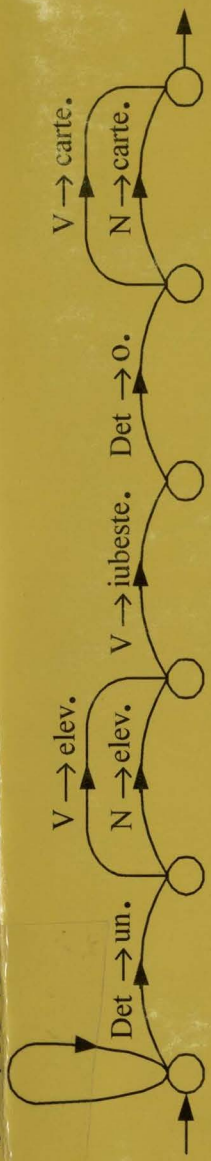
FLORENTINA HRISTEA

INTRODUCERE ÎN PROCESAREA LIMBAJULUI NATURAL CU APLICAŢII ÎN PROLOG



Editura Universităţii din Bucureşti

FLORENTINA HRISTEA

INTRODUCERE ÎN PROCESAREA LIMBAJULUI NATURAL CU APLICAȚII ÎN PROLOG

FLORENTINA HRISTEA

Este licenţiată a Facultăţii de Matematică - Universitatea din Bucureşti, secţia de informatică, în anul 1984. Este doctor în matematică al Universităţii din Bucureşti din anul 1996.

Autoare a cărţii "Sistemul de programe NORTON COMMANDER", Editura Tehnică, 1992. Autoare principală a cărţii "NORTON COMMANDER 4.0. NORTON UTILITIES 7.0 si 8.0", Editura Tehnica, 1994.

Participă ca responsabil de proiect din partea română la programul internaţional de cercetare în domeniul procesării limbajului natural "D-B-R MAT", având ca temă generală traducerea asistată de calculator.

Rezultatele practice obţinute în cadrul acestui program de cercetare sunt citate de Comisia Europeană (revista "Terminologie et Traduction", no.1, 1998, p. 140-163, autor Poul Anderson) şi menţionate ca reprezentând cea mai importantă contribuţie românească în domeniul **traducerii asistate de calculator.**

Coordonator al **"Centrului Regional de Informare** Pentru Cunoaşterea şi Standardizarea Resurselor şi a Instrumentelor Lingvistice Destinate Aplicaţiilor Avansate de Tehnologie a Limbajului**"**, înfiinţat de **Uniunea Europeana** în România, în cadrul programului de cercetare IST (prioritatea tematica IST-2000-8.1.5), coordonator al centrului de informare românesc corespunzător - RORIC (= "Romania – Regional Information Center").

FLORENTINA HRISTEA

INTRODUCERE ÎN PROCESAREA LIMBAJULUI NATURAL CU APLICAȚII ÎN PROLOG

EDITURA UNIVERSITĂȚII DIN BUCUREȘTI

2000

Referenți științifici: Prof. dr. SOLOMON MARCUS
Conf. dr. ing. ȘTEFAN TRĂUȘAN-MATU

Șos. Panduri, 90-92, București - 76235; Telefon/Fax 410.23.84
E-mail: editura@unibuc.ro
Internet: www.editura.unibuc.ro

Descrierea CIP a Bibliotecii Naționale
HRISTEA, FLORENTINA
Introducere în procesarea limbajului natural cu aplicații în PROLOG / Florentina Hristea
București: Editura Universității din București, 2000
Bibliogr.

ISBN 973-575-493-2

004.43 PROLOG

CUVÂNT ÎNAINTE

Lucrarea de față reprezintă cursul de procesare a limbajului natural pe care autoarea îl ține studenților din anul al patrulea ai secției "Informatică" a Facultății de Matematică - Universitatea din București.

Lucrarea se adresează, cu precădere, informaticienilor, adică programatorilor total lipsiți de cultură lingvistică. Tocmai de aceea ea își propune și să elucideze, în special prin intermediul notelor de subsol, o serie de termeni, majoritatea lingvistici, fără de care înțelegerea unui text referitor la procesarea limbajului natural, scris chiar și la un nivel introductiv, s-ar dovedi imposibil de realizat în profunzime. Acest lucru s-a făcut însă numai în măsura în care el era necesar înțelegerii textului de către nespecialiști, fără a constitui un scop în sine. Lucrarea se adresează, în egală măsură, lingviștilor care sunt familiarizați cu noțiunile elementare ale programării logice și ale limbajului Prolog, precum și tuturor acelora care sunt interesați de aspectele computaționale de bază ale studiului limbajului natural.

Alegerea limbajului Prolog pentru ilustrarea marii majorități a conceptelor introduse nu a fost întâmplătoare, întrucât considerăm că, dintre toate limbajele de programare disponibile în momentul de față, acesta este, foarte probabil, cel mai adecvat cerințelor impuse de prelucrarea limbajului natural. În acest sens, subscriem câtorva dintre argumentele favorabile acestui punct de vedere, expuse în [19], și anume: structuri de date extrem de complexe sunt ușor de construit și de modificat în Prolog, ceea ce facilitează reprezentarea structurilor sintactice și semantice, precum și a intrărilor lexicale; un program scris în Prolog se poate examina și modifica pe el însuși, ceea ce permite folosirea unor metode de programare foarte abstracte; Prologul este proiectat special pentru a se putea realiza reprezentarea cunoștințelor și este construit în jurul logicii de ordinul întâi, ceea ce face ca orice extensie a acestei logici să fie relativ ușor de implementat; un algoritm de căutare (de tip "depth-first") este încorporat în Prolog și este ușor de folosit în toate tipurile de analiză sintactică; unificarea este, de asemenea, încorporată în Prolog și poate fi în mod eficient folosită în construirea pas cu pas a structurilor de date ș.a. După cum se știe, limbajul Lisp nu prezintă decât primele două dintre aceste avantaje, în timp ce limbajele convenționale cum ar fi Pascal sau C nu dispun de nici unul dintre ele.

Cursul de față reprezintă o introducere în problematica lingvisticii computaționale și respectiv a procesării limbajului natural, realizată sub forma unei sinteze a trei dintre cele mai importante lucrări în domeniu, [4], [19], [29], dar folosind, în același timp, numeroase alte materiale și structurând informația gradat, în propria viziune a autoarei. Exemplele de programe preluate din [19] și [29] au fost reprogramate în SICStus Prolog, fiind testate pentru cazul limbii române. Sunt furnizate în mod constant exemple, atât pentru limba engleză, cât

şi pentru limba română, acolo unde acest lucru este posibil. Sunt prezentate teoriile şi tehnicile de bază ale lingvisticii computaţionale, în marea lor majoritate concepute şi testate pentru limba engleză şi se fac o serie de observaţii şi de comentarii privitoare la adaptarea acestora în cazul altor limbi şi, în mod special, al limbii române.

Dincolo de analiza tehnicilor fundamentale ale lingvisticii computaţionale şi ale prelucrării limbajului natural, lucrarea îşi propune clarificarea unor concepte şi a unor termeni de bază, precum şi crearea unei culturi generale în domeniu, prezentarea unui scurt istoric al contribuţiilor româneşti şi formarea unei viziuni de ansamblu asupra unui domeniu relativ controversat, în special datorită caracterului său interdisciplinar.

