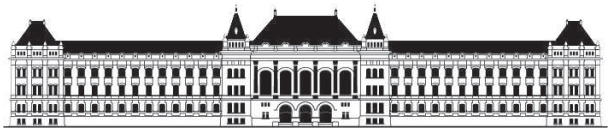


Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar
Számítástudományi és Információelméleti Tanszék

H-1117 Budapest
Magyar tudósok körútja 2. B-132.sz
Tel.: (36-1) 463-2585 Fax: (36-1) 463-3157

Prolog alapú következtetés

Zombori Zsolt

PhD tézisfüzet

Konzulens: Dr. Szeredi Péter

Számítástudományi és Információelméleti Tanszék
Villamosmérnöki és Informatikai Kar
Budapesti Műszaki és Gazdaságtudományi Egyetem
Budapest, Magyarország

Budapest, Magyarország
2013 Március

1. Bevezetés

A dolgozat fejezeteit összekötő bűvös szó az *okfejtés*, vagy *következtetés*. Következtetés alatt azt a képességet értjük, hogy a rendelkezésünkre álló tudás alapján levezessünk olyan igaz állításokat, melyek közvetlenül nem lettek megfogalmazva. Manapság már rengeteg olyan informatikai rendszer létezik, mely a tudást számítógépek számára kezelhető formában próbálja ábrázolni. Ezen rendszerek számára az automatikus következtetési támogatás különösen nagy jelentőségű: lehetővé teszi rejtett tudás valamint rejtett hibák feltárását. Emellett automatikus következtetés segítségével komplex felhasználói kérdéseket tudunk megválaszolni.

Disszertációmban két tudományterülettel foglalkozom. Az egyik terület leíró logikai (Description Logic – DL) következtetés. Leíró logikák alatt egy nyelvcsaládot értünk, mely népszerű tudásreprezentációs formalizmus és széles körben használják ontológiák építésére. Különféle következtetési módszerek kifejlesztésével és vizsgálatával foglalkoztunk, melyek lehetővé teszik kérdések megfogalmazását és megválaszolását DL ontológiák alapján. Ezen túlmenően módszereink segítségével el lehet dönteni egy ontológia konzisztenciáját.

A másik terület funkcionális programozási nyelvek típuselemzéséhez kötődik. Feladatunk, hogy egy bemenetként kapott programról eldöntsük, hogy tartalmaz-e típushibát, és amennyiben igen, jelezzük a programozó felé. Az egyes módszereket a Q programozási nyelv kapcsán vizsgáltuk. Statikus típuselemzés segítségével számos programhibát tudunk felfedezni fordítási időben, ezáltal nagy mértékben megkönnyítve a programfejlesztés folyamatát.

E két terület első ránézésre egészen másnak tűnik. Azonban ami mégis összekapcsolja őket, az a mögöttük levő következtetési feladat: mindkét esetben valamilyen kezdeti tudásból indulunk ki (leíró logikai axiómákból az egyik esetben, egy bemeneti programból és különféle típuskorlátozásokból a másikban), és ez alapján próbáljuk levezetni a bemenet valamilyen tulajdonságát automatikus következtetés segítségével.

1.1. Probléma megfogalmazás

A következőkben összegzem kutatómunkám motivációját és a főbb célkitűzéseimet.

Leíró logikai adatkövetkeztetés nagyméretű adathalmazok esetén. Leíró logikákat gyakran használnak tudás reprezentálására. Bár mára már számtalan leíró logikai következtető módszer létezik, ezek a módszerek nagyon érzékenyek a tudásbázisban megjelenő adathalmaz méretére. Számos alkalmazási területen azonban – mint például webes környezetben történő következtetés esetén – hatalmas adatmennyiséggel kell megbírkózni. Munkánk célja, hogy olyan új következtető módszereket keressünk, melyek ilyen környezetben is jól használhatóak. Különösen fontos, hogy a módszer futását ne befolyásolja a feladat szempontjából irreleváns adatok jelenléte és az adatainkhoz közvetlen adatbázis lekérdezések formájában fordulhassunk.

1. Elsődleges célom, hogy leíró logikai axiómákat elsőrendű logikai klózokká alakítsam, melyek a lehető legegyszerűbb szerkezetűek. A kapott klózok lehetőleg ne tartalmazzanak függvényjeleket.
2. A módszernek támogatnia kell a $\mathcal{SHIQ}$ nyelvet, mely a legelterjedtebb leíró logikai variáns.
3. Miután elkészül egy algoritmus a $\mathcal{SHIQ}$ nyelvhez, az eredményeket ki kell terjeszteni egyéb gazdagabb nyelvi elemek felé. Különösen hasznos lenne komplex szerephierarchiák támogatása, melyek gyakran megjelennek orvosi ontológiákban.
4. A kifejlesztett módszereket szeretném implementálni a DLog leíró logikai adatkövetkeztető rendszerben.

Optimalizált PTTP végrehajtás. A Prolog Technology Theorem Prover (PTTP) egy teljes és helyes elsőrendű tételbizonyító eljárás, mely a Prolog nyelvre épül. Ez a módszer fontos szerepet játszik a DLog rendszerben, így minden optimalizációs eljárás nagy mértékben meg tudja növelni a DLog hatékonyságát.

1. A legtöbb PTTP megvalósítás alkalmaz egy hurokkiküszöbölés nevű optimalizációs eljárást. Azonban ennek az eljárásnak a helyessége még nem lett bebizonyítva. Célom, hogy egy precíz bizonyítással igazoljam, hogy a hurokkiküszöbölés alkalmazása nem vezethet megoldásvesztéshez.

Statikus típuselemzés a Q funkcionális nyelvhez. A Q egy funkcionális, dinamikus típusrendszerű programozási nyelv, melynek roppant tömör és kivételekkel teli szintaxisa (és szemantikája) van. A Q nyelvet széles körben használják pénzügyi alkalmazásoknál, azonban nincs semmilyen beépített hibakeresési támogatás hozzá, ezáltal a programfejlesztés roppant nehézkes. Munkám célja, hogy elkészítsek egy eszközt a Q nyelv elemzésére, mely fordítási időben fedez fel típushibákat a programban.

1. Első lépésben fel kell térképezni a statikus típuselemzés megvalósíthatóságát a Q nyelv esetében. Ez magába foglalja azon korlátok meghatározását, melyeket a programozóknak be kell tartaniuk ahhoz, hogy egy típuselemző hatékonyan tudjon működni.

2. Következő lépésben meg kell tervezni egy olyan algoritmust, mely eldönti, hogy a programozó által a kódban elhelyezett típusinformáció helyes-e.

3. Ezután a feladat egy típuskikövetkeztető algoritmus készítése, mely nem vár semmilyen típusinformációt a programozótól, hanem közvetlenül a kód alapján próbálja kitalálni az egyes kifejezések típusát.

4. Az elkészült algoritmusokat imlementálni kell egy olyan eszközben, melyet partnerünk, a Morgan Stanley Business and Technology Centre, Budapest ipari környezetben is megbízhatóan tud használni.

1.2. Tézisfüzet áttekintés

A 2. fejezetben két olyan kalkulust mutatunk be, melyek segítségével el lehet dönteni egy leíró logikai tudásbázis konzisztenciáját. Az első, melyet *módosított kalkulus*-nak nevezünk, az elsőrendű rezolúció finomítása és az $\mathcal{ALCHIQ}$ DL nyelvet támogatja. Megmutatjuk, hogy hogyan lehet ezen kalkulus segítségével egy kétfázisú adatkövetkeztetést megvalósítani, ami jól skálázódok az adathalmaz növekedésével. Ez az eredmény könnyen kiterjeszthető a $\mathcal{SHIQ}$ nyelvre jól ismert módszerek segítségével.

Ezután egy olyan transzformációt mutatunk be, melynek segítségével egy $\mathcal{RIQ}$ nyelvű tudásbázisból egy kielégíthetőség szempontjából ekvivalens $\mathcal{ALCHIQ}$ nyelvű tudásbázist kapunk. A transzformációnak köszönhetően a következtetést el tudjuk végezni a módosított kalkulus segítségével. Az eredményeket összerakva egy olyan algoritmust kapunk, mely alkalmas $\mathcal{RIQ}$ nyelvi következtetésre és jól skálázódik az adathalmaz növekedésével.

A fejezet végén egy új kalkulust mutatunk be, melynek neve *DL kalkulus*. Itt a következtetési szabályok közvetlenül leíró logikai kifejezéseken vannak definiálva, megspórolva a köztes fordítási lépéseket elsőrendű logikára. A DL kalkulus segítségével el tudjuk dönteni, hogy egy $\mathcal{SHQ}$ terminológiai tudásbázis konzisztens-e.

