Mecanism de unificare pentru gramatici de limbaje naturale

Andrei RARES
Catedra de Calculatoare, Universitatea POLITEHNICA Bucuresti

Unificarea este un capitol întâlnit în orice abordare a prelucrarii limbajelor naturale. Limbajele naturale sunt definite, ca si limbajele de programare, prin gramatici specifice. Spre deosebire de limbajele de programare, limbajul uman este mult mai complex si prezinta un nivel ridicat de abstractizare si ambiguitate. De aceea au fost necesare structuri de date si proceduri speciale pentru a îl prelucra.

Gramaticile limbajelor naturale s-au lovit deseori de probleme aproape insurmontabile din punct de vedere al prelucrarii lor pe calculator. Pentru un elev este foarte usor, de exemplu, sa faca analiza sintactica a frazei "Ionel se uita la avionul de hârtie care plutea în aer si se minuna". Aceasta deoarece el întelege sensul frazei si îsi da seama ca singurul care se poate minuna este Ionel. Pentru un calculator însa, întelegerea sensului unui text este înca o problema departe de a fi rezolvata. Analiza sintactica facuta de el este limitata în principiu la cazurile în care nu exista ambiguitati si care nu necesita si o analiza semantica a textului.

Notiunea de unificare îsi are originile la mijlocul anilor '60, în munca de cercetare dusa de J. A. Robinson asupra demonstrarii automate de teoreme. Aplicarea propriu-zisa a acestei notiuni în domeniul lingvisticii computationale a început abia prin anii '80. În principiu, unificarea este operatia care determina daca doua structuri sunt compatibile prin construirea celei mai generale structuri compatibile cu amândoua. Ea este folosita la determinarea regulii de productie care se poate aplica asupra unei propozitii.

Cuvinte cheie: *arbore, categorie structurata, copiere întârziata, dag, duplicarea caii, formalismul PATR, limbaje naturale, unificare.*

1. Gramatici de unificare

Gramaticile de unificare sunt acele gramatici utilizate în prelucrarea limbajelor naturale si care folosesc unificarea ca operatie de baza. O regula de productie din cadrul unei astfel de gramatici arata la fel cu o regula de productie a unei gramatici obisnuite:

$S \rightarrow NP\ VP.$
<NP per> = <VP per>
<NP num> = <VP num>

unde S reprezinta Sentence (propozitie), NP reprezinta Noun Phrase (grup substantival) si VP reprezinta Verb Phrase (grup verbal).

În plus fata de o regula obisnuita, mai apar în exemplul prezentat doua constrângeri care precizeaza ca grupul substantival (NP), reprezentat într-o propozitie simpla de subiect, se acorda în persoana (per) si numar (num) cu grupul verbal (VP), reprezentat în general de predicat. Asadar, intervin anumite restrictii care trebuie verificate. Pentru aceasta avem nevoie de un lexicon în care, pe lânga cuvintele ca atare, sa fie incluse si caracteristicile fiecaruia în parte: persoana, numar, gen, caz s.a.m.d. Vom folosi în acest scop formalismul PATR de reprezentare a categoriilor structurate. În figura urmatoare este data descrierea cuvân-tului "carte" în formalismul PATR:

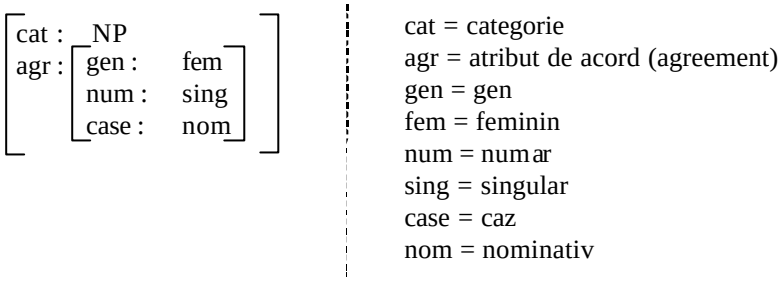

Fig. 1. Cuvântul "carte", reprezentat în formalismul PATR

Structura utilizata în majoritatea cazurilor pentru a reprezenta în memoria unui calculator o categorie structurata se numeste “dag”. Dag este prescurtarea de la “Directed Acyclic Graph”, adica graf aciclic orientat. Arcele unui dag sunt marcate cu etichetele atributelor, iar nodurile cu valorile lor. Dagul corespunzator cuvântului “carte” prezentat anterior este afisat în figura 2:

