Hatékony következtetés ontológiákon

Zombori Zsolt, *zombori@cs.bme.hu*
Lukácsy Gergely, *lukacsy@cs.bme.hu*
Szeredi Péter, *szeredi@cs.bme.hu*

Budapesti Műszaki és Gazdaságtudományi Egyetem
Számítástudományi és Információelméleti Tanszék

2008. május 29.

Bevezetés

Ebben a dolgozatban azt vizsgáljuk, hogy hogyan lehet a világhálón történő információkeresést hatékonyabbá, automatikusabbá tenni. Az 1. fejezetben röviden vázoljuk, hogy milyen elven működnek a webes keresőrendszerek, majd a 2. fejezetben bemutatjuk a *szemantikus web* koncepciót, mely a világhálón levő tartalomhoz metainformációt rendelve próbálja elősegíteni a gépi megértést. A 3. fejezetben áttekintjük, hogy hogyan lehet a metainformáció segítségével felépített webontológiákon következtetést végezni.

1. Keresés a világhálón

Napjainkban egyre kisebb probléma a kívánt információhoz való hozzájutás, viszont komoly kihívást jelent a rendelkezésre álló hatalmas információtömegből a számunkra releváns kiválasztása. Ez különösen igaz a világhálóra, ahol bárki bármit közzétehet, és az elérhető adatforrásoknak egészen apró töredékét is reménytelenül sokáig tartana végignézni.

A webes keresőrendszerek összegyűjtik azon oldalakat, melyek illeszkednek az általunk megadott keresési mintára. Sajnos még ennek végignézése sem lehetséges, egy tipikus lekérdezés több millió találatot ad. A keresők a lapok sorrendezésével próbálják megkönnyíteni dolgunkat: előre hozzák azon találatokat, melyeket a legrelevánsabbaknak tartanak.

Ha a felhasználó egy kérdésre keres választ, kitalálja, hogy milyen keresési mintával fog olyan oldalakat kapni, melyekből majd ki tudja nyerni a kívánt információt. A találatokat a gép sorrendezi, de nem tudja, hogy ténylegesen miről szól egy oldal, csupán ügyesen kihasznál néhány esetleges sajátosságot: például hogy ha egy oldalra sok hivatkozás történik, akkor releváns, vagy ha a keresett szó az oldal címében fordul elő, akkor az nagyobb relevanciára utal, mint ha egy lábjegyzetben lenne. A sorrendezési heurisztikák jól működnek, de ez jelentős mértékben köszönhető annak, hogy a világhálón levő információ nagyon redundáns, és ha el is veszítjük a releváns találatok többségét, még

így is jó eséllyel használható választ kapunk. Erre azonban nem lehet mindig számítani.

A keresés során az összes értelmezési feladat a felhasználó személyre hárul: neki kell kitalálnia, hogy a keresett információ milyen szintaktikai mintára illeszkedő dokumentumokban fog előfordulni, a visszaadott adatokat át kell olvasnia, hogy kinyerje a szükséges információt, illetve hogyha a válasz több dokumentumban elszórt információmorzsákból áll össze, akkor neki kell ezen morzsákat integrálnia a fejében. A 2. fejezetben a szemantikus világháló koncepció keretében azt vizsgáljuk, hogy hogyan lehetne a számítógépeket az értelmezési feladatokba is bevonni, ezáltal jelentősen könnyítve a felhasználók dolgát.

2. Szemantikus világháló és ontológiaépítés

Az információkeresést nagyban könnyítené, ha a számítógépek valamennyire értelmezni tudnák a világhálón levő tartalmat. A szemantikus világháló olyan irányzata a számítástudománynak, mely gépek által feldolgozható információval próbálja kiegészíteni a weboldalakat, ezáltal elősegítve az adatforrások összekapcsolását és rendszerezését illetve az automatikus információkeresést. A weboldalakhoz metainformációt társítunk, mely az oldal jelentéséről mond valamit a gép számára. Az adatforrások információtartalmát ezáltal egységesen tudjuk kezelni, és ontológiákat építhetünk belőlük. Az ontológiák leírják egy fogalomrendszeren belüli kapcsolatokat, függőségeket valamint a tárgyterületet benépesítő egyedek sajátosságait. A fogalmi kapcsolatok révén lehetőség nyílik egyedek olyan tulajdonságainak meghatározására, melyek sehol nem jelennek meg expliciten.

A következőkben vázolunk egy módszert webes tartalom címkézésére és ontológiákba szervezésére.