În funcţie de nivelul de pregătire al cititorului, precum şi de gradul de specializare în domeniu care se doreşte a fi atins, lucrarea poate fi parcursă în mai multe moduri. Într-o primă etapă ea poate fi citită cu ignorarea totală a notelor de subsol şi fără a se urma diversele trimiteri făcute în text. Un alt mod de a parcurge lucrarea, care are rolul de a oferi o imagine mai amplă asupra domeniului, precum şi clarificări suplimentare privitoare la unele dintre conceptele folosite, este acela al însoţirii lecturii de consultarea tuturor notelor de subsol, dar fără a se da curs trimiterilor din text. În fine, lucrarea poate fi parcursă cu urmărirea atât a notelor de subsol, cât şi a trimiterilor existente.

Structurată în şase capitole, lucrarea tratează concepte fundamentale clasice, dar, în egală măsură, abordează câteva dintre cele mai noi teme ale lingvisticii computaţionale, aducând discuţia în planul celor mai recente preocupări ale domeniului. Unele dintre aplicaţiile moderne pe care le aminteşte sau le descrie, în special în cadrul notelor de subsol (proiectul DBR-MAT şi parsing statistic pentru limba română, un parser bazat pe gramatici contextuale, WordNet ca bază de date lexicală interactivă şi ca reţea semantică), constituie subiecte de reflecţie şi eventuale teme de cercetare cu vaste posibilităţi de aprofundare, extindere şi generalizare.

Dintre diferitele niveluri la care se realizează analiza limbajului, lucrarea de faţă se concentrează asupra nivelului sintactic şi a celui semantic, precum şi a legăturii dintre acestea. Considerăm că un studiu de acest tip este absolut obligatoriu pentru realizarea unei introduceri în domeniu. Alte aspecte, cum ar fi cele legate de nivelul pragmatic al analizei limbajului sau de prelucrarea statistică a limbajului natural sunt numai amintite aici. Ele vor constitui obiectul unor cursuri speciale diferite şi deci a unor lucrări distincte.

Încheind acest curs, mulţumirile noastre se îndreaptă, în primul rând, spre profesorul Solomon Marcus, din al cărui îndemn am pătruns într-un domeniu de investigaţie pe cât de necunoscut, iniţial, pe atât de captivant în momentul de faţă. Domnia sa a avut amabilitatea să ne pună la dispoziţie şi o serie de materiale necesare în redactarea capitolului 3 al lucrării, motiv pentru care îi suntem încă o dată profund recunoscătoare.

În egală măsură, dorim să ne exprimăm gratitudinea faţă de prof. dr. Walther von Hahn (Universitatea din Hamburg, Catedra de limbaj natural)

pentru îndrumarea constant acordată și pentru nenumăratele ore de discuții științifice, care au avut rolul de a contribui la formarea noastră în acest domeniu.

Mulțumim, de asemenea, prof. dr. Dan Moldovan (Southern Methodist University, Dallas, Texas), precum și prof. dr. Jerry Hobbs (Artificial Intelligence Center, SRI International, Menlo Park, California), care au sprijinit acest demers și au avut amabilitatea de a ne pune la dispoziție o serie de materiale necesare în elaborarea capitolului 6 al lucrării.

Mulțumim colegului Marius Popescu, doctorand în domeniu, pentru numeroasele discuții purtate pe marginea acestui material.

Nu în ultimul rând, recunoștința noastră se adresează referenților științifici ai lucrării, care vor avea amabilitatea de a o citi și de a-și exprima opiniile asupra ei.

București, 24 august 2000 Autoarea

CUPRINS

CAPITOLUL 1

CONSIDERAȚII PRELIMINARE

Din punct de vedere științific, limbajul natural (uman) constituie obiectul de cercetare a numeroase discipline și, în primul rând, al lingvisticii sau "științei limbii". Același obiect de investigație interesează însă și filozofia, psiholingvistica, lingvistica matematică, lingvistica computațională etc.

Dintre acestea, lucrarea de față se concentrează asupra lingvisticii computaționale, al cărei principal scop este acela de a dezvolta *o teorie computațională a limbajului* folosind elemente ale informaticii (algoritmi, structuri de date etc.). Este de la sine înțeles că, în construcția unui model computațional al limbajului, trebuie folosit în mod eficient tot ceea ce se cunoaște deja prin intermediul celorlalte discipline.

Considerăm important faptul de a semnala încă de la început distincția care trebuie făcută între lingvistica computațională și lingvistica matematică, chiar dacă cele două discipline se întrepătrund adesea și, în orice caz, se completează reciproc. Este greșită, în opinia noastră, utilizarea acestor termeni ca fiind perfect echivalenți.

Lingvistica matematică este un termen general folosit pentru un număr de aplicații ale modelelor și procedurilor matematice în studiul lingvistic. Ca disciplină, lingvistica matematică urmărește să găsească un formalism matematic prin care să descrie limbajul natural și concepe teoreme, leme, corolare etc. prin care vrea să descrie fenomenele, schimbările limbii. Ea nu folosește tehnici computerizate.

Lingvistica matematică înseamnă, de fapt, studiul fenomenelor din limbă cu mijloace matematice [9]. Au fost examinate, spre exemplu, tipurile de opoziții lingvistice în corelație cu tipurile de mulțimi (finite sau infinite), problemă de interes în toate compartimentele limbii, dar cu precădere utilă în studiul lexicului. Modelele matematice ale limbii (asupra cărora vom reveni), sunt analitice și generative și reprezintă construcții matematice care reține unele aspecte relaționale ale fenomenelor lingvistice. Rolul lor este de a sistematiza unele noțiuni și relații deja cunoscute, precum și de a descoperi relații și moduri noi de organizare, care nu au putut fi puse în evidență prin alte mijloace.

Noua disciplină s-a constituit relativ recent (în jurul anului 1960), perioadă al cărei vis lingvistic a fost reprezentat în mare măsură de traducerea automată, fiind bine reprezentată de oamenii de știință ruși, dar și de alte naționalități. Ea apare ca o disciplină de intersecție, care este privită ca reprezentând atât lingvistică, cât și matematică.

În direcția fundamentării unei discipline autonome, cu baze teoretice, dar și cu o largă aplicabilitate practică, se remarcă și școala românească, prin aportul profesorului Solomon Marcus și al colaboratorilor săi.

Din punct de vedere științific, dezvoltarea lingvisticii matematice a fost favorizată, pe de o parte, de lingvistica structurală și de utilizarea metodei axiomatic-deductive iar, pe de altă parte, de importanța dobândită în matematică de teoria mulțimilor. Astfel, descrierea matematică a noțiunilor de bază ale lingvisticii s-a făcut cu precădere prin aplicarea teoriei mulțimilor. În unele cazuri se apelează la teoria algebrelor lui Boole și la teoria codurilor. Alte aplicații ale algebrei (în special teoria semigrupurilor libere) în lingvistica descriptivă sunt corelate cu studiul distribuției și al contextului. Un aspect important al lingvisticii matematice[1] este legat de statistică (teoria probabilităților în general, lanțuri Markov, legea lui Zipf) și a condus la conceperea unei noi metode de cercetare a vocabularului sau lexicului, numită statistică lexicală.

