi` függvényszimbólumot is. A fenti mondat ekkor pl. így formalizálható: `hiszi(Béla,fekszik(aKutya))=true`; itt az `=` az egyetlen predikátumszimbólum.

2.14 Kombinációk

2.14.1 A Nelson-Oppen Combination Method
Ez a módszer akkor használatos, ha két különböző teóriát (ilyen lehet pl. az elsőrendű logika és a lineáris aritmetika) együtt használó következtető rendszert készítünk. Nevezetesen, egy

módszert ad a kezünkbe, ami leírja, hogy hogyan lehet a két eltérő következtető módszert ötvözni.

2.14.2 Hipertablók

Elsőrendű logikai következtetések végzése olyan tabló módszerrel, melyben a formulák CNF-ben vannak. Ez a tabló kalkulus helyes és teljes. Módszerét tekintve nagyban emlékeztet a hiperrezolúcióra, ám azzal ellentétben pl. eldönti a Bernays-Schönfinkel osztályt (ezek azok a függvényszimbólumtól mentes formulák, ahol elegendő Skolem-konstansokat bevezetni). Viszont egyenlőségre nincs kiterjesztve.

2.15 Nem-elsőrendű megközelítések

Az alább felsorolt megközelítések – az eltérően definiált szemantika miatt – másfajta leírását teszik lehetővé a világnak, mint az elsőrendű logika. Azonban ezeknek közös hátrányuk, hogy nincs – vagy csak nagyjából körvonalazott – hozzájuk kiforrott következtetési mechanizmus.

2.15.1 Lineáris logika

A *lineáris logika* nagyon hasonlít az elsőrendű logikára, egy szemantikai különbséggel: a formulákat mint erőforrásokat képzeli el, és a bizonyítási lépések során felhasznált formulák valóban felhasználásra kerülnek – egy formulát egy bizonyítás során csak egyszer alkalmazhatunk.

Ennek a megközelítésnek a létalapja a következő példával szemléltethető: a „van tej” fennállása és a „van tej → van vaj” implikáció esetén az elsőrendű logika esetében kikövetkeztethető, hogy „van tej ÉS van vaj”. Amennyiben mint erőforrást képzeljük el a formulákat, a kettő kombinációjaként előálló „lehetséges világban” csak vaj van, tej nincs.

2.15.2 Intuitionistic logic

Ez a logika is csak szemantikáját tekintve különbözik az elsőrendű logikától: a predikátumok igazságértéke itt egybeesik bebizonyíthatóságukéval. Ez olyan érdekes következményekkel jár, mint pl. hogy a „$p \vee \neg p$” formula ebben a logikában nem feltétlenül igaz: ha sem p, sem $\neg p$ nem bizonyítható, akkor egyik sem igaz. Továbbá nem igaz az sem, hogy $\neg\neg p=p$, hiszen p jelentése „p bebizonyítható”, $\neg\neg p$ jelentése pedig „$\neg p$ nem bizonyítható be”. Az előbbiből következik ugyan az utóbbi, de fordítva ez nem áll fenn.

2.15.3 λ-kalkulus

A *lambda-kalkulus* megközelítése szerint „minden felfogható egyváltozós függvényként”; a definíció szerint másra nem is vagyunk képesek, mint függvényt definiálni. A logika maga nem tartalmaz sem függvény-, sem predikátumszimbólumokat; a természetes számokat („Church-számok”-ként való ábrázolással) szintén egyváltozós függvényként reprezentáljuk: az n szám megfelel annak a függvénynek, ami bemenetként kap egy f függvényt és visszaadja az f^n függvényt. Például a 0 ábrázolása a következő: $\lambda f(\lambda x(x))$; ennek olvasata „olyan függvény, amely az f input esetén visszaad egy olyan függvényt, mely az x input hatására visszaadja x-et magát”. Azaz, tetszőleges f függvényből az identikus függvényt készíti. Az 1 ábrázolása $\lambda f(\lambda x(fx))$, azaz az f függvényből mint inputból az a függvény készül, mely az x inputra visszaadja fx-et; ez nyilván maga f lesz, így ez valóban az f függvény első hatványa. A 2 átirata ennek megfelelően $\lambda f(\lambda x(ffx))$, stb.