2.1. Webes tartalom címkézése, az RDF koncepció

A világhálón rengeteg entitás található, melyekre együttesen erőforrásként hivatkozunk. Erőforrás lehet egy hírportál, egy weblap, annak egy része, de akár az a macska is, melyről az egyik oldal egyik bekezdése ír. Fontos azonban, hogy mindig egyértelmű legyen, hogy milyen entitásról beszélünk, így minden erőforrás rendelkezik (esetleg több) *Erőforrás Azonosítóval (Unified Resource Identifier, URI)*, melynek segítségével hivatkozhatunk rá.

2.1.1. RDF

Az RDF elképzelés azt próbálja megragadni, hogy az erőforrások nem csak önmagukban léteznek, hanem különféle kapcsolatok vannak közöttük. A kapcsolatokat rendezett hármasok segítségével fejezzük ki: az első elem egy erőforrás, melyre az adott állítás vonatkozik, a második elem egy tulajdonság, mely teljesül az első elemre (ez is erőforrás), végül a harmadik elem az állítás tárgya, mely lehet erőforrás vagy egy egyszerű literál (karakterlánc). Például a

```
[Dávid, ellenfele, Góliát]
```

hármas azt fejezi ki, hogy a 'Dávid' erőforrás a 'Góliát' erőforrással 'ellenfele' kapcsolatban áll. Az állításban szereplő erőforrások tetszőleges más hármasban

szerepelhetnek, így mindenféle egyéb tulajdonságuk lehet. Leírhatjuk, hogy Dávid fia Salamon (`[Dávid, fia, Salamon]`), hogy az 'ellenfele' erőforrás egy szimmetrikus reláció (`[ellenfele, típusa, szimmetrikus]`) vagy akár azt, hogy Góliátnak rövid élete volt (`[Góliát, élethossza, rövid]`). Ezekben a példákban az erőforrásokra beszédes nevekkel hivatkoztunk, de akármilyen azonosító használható. Azt, hogy egy erőforráshoz valamilyen konkrét név tartozik, újabb hármassal írjuk le, melyben a harmadik elem literál. Például az eddig 'Dávid'-ként hivatkozott erőforrásról kijelenthetjük, hogy az ő neve "Parittyás Dávid" és életkora 15:

```
[Dávid, neve, "Parittyás Dávid"]

[Dávid, életkora, "15"]
```

Itt most a 15-ös számot literálnak vettük. Ha viszont erről a számról ki akarnánk jelenteni, hogy páratlan, akkor létre kellene hozni egy olyan erőforrást, melynek 'számértéke' "15" és melynek 'paritása' "páratlan". Literálokról nem lehet további kijelentéseket tenni, így modellezési kérdés, hogy melyek azok az atomi építőelemek, melyeket nem akarunk tovább elemezni.

Egy erőforrás önmagában nem bír jelentéssel (hiszen csak egy azonosító tartozik hozzá), ám hármasok segítségével tetszőlegesen szövevényes kapcsolatrendszert építhetünk fel, és az ebben elfoglalt helye alapján már sok mindent el lehet róla mondani. Az RDF nyílt világot feltételez, azaz abból, hogy valami nincs kijelentve, még nem következik a negáltja. Tehát "bárki mondhat bármit", és folyamatosan jelenhetnek meg új erőforrások illetve új kapcsolatok, és csak azt tekintjük igaznak, mely igaz marad bármilyen későbbi bővítés esetén is.

Ahhoz, hogy ténylegesen "megértsünk" egy RDF leírást és következtetésre tudjuk felhasználni, legalább néhány erőforrásról kell, hogy rendelkezzünk valamilyen a priori ismerettel. Nézzük az alábbi állításokat:

```
[Dávid, ellenfele, Góliát]

[ellenfele, része, tiszteli]
```

azaz képzeljünk el egy olyan lovagias világot, melyben az ellenfelek tisztelik egymást. Szeretnénk kikövetkeztetni, hogy Dávid tiszteli Góliátot. Általános esetben ezt nem tehetjük meg, hiszen nincs semmi garancia, hogy aki a második kijelentést tette, az ugyanúgy értelmezi a 'része' kapcsolatot, mint mi. Bizonyos erőforrások jelentését ezért érdemes rögzíteni.

