Tárgymodellváltozatok a ReALIS nyelvi elemzéshez

Kilián Imre

ReALIS ESzNyK / PTE TTK Informatika Tanszék
7624 Pécs, Ifjúság útja 6.
kilian@gamma.ttk.pte.hu

Kivonat: Forrásnyelvek célnyelvre átalakítását (pl. fordítóprogramokban) a két metamodell közötti átalakítási szabályrendszerként értelmezhetjük. A ReALIS elmélet (http://lingua.btk.pte.hu/realispapers) esetében ez a ReALLan forrásnyelv, és a választott Prolog nyelvű tárgymodellváltozatok közötti leképezés megadását jelenti. Szövegfeldolgozási célokra Prolog nyelven általában a relációs tárgymodellt alkalmazzák, mert ez a nyelv jellegéből fakadóan a szövegnyelvtani szerkezet relációt nemcsak az adott (felismerési) irányban, hanem fordítva, szöveggenerálási irányban is képes kiszámítani. Hatékonysági okokból azonban még további tárgymodellváltozatokat is érdemes számításba venni. A következtetéses tárgymodell esetében az elemzett szöveg szavai tényállításokká, a ReALIS lexikonban ábrázolt nyelvtani információk szabályokká képződnek le, amelyek egy célállításból meghíva előállítják az elemzett szöveg nyelvtani szerkezetét. A Prolog logikán túli eszközeinek használatával a deduktívan megvalósított elemzési feladat abduktívan megvalósított szöveggenerálássá alakítható. A ReALIS lexikonban tárolt nyelvtanának, és az elemzési folyamat aszinkron jellegének a Prolog visszafelé következtető stratégiája helyett azonban jobban megfelel egy előre haladó modell. A cikkben tárgyalt Contralog modell a Prolog előre haladó kiterjesztése, amellyel magyar mondatok ReALIS elmélet szerinti elemzését mutatjuk be.

1 ReALLan: a ReALIS nyelvleíró nyelve

Természetes nyelvi megvalósítások egyik sarokköve a nyelvi információk leírási módja. Ezt célszerűen valamilyen nyelvleíró formális nyelven tehetjük meg. Ha csupán a szöveges kinézetet megadó valamelyik *nyelvtani formalizmusra* (pl. BNF) szorítkozunk, akkor a kinézet oltárán feláldozzuk az adatszerkezetet és annak az értelmezését. Objektumorientált rendszerekben a formális nyelv *metamodelljét* pl. UML-ben adjuk meg, amely a nyelv elemeit grafikus módon rögzíti, és amelyhez az *érvényességi szabályokat* az OCL megszorítás-leíró nyelvvel adhatjuk meg. A mi esetünkben a Prolog megvalósítás miatt a ReALLan a Prolog egy résznyelve, vagyis az alapnyelvtan egyfajta alkalmazói megszorítása. Mivel a Prolog típustalan, ezért erre a célra egy Prolog *típusleíró nyelvkiegészítést* (ReALType) valósítottunk meg.

A ReALLan nyelvleíró nyelven a rendszer teljes lexikalizmusa miatt a lexikonbéli elemekhez rendelhető nyelvtani információk rögzítésének szabályait lehet megadni. A nyelv alapvetően *jegyszerkezetes*, egy jegyszerkezet mátrix megadása alapvetően

Prolog listában, `JEGY:ÉRTÉK` párokkal lehetséges. Ehhez az általános leíráshoz képest a következő *bővítéseket* és *nyelvtani könnyítéseket* (syntactic sugar) tesszük lehetővé:

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

```
RDES1:=[argn(ord(-7,nei),cat(+2,noun),
        case(+2,nom)),
        argd(cat(+7,gqd))]
```

…kifejezés futásidőben egyesíti a Prolog változót, mint referenst a szövegben megfelelő helyzetben talált alanyesetű, főnévvel úgy, hogy a szerkezet általánosított kvantordeterminánsszerepben van.