Lingvistica computațională aplică tehnici computerizate în cercetarea lingvistică implicând, în felul acesta, utilizarea de *algoritmi, structuri de date* și *modele formale ale reprezentării și raționamentului*, precum și *tehnici ale inteligenței artificiale* (în special metode de reprezentare și de căutare). Unele dintre problemele specifice pe care le tratează sunt: identificarea structurii propozițiilor, modelarea raționamentului și a cunoașterii, stabilirea modului în care poate fi folosit limbajul natural pentru realizarea acestor obiective, traducerea asistată de calculator, prelucrări statistice ale limbajului (spre exemplu, parsing statistic sau analiză sintactică de natură stocastică) ș.a.

Scopul lingvisticii computaționale este, așadar, acela de a dezvolta o teorie computațională a limbajului folosind elemente ale informaticii (algoritmi, structuri de date etc.). Se pot desprinde cel puțin două *motivații* pentru dezvoltarea modelelor computaționale, cea de natură științifică și cea de natură practică sau tehnologică.

Motivația științifică urmărește să obțină o înțelegere mai bună asupra modului cum funcționează limbajul. Este un fapt bine cunoscut acela că nici una dintre celelalte discipline tradiționale nu dispune de mijloacele necesare pentru a trata această problemă în totalitate. O teorie completă, obținută prin combinarea tuturor ipotezelor dezvoltate de către diverse discipline ar fi mult prea complexă pentru a putea fi studiată exclusiv cu metodele tradiționale. Se încearcă, de aceea, modelarea algoritmică a acestor teorii complexe, pentru ca ele să poată fi programate și apoi testate pe calculator. Se crede că, prin intermediul calculatorului, se va putea obține o mai completă și mai profundă înțelegere asupra modului cum acționează limbajul uman. Modelele computaționale pot oferi idei deosebit de utile referitoare la comportamentul lingvistic (și deci uman) atât în prezent, cât și în viitor, idei care pot fi

[1] Pentru unele detalii referitoare la aceasta, vezi [9], s.v. *matematică (lingvistică~)*, p. 285.

exploatate de către psiholingviști. În același timp, nu putem să nu remarcăm faptul că limbile naturale au fost adesea studiate în special cu scopul de a fi predate altora, fiind ignorate principiile generale care stau la baza *tuturor* limbilor naturale. Între lingviști nu există încă un consens cu privire la multe dintre faptele, noțiunile și conceptele lingvistice de bază. Toate acestea au făcut să fie resimțită și mai mult necesitatea existenței unor modele computaționale generale referitoare la limbajul natural. Pe de altă parte, motivația practică sau de natură tehnologică enunță ideea că, datorită capacităților sale, utilizarea limbajului natural va revoluționa modul de folosire a calculatoarelor. În acest context, reamintim faptul că prin *procesarea limbajului natural* se înțelege acea tehnologie care creează și implementează modalități de a efectua diferite sarcini referitoare la limbajul natural. Spre exemplu, această tehnologie poate fi utilizată pentru a construi interfețe (bazate pe limbajul natural) cu baze de date, pentru a realiza traducerea automată (în special în domenii tehnice restrânse) ș.a.

1.1. Lingvistică matematică și lingvistică computațională; câteva contribuții românești

Lingvistica matematică are o îndelungată tradiție în țara noastră. Astfel, încă din secolul trecut, în România, lingviști cum ar fi Alexandru Cihac și Bogdan Petriceicu Hasdeu au anticipat folosirea metodelor statistice în lingvistică. În lucrarea sa "Limba în circulațiune" Hasdeu arată, probabil printre primii din lume, că importanța unui cuvânt este legată de frecvența cu care acesta este utilizat. Mulți lingviști români vor folosi ulterior parametri probabiliști dintre cei mai simpli, cum ar fi frecvența absolută și cea relativă, în special în domeniul lexicologiei. (O lucrare de acest tip, care reprezintă "o tipologie lexicală cantitativă, întocmită cu ajutorul statisticii moderne și privind limbile română și italiană" este [54]). În legătură directă cu *investigațiile de natură statistică*, s-au efectuat și diferite investigații lingvistice prin intermediul *teoriei informației*. Spre exemplu, energia informațională (un parametru introdus de O. Onicescu corespunzând energiei cinetice în același mod în care entropia informațională corespunde entropiei termodinamice) a unor texte poetice românești a fost investigată de S. Marcus [63], [64].

În ceea ce privește *lingvistica algebrică*, aceasta studiază două tipuri fundamentale de modele: generativ și analitic [58], [59]. Așa cum se arată în [65], "simplificând, am putea spune că, în cadrul unui model generativ, punctul de început îl reprezintă o anumită gramatică, în timp ce obiectul studiat este limbajul generat de această gramatică. Un model analitic prezintă o situație inversă. Aici punctul de plecare este o anumită limbă i.e. o anumită colecție de propoziții, în timp ce scopul studiului este să stabilească structura acestor

propoziții, elementele lor constitutive și relațiile dintre ele în cadrul propozițiilor."

Referitor la *modelele analitice*, remarcăm teoria generală privitoare la categoriile gramaticale dezvoltată de S. Marcus, cu specială referire la categoriile morfologice (partea de vorbire, genul, cazul), la diferite tipuri de configurații sintactice, la dependența sintactică și la subordonarea sintactică [65].

În ceea ce privește *modelele generative*, o monografie referitoare la cel mai simplu model generativ al unei limbi naturale este publicată de S. Marcus încă din 1964 [57]. În unul dintre capitolele acestei lucrări, dedicat relevanței gramaticilor cu stări finite în studiul limbilor naturale, se arată că relațiile de coordonare pot fi descrise prin intermediul gramaticilor cu stări finite, în timp ce relațiile de subordonare necesită tipuri mai generale de gramatici.

Două noi tipuri de gramatici generative au fost introduse de către S. Marcus. În [60], [61] este definită o gramatică contextuală exclusiv prin intermediul unei mulțimi finite de șiruri și de contexte. O serie de generalizări ale acestor gramatici sunt definite de către G. Păun, care investighează și puterea lor generativă [81], [82], [83]. Ultimele rezultate referitoare la gramaticile contextuale sunt prezentate în capitolul 3 al lucrării de față. Un amplu studiu referitor la gramaticile contextuale și la relevanța lor lingvistică este [105]. Noile cercetări privitoare la proiectarea unui parser (analizor sintactic) bazat pe gramatici contextuale (§3.6) le plasează pe acestea și în centrul domeniului lingvisticii computaționale.

Așa cum se remarcă și în [65], numeroase studii au fost dedicate gramaticilor independente de context și celor dependente de context. Dacă Orman [65] este interesat în mod special de gramaticile independente de context liniare și de structura derivărilor conform unei gramatici independente de context, Păun [65] se ocupă cu precădere de diferite clase de gramatici și limbaje dependente de context, obținute prin adoptarea anumitor restricții în folosirea regulilor. G. Păun obține, de asemenea, numeroase rezultate privind anumite măsuri ale complexității sintactice și concepe cele mai simple gramatici corespunzătoare unor importante limbaje definite pe un vocabular alcătuit dintr-un unic element.