Az RDF szabvány tartalmaz néhány meghatározott jelentésű erőforrást, melyekről részletesen olvashatunk [4]-ben. Mi most csak egy, gyakran használt és kitüntetett fontosságú erőforrást említünk meg: az `rdf:type` segítségével írhatjuk le, hogy egy erőforrás példánya egy osztálynak. Például azt az állítást, hogy Pistike okos, az alábbi hármas fejezi ki:

```
[Pistike, rdf:type, okos]
```

Azonban még mindig nagyon kevés eszköz van a kezünkben érdemleges következtetéshez.

2.1.2. RDF séma

Az RDF kitejesztéseként jött létre az *RDF séma (RDFS)*, mely nagyon leegyszerűsítve csupán annyi változást hoz, hogy bevezetünk néhány meghatározott

jelentéssel bíró erőforrást. Például két reláció közti tartalmazás leírására definiáljuk az `rdfs:subPropertyOf` erőforrást, mely a fenti példában a 'része' erőforrást helyettesíti. Ha tehát azt írjuk, hogy

```
[Dávid, ellenfele, Góliát]

[ellenfele, rdfs:subPropertyOf, tiszteli]
```

akkor mindenki számára egyértelmű, hogy az 'ellenfele' reláció a 'tiszteli' reláció része és ez alapján már tényleg ki lehet jelenteni, hogy Dávid tiszteli Góliátot. Ismét a teljesség igénye nélkül bemutatunk néhány RDFS erőforrást:

- Az `rdfs:Class` erőforrás egy osztály, melynek elemei éppen az osztályok. Segítségével kijelenthetjük például, hogy a 'Szép' erőforrás egy osztály: `[Szép, rdf:type, rdfs:Class]`[1].
- Az `rdfs:subClassOf` tulajdonság, azt fejezi ki, hogy egy osztály része egy másik osztálynak. Az `[Okos, rdfs:subClassOf, Sikeres]` hármas azt állítja, hogy mindenki, aki okos az egyben sikeres is.
- Az `rdfs:subPropertyOf` tulajdonságról már korábban láttuk, hogy tulajdonságok közti tartalmazást fejez ki. Például azt, hogy a barátok egyben ismerősök is a `[barátja, rdfs:subPropertyOf, ismerőse]` hármassal írhatjuk le.
- Az `rdfs:domain` és `rdfs:range` tulajdonságok segítségével megadhatjuk, hogy egy tulajdonság milyen típusú erőforrásokat köthet össze. Ha elő akarjuk írni, hogy barátság csak emberek között létezik, azt az alábbi két hármassal fejezzük ki: `[barátja, rdfs:domain, ember]`, `[barátja, rdfs:range, ember]`.

Az RDF Séma segítségével osztályok és tulajdonságok (relációk) hierarchiáját írhatjuk le. Ezt felhasználva egy kereső képes arra, hogy a sport kérdésre a fociról szóló cikkeket is visszaadja, vagy Magyarországra keresve megtalálja Debrecen honlapját. Azonban a nyelv továbbra is inkább csak tudásleíró keretrendszernek tekinthető, mely az erőforrások közti bonyolultabb kapcsolatokat nem fejezi ki.

Egy RDF illetve RDFS leírás logikailag hármasok halmazának tekinthető, melynek fizikai leírására többféle szabvány létezik. A legelterjedtebb egy XML alapú formalizmus, mellyel kényelmesen el tudjuk helyezni az információt a világhálón.

2.2. Az OWL nyelv

A *Web Ontology Language (OWL)* egy RDF alapú nyelv, melyet kifejezetten arra fejlesztettek ki, hogy a világháló leíró nyelve legyen és mára már W3C szabvány. Az RDFS-nél alkalmazott irányzatot követi, és további erőforrásokat definiál, melyek tisztázott, szabványos jelentéssel rendelkeznek. Ezáltal bizonyos logikai kapcsolatok leírásakor számíthatunk arra, hogy mindenki egységesen fogja őket értelmezni.

[1] Egy erőforrás osztály voltát már az RDF segítségével is ki lehetett fejezni impliciten úgy, hogy kijelentettük valamilyen egyedről, hogy eleme az erőforrásnak. Most viszont lehetőség van csak a metainformáció leírására, anélkül, hogy konkrét elemet adnánk.

Az OWL XML alapú és egy érvényes OWL dokumentum egyben érvényes RDF dokumentum is. A szintaxisra itt most nem térünk ki (lásd [4]), de a 3. fejezetben bemutatjuk, hogy milyen logikai tartalmat lehet segítségével kifejezni.