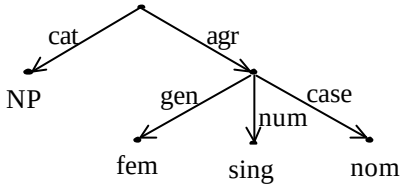

Fig. 2. Dag-ul asociat cuvântului "carte"

Se observa ca o categorie structurata consta dintr-o multime de atribute. Fiecare atribut este alcatuit dintr-o pereche de tipul <eticheta, valoare>. Eticheta este un simbol (în cazul de fata “cat”, “agr”, “gen”, “num”, “case”), iar valoarea poate fi un simbol atomic (“NP”, “fem”, “sing”, “nom”), simbolul vid (nil) sau o alta categorie structurata (a se vedea valoarea atributului etichetat cu “agr”). Prin urmare, categoriile structurate sunt definite recursiv si pot exista oricâte nivele de imbricare. Pentru simplificarea exprimarii, în cele ce urmeaza vom desemna prin “atributul x” atributul care are eticheta x.

2. Subsumare, generalizare si unificare

Definitie: Un dag A *subsumeaza* un dag B (“$A \subseteq B$”) daca si numai daca:

- fiecare atribut din A care are valoare atomica se regaseste identic în B;
- pentru oricare doua atribute din A care folosesc în comun aceeasi valoare, atributele corespondente din B folosesc la rândul lor în comun o valoare identica;
- valoarea fiecarui atribut compus din A subsumeaza valoarea atributului corespun-zator din B.

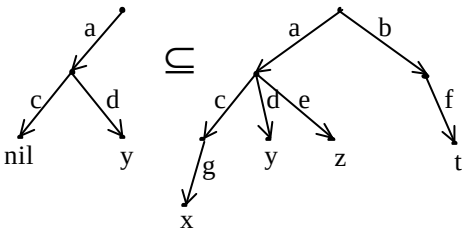

Fig. 3. Exemplul 1

Definitie: *Generalizarea* a doua dag-uri (A si B) este cel mai mare dag care subsumeaza ambele dag-uri ("A ∩ B").

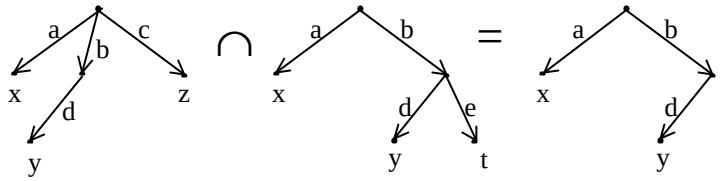

Fig. 4. Exemplul 2

Definitie: *Unificarea* a doua dag-uri (A si B) este cel mai mic dag care extinde ambele dag-uri ("A o B", sau "A ∪ B"), daca acest dag exista, altfel unificarea esueaza (sau "nu este definita").

În termeni matematici, date fiind doua dag-uri, D' si D",

$$D' \text{ o } D'' \Leftrightarrow (\exists)\ D \text{ a.î. } D' \subseteq D \text{ si } D'' \subseteq D \text{ si } (\lceil \exists)\ D_1 \text{ a.î. } D_1 \subseteq D \text{ si } D' \subseteq D_1 \text{ si } D'' \subseteq D_1.$$

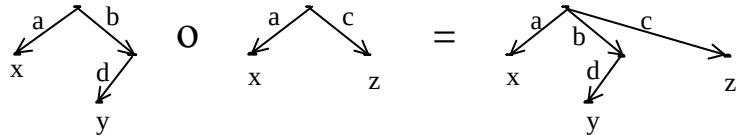

Fig. 5. Exemplul 3

3. Un algoritm de unificare si un contra-exemplu

Când un parser aplica o regula sintactica, el unifica anumite atribute ale cuvintelor frazei analizate pentru a verifica respectarea constrângerilor (referitoare la acordurile gramaticale) impuse de acea regula. Atunci când cuvintele sunt combinate pentru a forma propozitii si fraze, unificarea nu este aplicata direct cuvintelor din lexicon, întru-cât aceasta ar însemna modificarea irepara-bila a acestuia. Când este analizat un cuvânt dintr-un text, mai întâi este facuta o copie a corespondentului sau din lexicon, dupa care se aplica operatia de unificare asupra copiei. Costul efectuarii acestor copii (masurat în timp de calcul si spatiu de memorie) este mult mai mare decât cel al unificarii propriu-zise, prin urmare el reprezinta o buna parte din timpul necesar parsing-ului. De aseme-nea, o mare parte a efortului de copiere este risipita: atunci când o unificare esueaza, acest lucru se datoreaza de multe ori nepo-trivirii unor aspecte particulare ale dag-urilor (cum ar fi diferenta valorii unui atribut), nu a unor aspecte de ansamblu.