A *λ-kalkulus*ban definiálhatók a logikai konstansok és konnektívák is (mindannyian szintén egyváltozós függvényként definiálva), a fentihez hasonló módon. A kétváltozós $f(x,y)$ függvények felfoghatók olyan egyváltozós x-függvénynek, mely egy y-függvényt ad vissza, aminek belsejében mind x, mind y használható. Mivel a *λ-kalkulus*ban definiálható minden,

amit definiálni csak lehet (Turing-teljes), több programozási nyelv (pl. a LISP) ennek az alapjaira épül. Használata (főleg a felhasználóval való kommunikáció része) azonban módfelett kényelmetlen, és az emberi gondolkodástól nagyon távol áll, így ezt nem javasoljuk.

2.15.4 Boyer-Moore bizonyítási eljárás

Sok bizonyítási technikát ismer, ezek közül a leggyakrabban használtak:

- Egyszerűsítés (simplification)
- Destructor elimination
- Cross-fertilization
- Általánosítás (generalisation)
- Elimation of irrelevance
- Indukció (induction)

Ezen technikák és többféle heurisztika alkalmazásával szimbólumok helyettesítésével igyekszik megoldani a feladatait.
Az ACL-2 és az elődjének tekinthető Nqthm alkalmazza ezt a technikát. Többnyire szükséges emberi közreműködés is a bizonyítás eléréséhez.

2.15.5 Sequent deduction (LK), natural deduction (NK)

Az emberek matematikai következtetéseire hasonlító következtetési eljárás, mely helyes és teljes az elsőrendű logikában. Léteznek kiterjesztései más logikákra vonatkozóan is. Mind előre-, mind hátra következtetéshez használhatóak a szabályaik (melyekből viszonylag sok van), azonban többnyire csak az utóbbira szokták használni azokat.

2.15.6 Indukció

A matematikai teljes indukció használata viszonylag ritka a következtető rendszerek között. Egyedül az ACL-2 volt, mely a vizsgált rendszerek közül ezt a statégiát is alkalmazta. A módszer lényege, hogy belát egy kezdőformulát és egy továbblépési szabályt (Pl.: 0, 1 (szomszédos) négyzetszámok, különbségük (1) páratlan. Ha tudjuk, hogy $(n+2)^2$ és $(n+1)^2$ különbsége egy páros számmal tér el $(n+1)^2$ és n^2 különbségétől, akkor ebből már következik, hogy a szomszédos négyzetszámok különbsége páratlan. (Feltéve, hogy ismerjük a páratlan szám induktív definícióját.))

2.15.7 Higher-Order Logic

Ebben a logikában a függvényekre és a predikátumokra tehetünk kvantorokat, így kifejezhető például az, hogy a macskák minden olyan tulajdonsága, mely megegyezik a kutyákéval, az megegyezik a farkasokéval is.
Ebben a logikában azonban már nincs teljes bizonyítási eljárás. (Lehetnek olyan igaz állítások, melyeket nem tudunk bebizonyítani.)
Ehhez kapcsolódó rendszerek a TPS, valamint a HOL. (A HOL-t használja az Isabelle is.)

2.15.8 Modális logika – Modal logic

Az elsőrendű logikát egészíti ki a „*szükségszerűen*” (□), valamint az „*esetleg*” (◊) kifejezésekkel. Több kiterjesztését is vizsgálták, melyeket gyakran szintén modális logikának nevezik. Amivel ki lehet még bővíteni: „*kötelező*”, „*engedélyezett*”, „*tilos*”, „*jövőben állandó*”, „*jövőben egyszeri*”, „*múltban állandó(an)*”, „*múltban egyszer*”, „*x azt hiszi*”.