A 3. fejezetben bemutatjuk a *hurokkiküszöbölés* nevű optimalizációs eljárást. Segítségével elkerülhetőek bizonyos végtelen ciklusok a következtetés során. A hurokkiküszöbölés a legfontosabb optimalizáció PTTP alapú következtetés esetén.

A 4. fejezetben olyan következtető módszereket mutatunk be, melyeket Q nyelvű programok statikus típuselemzéséhez készítettünk. Először egy típusellenőrző módszert ismertetünk, mely felhasználói típusdeklarációk helyességét ellenőrzi. Ezután egy típuskikövetkeztető algoritmust vizsgálunk, melynek segítségével akkor is fel tudunk fedezni típushibákat, ha semmilyen felhasználó által megadott típusinformáció nem áll rendelkezésünkre.

Az 5. fejezetben bemutatjuk a DLog leíró logikai adatkövetkeztető rendszert, melynek segítségével $\mathcal{RIQ}$ nyelvű tudásbázishoz intézett lekérdezéseket tudunk megválaszolni.

Végezetül a 6. fejezetben ismertetjük a `qtchk` típuselemző rendszert, mely Q programok típushelyességét vizsgálja.

1.3. Publikációs Lista

Folyóirat cikkek

1. [20] Zsolt Zombori: A Resolution Based Description Logic Calculus. In ACTA CYBERNETICA-SZEGED 19: pp. 571-588. (2010)

2. [29] Zsolt Zombori, Péter Szeredi: Loop Elimination, a Sound Optimisation Technique for PTTP related Theorem Proving. In ACTA CYBERNETICA-SZEGED 20: pp. 441-458. Paper 3. (2012)

3. [18] Zsolt Zombori: Expressive Description Logic Reasoning Using First-Order Resolution. Submitted to JOURNAL OF LOGIC AND COMPUTATION.

Lektorált nemzetközi konferencia kiadványok

1. [19] Zsolt Zombori: Efficient Two-Phase Data Reasoning for Description Logics. In Artificial Intelligence in Theory and Practice II: World Computer Congress 2008. Milano, Italy, 2008.09.07-2008.09.10. Springer, pp. 393-402.(ISBN: 978-0-387-09694-0)

2. [26] Zsolt Zombori Zsolt, Gergely Lukácsy: A Resolution Based Description Logic Calculus. In Proceedings of the 22nd International Workshop on Description Logics (DL2009). Oxford, UK, 2009.07.27-2009.07.30. pp. 27-30.

3. [22] Zombori Zsolt: Two Phase Description Logic Reasoning for Efficient Information Retrieval. In 27th International Conference on Logic Programming (ICLP'11): Technical Communications. Lexington, USA, 2011.07.06-2011.07.10. Wadern: Schloss Dagstuhl Leibniz-Zentrum für Informatik, pp. 296-300.(ISBN: 978-3-939897-31-6)

4. [21] Zsolt Zombori: Two Phase Description Logic Reasoning for Efficient Information Retrieval. In Extended Semantic Web Conference (ESWC) 2011: AI Mashup Challenge 2011. Heraklion, Greece, 2011.05.29-2011.06.02. pp. 498-502.

5. [30] Zsolt Zombori, Péter Szeredi, Gergely Lukácsy: Loop elimination, a sound optimisation technique for PTTP related theorem proving. In Hungarian Japanese Symposium on Discrete Mathematics and Its Applications. Kyoto, Japan, 2011.05.31-2011.06.03. Kyoto: pp. 503-512.

6. [23] Zsolt Zombori, János Csorba, Péter Szeredi: Static Type Checking for the Q Functional Language in Prolog. In 27th International Conference on Logic Programming (ICLP'11): Technical Communications. Lexington, USA, 2011.07.06-2011.07.10. Wadern: Schloss Dagstuhl Leibniz-Zentrum für Informatik, pp. 62-72.(ISBN: 978-3-939897-31-6)

7. [3] János Csorba, Zsolt Zombori, Péter Szeredi: Using Constraint Handling Rules to Provide Static Type Analysis for the Q Functional Language. In Proceedings of the 11th International Colloquium on Implementation of Constraint and LOgic Programming Systems (CICLOPS 2011). Lexington, USA, 2011.07.10. Paper 5.

8. [25] Zsolt Zombori, János Csorba, Péter Szeredi: Static Type Inference for the Q language using Constraint Logic Programming. In Technical Communications of the 28th International Conference on Logic Programming (ICLP'12): Leibniz International Proceedings in Informatics (LIPIcs). Budapest, Hungary, 2012.09.04-2012.09.08. Dagstuhl: pp. 119-129. Paper 8. (ISBN: 978-3-939897-43-9)

9. [4] János Csorba, Zsolt Zombori, Péter Szeredi: Pros and Cons of Using CHR for Type Inference. In Proceedings of the 9th workshop on Constraint Handling Rules (CHR 2012). Budapest, Hungary, 2012.09.04 Paper 4.

Hazai konferencia kiadványok

1. [27] Zsolt Zombori, Gergely Lukácsy, Péter Szeredi: Hatékony következtetés ontológiákon. In 17th Networkshop Conference 2008. Dunaújváros, Hungary, 2008.03.17-2008.03.19.

2. [24] Zsolt Zombori, János Csorba, Péter Szeredi: Static Type Inference as a Constraint Satisfaction Problem. In Proceedings of the TAMOP PhD Workshop: TAMOP-4.2.2/B-10/1-2010-0009. Budapest, Hungary, 2012.03.09.

Oktatási segédanyagok

1. [28] Zsolt Zombori, Péter Szeredi: Szemantikus és deklaratív technológiák oktatási segédlet. Course handout. 2012. Budapest, pp. 1-32.

2. Rezolúció alapú leíró logikai következtetés

Először a 2.1. alfejezetben bemutatunk egy rezolúciós kalkulust, mely az alap-szuperpozíció módosításából származik és $\mathcal{ALCHIQ}$ leíró logikai következtetésre lett specializálva. Ezután a 2.2. alfejezetben kiterjesztjük a korábbi eredményeket a $\mathcal{RIQ}$ leíró logikai nyelvre. Végezetül a 2.3. alfejezetben egy olyan kalkulust ismertetünk, melyben a következtetési szabályok közvetlenül leíró logikai kifejezéseken vannak definiálva.

2.1. Kétfázisú következtetés

Követjük azt az irányvonalat, mely megjelenik [12]-ben, vagyis a komplexitás kézben tartása érdekében a következtetést két részre bontjuk. Az első lépés egy adatoktól független fázis, melyben csak az általános háttértudást leíró axiómák szerepelnek. Ezeket hívjuk terminológiai axiómáknak, melyek összessége alkotja a Tdobozt. A TDoboz általában viszonylag kicsi, viszont egészen komplex összefüggések is megjelenhetnek benne. A második fázisban megjelennek a konkrét egyedekre vonatkozó adataink, melyek összessége alkotja az Adobozt. Az Adoboz sok alkalmazás esetében tetszőlegesen nagy lehet, ám szerkezete egyszerű. A két fázist terminológiai- illetve adatkövetkeztetésnek nevezzük. Ez a szétválasztás drasztikusan növeli a következtetés hatékonyságát. Amellett, hogy a Tdoboz elég összetett, a benne megjelenő tudás általában nem vagy csak nagyon ritkán változik, míg az adataink folyamatosan módosulhatnak. Így ha sikerül a kizárólag a Tdobozt érintő lépéseket előre hozni, akkor még egy komplikált és lassú következtető algoritmus (mely szükséges a Tdoboz komplexitása miatt) használata is elfogadható, hiszen a nagyméretű Adoboz nem lassítja a következtetést. Továbbá ezen lépéseket csak egyszer kell végrehajtani, mintegy előfeldolgozási fázisként. Ezt követően az adatkövetkeztetést már egy gyors és lekérdezés orientált algoritmus segítségével tudjuk elvégezni. Valahányszor egy lekérdezés érkezik a rendszerhez, csak a második fázist kell megismételni, hogy a válasz tükrözze az Adoboz aktuális állapotát.

A következtető bemenete egy $\mathcal{SHIQ}$ leíró logikai tudásbázis, melyről el szeretnénk dönteni, hogy konzisztens-e. Könnyen belátható, hogy az összes szokásos leíró logikai következtetési feladat visszavezethető erre a problémára.

Első lépésben az axiómáinkat lefordítjuk elsőrendű klózok halmazára. Némi odafigyeléssel garantálható, hogy a kapott klózok az elsőrendű klózok egy valódi, jól karakterizálható részhalmazába tartozzanak. Ezt a halmazt $\mathcal{ALCHIQ}$ *klózoknak* nevezzük. Ezután a klózokat telítjük az alap-szuperpozíciós kalkulus módosított változatával. A módosított kalkulus fő erénye, hogy segítségével minden olyan következtetési lépést, mely függvényszimbólumokat érint, el tudunk végezni az Adobozhoz való hozzáférés előtt.