O solutie care reduce atât timpul de calcul, cât si spatiul de memorie ocupat este folosirea în comun a structurilor. Aceasta solutie a fost prezentata în 1985 de Martin Kay (cercetator la *Centrul de Cercetari Xerox de la Palo Alto* si la *Centrul pentru Studiul Limbajului si al Informatiei de la Universitatea Stanford*) si de Lauri Karttunen (cercetator la *Centrul de Inte-ligenta Artificiala al SRI International* si la *Centrul pentru Studiul Limbajului si al Informatiei de la Universitatea Stanford*), având la baza o idee a lui Martin Kay. Principalele trasaturi ale solutiei lor sunt:

- arborii binari ca mediu de stocare pentru categoriile structurate;
- copierea întârziata;
- adresarea relativa a nodurilor din arbore;
- strategii pentru mentinerea arborilor cât mai echilibrati.

3.1. Arborii binari ca mediu de stocare pentru categoriile structurate

Sa reluam exemplul cuvântului “carte” din figura 1. În memoria calculatorului, aceasta categorie structurata poate fi reprezentata ca arbore binar în felul urmator (a se considera numai arborele a carui radacina este eti-chetata cu "*original*":

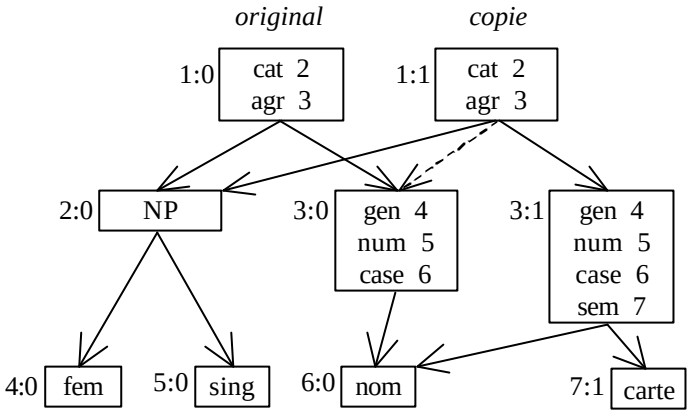

Fig. 6. Exemplul 4

Numerotarea nodurilor (este vorba de numarul dinaintea semnului ":") este facuta astfel: radacina este etichetata cu 1, iar pentru un nod cu o valoare oarecare n (n≥1), fiul din stânga va fi etichetat cu 2*n, iar fiul din dreapta cu 2*n+1. Numerele care apar în dreapta fiecarui atribut se refera la indexul nodului în care se afla valoarea atributului. Avem în acest caz o adresare absoluta a nodurilor. O conditie obligatorie este ca radacina arborelui binar sa contina radacina dag-ului.

3.2. Copierea întârziata

Spre deosebire de copierea obisnuita, în copierea întârziata este duplicata (initial) numai radacina arborelui, restul arborelui fiind utilizat în comun (vezi Fig. 6, fara nodurile *3:1*, *7:1* si sagetile adiacente lor). Pentru a putea recunoaste care radacina este cea originala, au fost adaugati, pe lânga indecsii nodurilor, indecsi numerici care indica generatia radacinii de care apartin nodurile respective. Indecsii de generatie apar în dreapta indecsilor nodurilor, separati prin semnul “:”.

Daca un nod (care nu este radacina) este pe cale de a suferi transformari, atunci, pentru a nu afecta structura originala, vor fi duplicate toate nodurile de pe calea de la radacina si pâna la nodul respectiv inclusiv. De ase-menea, sunt actualizate indexurile de ge-neratie ai nodurilor duplicate, corespunzator cu radacina de care apartin.