3 Tesztek bemutatása

Ezek a tesztek már a rendszerek sebességét igyekeztek összehasonlítani, azonban túlzottan messzemenő következtetéseket nem lehet az eredményekből levonni, mivel többféle

lehetséges beállítás létezik mindegyik rendszer esetében, így az egyik gyengébb szereplése nem jelenti azt, hogy valóban minden körülmények között lassabb, mindössze azt jelenti, hogy a másik rendszernél sikerült egy viszonylag jó beállítást találnunk az adott problémá(k)ra.

3.1 Steamroller

Egy logikai feladvány; teljes szövege a következő:
„A farkasok, rókák, madarak, csigák és hernyók állatok, és mindegyik létező faj (van példányuk). A gabona növény; ez is létező faj. Minden állat vagy megeszik minden növényt, vagy minden nála kisebb olyan állatot, ami eszik növényt. A farkasok nem esznek sem rókát, sem gabonát. A madár megeszi a hernyót, de a csigát nem. A csiga és a hernyó is eszik növényt. A farkasnál kisebb állat a róka, a rókánál a madár, a madárnál pedig a hernyó és a csiga is kisebb. Ebből következően van olyan állat, aki eszik növény-evő állatot."
A következők miatt szokott kemény dió lenni ez a kérdés:

- A következtető rendszerek nagy részében a „minden állat vagy megeszik minden növényt, vagy minden nála kisebb olyan állatot, ami eszik növényt" mondat nem formalizálható, a fejoldalon lévő diszjunkció miatt.
- A „csiga és a hernyó is eszik növényt" pontos formalizálásához egy egzisztenciális kvantort vagy ekvivalensen egy Skolem-függvényt kell definiálni; ehhez már elsőrendűnek kell lennie a motornak.
- A Skolem-függvényekkel az is lehet probléma, hogy a rendszerek némelyike végtelen levezetésbe szalad, amikor az f függvényt elkezdi visszahelyettesíteni önmagába.

(Egy – legrövidebb – megoldás a következő: a hernyó eszik növényt, és kisebb a madárnál, ami viszont őt nem eszi meg; tehát a madár minden növényt megeszik, így a gabonát is. A farkas nem eszik gabonát, így minden nála kisebb növényevőt megeszik. A rókát – ami kisebb, mint a farkas – nem eszi meg, így a róka nem növényevő, azaz a róka megeszik minden nála kisebb növényevőt. Például a madarat is, így a róka olyan állat, ami megeszi a gabona-evő madarat. Tehát a levezetés még hosszú is – ez szintén problémát jelenthet.)
Megjegyzés: ez a feladvány megegyezik a PUZ031+1 teszttel.

3.2 Egyenlőség

Ez lényegében néhány egyszerű teszt az egyenlőség és a függvényszimbólumok helyes kezelésének ellenőrzésére.

3.3 Lánc

Egy következtetések láncolatából álló szabályrendszer, mely végén azt kérdezzük a rendszertől, hogy az első részből következik-e az utolsó.
Két paraméter érték mellett jegyeztük fel a következtetéshez szükséges időt 5 és 100 esetében.

3.4 Háromszög

Két változata van, az egyik egy formulából indul ki, majd egy-egy konjunkcióval a fejben folyamatosan növekszik. A maximális méret elérése után a végső konjunkcióra kérdezünk rá.
A másik változat lényegében ugyanez konjunkció helyett diszjunkcióval.
Mindkét változatot 8-as paraméterérték mellett teszteltük.

3.5 Rombusz

Két változata van, az egyik egy formulából indul ki, majd egy-egy konjunkcióval a fejben folyamatosan növekszik. A maximális méret után konjunkciókkal a törzsben fokozatosan lecsökken. A végső következtetéshez az összes ágon végig kell mennie. (Nincsenek felesleges

ágak.) A másik változatában a konjunkciók helyett diszjunkciók vannak. Ezt nem lehet kifejezni csak a elsőrendű logikát támogató rendszerekben. A megoldáshoz az összes szabályt fel kell használnia, nincs felesleges szabály benne.
A konjunkciós változat paramétere 6, míg a diszjunkciósé 3, illetve 6 volt.