2 Tárgymodell: Horn-klózok

A tárgymodellek leírásához érdemes rögtön az átalakítási szabályrendszert is hozzákapcsolni. Ha a szigorú objektumorientáltság elvei mellett maradunk, akkor ez úgy történik, hogy a forrás- és a célkörnyezet metamodelljét kapcsolatnyalábbal kapcsoljuk össze, melyet az átalakítások szabályait rögzítő OCL-megszorításokkal látunk el. Bár most nem kívánjuk az UML modelleket bemutatni, a metamodellek és az átalakító relációk fogalma a modellező eszköztől független, és a Prologhoz kötődő környezetben is alkalmazható úgy, hogy a forrás- és célkörnyezet fogalmait, valamint a közöttük megvalósítandó *átalakítási relációt* adjuk meg.

A célkörnyezet a *Horn-klózok osztálya*. Ez az elsőrendű logika azon részosztálya, amelyekben a klózok következményoldalán több literál diszjunkciója helyett legfeljebb *egyetlen literál* állhat.

$p_1; p_2;\ldots p_k :- n_1, n_2,\ldots, n_1.$

A részosztály azért figyelemre méltó, mert a Prolog programozási nyelv is ezt használja úgy, hogy a következtetéseket a háttérben egy rögzített stratégiájú, rezolúciós tételbizonyító végzi. A visszafelé haladó, lineáris-, egység- és alaprezolúciós stratégia tételbizonyításra gyengécskének tűnik, de cserébe a nyelv nem logikai eszközeivel meglehetősen rugalmas és magasszintű működés írható elő.

A ℜeALIS céljaira a Horn-klózokra alapuló relációs és következtetéses tárgymodellt is, ez utóbbira pedig a Prolog eredeti, *visszafelé haladó*, ill. a Horn-klózok újonnan kifejlesztett, *előre haladó* értelmezésére alapuló tárgymodellt is kidolgoztuk.

2.1 Relációs tárgymodell

A Horn-klózok relációs tárgymodell szerinti alkalmazásakor egy program bemenet/kimenet relációját egy adott Prolog szabály számítja ki. Ha egy reláció több részrelációból van összetéve, akkor azokat a szabály feltételében nevezzük meg úgy, hogy a be- és kimenő paraméterek egymáshoz láncszerűen kapcsolódnak. Az ilyen szerepű változókat a Prolog programozók *akkumulátorpárnak* nevezik.

```
reláció(BE,KI):-
  rész1(BE,TMP1),rész2(TMP1,TMP2),…,részN(TMPN-1,KI).
```

A Definite Clause Grammar (DCG) formalizmus relációs tárgymodell szerinti nyelvtani elemzésekor a <bemenet-elemzetlen szöveg> párt használjuk akkumulátorként, a tetszőleges argumentumszerkezethez az akkumulátorpárt pedig a DCG előfordító maga hozza létre.

```
nonterm(…)→nonterm1(…),nonterm2(…),…,nontermN(…).

nonterm(…,BE,KI):-
  nonterm1(…,BE,TMP1),nonterm2(…,TMP1,TMP2),…,
  nontermN(…,TMPN-1,KI).
```

A megoldás egyik hátránya: a *relációk nemdeterminisztikus kiértékelése* miatt az eredményreláció számossága legrosszabb esetben az egyes részrelációk számosságának a szorzata is lehet. Ha viszont a szorzatban az első részreláció számossága nagyobb, akkor a nemdeterminizmus visszalépéses kezelése miatt egészen az első relációig tartó, ún. *mély visszalépés* történik.