Sa presupunem ca la structura cuvântului “carte” este adaugat un nou atribut, rezul-tând structura urmatoare:

$$\begin{bmatrix} \text{cat} : \text{NP} \\ \text{agr} : \begin{bmatrix} \text{gen} : \text{fem} \\ \text{num} : \text{sing} \\ \text{case} : \text{nom} \\ \text{sem} : \text{carte} \end{bmatrix} \end{bmatrix}$$

Fig. 7. Exemplul 5

Mai presupunem ca vrem ca aceasta schimbare sa afecteze numai copia, nu si origi-nalul. Arborii rezultati vor fi cei din figura 6, fara sageata punctata. Arborele modificat (copia) contine cât mai mult posibil din arborele original. În acest fel efortul de copiere al arborilor este redus la minim.

3.3. Adresarea relativa

Autorii algoritmului de fata, din motive mai mult sau mai putin întemeiate, au ales ca adresarea nodurilor în arbore sa se faca prin utilizarea unei perechi de indecsi relativi la nodul din care se face aceasta adresare, nu relativi la radacina ca pâna acum. Aceasta în special datorita faptului ca pentru a cauta un nod, în cazul adresarii absolute, trebuie pornit mereu de la radacina, în loc de a porni de la nodul curent. Iata cum arata arborii binari din figura anterioara (îi vom mai numi si "dag-uri binare", printr-o fortare de limbaj), în cazul utilizarii indecsilor relativi:

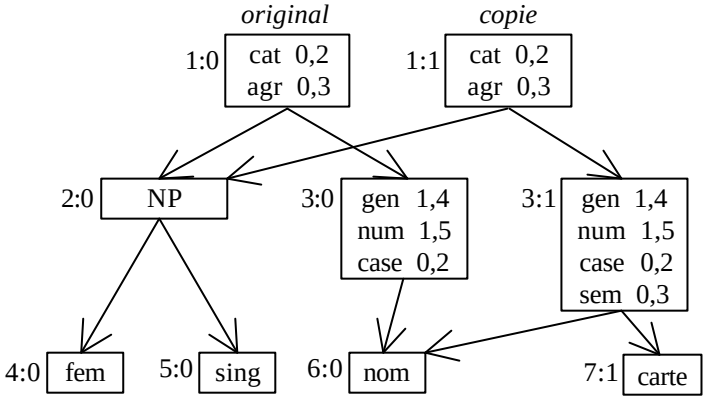

Fig. 8. Exemplul 6

Indecsii relativi aflati în dreapta fiecarui atribut arata calea care trebuie parcursa de la nodul curent pâna la nodul care contine valoarea atributului din nodul curent. Primul numar din perechea de indecsi reprezinta numarul de noduri care trebuie parcurse în sus pâna la cel mai apropiat stramos comun al celor doua noduri. Al doilea numar, reprezentat în binar, este calea de la acest stramos comun la nodul care contine valoarea atributului din primul nod. Aso-ciind cifra 0 cu "STÂNGA" si cifra 1 cu "DREAPTA" si neglijând prima cifra (care este 1 în toate numerele binare), obtinem aceasta cale. De exemplu, în cazul nodului 5 (101), eliminând prima cifra (1), ne ramâne "01", care se traduce prin "STÂNGA-DREAPTA", adica exact calea de la radacina la nodul 5.

Se observa în Fig. 8 ca atributele "cat" si "case" au aceeasi pereche de indecsi relativi. Acest lucru se datoreaza faptului ca ele se situeaza în aceeasi pozitie, relativ la nodurile care contin valorile lor.

3.4. Echilibrarea arborilor

Pentru a economisi cât mai mult din efortul de copiere, se "încuibeaza" (include) unul din arbori în celalalt, într-un mod cât mai echilibrat cu putinta. Un arbore are un timp mediu de cautare a unui nod cu atât mai bun cu cât este mai echilibrat, deoarece acest timp este proportional cu înaltimea arbo-relui. În ciuda eforturilor de a echilibra rezultatul final, aceasta "altoire" produce un dezechilibru care poate mari semnificativ timpul de cautare.