Az eredeti $\mathcal{SHIQ}$ tudásbázis nem tartalmaz függvényjeleket. Azonban, mikozben az axiómákat elsőrendű klózokká alakítjuk, az egzisztenciális kvantifikáció kiküszöbölése érdekében új skolem függvényeket vezetünk be. Ez az Adobozt nem érinti, így minden a függvényekhez kötődő tudás a Tdobozban van. Miután telítettük a Tdobozt a módosított kalkulus segítségével, be lehet látni, hogy a függvényjeleket tartalmazó klózok nem játszanak már szerepet a későbbi lépések során, ezért ezeket teljesen el lehet hagyni. Ezáltal a következtetés második fázisa nagy mértékben egyszerűsödik.

Propozíció 1. *A módosított kalkulus helyes és teljes. Igaz továbbá, hogy tetszőleges $\mathcal{ALCHIQ}$ klózhalmaz telítése véges sok lépésben befelyeződik.*

2.1.1. Kétfázisú következtetés megvalósítása

A módosított kalkulus segítségével a kövekeztetést két lépésben valósítjuk meg, melyet az 1. algoritmusban foglalunk össze. Az (1) - (3) lépések alkotják az első fázist, a (4) lépés pedig az adatkövetkeztetés. Fontos azonban, hogy az adatkövetkeztetéshez nem szükségszerű a módosított kalkulus alkalmazása: bármilyen olyan következtetési módszer, mely hatékonyan ki tudja használni, hogy nincsenek a klózokban többé függvényjelek, alkalmazható.

Algorithm 1 $\mathcal{SHIQ}$ következtetés

1. A $\mathcal{SHIQ}$ Tdobozt átalakítjuk $\mathcal{ALCHIQ}$ klózokká.
2. A klózokat telítjük a módosított kalkulus segítségével.
3. Elhagyjuk a függvényjeleket tartalmazó klózokat.
4. Az Adobozból származó klózokat is bevonjuk és telítjük a teljes klózhalmazt.

Tétel 1. *Az 1. algoritmus egy helyes, teljes és véges $\mathcal{SHIQ}$ leíró logikai tételbizonyító eljárás.*

2.1.2. A függvényjelek kiküszöbölésének előnyei

Több szempontból is előnyös, ha megszabadulunk a függvényjelektől az Adobozhoz való hozzáférés előtt. Először, növekszik a **hatékonyság**. Mindazon Adoboztól független következtetési lépést, melyet az adatok bevonása után végzünk el, meg kell ismételni a klózokban szereplő változók minden lehetséges behelyettesítésére. Másrészt, a következtetés **biztonságosabbá** válik. Egy olyan lekérdezésből kiinduló következtető, mely találkozhat függvényjelekkel, könnyen végtelen ciklusba eshet. Külön figyelmet kell fordítani arra, hogy ne fordulhasson elő a függvényjelek végtelen egymásba ágyazódása a következtetés során. Végezetül, függvényjelek nélkül a következtetés lényegesen **egyszerűbb**. Számos módszer nem alkalmazható közvetlenül függvényjelek jelenlétében.

2.2. $\mathcal{RIQ}$ következtetés visszavezetése $\mathcal{ALCHIQ}$ következtetésre

A $\mathcal{RIQ}$ egy leíró logikai nyelv, melyet úgy kapunk, hogy a $\mathcal{SHIQ}$ nyelvet kiterjesztjük komplex szereptartalmazási axiómák bevezetésével. A $\mathcal{SHIQ}$ nyelvben ki tudunk fejezni olyan összefüggéseket, mint $(\text{apa} \sqsubseteq \text{rokon})$, vagyis hogy az apa egy rokon. Komplex szereptartalmazási axiómák esetén megengedjük szerepek kompozícióját tartalmazási kifejezések bal oldalán. Például, optimista módon kijelenthetjük, hogy $(\text{házastárs} \circ \text{anya} \sqsubseteq \text{barát})$, vagyis hogy az anyós egyben barát is. Ez jelentősen megnöveli a nyelv kifejezőerejét és különösen nagy jelentőségű orvosi ontológiák esetében. Az azonban jól ismert, hogy a következtetés komplexitása is növekszik, méghozzá a bemenet méretének exponenciális függvényében.

Elkészítettünk egy algoritmust, melynek segítségével $\mathcal{RIQ}$ tudásbázisunkat leképezzük egy kielégíthetőség szempontjából ekvivalens $\mathcal{ALCHIQ}$ tudásbázisra. Az átalakítás exponenciális az eredeti tudásbázis méretében, azaz asszimptotikusan optimális. Módszerünk segítségével A $\mathcal{RIQ}$ nyelvű következtetés visszavezethető $\mathcal{ALCHIQ}$ nyelvű következtetésre.

2.2.1. Egy példa

Először egy apró példán keresztül próbáljuk szemléltetni a transzformációt. A példa feltételezi a leíró logikai szintaxis minimális ismeretét. Tegyük fel, hogy tudásbázisunk egyetlen szereptartalmazási axiómát tartalmaz:

$$PQ \sqsubseteq R$$

ahol R, P, Q szerepnevek. Ez az axióma – egyebek mellett – azt is állítja, hogy ha egy x egyedre teljesül a $\forall R.C$ tulajdonság valamilyen C fogalomra, akkor mindazon egyedek, akik kapcsolódnak x-hez egy $P \circ Q$

szerepláncon keresztül, ki kell elégíteniük a C fogalmat. Ezt könnyen megfogalmazhatjuk egy fogalomtartalmazási axióma segítségével:

$$\forall R.C \sqsubseteq \forall P.\forall Q.C$$

evvel ekvivalens módon bevezethetünk új fogalomneveket, hogy elkerüljük az összetett fogalmak egymásba ágyazódását:

$$\forall R.C \sqsubseteq X_1$$
$$X_1 \sqsubseteq \forall P.X_2$$
$$X_2 \sqsubseteq \forall Q.C$$

Ezen axióma azonban kizárólag a C fogalom helyes terjedését garantálja, és ugyanilyen axiómákra van szükségünk minden egyéb fogalom esetén. Azonban elég azokra a fogalmakra szorítkoznunk, melyek megjelennek a tudásbázisban unverzális szerepkorlátozással előálló fogalmakban. Például, ha a Tdoboz az alábbi fogalomtartalmazási axiómákat tartalmazza:

$$D \sqsubseteq \forall R.C$$
$$\top \sqsubseteq \forall R.D$$

akkor látható, hogy csak a C és D fogalmak jelennek meg univerzális korlátozásban. Ennek megfelelően a fenti axiómákat mind C mind D esetében hozzáadjuk a fogalomtartalmazási axómákhoz és elhagyjuk a szereptartalmazási axiómát. Az alábbi Tdobozt kapjuk:

$D \sqsubseteq \forall R.C$	$\top \sqsubseteq \forall R.D$
$\forall R.C \sqsubseteq X_1$	$\forall R.D \sqsubseteq Y_1$
$X_1 \sqsubseteq \forall P.X_2$	$Y_1 \sqsubseteq \forall P.Y_2$
$X_2 \sqsubseteq \forall Q.C$	$Y_2 \sqsubseteq \forall Q.D$

A két tudásbázisnak más a szignatúrája (más fogalomnevek fordulnak elő benne), így különböző a modelljeik halmaza, azonban ha az egyik kielégíthető, akkor a másik is az. Ezt úgy bizonyítjuk, hogy belátjuk, hogy az egyik modelljéből elő lehet állítani a másik modelljét.

2.2.2. Fogalomtartalmazási axiómák generálása

Módszerünk erősen épít [8]-re, mely egy tabló alapú döntési eljárást ad $\mathcal{R}\,I\,\mathcal{Q}$ tudásbázis konzisztenciájának eldöntésére. Minden R szerephez, mely megjelenik a tudásbázisban, a szerzők definiálnak egy nemdeterminisztikus véges B_R automatát, mely reprezentálja mindazon szerepláncokat, melyeket tartalmaz az R szerep. Ezen automatákat felhasználják a tabló fa építésében arra, hogy nyomon kövessék a különféle szerepláncokat. Mi ezen automatákat használjuk arra, hogy előállítsunk egy $\mathcal{ALCH}\,I\,\mathcal{Q}$ tudásbázist. Ennek köszönhetően a szerephierarchia kezelése nem kötődik többé a tabló algoritmushoz, így bármilyen olyan módszer, mely képes eldönteni egy $\mathcal{ALCH}\,I\,\mathcal{Q}$ tudásbázis konzisztenciáját, felhasználható $\mathcal{R}\,I\,\mathcal{Q}$ következtetésre. Többek között a 2.1. alfejezetben bemutatott algoritmus is alkalmazható.