3. Leíró logikai következtetés ontológiákhoz

Az OWL szintaxisának kialakulásakor fontos szempont volt, hogy legyen egy jól használható logikai háttér, melyben tárgyalni lehet az ontológiákat. Ezt a hátteret nyújtják a *leíró logikák*, melyet a 3.1. szakaszban röviden összefoglalunk.

Egy leíró logikai tudásbázist szeretnénk következtetésre felhasználni, azaz el szeretnénk tudni dönteni, hogy milyen állítások következnek belőle. A 3.2. és a 3.3. szakaszokban bemutatunk két következtetési módszert, majd a 3.4. szakaszban vázoljuk az általunk készített DLog következtető rendszer működési elvét.

3.1. Leíró logikák

A nyelvcsalád kétfajta építőelemet használ: vannak *fogalmak* és a *szerepek* melyek az RDF-nél korábban megismert osztályoknak illetve tulajdonságoknak (bináris relációknak) feleltethetőek meg.

Egy leíró logikai állítás vagy egy szabályszerűséget fejez ki (*terminológiai állítás*), vagy pedig konkrét egyedekről mond valamit (*adatállítás*). A tudásbázis így két jól elkülöníthető részre bontható: egy *terminológiai dobozra* és egy *adatdobozra*.

Egyszerű fogalmakból lehetőség van összetett fogalmak képzésére. Hogy pontosan milyen konstrukciók engedélyezettek, függ az aktuális leíró logikai nyelvtől.

Az $\mathcal{ALC}$ nyelvben lehetőség van fogalmak *uniójának* ($C \sqcup D$), *metszetének* ($C \sqcap D$) és *komplemensének* ($\neg C$) képzésére, valamint két további konstrukcióra, mely a fogalomba tartozó egyedek más egyedekhez fűződő viszonyát korlátozza. Ha egy egyedhez az R tulajdonságon keresztül kapcsolódik egy másik egyed, akkor azt az ő *R-követőjének* hívjuk. *Értékkorlátozással* ($\forall R.C$) olyan fogalom állítható elő, melyben az elemek összes R-követője a C fogalomba tartozik. A *létezési korlátozás* ($\exists R.C$) pedig azt fejezi ki, hogy minden elemnek van olyan R-követője, mely C-be tartozik. A nyelvhez tartozik két kitüntetett fogalom: a *top* ($\top$), mely az ontológia összes elemét tartalmazza, valamint a *bottom* ($\bot$), ami az üres fogalomnak felel meg.

Fogalomtartalmazási axiómák ($C \sqsubseteq D$) segítségével leírhatjuk a fogalmak hierarchikus kapcsolatrendszerét, kifejezve olyan bölcsességeket, mint például hogy akinek minden gyereke boldog az maga is boldog, vagy aki gazdag, annak létezik gazdag őse.

Számtalan egyéb nyelvi kiterjesztés létezik ([1]), ezekre nevük kezdőbetűjével szokás hivatkozni. A következőkben felsoroljuk a legfontosabbakat:

- $\mathcal{S}$: egy szerepről kijelenthetjük, hogy tranzitív ($Trans(R)$).
- $\mathcal{H}$: a szerepek hierarchiába rendezhetőek, akár a fogalmak ($R \sqsubseteq S$).
- $\mathcal{I}$: az inverz szerepkonstruktor segítségével hivatkozhatunk tetszőleges szerep inverzére (R^-).

- $\mathcal{O}$: definiálhatunk úgynevezett *nominálisokat*, melyek pontosan egy elemmel rendelkező fogalmak.
- $\mathcal{N}$: megmondhatjuk, hogy egy egyednek legfeljebb vagy legalább mennyi R-követője van, ahol R valamilyen szerep. Ezt a konstrukciót egyszerű számosságkorlátozásnak nevezzük ($\leq nR$, $\geq nR$).
- $\mathcal{Q}$: az előző konstruktor általánosítása a minősített számosságkorlátozás. Nem csak az R-követők számát korlátozzuk, hanem a valamilyen C tulajdonsággal rendelkező R-követők számát. Tehát nem csak azt mondhatjuk, hogy a gazdagoknak legfeljebb két gyereke van, hanem azt is, hogy legfeljebb 1 gyerekük sikeres ($\leq nR.C$, $\geq nR.C$).

Az OWL a $\mathcal{SHOIN}$ nyelvet valósítja meg, azaz szerepekről kijelenthető tranzitivitásuk, építhetünk szerephierarchiát, használhatjuk az inverz műveletet mint egyszerű szerepkonstruktort és csak az egyszerű számosságkorlátozás megengedett.