A ℜeALIS relációs tárgymodell szerinti megvalósításában a bemenő paraméter az elemzendő szöveg, a kimenő pedig a szövegnek megfelelő logikaikifejezés-szerkezet. Értelmes részrelációk lehetnek: szóalaktani, nyelvtani-szemantikai elemzés, ill. pragmatika. Ilyen értelmezés mellett ugyanazt a szabályt használhatjuk elemzésre, (ha híváskor TEXT adott, LOGEXPR viszont változó), illetve szöveggenerálásra is (ha híváskor TEXT változó, de LOGEXPR adott).

```
text2logic(TEXT,LOGEXPR):-
  morphology(TEXT,MORPHLIST),
  syntaxSemantics(MORPHLIST,PUREEXPR),
  pragmatics(PUREEXPR,LOGEXPR).
```

Sajnos a relációs tárgymodell és az ezzel összefüggő Prolog DCG formalizmus a mi céljainkra nemigen alkalmas. A ℜeALIS környezeti feltételei (pl. vonzatok bizonyos távolságban) csak úgy lennének elemezhetők, ha azokat a bemenő szövegben előre-hátra mozgással ellenőriznénk. Ennek a megvalósítása is körülményes, és komoly hatékonysági aggályokat is felvet.

A ℜeALIS megvalósítás célkitűzése a szöveg és a diskurzusreprezentáció közötti reláció kiszámítása. Ez (Prolog-szerű értelmezésben) mindkét irányú kapcsolatot jelenti. Ha a szöveg adott, akkor a program azt a reprezentációs kifejezést számítja ki,

amely az adott logikai rendszerben és az interpretáló belső tudatállapotát leíró tudásbázisban (ontológiában) kiértékelhető, bizonyítható, vagy hozzávehető a tudásbázishoz. Az ellenkező irányban: ha a tudáskezelő összetevő által (pl. egy kérdésre adott válaszként) egy logikai kifejezést kapunk, akkor a reláció a szöveg képét állítja elő.

A megoldás másik hátránya, hogy a szöveg legalább egy bekezdésnyi, de esetleg akár több oldalnyi hosszú is lehet. Ez egyrészt a feldolgozás időigényét behatárolja, másrészt a hosszú bemenő adatokon az igen mély visszalépések csökkenthetik az elemzés hatékonyságát. Harmadrészt a szélsőségesen összetett adatszerkezetek sok Prolog-megvalósítás fizikai határait is feszegethetik (pl. veremtúlcsordulást okozhatnak).

2.2 Következtetéses tárgymodell Horn-klózokon

A következtetéses tárgymodell esetében a bemenő szöveget nem listaparaméterként, hanem *tényállításokként* ábrázoljuk. A cikkben feltételezzük, hogy a szóalaktani elemzés már megtörtént, és már csak a nyelvtani-szemantikai elemzés van hátra.

```
word(peter,1,1,noun('Péter',proper,nom,sing-3)).
word(peter,1,2,verb('hasonlít',[], decl, pres, sing-3)).
word(peter,1,3,noun('az',pro(point),sub,sing-3)).
word(peter,1,4,art(def,cons)).
word(peter,1,5,adj('vörös')).
word(peter,1,6,adj('ukrán')).
word(peter,1,7,adj('futó')).
word(peter,1,7,noun('bajnok',common,sub,sing-3)).
```

A ℜeALLan szabályok követel-kínál mechanizmusa szinte kínálja magát arra, hogy Horn-klózokká képezzük le őket. Az alábbi klóz pl. a 'hasonlít' ige és kötelező vonzatai közötti kapcsolatot írja le.

```
regArg2(ID,S,XV,verb('hasonlít',[],MODE,VTIME,AGR),
             XS,noun(SUBJ,SKIND,nom,AGR),-7,
             XO,noun(OBJ,OKIND,sub,OAGR),7):-
  verb(ID,S,XV,'hasonlít',[],MODE,VTIME,AGR),
  gqdet(ID,S,XS,SUBJ,SKIND,nom,AGR), order(XV,XS,-7,nei),
  gqdet(ID,S,XO,OBJ,OKIND,sub,OAGR), order(XV,XO,7,nei).
```

Szintén Horn-klózok írják le a ℜeALIS σ (sigma) függvényének megfelelő eventuális kifejezések részkifejezésekből történő felépítését is.

```
sigma3(ID,S,XV,TIME,SUB,OB,CLAUSE):-
  regArg2(ID,S,XV,verb('hasonlít',[],_MODE,VTIME,_AGR),
       XS,SUBJ,_PRS,XO,OBJ,_PRO), {TIME =.. [VTIME,_]},
  sigma3(ID,S,XS,TIME,SUB,CLAUSE,
           (desire(TIME,SUB,OB):-CONS)),
  sigma3(ID,S,XO,TIME,OB,CONS).
```