Propozíció 2. *Tetszőleges $\mathcal{R}$ reguláris szerephierarchia és I interpretáció esetén teljesül, hogy I pontosan akkor modellje $\mathcal{R}$-nek, ha minden (esetlegesen inverz) S szerepre, mely előfordul $\mathcal{R}$-ben, minden $w \in L(B_S)$ szerepláncra és minden $\langle x,y\rangle \in w^I$ egyedekre teljesül, hogy $\langle x,y\rangle \in S^I$.*

A 2. propozíció azt fejezi ki, hogy két egyed pontosan akkor S-összekötött, ha van köztük egy w szereplánc, melyet B_S elfogad.

A következőkben felhasználjuk a tudásbázis fogalmi lezártját (concept closure – $clos(KB)$), amely tartalmazza a KB tudásbázisban előforduló összes részfogalmat. Minden $\forall R.C \in clos(KB)$ fogalomhoz és B_R automata minden s állapotához bevezetünk egy új $X_{(s,R,C)}$ fogalmat. A B_R automata kezdeti és elfogadó állapotához tartozó új fogalmakat $X_{(start,R,C)}$ és $X_{(stop,R,C)}$ jelöli.

Definíció 1. *Egy tetszőleges, $\mathcal{R}\mathcal{I}\mathcal{Q}$ nyelvű KB tudásbázis esetén $\Omega(KB)$ egy olyan $\mathcal{ALCHIQ}$ tudásbázist jelent, melyet az alábbiak szerint konstruálunk meg:*

- *$\Omega(KB)_\mathcal{T}$-t $KB_\mathcal{T}$-ből kapjuk minden olyan $w \sqsubseteq R$ szereptartalmazási axióma elhagyásával, melyekben R nem egyszerű szerep; valamint az alábbi axiómák hozzáadásával minden $\forall R.C \in clos(KB)$ fogalomra:*
  1. $\forall R.C \sqsubseteq X_{(start,R,C)}$
  2. $X_{(p,R,C)} \sqsubseteq X_{(q,R,C)}$ *for each* $p \xrightarrow{\varepsilon} q \in B_R$
  3. $X_{(p,R,C)} \sqsubseteq \forall S.X_{(q,R,C)}$ *for each* $p \xrightarrow{S} q \in B_R$
  4. $X_{(stop,R,C)} \sqsubseteq C$
- $\Omega(KB)_\mathcal{A} = KB_\mathcal{A}$

Tétel 2. *KB pontosan akkor kielégíthető, ha $\Omega(KB)$ kielégíthető. Igaz továbbá, hogy $\Omega(KB)$ mérete felülről becsülhető KB méretének exponenciális függvényével.*

Az Ω transzformáció $\mathcal{R}\mathcal{I}\mathcal{Q}$ tudásbázisból $\mathcal{ALCHIQ}$ tudásbázist képez. A 2. tétel azt állítja, hogy a transzformáció megőrzi a tudásbázis konzisztenciáját és legfeljebb exponenciális mértékben növeli a tudásbázis méretét. Ezáltal az $\mathcal{ALCHIQ}$ következtetéshez kifejlesztett módosított kalkulus (2.1. alfejezet) felhasználható a következtetés elvégzésére.

2.3. Egy rezolúció alapú leíró logikai kalkulus

Ebben az alfejezetben egy új rezolúciós kalkulust mutatunk be, melynek neve *DL kalkulus*, és ami alkalmas egy $\mathcal{SHQ}$ Tdoboz konzisztenciájának eldöntésére. A kalkulus újdonsága, hogy közvetlenül leíró logikai axiómákon lett definiálva. A következtetés így egy magasabb absztrakciós szinten történik, kevesebb átalakítási lépéssel és emberi szemmel jobban átlátható levezetésekkel.

Első lépésben meghatározunk egy fogalomhalmazt, melyet ki kell elégítenie minden egyednek ahhoz, hogy a Tdoboz igaz legyen. Ezután következtetési szabályokat vezetünk be, melyek két fogalomból egy új fogalmat vezetnek le. A szabályok segítségével telítjük a tudásbázisunkat, azaz addig alkalmazzuk őket amíg lehetséges, és a konklúziót hozzáadjuk a halmazhoz. A telítésben fontos szerepe van a redundancia kiküszöbölésének: valahányszor egy olyan fogalommal találkozunk, mely általánosabb, mint egy másik fogalom, az általánosabb fogalmat elhagyjuk. Belátjuk, hogy a telítés mindig végetér, valamint hogy a tudásbázis pontosan akkor ellentmondásos, ha a telített halmaz tartalmazza az üres fogalmat ($\bot$).

Egy *bool fogalom* nem tartalmaz szerepeket (csak negáció, unió és metszet megengedett). Bool fogalmak jelölésére nagybetűket használunk az abc elejéről ($A, B, C \ldots$). A későbbiekben mindig feltételezzük, hogy a bool fogalmak a lehető legegyszerűbb diszjunktív normál formában vannak. Például az $A \sqcup A \sqcup (B \sqcap \neg B \sqcap C)$ fogalom helyett A illetve $A \sqcap \neg A$ helyett $\bot$ szerepel. Valahányszor a következtetési szabályok nem őrzik meg a diszjunktív normálformát, alkalmazzuk a *dnf* operátort.

Beveteztünk egy teljes rendezést ($\succ$) leíró logikai kifejezéseken. Adott N halmaz esetén azt mondjuk, hogy $C \in N$ fogalom *maximális* N-ben, ha C nagyobb ($\succ$ alapján) mint bármely más fogalom N-ben. Mivel a rendezés teljes, tetszőleges véges N halmaznak van egy egyedi maximális eleme.

$\mathcal{SHQ}$-fogalmak. Az $\mathcal{SHQ}$ Tdobozt *$\mathcal{SHQ}$-fogalmak* egy halmazára képezzük le, melyeket az alábbiak szerint definiálunk:

$$
\begin{array}{ll}
C & \text{(bool fogalmak)} \\
C \sqcup \bigsqcup (\leq nR.D) & (\leq\text{-max fogalmak}) \\
C \sqcup \left(\bigsqcup (\leq nR.D) \right) \sqcup (\geq kS.E) & (\geq\text{-max fogalmak})
\end{array}
$$

ahol a C, D, E bool fogalmak és diszjunktív normálformájúak.

Következtetési szabályok. A következtetési szabályokat az 1. ábrán láthatjuk, ahol C_i, D_i, E_i (esetlegesen üres) bool fogalmak. W_i egy tetszőleges $\mathcal{SHQ}$-fogalom, akár az üres fogalom is lehet. Két diszjunktív fogalmat mindig a maximális tagok alapján rezolválunk, így a rendezésünk egy kiválasztási függvényt eredményez. Mivel a rendezés teljes, mindig kiválasztható az az egyedi maximális tag, ami szerint rezolválni lehet.

Sz1 $\dfrac{C_1 \sqcup (D_1 \sqcap A) \qquad C_2 \sqcup (D_2 \sqcap \neg A)}{C_1 \sqcup C_2}$

ahol $D_1 \sqcap A$ maximális $C_1 \sqcup (D_1 \sqcap A)$-ban

és $D_2 \sqcap \neg A$ maximális $C_2 \sqcup (D_2 \sqcap \neg A)$-ban

Sz2 $\dfrac{C \qquad W \sqcup (\geq nR.D)}{W \sqcup (\geq nR.dnf(D \sqcap E))}$

ahol E-t Sz1 alkalmazásával kapjuk C és D premisszákkal

Sz3 $\dfrac{W_1 \sqcup (\leq nR.C) \qquad W_2 \sqcup (\geq kS.D)}{W_1 \sqcup W_2 \sqcup (\geq (k-n)S.dnf(D \sqcap \neg C))}$

$n < k, S \sqsubseteq^* R$, $(\leq nR.C)$ maximális $W_1 \sqcup (\leq nR.C)$-ben

és $(\geq kS.D)$ maximális $W_2 \sqcup (\geq kS.D)$-ben

Sz4 $\dfrac{W_1 \sqcup (\leq nR.C) \qquad W_2 \sqcup (\geq kS.D)}{W_1 \sqcup W_2 \sqcup (\leq (n-k)R.dnf(C \sqcap \neg D)) \sqcup (\geq 1S.dnf(D \sqcap \neg C))}$

$n \geq k, S \sqsubseteq^* R$, $(\leq nR.C)$ maximális $W_1 \sqcup (\leq nR.C)$-ben

és $(\geq kS.D)$ maximális $W_2 \sqcup (\geq kS.D)$-ben

1. ábra. DL kalkulus TBox következtetési szabályok

Sz1 megfelel a klasszikus rezolúciós lépésnek és Sz2 segítségével ugyanezt a következtetést el tudjuk végezni olyan egyedek esetében, melyek létezését egy $\geq$-fogalom írja elő. Sz3 és Sz4 $\geq$-fogalmak és $\leq$-fogalmak egymásrahatásához kapcsolódik. Ha egy egyed kielégíti a $\leq nR.C$ fogalmat, valamint a $\geq kS.D$ fogalmat, ez ütközéshez vezethet, amennyiben C tartalmazza D-t. Ebben az esetben $D \sqcap \neg C$ nem kielégíthető, ami vagy ellentmondáshoz vezet amennyiben $n < k$ (Sz3) vagy egy kisebb számosságú korlátot eredményez (Sz4). Ha több $\geq$-fogalom és egy $\leq$-fogalom együttesen ellentmondásos, akkor minden $\geq$-fogalomból levezetünk egy kisebb számosságú $\leq$-fogalmat (Sz4) mindaddig, amíg a $\leq$-fogalom teljesen el nem tűnik (Sz3), az üres fogalmat eredményezve.