Sa reluam ultimul exemplu, considerând însa ca nu extindem un dag binar existent, ci ca unificam doua dag-uri binare (de fapt copiile lor):

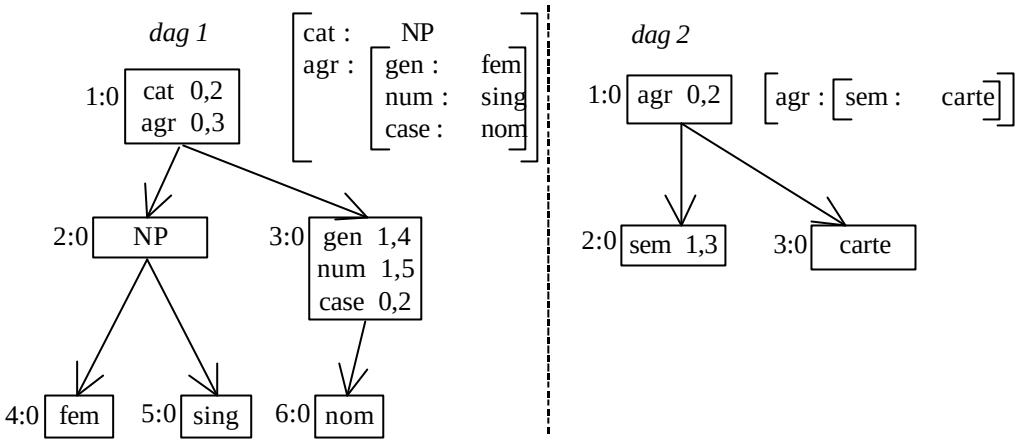

Fig. 9.

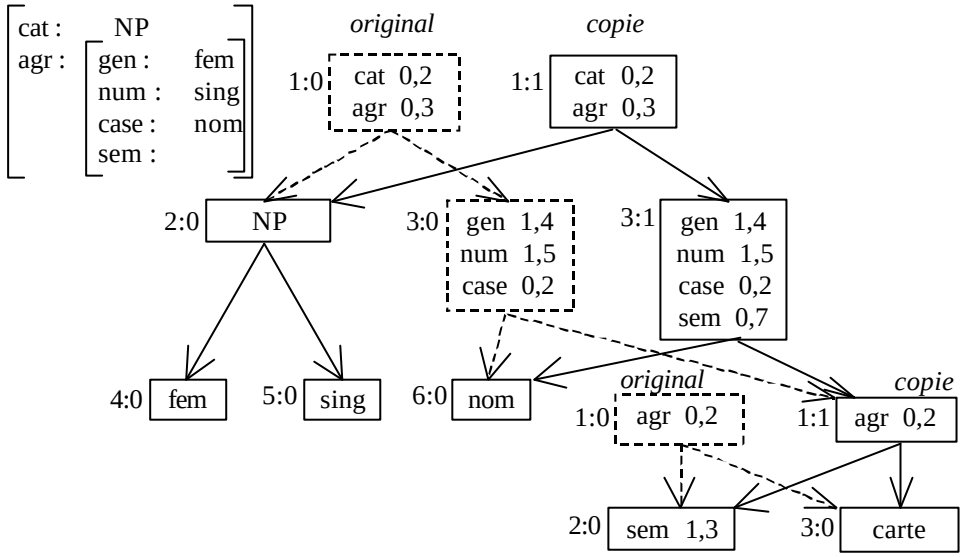

Fig. 10. Rezultatul unificarii

Întrucât am inserat dag-ul 2 în cel mai "înalt" loc posibil, înaltimea arborelui rezul-tat este mai mica decât în cazul în care łam fi atasat, de exemplu, la nodul *4:0*.

3.5. Un contraexemplu

Ceea ce nu se spune în algoritm si nu apare în desenele explicative ale autorilor, dar apare evident la încercarea de a implementa algoritmul, este faptul ca, pe lânga arcele care pleaca de la fiecare nod catre fiii sai, mai exista un arc (cel putin) care pleaca spre parintele nodului. Daca nu ar exista acest arc, nu am putea pleca de la eticheta unui atribut catre valoarea sa suind mai întâi câteva noduri în sus. Arcul de la un nod catre parintele sau îl vom desena punctat.