Matematicienii, informaticienii și lingviștii români au, de asemenea, numeroase contribuții importante în domeniul *poeticii matematice și computaționale*, extinzându-și aria de preocupări prin conceperea unor modele matematico-lingvistice pentru diverse domenii ale științei și artei. O lucrare de referință este [62], în a cărei primă parte S. Marcus construiește o serie de modele matematice ale limbajului poetic. În cea de-a doua parte a cărții, constând numai din capitolul VIII, este propusă o abordare matematică a teatrului. Întreaga lucrare este concepută sub semnul *structurilor algebrice ale limbajului*.

Domeniul lingvisticii computaţionale, în care se înscrie şi lucrarea de faţă, a fost inaugurat în România de către Gr. C. Moisil, care, în numeroase articole, printre care [72], [73], [74], a dezvoltat ceea ce el a numit "gramatica mecanizată a limbii române", considerând aceasta un studiu preliminar (şi absolut obligatoriu) traducerii automate. Moisil a oferit descrieri riguroase ale conjugării verbelor şi ale declinării substantivelor şi adjectivelor în limba română, inclusiv prin introducerea unor inovaţii metodologice (cum ar fi metoda literelor variabile, folosită pentru a trata analogul grafic a ceea ce se numeşte *alternanţă fonologică* sau *morfologică*). El a oferit şi o tratare logică a conjuncţiei *şi*, precum şi a microsintaxei verbului românesc [75]. Rezultatele cercetărilor sale, consacrate limbii scrise, s-au dovedit ca având o oarecare relevanţă şi pentru limba vorbită. Aplicarea directă a gramaticii mecanizate a lui Moisil la realizarea unui algoritm de traducere automată din engleză în română a fost realizată în 1965 de către Nistor-Domonkos [79], care a folosit în acest scop calculatorul MECIPT al Institutului Politehnic din Timişoara. Din păcate, acest prim experiment românesc privitor la traducerea automată nu a fost continuat şi dezvoltat ulterior.

O sinteză a realizărilor româneşti în domeniul lingvisticii computaţionale, până în anul 1978, se găseşte în [65].

În ultimii ani se remarcă rezultatele obţinute în domeniul ingineriei limbajului de către un colectiv de cercetători ai Academiei Române (www.racai.ro), coordonat de c.p.I dr. Dan Ioan Tufiş, m.c. al Academiei Române. D. Tufiş şi colectivul pe care îl coordonează (în cadrul Centrului pentru Cercetări Avansate în Învăţarea Automată, Prelucrarea Limbajului Natural şi Modelare Conceptuală al Academiei Române) au dezvoltat primul corpus românesc standardizat, precum şi resursele de limbaj asociate (cum ar fi un dicţionar românesc, morfologia flexionară românească, mulţimi de tag-uri pentru dezambiguizare statistică etc.). O sinteză referitoare la resursele lingvistice computaţionale existente pentru limba română se găseşte în [100]. Pentru mai multe informaţii poate fi consultat [52]. Aceste resurse lingvistice computaţionale româneşti sunt considerate de expertul Uniunii Europene Poul Andersen [5] ca reprezentând adevărate "materiale de construcţie" ale traducerii automate în România, existenţa lor făcând ca infrastructura necesară traducerii automate să existe în ţara noastră.

Acelaşi autor [5] menţionează proiectul DBR-MAT, realizat de un colectiv de la Facultatea de Matematică a Universităţii din Bucureşti (I. Văduva, F. Hristea, M. Popescu), ca fiind singurul proiect de cercetare românesc care se orientează în mod direct către traducerea automată (MT - de la "machine translation") şi traducerea asistată de calculator (MAT - de la "machine aided translation"). Proiectul DBR-MAT realizează (1996-1998) un sistem inteligent de traducere asistată de calculator (MAT) pentru limbi diferite din punct de vedere structural. Scopul declarat al proiectului este investigarea şi implementarea pilot a unui sistem de tip MAT care să combine o abordare

bazată pe procesarea cunoștințelor cu metode statistice în prelucrarea limbajului natural. Proiectul a fost finanțat de către Fundația Volkswagen și este realizat de Universitatea din Hamburg, Academia de Științe din Bulgaria și Universitatea din București (prin Facultatea de Matematică, catedra de informatică). Coordonatorul proiectului este profesorul Walther von Hahn de la Universitatea din Hamburg (vhahn@nats.informatik.uni-hamburg.de), iar coordonator al părții române este dr. Florentina Hristea (fhristea@mailbox.ro). Detalii asupra unora dintre rezultatele românești ale acestui proiect de cercetare pot fi văzute în capilolul 3 al lucrării de față (§3.5).

1.2. Procesarea limbajului natural

Procesarea limbajului natural reprezintă o *tehnologie* (adică un ansamblu de procese, metode, operații) care creează și implementează modalități de a executa diferite sarcini referitoare la limbajul natural (cum ar fi construcția unor interfețe - bazate pe limbaj natural - cu baze de date, traducerea automată ș.a.). Procesarea limbajului natural reprezintă și astăzi o problemă dificilă și în cea mai mare parte nerezolvată. Găsirea unei tehnologii adecvate este extrem de grea datorită naturii multidisciplinare a problemei, fiind implicate următoarele științe și domenii: lingvistică, psiholingvistică, lingvistică computațională, filozofie, informatică, în general, și inteligență artificială, în mod special, etc. *Ingineria limbajului natural* se ocupă de implementarea unor sisteme de mare anvergură.

Aplicațiile procesării limbajului natural se înscriu în trei mari categorii:

- aplicațiile bazate pe text, dintre care amintim:
  - clasificarea documentelor (și respectiv găsirea documentelor legate de anumite subiecte);
  - regăsirea informației (căutarea unor cuvinte-cheie sau concepte);
  - extragerea informației (legate de un anumit subiect, deci de un anumit cuvânt-cheie);
  - înțelegerea textelor (care presupune o analiză profundă a structurii acestora);
  - traducerea automată și traducerea asistată de calculator dintr-o limbă în alta;
  - alcătuirea de sinteze;
  - achiziția de cunoștințe.
- aplicațiile bazate pe dialog, care implică comunicarea între om și mașină, aplicații cum ar fi sistemele de învățare, sistemele de interogare și răspuns la întrebări, rezolvarea problemelor, controlul (bazat pe limba vorbită) al unui calculator ș.a.

- procesarea vorbirii
  (Este important să facem distincția între problemele de *recunoaștere a vorbirii* și cele de *înțelegere* a limbajului. Astfel, trebuie să remarcăm încă de la început faptul că un sistem de recunoaștere a vorbirii nu folosește nici un element de înțelegere a limbajului. Recunoașterea vorbirii se ocupă numai de identificarea cuvintelor vorbite provenind de la un semnal dat, nu și de înțelegerea mesajului, adică a modului în care aceste cuvinte sunt folosite în procesul de comunicare. Pentru a deveni un sistem de înțelegere a limbajului, un dispozitiv de recunoaștere a vorbirii trebuie să furnizeze intrarea sa unui sistem de înțelegere a limbajului natural, operație care produce un așa-numit "sistem de înțelegere a limbajului vorbit". Caracteristica de bază a oricărui sistem de înțelegere este aceea că el realizează o reprezentare a înțelesului propozițiilor într-un limbaj de reprezentare, care poate fi utilizat în vederea unei procesări ulterioare).