Telítés. Telítjük a tudásbázist, azaz a következtetési szabályok (1. ábra) ismételt alkalmazásával új fogalmakat vezetünk le. A következmény (esetleg néhány egyszerűsítő lépés után) mindig egy $\mathcal{SHQ}$-fogalom.

Propozíció 3. *Az $\mathcal{SHQ}$-fogalmak halmaza zárt az 1. ábra szabályaira és csak véges sok $\mathcal{SHQ}$ -fogalom vezethető le egy véges Tdobozból kiindulva.*

Tétel 3. *A DL kalkulus helyes és teljes.*

Legyen $\mathcal{T}$ egy $\mathcal{SHQ}$ Tdoboz. Legyen $Sat_{\mathcal{T}}$ az a fogalomhalmaz, melyet $\mathcal{T}$ $\mathcal{SHQ}$ -fogalmakra történő átalakításával és DL kalkulus szerinti telítésével kapunk. A telítés mindig véget ér. Teljesül továbbá, hogy amennyiben $Sat_{\mathcal{T}}$ nem tartalmazz az üres fogalmat ($\bot$), akkor $\mathcal{T}$ kielégíthető.

3. Hurokkiküszöbölés, egy helyes optimalizációs eljárás

A PTTP (Prolog Technology Theorem Prover [16]) egy olyan helyes és teljes elsőrendű logikai tételbizonyító eljárás, mely a Prolog nyelvre épül. Egy elsőrendű klózhalmazból készítünk egy Prolog programot, majd ezen program végrehajtása végzi el a következtetést.

A PTTP-ben minden elsőrendű klózhoz hozzárendeljük Horn-klózok egy halmazát, melyeket a klóz *kontrapozitívjainak* hívunk. Az $L_1 \vee L_2 \vee \cdots \vee L_n$ klózhoz összesen n kontrapozitív tartozik, melyek alakja $L_k \leftarrow \neg L_1, \ldots, \neg L_{k-1}, \neg L_{k+1}, \ldots, \neg L_n$, ahol $1 \leq k \leq n$. A kettős negáció kiküszöbölése után a maradó negált literáloktól is megszabadulunk új predikátumnevek segítségével: minden P predikátumnévhez bevezetünk egy *not_P* predikátumnevet és $\neg P(X)$ minden előfordulását lecseréljük *not_P*(X) -re. Például az $A(X) \vee \neg B(X) \vee \neg R(X,Y)$ klózt lefordítjuk három Prolog szabályra, melyek mindegyikének más a feje:

```
A(X)       :-  B(X), R(X,Y).
not_B(X)   :-  not_A(X), R(X,Y).
not_R(X,Y) :-  not_A(X), B(X).
```

A kontrapozitívok használatának köszönhetően az elsőrendű klóz minden literálja megjelenik valamelyik Horn-klóz fejében. Ez garantálja, hogy bármelyik literál részt vehet a rezolúciós lépésben a Prolog korlátozott kiválasztási szabályának ellenére.

Tegyük fel, hogy be akarjuk bizonyítani A-t és a végrehejtás során találkozunk a $\neg A$ részcéllal. Ez azt jelenti, hogy eddigre egy olyan szabályt vezettünk le, mely szerint egy $\neg A$-val kezdődő célsorozatból következik A:

$$\texttt{A} \leftarrow \texttt{not\_A}, \texttt{P}_1, \texttt{P2}, \ldots \texttt{P}_k.$$

Ami logikailag ekvivalens az alábbi elsőrendű klózzal:

$$A \vee A \vee \neg P_1 \vee \neg P2 \vee \cdots \vee \neg P_k$$

Innen azonnal látjuk, hogy A két előfordulását össze lehet vonni, vagyis nem kell bebizonyítanunk a `not_A` részcélt. Ezt a lépést hívjuk *ős rezolúciónak* [10].

Két további jellegzetessége van a PTTP megközelítésnek. Először, a végtelen bizonyítási ágak elkerülése érdekében iteratívan mélyülő keresést alkalmazunk a Prolog szokásos szélességi keresése helyett. Másodszor, a legtöbb Prolog rendszerrel ellentétben a PTTP előfordulás ellenőrzést (occurs check) alkalmaz egyesítés előtt, azaz például az X és $f(X)$ kifejezések egyesítését nem engedélyezi, mivel ez végtelen mélységű kifejezést eredményezne.

Összefoglalva, a PTTP öt módszert alkalmaz ahhoz, hogy Prolog alapon teljes elsőrendű következtetést végezzen: kontrapozitívok, negált literálok átnevezése, ős rezolúció, iteratívan mélyülő keresés, előfordulás ellenőrzéses egyesítés.

A DLog rendszer [11], melyet az 5. fejezetben fogunk ismertetni, tekinthető a PTTP specializációjának leíró logikai következtetésre. A DLog egy két fázisú következtetést végez, melyben az első fázis a 2.1. alfejezetben bemutatott módosított kalkulust használnja, míg a második fázis a PTTP-re épül.

A hurokkiküszöbölés egy olyan módszer, mely megakadályozza, hogy egy logikai program újra és újra ugyanazt a célt próbálja bebizonyítani, ezáltal elkerülve bizonyos végtelen ciklusokat a következtetésben. Bár mind a PTTP, mind a DLog használja ezt az optimalizációt, ezidáig nem lett bebizonyítva a helyessége. Ezt sikerült pótolnom, azaz precíz, formális bizonyítás segítségével beláttam, hogy a hurokkiküszöbölés nem vezet megoldásvesztéshez, alkalmazása megengedett.

Definíció 2 (Hurokkiküszöbölés)**.** *Legyen P egy Prolog program és G egy Prolog cél. P végrehajtása G céllal, a* hurokkiküszöbölés *alkalmazása mellett egy olyan kiterjesztett végrehajtást jelent, ahol egy végrehajtási ágat meghiúsulással lezárunk valahányszor olyan H céllal találkozunk, ami szintaktikusan azonos egy nyitott részcéllal (melynek tehát elkezdtük a bizonyítását, de nem fejeztük meg be).*

A hurokkiküszöbölés roppant intuitív módszer. Ha például be szeretnénk látni a G célt és rájövünk, hogy a bizonyítás részeként be kell bizonyítanunk ugyanezt a G célt, akkor nem érdemes továbbmenni a bizonyításban, mivel 1) vagy végtelen ciklusba esünk és sose fejeződik be a bizonyítás vagy 2) valahogy bebizonyítjuk a G cél második előfordulását olymódon, amit azonban közvetlenül fel lehet használni az első előfordulás bizonyítására. A helyzet azonban bonyolultabbá válik az ős rezolúció alkalmazása mellett. Ugyanis a két G cél különbözik az őseik halmazában, és előfordulhat, hogy G második előfordulását azon ősök segítségével tudtuk bizonyítani, melyekkel az első előfordulás nem rendelkezik. Munkánk során az derült ki, hogy bár valóban elő lehet állítani az G első előfordulásának bizonyítását a második előfordulás bizonyításából, a kapott bizonyítás nagy mértékben különbözhet az eredetitől.

Tétel 4. *Minden teljes PTTP bizonyításhoz, mely tartalmaz hurkokat, létezik egy teljes PTTP bizonyítás amely mentes minden huroktól.*

A 4. tétel igazolja a hurokkiküszöbölés használatát, melynek köszönhetően a keresési tér nagy mértékben csökken. További nagyon fontos haszna, hogy a DLog következtető estetében kizárólag ilyen hurkok eredményezhetnek végtelen futást,vagyis a DLog kiegészítve a hurokkiküszöböléssel garantáltan leáll. Ennek köszönhetően az iteratívan mélyülő keresés helyett használhatjuk a lényegesen hatékonyabb mélységi keresést.

4. Típuselemzés a Q funkcionális nyelvhez

A következőkben egy olyan eszköz tervezését és fejlesztését ismertetem, melyet a Q vektorfeldolgozó nyelv típuselemzésére készítettünk. Kiemelném munkánk két erényét: 1) elkészítettünk egy típusnyelvet, melynek segítségével Q programokat lehet kiegészíteni típusdeklarációkkal, ezáltal jobban dokumentált és jobban karbantartható kódot eredményezve, valamint 2) eszközünk megvizsgálja a Q programokat típushelyesség szempontjából fordítási időben jelez esetleges típushibákat.