Sa consideram exemplul urmator. Vrem sa unificam doua categorii structurate:

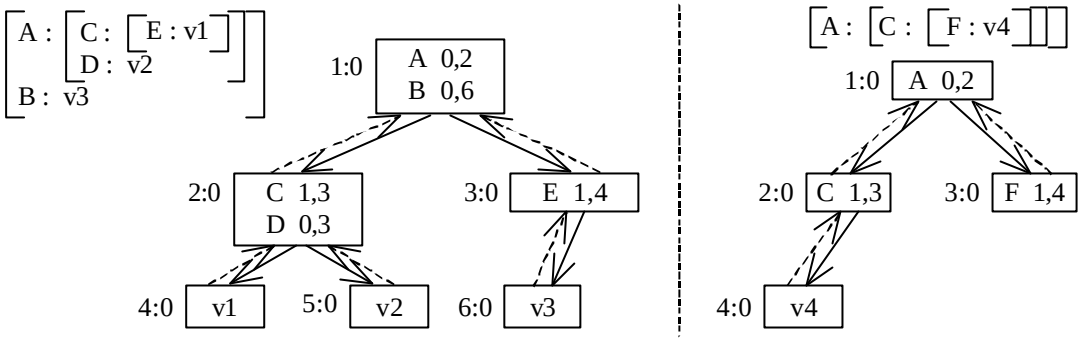

Fig. 11.

Teoretic, rezultatul unificarii este:

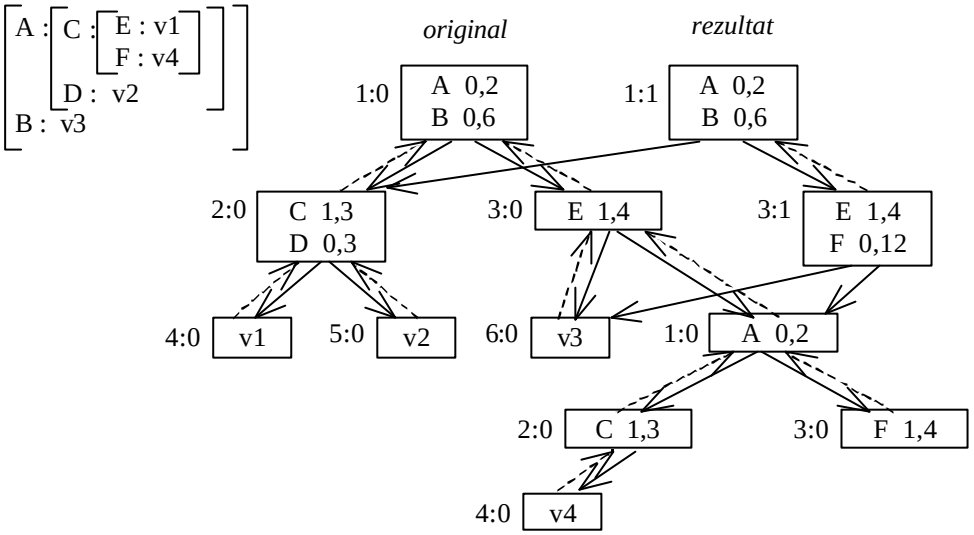

Fig. 12.

Daca ne uitam însa pe arborele rezultat (având radacina *1:1*) si încercam sa ajungem la atributul F pornind de la atributul A, vom constata ca în loc sa ajungem în nodul *3:1*, ajungem în vechiul nod etichetat *3:0*, care contine numai atributul E. Acest lucru se întâmpla deoarece atunci când pornim în sus de la atributul C (nodul *2:0*), aflat pe calea de la A la F, ajungem în radacina originala a arborelui, nu în ceea duplicata. Ori, numai radacina duplicata este cea care contine referinta catre nodul cu atributele E si F (*3:1*). Prin urmare, rezultatul va fi diferit de ceea ce asteptam. De fapt, o parte din necazuri provine din faptul ca noi repre-zentam un arbore multi-cai printr-un arbore binar. În desenul de dedesubt este dat cuvântul "carte" în cele doua tipuri de repre-zentare: dag obisnuit (îl vom numi simplu "dag") si dag binar. De data aceasta sagetile punctate sunt utilizate pentru a evidentia modul în care arborele logic (dag-ul) este mapat pe arborele fizic (dag-ul binar):

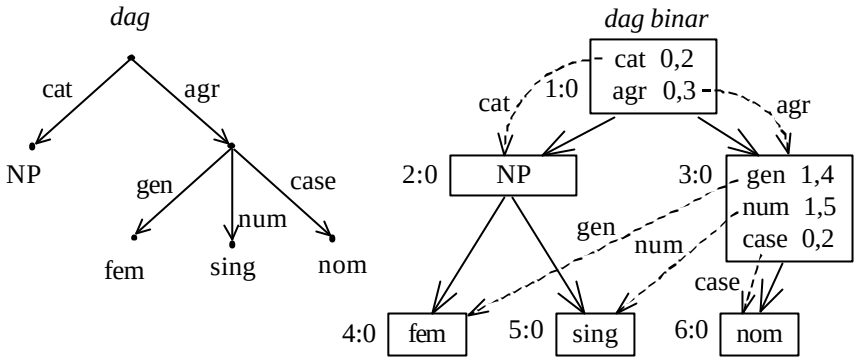

Fig. 13.