1.2.1. Niveluri ale procesării limbajului natural

Structura oricărei limbi naturale (umane) se împarte în cinci niveluri diferite: *fonologie, morfologie, sintaxă, semantică* și *pragmatică*. Aceste niveluri coincid cu formele de cunoștințe relevante și, în același timp, necesare pentru înțelegerea limbajului natural. Dintre toate acestea, lucrarea de față se concentrează asupra nivelului sintactic și a celui semantic, precum și a legăturii existente între acestea. Înainte de a trece la studiul sintaxei și al semanticii computaționale, ne propunem o scurtă incursiune la toate nivelurile analizei limbajului.

Fonologia se ocupă cu studiul *fonemelor* (ca entități abstracte, care se realizează printr-o infinitate de sunete). Cunoștințele fonetice și fonologice sunt cruciale pentru sistemele bazate pe vorbire (în procesarea vorbirii).

Morfologia, în gramatica tradițională, este o știință a cuvântului urmărit sub aspect gramatical, adică sub aspectul variației formei sale (al flexiunii) pentru exprimarea diverselor categorii gramaticale, prin opoziție cu sintaxa, care studiază combinarea cuvintelor și funcțiile pe care acestea le iau în cadrul combinațiilor. Așa cum se remarcă în [9], odată cu apariția structuralismului, interesul cercetătorilor s-a deplasat de la *cuvânt* la *morfem*, ca unitate minimală de expresie purtătoare de semnificație lexicală sau gramaticală[2]. Efectul acestei

[2] **Morfemul** este o unitate fonică și semnificativă indivizibilă în alte unități mai mici, dotate cu semnificație. El reprezintă unitatea structurală de bază a limbajului, care are asociat un înțeles sau sens.

deplasări îl constituie estomparea graniței prea rigide dintre morfologie și sintaxă, căci ambele discipline studiază combinații de morfeme și comportă tipuri asemănătoare de relații, având ca efect apariția *morfosintaxei*. În măsura în care unii structuraliști (L. Bloomfield, J. Vendryes) păstrează autonomia celor două discipline, morfologia studiază regulile care guvernează structura internă a cuvintelor atât în cadrul *flexiunii*, cât și al *formării cuvintelor*, incluzând o *morfologie flexionară* și una *derivațională* [9].

Granița dintre flexiune (care furnizează diferitele forme ale unui cuvânt) și derivare (care produce cuvinte noi pornind de la cele existente) este uneori neclară. O diferență esențială o constituie aceea că numai derivarea poate introduce o schimbare de sens (prin introducerea de cuvinte noi). O altă deosebire constă în faptul că formele derivative ar putea să nu existe, în timp ce formele flexionare nu lipsesc aproape niciodată.

În limba engleză, pentru care s-au dezvoltat aproape toate teoriile computaționale, flexiunea este mult mai simplă decât derivarea (prin contrast, de pildă, cu limba latină, dar și cu unele limbi ca rusa, japoneza sau finlandeza). Simplitatea flexiunii în limba engleză face ca majoritatea cercetătorilor din domeniul lingvisticii computaționale să neglijeze morfologia.

Conceptul de *morfem* este foarte controversat și cunoaște mai multe accepții. Evoluția noțiunii de *morfem* a fost, pe larg, discutată de Paula Diaconescu în [21]. Dintre numeroasele accepții sub care circulă acest concept, am ales-o pe aceea care este frecvent întâlnită în lingvistica clasică și care ni se pare, în același timp, foarte apropiată tehnicilor computaționale. Conform acestei definiții a morfemului există o *semnificație lexicală* și o *semnificație gramaticală*, pe care le vom ilustra printr-un exemplu. Cuvântul *gânduri* este format din rădăcina cuvântului, care este *gând* și desinența de plural *uri*. Rădăcina cuvântului este purtătoare de sens lexical (*gând* este un *morfem lexical*), iar desinența de plural este purtătoare a unui sens gramatical (*-uri* este un *morfem gramatical*). Acest din urmă sens (gramatical) arată că este vorba despre un substantiv *neutru*, la *plural*, nearticulat.

Altfel spus, morfemul este un termen generic pentru *rădăcină* și *afixe* (prefixe, sufixe și desinențe).

Noțiunea de morfem este adesea alăturată celei de fonem. **Fonemul** este o unitate lingvistică minimală având capacitatea de a distinge semnificații (lexicale sau gramaticale), dar lipsită prin ea însăși de semnificație. Spre exemplu, fiecare literă a cuvântului *gând* reprezintă un fonem. Litera *m* din cuvântul *masă* este un fonem, în timp ce litera *m* inițială din *m-am dus* reprezintă un morfem (*mă*).

O altă distincție care trebuie făcută este aceea dintre *desinență* și *terminație*. Astfel, în timp ce desinența are rol gramatical (indică, spre exemplu, cazul, genul, numărul, persoana), terminația reprezintă ultimul sunet, fără a avea rol gramatical. Uneori desinența și terminația se confundă, alteori nu. Spre exemplu, *ă* final din *masă* reprezintă, în egală măsură, o terminație și o desinență (pentru că exprimă numărul singular și cazul nominativ-acuzativ al substantivului, precum și faptul că acesta este nearticulat). În cazul substantivului *gânduri*, *-uri* reprezintă o desinență, iar *i* final este terminația cuvântului.

Pentru procesarea eficientă a altor limbi naturale, inclusiv a limbii române, este însă necesară dezvoltarea unor modele computaționale adecvate ale morfologiei fiecărei limbi. În cazul limbii române, un model al morfologiei ei flexionare este creat și implementat la Centrul de Studii Avansate al Academiei Române (www.racai.ro) sub coordonarea lui D. Tufiș.

Indiferent dacă avem de-a face cu morfologia flexionară sau cu cea derivațională, putem spune că, din punct de vedere computațional, nivelul morfologic al limbii se ocupă de modul în care sunt alcătuite cuvintele pornindu-se de la unitățile de bază numite *morfeme*.

Sintaxa reprezintă, într-o concepție asupra organizării stratificate, pe niveluri, a limbii, acel nivel [9] a cărui organizare se desfășoară între cuvânt, ca unitate minimală și combinațiile acestuia: propoziții/fraze, ca unități maximale. Cunoștințele sintactice se referă la modul în care pot fi alăturate cuvintele pentru a forma propoziții corecte și determină care este rolul structural al fiecărui cuvânt în cadrul propoziției, precum și ce grupuri sintactice intră în componența altori grupuri sintactice (vezi §1.3). Prin urmare, nivelul sintactic determină rolul structural al fiecărui cuvânt în interiorul unei propoziții, precum și relațiile dintre propoziții în cadrul frazei.

Sintaxa (construcția propozițiilor) reprezintă nivelul cel mai de jos la care limbajul natural (uman) este, în mod constant, creator. Vorbitorii unei limbi creează mult mai rar unități fonice și lexicale. În schimb, sunt concepute în mod constant noi propoziții și fraze. Acest tip de creativitate deosebește sintaxa atât de fonologie, cât și de morfologie. (În timp ce putem alcătui o listă cu fonemele unei limbi, cu unitățile ei lexicale și cu regulile morfologice, nu există nici o modalitate de a întocmi o listă similară a structurilor de propoziție admisibile într-o limbă dată. Atâta timp cât nu este impusă o limită asupra lungimii propozițiilor, se poate demonstra că numărul structurilor de propoziție admisibile este infinit.)