3.6 Kizárás

Egy egyenlőséget is használó teszt, mely arról szól, hogy ha egy osztály két elemből áll, két elemet megadunk, hogy az osztály elemei, kikötjük, hogy a két elem nem egyenlő, és állítjuk, hogy nem az egyik elemre vagyunk kíváncsiak. Ekkor megkapjuk-e a másik elemet válaszként? (Illetve a másik elemre igaz választ kapunk-e?)

3.7 Az OWL következtető motorok tesztjei

A http://www.w3.org/TR/owl-test/ webcímen elérhető a W3C konzorcium ajánlása az OWL ontológianyelvben következtető motorok teszteléséhez. Ez egy nagy tesztgyűjtemény; azért esett rá a választásunk, mert eleve egy ontológianyelvhez készült. A tesztek közül kiválogattuk azokat, melyek valóban a motort tesztelik (ugyanis egy jelentős részük arra ment rá, hogy a motor felismeri-e a szintaktikailag helytelen OWL file-okat, és jól sorolja-e be a Lite, DL, Full osztályok valamelyikébe – ez jelenleg nem fontos kérdés). Ez 57 különböző tesztet jelent, melyek között vannak könnyebbek és nehezebbek is szép számmal. A kivitelezéshez kibővítettük a logikai frameworköt egy olyan segédosztállyal, mely OWL-es konstruktumokat fordít elsőrendű logikába.
A teszteket kétféleképpen hajtottuk végre. Először minden tesztadatot hozzátettünk a rendszerhez, majd az egyikhez tartozó kérdést feltettük a következtető rendszerhez. A másik méréskor már egyenként, csak az adott tesztet és a kérdést adtuk át a következtetőnek és ehhez is meghatároztunk időeredményeket (sajnos ez utóbbi tesztet már nem hajtottuk végre az EPILOG esetében, mivel nagyon kényelmetlen volt a tesztelése).

3.8 TPTP-s tesztek

Nem az összes tesztet futtattuk, mivel nagyon sok tesztet tartalmaz. Azonban igyekeztünk úgy válogatni, hogy a példák hasonlítsanak az esetleges ontológiai nyelveken leírt tudáshoz. Néhány olyan tesztet is választottunk, amelyek nem kapcsolódnak hozzá, azonban a nagy adathalmazon következtetési képességeket mérte.
A TPTP verziószáma: 3.0.1
Az állománynevek végződése:
+1 – a teszt nem feltétlenül van klóz normál formában.
-1 – a teszt CNF-ben van.
KRS:
Tudásreprezentáló rendszerekkel kapcsolatos tesztek.
MGT:
Vezetéssel (irányítással, management) kapcsolatos tesztek.
NLP:
Természetes nyelv feldolgozáshoz kapcsolódó tesztek.
PLA:
Tervezéssel kapcsolatos tesztek.
PUZ:
Rejtvények, fejtörők.
SYN:
Mesterséges tesztek, melyek a kivételes eseteket, valamint esetleges tipushibákat és teljesítményt mérnek. (Nem kapcsolható hozzájuk jelentés.)

Ez EPILOG-ot ezen a teszten már nem futtattuk a tesztelés kényelmetlen volta miatt. (Valószínű, hogy ezeken a teszteken sem teljesített volna jobban a többi egyenlőséges elsőrendű nyelvet használó rendszernél.) Szintén kihagytuk a SNePSLOG-ot, Carine-t és a JTP-t az egyenlőség kezelésének hiánya miatt.

4 Példa a következtetés irányára

Ebben a részben egy példán keresztül igyekszünk bemutatni a különbséget a három megközelítés között. A példa egy kicsit csal, mert azt sugallja, hogy a kétirányú keresés esetén mindig az optimális munkát végezzük, azonban ez nincs így. Akár több munkát is végezhet, mint a másik kettő, ha rossz a heurisztikája, ám azoké megfelelő.
Az alábbi példán vezetjük végig a következtetést:

Szabályok

1. ∀x: ember(x)→halandó(x)
2. ∀tej(x)→¬kőszikla(x)
3. ∀y: halandó(x)→¬kőszikla(x)
4. ember(Arisztotelész)
5. ember(Platón)
6. ember(Püthagorasz)
7. férfi(Szókratész)
8. tej(vaj)
9. tej(kefír)
10. tej(tejföl)
11. tej(joghurt)
12. ∀x: nő(x)→ember(x)
13. ∀x: férfi(x)→ember(x)

Kérdés

kőszikla(Szókratész)?