A fenti állítás eredményeképpen a mondat logikai alakjaként a következőket kapjuk. (A kettős implikáció egy egyszerű normáló program segítségével átalakítható feltételek konjunciójává.)

```
CLAUSE=((similar(pres(T),SUB,OB):-
           run(T, OB),ukrain(T, OB),
           red(T, OB),champion(T, OB)):-
             name(T,SUB,'Peter'))
```

2.3 Visszafelé haladó tárgymodell (Prolog)

A visszafelé haladó tárgymodell magát a Prolog értelmezőt használja következtető motorként úgy, hogy az általános következtetéses tárgymodellt használja. Ebben a megközelítésben az elemzést a logikai alakra változóként hivatkozó *célállítás* hívásával indítjuk. Ha visszavezethető a célállítás a szöveget rögzítő tényállításokra, akkor a mondat elemezhető volt, és a közben elvégzett változóhelyettesítésekből kiadódik a célállításban szereplő logikai alak is.

A megközelítés egyik hátránya, hogy a bizonyításhoz *hipotézist* kell felállítani, ez gyakorlatilag a célállítás. A bizonyítás időpontjában már minden ténynek ismertnek kell lennie – a rendszer nem alkalmas csővezeték- (pipe) -szerű feldolgozásra.

Másrészt viszont a visszafelé bizonyítás logikája szerint még az ismétlődő részbizonyításokat is újra és újra elvégzi, ezzel romlik a hatékonysága.

A fentebb vázolt tárgymodell alapvetően *deduktívan, felismerőként* használható, mégis kicsi módosítással *abduktív, szöveggenerátor* célú használatra is alkalmas. Ha a célállítást a logikai alak megadásával, de hiányzó szövegkép-tényállításokkal indítjuk, akkor a visszafelé bizonyítás során előbb-utóbb a tényállítások szintjéig ér. Ha az üres tényállításokat *visszaléptethető állításfelvétellel* (`assert`) valósítjuk meg, akkor a program végeredményben abduktív bizonyítást fog végezni.

```
word(ID,S,X,WORD):-
  (assert(w(ID,S,X,WORD));
  retract(w(ID,S,X,WORD)), fail).
```

2.4 Contralog: Horn-klózok előre haladó értelmezése Prologban

A Contralog tervezésekor cél volt, hogy az előre- és visszafelé haladó működés integrálható legyen úgy, hogy a logikai forrásnyelv ugyanaz (a Horn-klózok nyelve), amit részben maga a Prolog visszafelé haladóan, részben pedig az előrehaladó motor akként értékelhet ki. A kétféle rezolúciós stratégia pedig a programozó által vezérelhetően legyen váltható: egyrészt a Prologból legyen meghívható az előrehaladó motor, másrészt az előrehaladó végrehajtásból legyen meghívható a Prolog.

A Contralog programnyelv a Horn-klózok nyelvét (a Prolog nyelvet) előrehaladó stratégiát megvalósítva képezi le a Prolog nyelvre magára úgy, hogy egy inkrementális fordítóprogram a beolvasott Contralog-szabályokat Prolog-szabályokká fordítja le, és a szabványos Prolog futtatókörnyezetben működteti. [4]

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

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

A Prolog-Contralog kapcsolatot kétféleképpen lehet működtetni:

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

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

- minden tárgymodulban létrehoztunk egy, a *munkatáblát teljesen törlő* Prolog eljárást, amit a `MODULE:clean` hívással indíthatunk.
- egyes tények kikövetkeztetésekor *letilthatjuk a következtetést* az adott szálon (a tényt a munkatáblán tároljuk ugyan, de a megfelelő tüzelő eljárásokat nem hívjuk meg). Ezt a működést a `:- lazy NAME/ARITY.` deklaráció hatására válthatjuk ki.
- egyes tények kikövetkeztetésekor az *azonos névjegyű tényeket mind töröljük* a munkatábláról (`:- var NAME/ARITY.`), vagy egyes argumentumokat – a relációs technológiához hasonlóan – kulcsként tekintve, csak az azonos kulcsú tényt töröljük. Ezt a `:- key(NAME(KEYVECTOR)).` deklarációval válthatjuk ki, ahol a `KEYVECTOR` szerkezet egy argumentumlista, ahol a „+” jel azt jelzi, hogy az argumentum kulcsként szerepel, a „-” pedig azt, hogy nem.