Noam Chomsky (1957) este primul care evidențiază această idee. El introduce *gramatica generativă*, care descrie propozițiile furnizând reguli de construcție a lor (vezi capitolul 3 al lucrării de față). Astfel de reguli vor deveni standard nu numai în lingvistică, ci și în informatică, cu precădere în proiectarea compilatoarelor. Ceea ce este interesant și, în egală măsură, crucial, în cadrul acestei teorii, este faptul că o mulțime finită de reguli poate descrie un număr infinit de propoziții.

Procesul de recunoaștere a structurii unei propoziții de către un calculator se numește ***parsing***. Aceasta este analiza sintactică computațională, la care ne vom referi în capitolul 4 al lucrării de față. În esență, vom spune că, pentru a analiza sintactic o propoziție, un calculator trebuie să o împerecheze pe aceasta cu regulile care o generează. Acest proces de împerechere poate fi realizat în manieră *top-down* (de sus în jos) sau *bottom-up* (de jos în sus). capitolul 4 al lucrării se va referi în detaliu la ambele tipuri de tehnici, precum și la unele combinații ale lor care și-au dovedit eficiența.

Aşa cum se remarcă în [19], analiza sintactică de acest tip a propoziţiilor englezeşti a fost studiată pe larg şi este practic considerată o problemă încheiată. Astăzi se poartă discuţii exclusiv asupra modului *cel mai eficient* în care se poate realiza procesul de parsing în cazul limbii engleze şi nu referitor la faptul că acest lucru este sau nu posibil. Tocmai de aceea, capitolul 4 al lucrării realizează o prezentare amănunţită a principalelor tehnici de parsing existente. Analiza sintactică computaţională referitoare la alte limbi nu a fost investigată la fel de riguros şi ea rămâne, în multe cazuri, o problemă deschisă. Cu atât mai mult cu cât majoritatea tehnicilor de parsing existente se bazează pe ordinea fixată a cuvintelor şi nu lucrează la fel de eficient pentru limbile în care ordinea cuvintelor este extrem de variabilă, cum ar fi latina, rusa sau finlandeza. capitolul 3 al lucrării menţionează, de pildă, o tehnică de parsing pentru limba română bazată pe gramatici care nu sunt de tip generativ (vezi §3.5).

Semantica este [9] o ramură a lingvisticii, dar şi a altor ştiinţe (filozofie, logică, psihologie) al cărei obiect de studiu este *sensul*[3], unitate greu de abordat

---

[3] Şi în legătură cu acest concept există numeroase probleme relativ controversate în lingvistică. Astfel de probleme au făcut necesară apariţia lingvisticii computaţionale, pe de o parte, iar, pe de altă parte, ele reprezintă şi câteva dintre dificultăţile pe care le întâmpină această disciplină relativ nouă. Unul dintre foarte controversatele concepte îl constituie acela de *sens*, ceea ce face ca semantica să se numere printre cele mai interesante dintre ramurile lingvisticii.

Leonard Bloomfield remarca, în cartea sa de referinţă "Language" din 1933, că "sensul nu poate fi definit în termenii ştiinţei noastre". (Este vorba, desigur, despre *lingvistică* sau *ştiinţa limbii*). De altfel, încă din anul 1923, C.K. Ogden şi I.A. Richards colecţionaseră, în cartea lor "The Meaning of Meaning", 23 de definiţii ale cuvântului *meaning* (sens). Iată cum, încă de la acea dată, se făcea simţită nevoia apariţiei unei noi discipline, mai riguros ştiinţifică şi cu o putere mai mare de formalizare, aşa cum este lingvistica computaţională. Devine evident faptul că, pentru a reprezenta sensul, este nevoie de un limbaj mai precis decât cel natural. Instrumentele pentru a crea un astfel de limbaj vin din matematică, în general, din logică şi din limbajele formale, în mod special. *Limbajele de reprezentare* create (vezi capitolul 6) trebuie să aibă anumite proprietăţi, şi anume: să fie precise şi neambigue, să păstreze structura intuitivă a propoziţiilor. Astfel, reprezentarea obţinută trebuie să păstreze structura intuitivă a propoziţiilor limbajului natural pe care îl modelează, aceasta însemnând, de pildă, că propoziţiile cu structuri similare trebuie să aibă *reprezentări structurale* asemănătoare.

În timp ce un vorbitor nu ia în consideraţie fiecare dintre sensurile unui cuvânt pentru a înţelege sensul unei propoziţii sau fraze, un program de calculator trebuie să ia în consideraţie, în mod explicit, pe fiecare dintre acestea.

Un cuvânt se poate defini ca fiind *ambiguu* din punct de vedere semantic dacă el poate fi asociat mai multor sensuri.

Sens rămâne şi astăzi un concept ambiguu, greu de definit, obiect de studiu al semanticii, dar şi al altor discipline lingvistice şi nelingvistice. Una dintre definiţiile cele mai simple şi relativ uzuale ale cuvântului *sens* este următoarea: însuşirea reflectată în mintea noastră a unui obiect (lucru, fiinţă, acţiune, proces etc.).

dintr-o perspectivă unică şi unitară. Ca disciplină lingvistică, semantica este ultima creată (în sec. al XIX-lea). În funcţie de diversele aspecte ale sensului luate în consideraţie, se delimitează *semantica lingvistică* şi *semantica aparţinând altor ştiinţe*, chiar dacă interferenţele dintre diferitele tipuri de semantică sunt curente. Aceste aspecte vor fi analizate în capitolul 6 al lucrării.

Reţinem, încă de la început, faptul că nivelul semantic se referă la sensul cuvintelor şi al propoziţiilor, precum şi la modul de combinare a semnificaţiilor cuvintelor pentru a forma semnificaţia unei întregi propoziţii. La acest nivel, semantica computaţională face un *studiu al sensului independent de context*. Cu alte cuvinte, interesează sensul pe care o propoziţie îl are fără legătură cu contextul în care ea a fost utilizată.

Pragmatica se ocupă de *utilizarea limbii în context*. Astfel, conform [9], pragmatica este o disciplină al cărei obiect îl constituie limba, privită nu ca sistem de semne, ci ca acţiune şi interacţiune comunicativă. Pragmatica examinează, cu precădere, efectele diverselor componente ale contextului asupra producerii şi receptării enunţurilor, atât sub aspectul structurii, cât şi al semnificaţiei acestora. În procesarea limbajului natural nivelul pragmatic tratează folosirea propoziţiilor în diverse situaţii (contexte), precum şi modul în care contextul influenţează interpretarea unei propoziţii.

Pragmatica a apărut ca o reacţie atât faţă de lingvistica chomskyană, cât şi faţă de pozitivismul logic, iar domeniul ei de investigaţie a cunoscut ulterior o continuă extindere şi diversificare.

Între semantică şi pragmatică există o relaţie de complementaritate, ultima atribuind un rol esenţial contextului. Pragmatica are o problematică proprie[4], care include aspecte referitoare la: organizarea pragmatică a discursului (acte de vorbire, forme ale implicitului conversaţional - presupoziţii, implicaturi etc.), principiile şi strategiile comunicative, analiza conversaţiei. Un asemenea tip de analiză este îngreunat de diverşi factori, cum ar fi *relaţia de* ***referinţă***, extrem de frecventă în fragmentele de discurs aparţinând limbajului natural. Astfel, se ştie că, într-o limbă dată, un enunţ comportă o referinţă la o anumită stare de lucruri sau la o anumită persoană. Relaţia de referinţă, ca relaţie particulară dintre limbă şi lume, nu se poate realiza decât prin context. Contextul localizează în timp şi spaţiu, cuantifică, determină gradul de generalitate al unui cuvânt.