Gyakorlati fontossága miatt meg kell még említeni a $\mathcal{SHIQ}$ nyelvet, mely kicsit szűkebb is (nincsenek benne nominálisok) és kicsit bővebb is (minősített számosságkorlátozás megengedett), mint az OWL nyelve.

3.2. A tabló algoritmus

A *tabló algoritmus* a legáltalánosabban használt következtetési eljárás leíró logikákhoz. Alapváltozata egy fogalom kielégíthetőségét dönti el azáltal, hogy megpróbál modellt építeni hozzá. Az algoritmus véges időben befejeződik vagy úgy, hogy előállítja a tudásbázis egy olyan modelljét, melyben a vizsgált fogalom nem üres (ekkor a fogalom kielégíthető), vagy pedig, hogy kimerítő módon megvizsgálja az összes lehetséges modelljét az adott fogalomnak és belátja, hogy mindegyik ellentmondásos. Az algoritmus csekély módosítással felhasználható adatkövetkeztetésre is, ekkor azt határozza meg, hogy a tudásbázis konzisztens-e.

A leggyakoribb adatkövetkeztetési feladat a példénykikeresés, vagyis mikor egy C fogalomról el szeretnénk dönteni, hogy mely egyedekre vonatkozik. Ilyenkor vesszük a tudásbázisban előforduló összes a egyednevet egyesével, és a $\neg C(a)$ állítást hozzávéve az axiómákhoz, a tabló algoritmussal eldöntjük, hogy a kapott halmaz konzisztens-e. Amennyiben ellentmondásra jutunk, akkor $C(a)$ következik a tudásbázisból.

Nagy adathalmazok esetén ez a következtetési módszer komoly nehézségekbe ütközhet, mivel egyesével meg kell vizsgáljuk az összes egyedet, hogy eldöntsük, beletartozik-e az adott fogalomba. A világhálón történő keresés esetén ez azt jelenti, hogyha például a magyar focistákra vagyunk kíváncsiak, akkor az összes Weben előforduló egyedet meg kell vizsgálni! Erre a gyakorlatban nincs lehetőség. Adatkövetkeztetési feladatokhoz tehát a Tabló algoritmus nem mindig használható.

3.3. Rezolúció alapú következtetés

A leíró logikai nyelvek részét képezik az elsőrendű logikának. Így nem okoz problémát közvetlenül átalakítani az axiómákat elsőrendű formulákká. Ezáltal

használhatunk valamilyen elsőrendű tételbizonyító eljárást, mint például rezolúciót.

Ennek a megközelítésnek az az előnye, hogy már nagyon régóta foglakoznak elsőrendű tételbizonyítással, jól kiforrt és optimalizált módszerek léteznek, melyeket felhasználhatunk. Hátránya azonban, hogy az elsőrendű logika nem eldönthető, így általános esetben semmilyen elsőrendű tételbizonyító eljárásról nem garantálható, hogy valaha is befejeződik. Könnyen elveszíthetjük a szűkebb résznyelv nyújtotta előnyöket.

Egy kis trükköt alkalmazva azonban mégis végessé lehet tenni a rezolúció menetét. A tudásbázist át lehet alakítani egy olyan ekvivalens axiómahalmazra, melyeket lefordítva elsőrendű logikára, jól meghatározott struktúrájú formulákat kapunk ([3]). A tudásbázis jelentése nem módosul, csupán formai változást eszközölünk. Az eredő formulák szabályos struktúrájának köszönhetően meg lehet határozni egy olyan véges méretű formulahalmazt, mely tartalmazza a kiinduló formulák minden lehetséges következményét, így garantáltan véges időben befejeződik a futás. Ezt a módszert alkalmazza a következő fejezetben bemutatásra kerülő DLog rendszer valamint a KAON2 következtető ([3]).

3.4. A DLog rendszer

A DLog egy saját fejlesztésű következtető, mely a $\mathcal{SHIQ}$ leíró logikai nyelvet támogatja, és részletes leírása megtalálható [2]-ben és [5]-ben. A program Prolog nyelven íródott. Először a 3.3. szakaszban említett transzformáció segítségével átalakítjuk a tudásbázist elsőrendű klózhalmazzá. Az utána következő következtetés két jól elkülöníthető szakaszra bontható: egy terminológiai telítésre és egy adatkövetkeztetésre.