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

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

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

A fenti esetben ez a következő Prolog-kód létrehozását jelenti:

```
fire_b:- assert(b), test_b.
fire_c:- assert(c), test_c.

test_b:- c, fire_a.
test_c:- b, fire_a.
```

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

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

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

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

2.5 Előre haladó tárgymodell (Contralog)

Az előre haladó tárgymodell esetében a *szabályalkalmazási rohamokat* (burstout) az egyes mondatelemek, mint tények felvétele (beérkezése) indítja. A tények érkezhetnek *aszinkron módon*, időben elcsúsztatva, sőt akár tetszőleges sorrendben is: egy következtetési lépés akkor történik meg, ha minden feltétel megérkezett és rendelkezésre áll. Bár van lehetőség a *következtetési fa ágainak nyírbálására*, a következmények a teljes gazdagságukban előállnak, ha ezekből néhány illeszkedik a megadott célállításokra, akkor a következtetés leáll.

A modell előnye, hogy az egyszer bebizonyított tényeket tároljuk, és azokat akárhányszor fel lehet még használni.

Sajnos az előrehaladó modell abduktív módon szöveggenerálásra történő használata nem látszik kézenfekvőnek.

3 Értékelés

A tesztmondatok elemzése a bemutatott modellváltozatok alapján elegendő tapasztalatot szolgáltatott. A következő lépés a ℜeALLAN-Horn-klóz fordítóprogram megírása lehet. Károly Márton munkájában az elemzési modellt modalitások beépítésével egészíti ki. A modalitások kezelése pedig kijelöli az utat a háttérben alkalmazott tudástár összetevő megtervezéséhez – egy *multimodális többszereplős logikai következtető rendszer* képében.

A szerzőt e cikk alapjait jelentő kutatásaiban az OTKA T60595 sz. projektje támogatta, a konferencia-részvételt pedig a TÁMOP-4.2.1.B-10/2/KONV/2010/ KONV-2010-0002 (A Dél-dunántúli régió egyetemi versenyképességének fejlesztése).

Itt szeretnék köszönetet mondani a ℜeALIS projektbéli munkatársaimnak, Alberti Gábornak, Kleiber Juditnak és Károly Mártonnak a nyelvészeti információk önzetlen átadásáért és a jól célzott, és egyben megfelelően adagolt, a cikk végső példányára is kiható megjegyzéseikért.

Hivatkozások

1. Clockshin-Mellish: Programming in Prolog. Springer Verlag, Berlin, Heidelberg, New York (1994)
2. Alberti, G.: ℜeALIS. Interpretálók a világban, világok az interpretálóban. Akadémiai Kiadó, Budapest (2011)
3. Alberti, G., Kilián, I.: Vonzatkeretlisták helyett polaritásos hatáslánccsaládok - avagy a ℜeALIS σ függvénye. In: VII. Magyar Számítógépes Nyelvészeti Konferencia. Szegedi Tudományegyetem, Informatikai Tanszékcsoport, Szeged (2010) 113–127
4. Kilián, I.:: Contralog: egy előre haladó, Prolog-konform következtető motor, és alkalmazása ℜeALIS nyelvi elemzésre. In: SzámOkt 2011. konferencia kiadványa. Erdélyi Magyar Műszaki Tudományos Társaság, Kolozsvár (2011) 199–205
5 Nakashima, H.: Term Description: A Simple Powerful Extension to Prolog Data Structures Electrotechnical Laboratory, Umezono, 1-1-4, Ibaraki, Japan (1985)