4.1 Előre következtetés

4, 1 ⇒ halandó(Arisztotelész) (14)
5, 1 ⇒ halandó(Platón) (15)
6, 1 ⇒ halandó(Püthagorasz) (16)
8, 2 ⇒ ¬kőszikla(vaj) (17)
9, 2 ⇒ ¬kőszikla(kefír) (18)
10, 2 ⇒ ¬kőszikla(tejföl) (19)
11, 2 ⇒ ¬kőszikla(joghurt) (20)
13, 3 ⇒ ¬kőszikla(Arisztotelész) (21)
14, 3 ⇒ ¬kőszikla(Platón) (22)
15, 3 ⇒ ¬kőszikla(Püthagorasz) (23)
7, 13 ⇒ ember(Szókratész) (24)
24, 3 ⇒ ¬kőszikla(Szókratész)

4.2 Hátra következtetés

kőszikla(Szókratész)?
2 – tej(Szókratész)?
8, 9, 10, 11, 12 – ¬tej(Szókratész) (14)
3 – ember(Szókratész)?
12 – nő(Szókratész)? (15)
1-13 – ¬nő(Szókratész)
13 – férfi(Szókratész)?
7, 13 ⇒ ember(Szókratész) (16)
16, 3 ⇒ ¬kőszikla(Szókratész)

4.3 Vegyes következtetés

Hátra	*Előre*
kőszikla(Szókratész)?	7, 13 ⇒ ember(Szókratész) (14)
tej(Szókratész)?	14, 3 ⇒ ¬kőszikla(Szókratész)
Ember(Szókratész)?	

VIII Felhasznált irodalom

G. Aguilera, I. P. de Guzmán, M. Ojeda-Aciego, A. Valverde: *Reductions for Non-Clausal Theorem Proving*, 2001.

Alessandro Armando, Silvio Renise, Michaël Rusinowitch: *A Superposition Based Methodology to Design Satisfiability Decision Procedures*, 2001

Peter Baumgartner: *Hyper Tableaux – the Next Generation*, 1998.

Robert S. Boyer, Matt Kaufmann, J. Strother Moore: *The Boyer-Moore Theorem Prover and Its Interactive Enhancement*, 1993

Richard Fikes, Gleb Frank, Jessica Jenkins: *JTP: A System Architecture and Component Library for Hybrid Reasoning*, 2003

James Garson: *Modal Logic*, 2003

Hashim Habiballa: *Non-Clausal Resolution Theorem Proving for Desceiption Logic*, 2005.

Chung Hee Hwang, Lenhart K. Schubert: *Episodic Logic: A Situational Logic for Natural Language Processing*, 1993

William McCune: *Otter 3.3 Reference Manual*, 2003

Robert A. Orchard: *NRC FuzzyJ Toolkit for the Java(tm) Platform User's Guide*, 2003

Frederic Portoraro: *Automated reasoning*, 2001

Stuart Russel, Peter Norvig: *Mesterséges intelligencia modern megközelítésben*, 2000

Stuart C. Shapiro: *SNePS 2.6.1 User's Manual*, 2004

Stephanie Schaeffer, Chung Hee Hwang, John de Haan, Lenhart K. Schubert: *EPILOG: The Computational System for Episodic Logic PROGRAMMER'S GUIDE*, 1993

Stephan Schulz: *E 0.8x User Manual*, 2004

Stickel, M.E. *A Prolog technology theorem prover: a new exposition and implementation in Prolog*. Theoretical Computer Science 104 (1992), 109-128.

Christoph Weidenbach: *The Theory of SPASS Version 2.0*, 2002