A típuselemző eszközt két fázisban fejlesztettük. Első lépésben egy *típus ellenőrző* rendszert készítettünk: a programozótól elvárjuk, hogy deklarálja a változók típusát (megfelelő Q kommentek formájában) és mi megvizsgáljuk, hogy nincs-e ellentmondás a deklarációk és a kód között. A második lépésben továbbmentünk típusellenőrzésről *típuskikövetkeztetés felé*: mindenféle felhasználói típusinformáció nélkül próbáljuk meghatározni az egyes kifejezések típusát, illetve lehetséges típusainak halmazát.

A legfontosabb feladat a hibák feltárása és jól értelmezhető hibaüzenetek generálása. Eszközünk segítségével számos olyan hibát fel lehet tárni, melyek futtatás közben könnyen elsikkadnak, ezáltal használata nagymértékben segíti a programfejlesztés folyamatát.

A típuskövetkeztetést korlát logikai eszközökkel végezzük: az erdeti feladatot átfogalmazzuk mint egy korlát kielégítési problémát (Constraint Satisfiction Problem – CSP), melyet a Prolog kiterjesztésével oldunk meg (Constraint Handling Rules [5], [14]).

4.1. Típusellenőrzés

Típusellenőrző algoritmusunk két követelményt támaszt a Q programozók felé: egyrészt minden változó típusát deklarálni kell, másrészt csak tömör – változó mentes – típusdeklarációk megengedettek. Mindkét követelmény megszűnik majd a típuskikövetkeztető módszer esetében.

A 2. algoritmus összegzi a típuselemző komponens működését. A bementő program absztrakt szintaxis fa (AST) reprezentációjából indulunk ki, melyet az elemző komponens állít elő. A fa minden csomópontja egy részkifejezésnek felel meg és célunk, hogy minden csomóponthoz egy típust rendeljük. Bizonyos csomópontok típusát azonnal tudjuk: a változók típusát a programozó megadta, valamint tudjuk a beépített függvények és az atomi kifejezések típusát. Az elemző ezek alapján kideríti a többi kifejezés típusát.

A korlátokat a Prolog CHR [14] könyvtárának segítségével kezeljük. Minden korláthoz tartozik néhány korlát kezelő szabály. Amennyiben az argumentumokról elég típusinformáció áll rendelkezésünkre, egy megfelelő szabály felébred. A szabály meghatározhatja valamilyen kifejezés típusát, meghívhat újabb korlátokat vagy megállapíthatja, hogy hiba történt.

Abban az esetben, ha minden változóhoz tartozik egy típusdeklaráció, a típuselemzés kezdetén már ismerjük az absztrakt fa minden levelének típusát. Ez azért van így egy levél vagy egy atomi kifejezést vagy egy változót reprezentál. Miután a levelek típusa ismert, a szabályok azonnal felterjesztik a szükséges típusinformációt a belső csomópontok felé.

4.2. Típuskikövetkeztetés

A típuselemző készítésének második fázisában eltöröltük a korábbi megkötéseket: a programozónak már nem kötelező típusdeklarációkat használnia, és ha mégis megteszi, akkor használhat típusváltozókat. Rugalmasabb megoldást kerestünk, ahol az elemző nem vár el semmit, csak felhasználja a rendelkezésre álló információt, és a lehető legtöbbet kikövetkezteti.

Algorithm 2 A típusellenőrzés algoritmusa

1. Az absztrakt szintaxis fa minden csomópontjához hozzárendelünk egy típusváltozót.

2. Bejárjuk a fát és korlátokat fogalmazunk meg. Minden programkifejezés esetén van egy korlát, mely leírja, hogyan áll elő a típusa a részkifejezései típusának függvényében. Vagyis a korlátok meghatározzák, hogy az absztrakt fa egyes csomópontjaihoz milyen típust kell hozzárendelni a gyerekekhez rendelt típusok függvényében.

3. A korlát következtetés segítségével automatikusan

   - terjesztjük a korlátokat az absztrakt fában,
   - levezetjük az ismeretlen típusokat,
   - észleljük az ütközéseket, vagyis a típushibákat.

   A levél csomópontok típusa alapján meg tudjuk határozni a közvetlen szülőjük típusát. Ettől felébrednek az egyel fejjebbi szinten levő korlátok, melyek a következő lépésben meghatározzák a legfeljebb kettő mélységű belső csomópontok típusát. A folyamat automatikusan folytatódik, míg végül meghatározzuk az összes csomópont típusát.

4. Ha típusütközés van, megjelöljük az adott csomópontot. A hibás csomópont minden őse is hibás lesz – azonban csak a legmélyebben levő csomóponthoz, azaz a legszűkebb hibás kifejezéshez készítünk hibajelentést.

5. A fa bejárása során minden csomóponthoz, mely típushelyes kifejezést reprezentál, hozzárendelünk egy típust. A típushozzárendelés kielégíti az összes korlátot.

CSP. A típuskikövetkeztetés feladatát mint egy korlátkielégítési problémát (CSP) fogalmazzuk meg, melyet korlát logikai programozás segítségével oldunk meg. Minden programkifejezéshez hozzárendelünk egy CSP változót, melyekhez tartozik egy értékkészlet. Kezdetben minden értékkészlet az összes típus halmaza. A különféle típusmegkötéseket korlátokként fogalmazzuk meg, melyek szűkítik változók tartományait. Az elemzőnek az a feladata, hogy olyan értéket rendeljen a változókhoz a megfelelő tartományukból, melyek kielégítik a korlátokat.

Korlátok. A típuselemző bejárja az absztrakt szintaxis fát és korlátokat fogalmaz meg. Azon korlátok, melyek a CSP változók tartományát írják le, különösen fontosak, ezért *elsődleges korlátoknak* nevezzük őket. Ezek alsó és felső korlátokként jelentkeznek. A többi korlátot *másodlagos korlátnak* nevezzük. A másodlagos korlátok több változót kötnek össze és amint argumentumaik kellően behelyettesítődnek, elsődleges korlátokat generálnak, szűkítve a tartományokat.

Korlát következtetés. A korlát következtetés egy *produkciós rendszerre* [13] épül, ami HA-AKKOR szabályok halmaza. Karbantartunk egy *korláttárat* mely tartalmazza az összes korlátot. Kezdetben azok a korlátok vannak benne, melyeket az absztrakt fa bejárása során felvettünk. Egy produkciós szabály akkor tüzel, ha bizonyos korlátok benne vannak a tárban és ez azt eredményezi, hogy bizonyos korlátokat hozzáadunk vagy elveszünk a tárból. Másképp fogalmazva (CHR teminológiát használva), minden szabálynak van egy fej része, melyben azt határozzuk meg, hogy milyen szabályok jelenléte esetén kell tüzelnie, és egy törzs része, mely a tüzelés során hozzáadott korlátokat tartalmazza. A törlésre ítélt korlátok a fej egyik részéhez tartoznak. Megadható továbbá egy őrfeltétel, melynek segítségével komplex tüzelési feltételt lehet meghatározni.

Célunk, hogy végezetül megszabaduljunk minden másodlagos korláttól a szabályok ismételt tüzelése során. Ha ez sikerül, akkor az elsődleges korlátok által meghatározott tartományok megadják minden kifejezéshez a lehetséges típusok halmazát. Amennyiben egy tartomány üres, típushibára jutottunk. Különben a programot típushelyesnek tekintjük.

Amennyiben másodlagos korlátok maradnak a tárban, *címkézést* alkalmazunk. Címkézés során az egyes CSP változókhoz szisztematikusan értékeket rendelünk a tartományaikból. A hozzáréndelés felébreszti a produkciós szabályokat. Ha ütközésre jutunk, akkor visszalépünk és új hozzárendelést keresünk. Végezetül vagy találunk egy konzisztens és teljes változóhozzárendelést, mely kielégíti az összes korlátot, vagy pedig ennek hiányát állapítjuk meg és típushibát jelzünk.