A terminológiai telítés során nem nyúlunk az adatokhoz, csak a tudásbázis szabályaival foglalkozunk és előállítjuk azok minden lehetséges következményét elsőrendű rezolúció felhasználásával. Ez egy igen lassú bottom-up következtetési fázis, de szerencsére nem kell nagyon nagy klózhalmazzal dolgozunk. A tudásbázis döntő részét az adatok teszik ki, szabályokból pedig kevesebb van: egy tipikus alkalmazás esetén néhány száz szabály már soknak számít, míg az adatállítások száma sok millió is lehet. Az első fázis végére a kapott klózok jelentős része nem vehet részt olyan következtetési lépésben, melyben adatok is megjelennek, így ezeket – mivel minden adatfüggetlen lépést már elvégeztünk – el lehet hagyni. Eredményül egy roppant egyszerű szintaxisú klózhalmazt kapunk.

Az eredő klózokból a DLog fordító modulja előállít egy Prolog programot, még mindig az adatok érintése nélkül. Ennek a programnak a futása fogja a tényleges adatkövetkeztetést elvégezni, kihasználva a Prolog roppant hatékony, top-down tételbizonyító mechanizmusát. A bizonyítás során csak azok az adatok kerülnek megvizsgálásra, melyek relevánsak a feltett kérdés szempontjából, így például ha a magyar focistákra vagyunk kíváncsiak, akkor nem fogunk időt vesztegetni olyan egyedre, melyről nincs kijelentve, hogy magyar vagy hogy focista – vagy valami más, amiről ezen két tulajdonságra lehetne következtetni.

A teszteredmények azt mutatták, hogy a DLog jelentősen gyorsabb, mint a többi leíró logikai következtető, amennyiben nagyon nagy adathalmazzal kell dolgozni. Ez elsősorban az adatok hatékony kezelésének köszönhető, vagyis hogy soha nem kell a teljes adathalmazt beolvasni, csak azokat, melyek ténylegesen befolyásolják a választ. Ez az architektúra lehetőséget ad arra, hogy az adatok

külső adatbázisokban legyenek tárolva és az adatkövetkeztetés során célzott lekérdezésekkel fordulhassunk hozzájuk. Webes alkalmazásoknál, ahol éppen az adatok óriási mennyisége a legjellemzőbb, különösen fontos ez a tulajdonság.

Összefoglalás

A világhálón lévő hatalmas adattömegben egyre nagyobb kihívást jelent a keresett információ megtalálása. A szemantikus web irányzat célja, hogy a számítógépek ne csak olvasni, hanem valamennyire értelmezni is tudják a webes tartalmat. Ezáltal lehetővé válik, hogy ne keresési mintákat, hanem kérdéseket tehessünk fel a keresőnek, ami aztán esetlegesen több forrás integrálásával és hosszas következtetés révén felkutatja a választ.

Az adatokhoz metaadatokat rendelünk, melyek szabványos és a gép által feldolgozható módon leírják, hogy az adat mire vonatkozik. Az RDF egy általános mechanizmus metainformáció leírására. Ennek kiterjesztéseként született meg az OWL nyelv, melynek segítségével a források információtartalma ontológiákba szervezhető.

Az OWL nyelven megfogalmazott állítások közvetlenül megfeleltethetőek leíró logikai állításoknak, így a leíró logikai következtetők felhasználhatóak a világhálón való keresés hatékonyabbá tételére. Példaként bemutattuk a DLog rendszer vázlatos felépítését, mely hatékony tud maradni igen nagy adathalmazokon történő következtetés esetén is.

Hivatkozások

[1] Ian Horrocks, Oliver Kutz, and Ulrike Sattler, *The Even More Irresistible SROIQ*, KR (Patrick Doherty, John Mylopoulos, and Christopher A. Welty, eds.), AAAI Press, 2006, pp. 57–67.

[2] Gergely Lukácsy and Péter Szeredi, *Efficient description logic reasoning in Prolog: the DLog system*, Tech. report, Budapest University of Technology and Economics, January 2008, Submitted to Theory and Practice of Logic Programming.

[3] Boris Motik, *Reasoning in Description Logics using Resolution and Deductive Databases*, Ph.D. thesis, Univesität Karlsruhe (TH), Karlsruhe, Germany, January 2006.

[4] Szeredi Péter, Lukácsy Gergely, and Benkő Tamás, *A Szemantikus Világháló Elmélete és Gyakorlata*, TypoTeX, Budapest, 2005.

[5] Zombori Zsolt, *Hatékony rezolúció alapú következtetés leíró logikákhoz nagy adatdobozok mellett*, Tdk dolgozat, BME VIK, Számítástudományi és Információelméleti Tanszék, Budapest, 2007.