Devine evident faptul ca utilizarea arborilor binari pentru a stoca dag-uri reprezinta un mod simplu de a complica lucrurile. Dupa una sau mai multe unificari, manipularea structurii rezultate devine foarte grea si neeconomica, atât în privinta spatiului ocu-pat, cât si a timpului consumat.

4. O alternativa: algoritmul de unificare bazat pe reprezentare directa si copiere întârziata

Deseori, atunci când vrem sa rezolvam o problema si lucrurile se complica la fiecare pas, solutia este simpla si evidenta (daca o descoperi). Algoritmul pe care îl propun în locul celui anterior l-am conceput împreuna cu colegul si prietenul meu Radu Palanca. Solutia noastra are la baza urmatoarele principii:

- reprezentarea directa a categoriilor structurate, ca arbori multicai (arborele fizic este chiar arborele logic, adica dag-ul asociat structurii);
- copierea întârziata, bazata pe aceleasi principii ca în cazul dag-urilor binare.

Revenind la exemplul "carte", dag-ul corespunzator si reprezentarea lui exacta în memoria calculatorului sunt date în figura 14 (implementarea a fost facuta în LISP). O copie a unui dag va diferi de acel dag numai prin nodul radacina. Numai în cazul în care este pe cale a se produce o modificare asupra unui nod din interiorul copiei, acest nod este duplicat, împreuna cu toate nodurile de pe calea pâna la radacina. Pentru a putea sti care este radacina de care apartin nodurile, ele sunt etichetate cu numere. Nodurile unui arbore vor purta acelasi numar ca si radacina de care apartin. Numerele cu care sunt etichetate radacinile sunt unice în spatiul dag-urilor. În acest scop este folosit un contor universal, incrementat ori de câte ori sunt create un nou dag sau o noua copie a unui dag. Iata cum ar putea arata dag-ul din Fig. 13, precum si cel cu care dorim sa îl unificam:

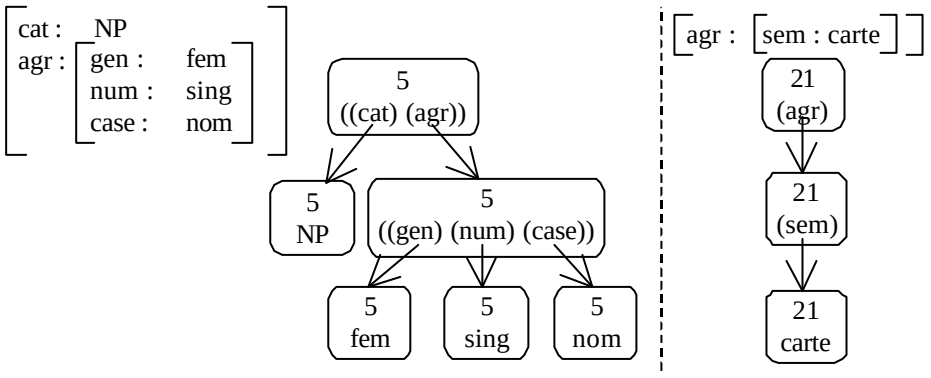

Fig. 14.

Rezultatul unificarii lor este:

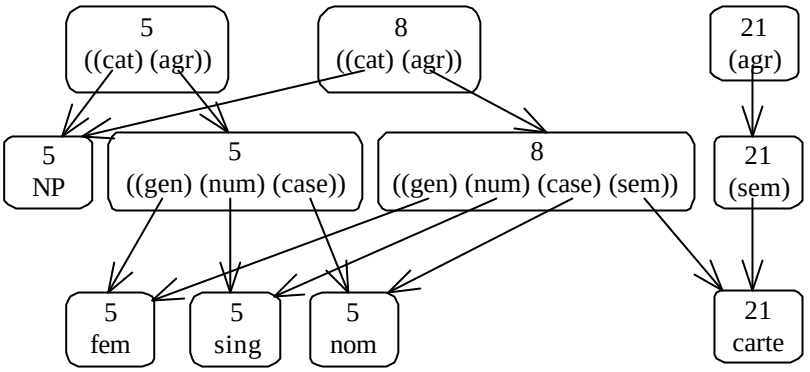

Fig. 15.