5. A DLog leíró logikai következtető rendszer

A DLog [11] egy Prolog nyelven írott leíró logikai következtető, mely kétfázisú rezolúciós következtetést valósít meg. A rendszer a $\mathcal{RIQ}$ nyelvet támogatja. A 2. fejezetben leírtak szerint a bemeneti tudásbázist az első fázisban átalakítjuk függvényjel mentes elsőrendű klózok halmazára. A második fázisban ezen klózokból készítünk egy Prolog programot a PTTP technológia alapján, majd ennek a programnak a futása végzi el a tényleges adatkövetkeztetést. Az adatkövetkeztetés fókuszált, ami azt jelenti, hogy a lekérdezésből indul ki és az Adoboznak csak azon részét érinti, amely lényeges a válasz szempontjából. Hogy mi lényeges, azt a Tdobozból nyert klózok segítségével határozzuk meg. Ezáltal a DLog futását nem befolyásolja nagy mennyiségű irreleváns adat jelenléte. Az Adobozt közvetlen adatbázis lekérdezések formájában tudjuk elérni így nem kell az egészet betölteni a memóriába. Tudomásunk szerint a DLog az egyetlen olyan leíró logikai következtető, melynek nem kell végigolvasnia a teljes Adobozt. Ennek köszönhetően a DLog jól használható akkor is, ha nagy adatbázisokban tárolt Adoboz felett kell következtetni. A DLog utolsó stabil változata, mely a $\mathcal{SHIQ}$ nyelvet támogatja, elérhető a `http://dlog-reasoner.sourceforge.net` weblapról.

Az első következtetési fázis független az Adoboztól és a lekérdezéstől. Ezért mindaddig, amíg a Tdoboz változatlan, az első fázist elég egyszer, egy előfeldolgozási fázis gyanánt elvégezni. Ezáltal az első fázis sebessége nem kritikus, hiszen nem befolyásolja a rendszer lekérdezésekre adott válaszidejét.

Terminológiai következtetés. A Tdoboz telítési modul a bemenet Tdoboz részéből indul ki és átalakítja az alábbi típusú elsőrendű klózok halmazára:

$$\neg R(x,y) \vee S(y,x) \tag{c11}$$

$$\neg R(x,y) \vee S(x,y) \tag{c12}$$

$$\mathbf{P}(x) \tag{c13}$$

$$\mathbf{P_1}(x) \vee \bigvee_i (\neg R(x,y_i)) \vee \bigvee_i \mathbf{P_2}(y_i) \vee \bigvee_{i,j} (y_i = y_j) \tag{c14}$$

A transzformáció a 2.1.alfejezetet követi, megvalósítva a következtetés első fázisát. A kimenő klózok szintaxisa meglehetősen egyszerű, aminek köszönhetően a második fázisban a PTTP nagy mértékben optimalizált és specializált változatát alkalmazhatjuk. A legfontosabb haszna az Tdoboz telítésnek, hogy megszabadulunk a függvényjelektől.

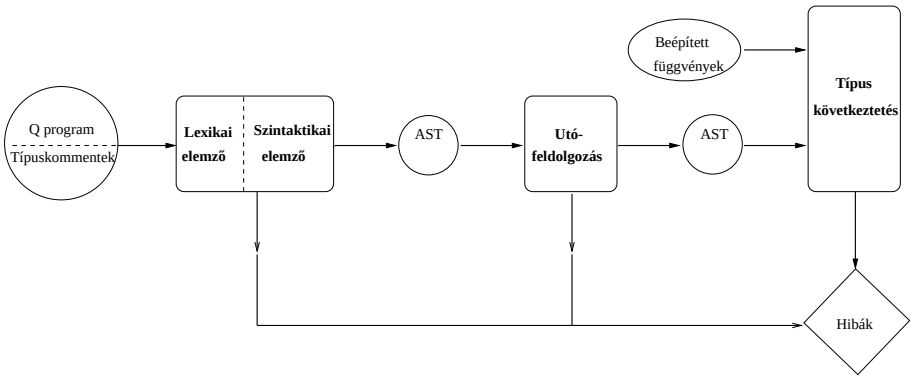
Jövőbeli feladatok. Egyik legsűrgősebb feladat, mely előttünk áll, a rendszer interfészének bővítése. Jelenleg kizárólag a DIG [1] formátumot támogatjuk mind a tudásbázis mind a lekérdezés számára. Szeretnénk elkészíteni egy OWL interfészt ([7], [6]) a rendszerhez. Továbbá, bár elkészült egy programkomponens, mely lehetővé teszi az Adoboz adatbázisból történő elérését ([9]), ez még nem lett rendesen beépítve a teljes rendszerbe. Ezen feladatok elvégzése után a rendszert alapos tesztelésnek kell alávetni, hogy össze tudjuk hasonlítani más leíró logikai következtetőkkel, mint például a RacerPro, Pellet, Hermit, KAON2 rendszerek.

A fent említett gyakorlatiasabb problémák mellett szeretnénk kiterjeszteni a rendszert még komplexebb nyelvi elemek felé a $\mathcal{RIQ}$-n túl, lehetőség szerint minél jobban megközelítve a $\mathcal{SROIQ}(\mathcal{D})$ nyelvet, mely az OWL2 [6] nyelv hátterét adja.

6. A `qtchk` statikus típusellenőrző rendszer

Elkészítettünk egy `qtchk` nevű Prolog programot, mely megvalósítja a 4. fejezetben bemutatott típuselemző módszert. A rendszert három részre lehet bontani:

- 1. Fázis: lexikai és szintaktikai elemzés
  A bemenő Q programot a lexikai elemző tokenek sorozatára bontja. Ezután a szintaktikai elemző előállítja a program belső reprezentációját, amely egy absztrakt szintaxis fa.
- 2. Fázis: utófeldolgozás
  Számos apró transzformációs lépéssel egyszerűsítjük az absztrakt szintaxis fát, feloldva különböző környezetfüggőségi problémákat.
- 3. Fázis: típuselemzés
  A típuselemző komponens bejárja az absztrakt fát és korlátokat vesz fel. Ezen fázis felhasználja a felhasználói típusdeklarációkat valamint a beépített függvények típusait is. Az előre elkészített korlátkezelő szabályokon keresztül automatikusan beindul a korlát következtetés, melynek végére minden kifejezéshez hozzárendelődik egy olyan típus mely kielégíti a korlátokat.



2. ábra. A típuselemző felépítése

A rendszer architekturája a 2. ábrán látható. Minden fázisban gyűjtjük az esetleges hibákat. A program kimenete ezen hibák összessége, mely tartalmazza a hibák pontos helyét, valamint típushibák esetén egy rövid indoklást, hogy miért gondolja azt a rendszer, hogy egy kifejezés hibás.

A `qtchk` rendszer mind SICStus Prolog 4.1 [15] mind SWI Prolog 5.10.5 [17] alatt fut. Több mint 8000 sor kódot tartalmaz.[1] A Q nyelv tele van kivételekkel és furcsaságokkal és rengeteg beépített függvénye van, ezért egy nagyon komplex korlát rendszert kellett megvalósítanunk, több mint 60 korlát használatával. A rendszerhez elkészült egy részletes kézikönyv [2], mely tartalmaz számos példát valamint a Q nyelv konkrét szintaxisát.

7. Összefoglalás és a tézisek felsorolása

Ebben a tézisfüzetben röviden bemutattam a leíró logikai következtetéshez valamint a statikus típuselemzéshez kapcsolódó munkánkat. Bár ezen területek sok mindenben különböznek, mindkét esetben automatikus következtetéssel oldjuk meg a feladatot. Módszereink kihasználják és kiegészítik a logikai programozás eszköztárát, így roppant kényelmes volt a Prolog választása megvalósítási nyelvként. Az elkészült rendszerek – Dlog és `qtchk` – demonstrálják, hogy a Prolog nyelv beépített következtető mechanizmusa jól használható különféle okfejtési feladatok elvégzésére. A következőkben megfogalmazom a saját hozzájárulásomat a bemutatott eredményekhez.

[1] A kódot szívesen megosztjuk bármely érdeklődővel.

Tézis 1. Terveztem egy olyan módszert, melynek segítségével leíró logikai axiómákból függvényjelmentes elsőrendű klózokat állítunk elő. A módszert megvalósítottam a DLog leíró logikai következtető rendszerben. [20, 18, 19, 26, 21]

Tézis 1.A. Terveztem egy elsőrendű rezolúciós kalkulust (*módosított kalkulus*), amely az alap szuperpozíció módosításával jött létre. Beláttam, hogy a kalkulus helyes, teljes és véges futású bármely kiinduló $\mathcal{ALCHIQ}$ klózhalmaz esetén. Ezen eredmény teszi lehetővé a DLog rendszer kétfázisú következtetését: a komplex Tdoboz következtetés függetlenné válik az Adobozon végzett következtetéstől. [18, 19, 21]

Tézis 1.B. Terveztem egy olyan transzformációt, melynek segítségével egy $\mathcal{RIQ}$ tudásbázist leképezünk egy $\mathcal{ALCHIQ}$ tudásbázisra, megszabadulva a komplex szereptartalmazási axiómáktól. Beláttam, hogy a tranformáció helyes, azaz a kezdeti tudásbázis pontosan akkor kielégíthető, ha a kapott tudásbázis az. Ennek köszönhetően számos következtető módszer, melyeket $\mathcal{ALCHIQ}$ következtetésre terveztek, felhasználható a gazdagabb $\mathcal{RIQ}$ nyelvre is. [18]

Tézis 1.C. Terveztem egy leíró logikai kalkulust, a DL kalkulust, mely el tudja dönteni egy $\mathcal{SHQ}$ Tdoboz konzinsztenciáját. Beláttam, hogy a kalkulus helyes, teljes és véges futású. A DL kalkulus egy alternatív következtetési módszer a széles körben elterjedt tabló algoritmus mellett. [20, 26]

Tézis 1.D. Implementáltam a módosított kalkulust valamint a $\mathcal{RIQ}$ -ból $\mathcal{ALCHIQ}$ -ba képző transzformációt a DLog rendszer Tdoboz telítési moduljában. Ez képezi a rendszer első következtetési fázisát.