Referinţa poate interveni în cadrul aceleiaşi propoziţii (***Ion** a afirmat că **el** nu crede asta*) sau cu privire la propoziţii diferite, în virtutea diverselor *proceduri de anaforizare*[5] existente. În propoziţia

---

Remarcăm faptul că, în accepţia lucrării prezente, cuvintele *sens* şi *înţeles* sunt folosite ca sinonime.

[4] Pentru unele detalii privitoare la problematica domeniului pragmaticii, vezi [9], s.v. *pragmatică*, p. 373 şi capitolul 6 al lucrării de faţă, p. 243.

[5] Procedurile de anaforizare sunt legate de modul în care intervine *anafora*. Conform [9], *anafora* este un "fenomen sintactico-semantic constând în reluarea printr-un

Ion crede că a găsit pălăria lui.

pronumele *lui* ar putea să se refere la *Ion* (care intervine în aceeaşi propoziţie) sau la un alt posesor al pălăriei, o cu totul altă persoană, care a fost menţionată într-o propoziţie anterioară. Numai contextul poate ajuta în dezambiguizare.

Foarte delicată este elucidarea referinţelor şi a coreferinţelor realizate prin intermediul pronumelor. Tocmai de aceea, la nivelul discursului, care se ocupă de modul în care propoziţiile imediat premergătoare afectează interpretarea propoziţiei următoare, un aspect important se referă la interpretarea pronumelor şi la aspectul temporal al informaţiei vehiculate.

O altă formă de cunoaştere relevantă pentru înţelegerea şi prelucrarea limbajului natural o constituie *cunoaşterea universului*, prin care înţelegem cantitatea vastă de cunoştinţe necesare în vederea înţelegerii textelor. Acestea includ cunoştinţele generale despre structura lumii înconjurătoare la care face referire vorbitorul pentru ca, spre exemplu, să poată face faţă conversaţiei, precum şi informaţiile pe care trebuie să le aibă un vorbitor despre partenerii săi de discuţie.

Actualmente, în domeniul procesării limbajului natural, există trei mari *direcţii* reprezentate de sintaxa computaţională, semantica computaţională şi generarea limbajului (care studiază modul în care maşina generează text coerent).

1.3. Noţiunea de grup sintactic

După cum am arătat deja, cunoştinţele sintactice se referă la modul în care pot fi alăturate cuvintele pentru a forma propoziţii corecte şi determină care este rolul structural al fiecărui cuvânt în cadrul propoziţiei, precum şi ce grupuri sintactice intră în componenţa altor grupuri sintactice.

În sintaxa modernă, prin *grup* în general se înţelege [9] un component al structurii propoziţiei constituit în jurul unui centru (sau cap) de grup: verb, nume (substantiv), adjectiv, adverb, prepoziţie, grup a cărui coeziune sintactico-semantică este asigurată prin constrângeri de formă gramaticală (de caz, de prepoziţie, de topică, de acord), dar şi prin rolurile semantico-tematice impuse de centru determinanţilor.

---

substitut (sau anaforic) a unui termen plin referenţial, exprimat anterior, numit antecedent. Relaţia antecedent - substitut, numită <<relaţie anaforică>> sau <<interpretativă>>, procură referinţa substitutului, component care, în afara contextului, este lipsit de referinţă proprie". În lucrări mai noi "este desemnată prin anaforă şi clasa de cuvinte fără referinţă proprie, alcătuită, mai ales, din pronume şi adverbe pronominale, care primesc referinţa de la un component exprimat anterior, numit antecedent".

În cele ce urmează, vom opera cu noțiunea de **grup sintactic** în general și, în particular, cu noțiuni ca grup nominal, grup verbal, grup prepozițional ș.a. și vom folosi abrevierile englezești consacrate. Astfel, în funcție de *clasa morfologică a centrului*, se disting:

- *grupul verbal*, care va fi notat VP (după engl. *verb phrase*);
- *grupul nominal*, care va fi notat NP (după engl. *noun phrase*);
- *grupul adjectival*, care va fi notat ADJP (după engl. *adjectival phrase*);
- *grupul adverbial*, care va fi notat ADVP (după engl. *adverbial phrase*);
- *grupul prepozițional*, care va fi notat PP (după engl. *prepositional phrase*).

Spre exemplu, în limba română, în jurul verbului *a plăcea*, se poate constitui un grup verbal (VP) de forma

îmi place cartea

în structura căruia centrul atrage doi actanți (sau două argumente), iar coeziunea se asigură prin restricțiile de caz și de topică impuse acestor argumente (dativul antepus și nominativul postpus), dar și prin atribuirea funcțiilor de obiect indirect și de subiect.

În terminologia românească, termenul *grup* este preferat celui de frază (engl. *phrase*) pentru a evita confuzia cu accepția tradițională a termenului *frază*. În gramatica engleză termenul *phrase* desemnează o îmbinare de cuvinte, fără subiect sau predicat, care funcționează ca o singură parte de vorbire. Unii autori utilizează și termenul de *categorie sintactică* pentru a desemna aceste grupuri de cuvinte. Vom reveni asupra acestor noțiuni în capitolul 3 al lucrării, atunci când vom discuta despre gramatici și respectiv despre categorii (gramaticale și lexicale).

Precizăm că la baza conceptului de *grup sintactic*[6] se află *sintagma*, întrucât grupul sintactic reprezintă o unitate alcătuită, ca și sintagma, în jurul unui centru sau cap de grup. Singura deosebire este aceea că structura binară a sintagmei constituie numai una dintre realizările posibile ale grupului sintactic, acesta cunoscând și organizări mult mai ample decât cele de tip binar. (Lingviștii francezi utilizează în mod frecvent termenul de sintagmă și cu sensul de grup, lărgindu-i accepția și dincolo de structurile binare. În contextul actual ne vom raporta însă la accepția clasică, în care sintagma reprezintă o structură binară și doar una dintre posibilitățile de realizare ale grupului sintactic.)

[6] Vezi și [9], s.v. *grup*, p. 232.

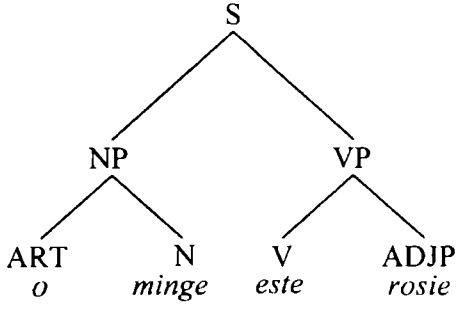
Grupurile sintactice sunt folosite în *reprezentarea structurii sintactice a unei propoziții.*

Structura sintactică a propoziției indică modul în care cuvintele din propoziție sunt legate unele de altele. Ea arată cum sunt grupate cuvintele, cum anumite cuvinte modifică pe altele și care dintre cuvinte sunt de importanță centrală în cadrul propoziției. În plus, sintaxa poate identifica tipurile de relații care există între grupurile sintactice și poate reține diverse informații despre structura propoziției, care vor fi necesare în prelucrarea ulterioară.