Duplicarea caii de la radacina pâna la un nod se face pornind de la radacina-copie (care nu va mai fi duplicata, întrucât ea este deja un duplicat) si mergând în jos, în paralel mergându-se si pe arborele original, fara a duplica vreun nod pentru moment. În acest fel nu mai avem nevoie de arce catre nodul parinte. În momentul în care pe ambii arbori se ajunge la un nod comun, se începe duplicarea de la acel nod în jos, inclusiv nodul respectiv. Pentru a putea reconstitui calea de la radacina la nodul care a provocat duplicarea, este pastrata permanent o lista cu nodurile parcurse pâna la un moment dat (de la radacina pâna la nodul curent).

Acest al doilea algoritm este, fata de primul, mai simplu, mai eficient si mai elegant. Economia de timp si spatiu este reala. Conform unor calcule estimative pe care nu am loc sa le redau aici, spatiul de memorie ocupat de cel de al doilea algoritm este de cel putin 2-3 ori mai mic, iar timpii de creare si unificare a dag-urilor este cu un ordin de marime mai mic. De altfel, experientele au aratat, de exemplu, un timp de creare a dag-urilor de 10-20 ori mai mic în cazul celei de a doua metode. În plus, implementarea celui de-al doilea algoritm a necesitat mai putin de o treime din timpul necesar pentru primul algoritm.

5. Concluzii

Al doilea algoritm este superior primului din toate punctele de vedere:

- spatiu ocupat de structura dag-ului;
- timp de constructie a dag-urilor;
- timp de unificare a dag-urilor;
- efort de implementare a algoritmului.

As putea adauga la aceasta lista si usurinta întelegerii algoritmului. Este mult mai sim-plu sa lucrezi cu arborii multipli reprezentati asa cum sunt ei decât "proiectati" pe un arbore binar.

Desi nu am întâlnit cel de-al doilea algoritm descris nicaieri, el poate sa fi existat dinainte de a îl concepe noi, dar sa nu îl fi întâlnit. Algoritmul de fata (ma refer la cel nou) poate fi utilizat pentru a implementa si alte operatii decât unificarea sau extensii ale unificarii. Se poate imagina chiar si o para-lelizare a acestui algoritm.

Prelucrarea limbajelor naturale nu consta numai din operatia de unificare. Ea implica o multime de aspecte, cum ar fi: parsing, inferente logice, achizitia, reprezentarea si analiza cunostintelor, traducerea automata a textelor s.a.m.d. Desi este un domeniu intens studiat în ultimii cincisprezece ani, puterea de calcul mica a procesoarelor si capacitatile de stocare reduse au facut ca rezultatele practice sa fie destul de timide, cel putin pâna acum câtiva ani. Masinile cu comanda verbala au capacitati limitate de "întelegere", iar traducatoarele automate fac înca destule greseli.

Asadar, în acest moment paharul este pe jumatate gol, sau pe jumatate plin, în functie de cât de pesimisti sau optimisti suntem. Daca ne

gândim însa la legea lui Moore, confirmata în timp, care spune ca la fiecare 18 luni se dubleaza capacitatea proce-soarelor, nu putem sa fim decât optimisti. Peste 10 ani calculatoarele vor avea deja posibilitati rezonabile de a analiza un text. La fel de adevarat este si faptul ca notiunea de "rezonabil" va evolua în aceasta perioada de timp, chiar daca nu dupa aceeasi lege.

Bibliografie

Gazdar G, Mellish C. - *Natural Language Processing in LISP* - Addison-Wesley Publishing Company, 1989.

Shieber S.M., Karttunen L., Pereira F., Kay M. - *More Notes from the Unification Underground: A second Compilation of Papers on Unification-Based Formalisms* - 20 August 1985.

Trausan-Matu S. - *Prelucrarea limbajelor naturale* - note de curs.