Tézis 2. Beláttam a hurokkiküszöbölés helyességét, mely egy fontos optimalizáció PTTP alapú tételbizonyításnál. [29, 30]

Tézis 2.A. Azonosítottam a logikai programokban azt a három tulajdonságot, ami végtelen futáshoz vezethet: függvényjelek, új változók folyamatos megjelenése és hurkok. Beláttam, hogy ezek közül egyedül hurkok jelenhetnek meg DLog programokban. Ebből az következik, hogy a hurokkiküszöbölés biztosítja a DLog véges futását. [29, 30]

Tézis 2.B. Precíz bizonyítással igazoltam a hurokkiküszöbölés helyességét. A bizonyítás épít egy úgynevezett kifordítási módszerre, melynek segítségével alternatív bizonyításokat tudunk meghatározni ugyanazon célhoz PTTP programokban. Ezen eredményből következik, hogy bármely célhoz, melyet be lehet bizonyítani PTTP-vel, létezik olyan bizonyítás, mely nem tartalmaz hurkokat. [29, 30]

Tézis 3. Terveztem egy statikus típuselemző algoritmust Q nyelvű programok típushelyességének vizsgálatára. Az algoritmust implementáltam a `qtchk` rendszer típuselemző moduljában. [23, 3, 25, 24, 4]

Tézis 3.A. Terveztem egy típusellenőrző módszert: a program változóihoz hozzárendelt felhasználói típusdeklarációk alapján meghatározzuk az összetett kifejezések típusát. [23, 3]

Tézis 3.B. Terveztem egy típuskikövetkeztető módszert: nem várunk el semmiyen típusinfromációt a felhasználótól, hanem közvetlenül a kód alapján próbáljuk kitalálni az egyes kiejezések típusát illetve le-

hetséges típusainak halmazát. Ezt úgy valósítjuk meg, hogy a feladatot átfogalmazzuk korlát kielégítési problémaként. [25, 24, 4]

Tézis 3.C. Implementáltam mind a típusellenőrző, mind a típuskikövetkeztető algoritmusokat a `qtchk` rendszer típuselemző komponensében. Az implementációban korlát logikai programozást használunk, a Prolog nyelv Constraint Handling Rules kiterjesztését. [23, 3, 25, 24, 4]

Hivatkozások

[1] S. Bechhofer, R. Moller, and P. Crowther. The dig description logic interface. In *In Proc. of International Workshop on Description Logics*, 2003. `citeseer.ist.psu.edu/690556.html`.

[2] János Csorba, Péter Szeredi, and Zsolt Zombori. *Static Type Checker for Q Programs (Reference Manual)*, 2011. http://www.cs.bme.hu/~zombori/q/qtchk_reference.pdf.

[3] János Csorba, Zsolt Zombori, and Péter Szeredi. Using constraint handling rules to provide static type analysis for the q functional language. In *Proceedings of the 11th International Colloquium on Implementation of Constraint and LOgic Programming Systems (CICLOPS 2011)*, 2011.

[4] János Csorba, Zsolt Zombori, and Péter Szeredi. Pros and cons of using CHR for type inference. In Jon Sneyers and Thom Frühwirth, editors, *Proceedings of the 9th workhop on Constraint Handling Rules (CHR 2012)*, pages 16–31, September 2012.

[5] Th. Fruehwirth. Theory and Practice of Constraint Handling Rules. In P. Stuckey and K. Marriot, editors, *Journal of Logic Programming*, volume 37(1–3), pages 95–138, October 1998.

[6] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. OWL 2: The next step for OWL. *Web Semant.*, 6:309–322, November 2008.

[7] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: the making of a web ontology language. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1):7 – 26, 2003.

[8] Ian Horrocks and Ulrike Sattler. Decidability of SHIQ with complex role inclusion axioms. In Georg Gottlob and Toby Walsh, editors, *IJCAI*, pages 343–348. Morgan Kaufmann, 2003.

[9] Balázs Kádár, Gergely Lukácsy, and Péter Szeredi. Large scale semantic web reasoning. In *Proceedings of the 3rd International Workshop on Applications of Logic Programming to the Web, Semantic Web and Semantic Web Services (ALPSWS2008), Udine, Italy*, pages 57–70, December 2008.

[10] R. Kowalski and D. Kuehner. Linear resolution with selection function. *Artificial Intelligence*, 2:227–260, 1971.

[11] Gergely Lukácsy and Péter Szeredi. Efficient Description Logic reasoning in Prolog: The DLog system. *Theory and Practice of Logic Programming*, 9(03):343–414, 2009.

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

[13] A. Newell and H.A. Simon. *Human Problem Solving*. Prentice Hall, Englewood Cliffs, 1972.

[14] Tom Schrijvers and Bart Demoen. The K.U.Leuven CHR system: implementation and application. In *First Workshop on Constraint Handling Rules: Selected Contributions*, pages 1–5, 2004.

[15] SICS. *SICStus Prolog Manual version 4.1.3*. Swedish Institute of Computer Science, September 2010.
`http://www.sics.se/sicstus/docs/latest4/html/sicstus.html`.

[16] Mark E. Stickel. A Prolog technology theorem prover: a new exposition and implementation in Prolog. *Theoretical Computer Science*, 104(1):109–128, 1992.

[17] Jan Wielemaker, Tom Schrijvers, Markus Triska, and Torbjörn Lager. SWI-Prolog. *TPLP*, 12(1-2):67–96, 2012.

[18] Zsolt Zombori. Expressive description logic reasoning using first-order resolution. *Journal of Logic and Computation*. Submitted for publication.

[19] Zsolt Zombori. Efficient two-phase data reasoning for description logics. In *IFIP AI*, pages 393–402, 2008.

[20] Zsolt Zombori. A resolution based description logic calculus. *Acta Cybern.*, pages 571–588, 2010.

[21] Zsolt Zombori. Two phase description logic reasoning for efficient information retrieval. In Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors, *ESWC (2)*, volume 6089 of *Lecture Notes in Computer Science*, pages 498–502. Springer, 2010.

[22] Zsolt Zombori. Two Phase Description Logic Reasoning for Efficient Information Retrieval . In John Gallagher and Michael Gelfond, editors, *Technical Communications of the 27th International Conference on Logic Programming (ICLP'11)*, volume 11 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 296–300, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[23] Zsolt Zombori, János Csorba, and Péter Szeredi. Static Type Checking for the Q Functional Language in Prolog. In John Gallagher and Michael Gelfond, editors, *Technical Communications of the 27th International Conference on Logic Programming (ICLP'11)*, volume 11 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 62–72, Dagstuhl, Germany, 2011. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[24] Zsolt Zombori, János Csorba, and Péter Szeredi. Static Type Inference as a Constraint Satisfaction Problem. In *Proceedings of the TAMOP PhD Workshop: TAMOP-4.2.2/B-10/1-2010-0009*, Leibniz International Proceedings in Informatics (LIPIcs), Budapest, Hungary, 2012.

[25] Zsolt Zombori, János Csorba, and Péter Szeredi. Static Type Inference for the Q language using Constraint Logic Programming. In Agostino Dovier and Vítor Santos Costa, editors, *Technical Communications of the 28th International Conference on Logic Programming (ICLP'12)*, volume 17 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 119–129, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[26] Zsolt Zombori and Gergely Lukácsy. A resolution based description logic calculus. In *Description Logics*, 2009.

[27] Zsolt Zombori, Gergely Lukácsy, and Péter Szeredi. Hatékony következtetés ontológiákon. In *17th Networkhop Conference 2008*, Budapest, Dunaújváros, 2008.

[28] Zsolt Zombori and Péter Szeredi. Szemantikus és deklaratív technológiák oktatási segédlet. Course handout.

[29] Zsolt Zombori and Péter Szeredi. Loop elimination, a sound optimisation technique for pttp related theorem proving. *Acta Cybernetica*, 20(3):441–458, 2012.

[30] Zsolt Zombori, Péter Szeredi, and Gergely Lukácsy. Loop elimination, a sound optimisation technique for pttp related theorem proving. In *Hungarian Japanese Symposium on Discrete Mathematics and Its Applications*, pages 503–512, Kyoto, Japan, 2011.