Majoritatea *reprezentărilor sintactice* ale limbajului sunt bazate pe noțiunea de *gramatici independente de context.* Acestea reprezintă structura propoziției în termeni de *grupuri sintactice*, care, la rândul lor, sunt părți componente ale altor grupuri sintactice. Informația de acest tip este adesea prezentată sub formă de *arbore.* Iată, spre exemplu, reprezentarea sintactică a propoziției

O minge este roșie.

în care, pe lângă abrevierile introduse deja corespunzător grupurilor sintactice, mai intervin următoarele notații: S pentru grupul sintactic care reprezintă întreaga propoziție (de la engl. *sentence*), ART pentru articol, N pentru substantiv (de la engl. *noun*) și V pentru verb:



1.4. Exemple de dificultăți ale procesării limbajului natural

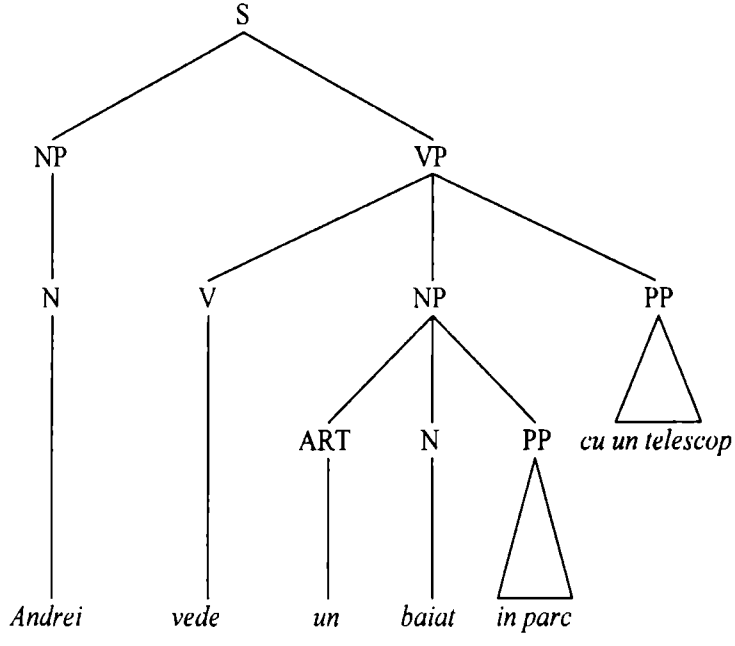
O dificultate fundamentală în procesarea limbajului natural o constituie **ambiguitatea**, adică posibilitatea de a da două sau mai multe interpretări unei

construcții sau unui component al ei. Există teorii diferite referitoare la ambiguitate. Aceste teorii detectează diverse tipuri de ambiguitate la toate nivelurile limbii. În cele ce urmează, vom adopta punctul de vedere conform căruia, la nivel sintactic și respectiv semantic, există trei tipuri majore de ambiguitate, după cum urmează:

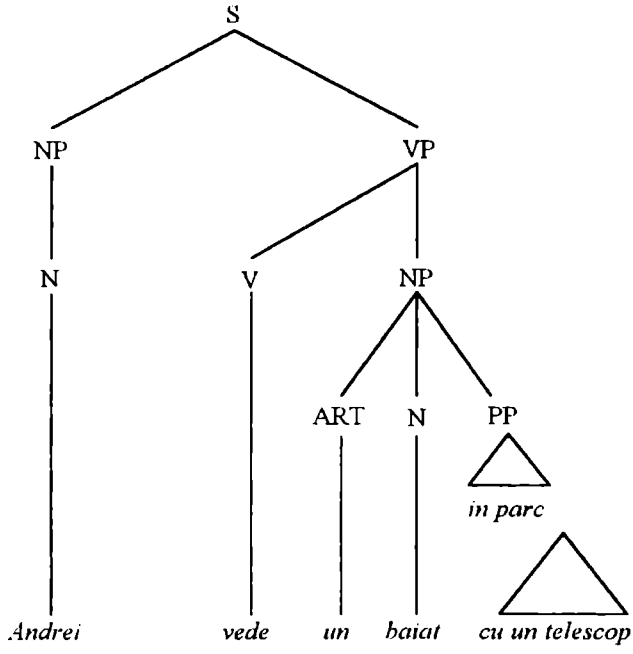
- **ambiguitatea structurală**, care intervine atunci când un parser (analizor sintactic) detectează mai mult de o structură posibilă corespunzător unei propoziții. Spre exemplu, propoziția

Andrei vede un băiat în parc cu un telescop.

poate avea următoarele două structuri, rezultate în urma analizei sintactice:



sau

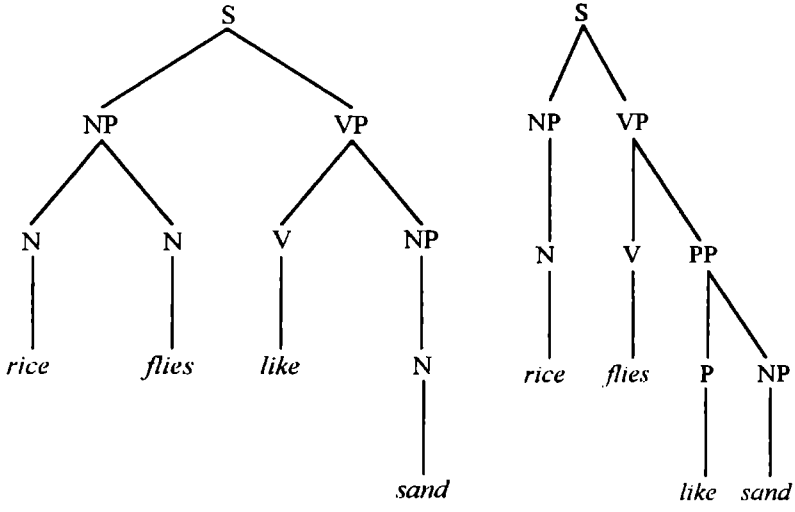


- **ambiguitatea lexicală**, ce intervine atunci când unui cuvânt îi corespund mai multe părți de vorbire, adică atunci când un cuvânt aparține la două sau mai multe categorii lexicale diferite. Majoritatea cuvintelor englezești, spre exemplu, pot aparține mai multor părți de vorbire, ceea ce determină existența unui mare număr de posibilități pentru generarea ambiguității. Un exemplu celebru [4] pentru limba engleză îl constituie propoziția

Rice flies like sand.

în care cuvântul *rice* se referă fie la orez (care zboară ca nisipul), fie la muștele de orez (insecte), adică la *rice flies* (cărora le place nisipul).

Observăm că ambiguitățile lexicale pot conduce la structuri diferite rezultate în urma procesului de parsing (analiză sintactică), deci la ambiguități structurale. În cazul propoziției anterioare, cele două reprezentări structurale posibile ale acesteia sunt:



Uneori este posibilă utilizarea unor *reguli gramaticale* (cum ar fi acordul predicatului cu subiectul) pentru a realiza dezambiguizarea. Un exemplu la fel de cunoscut [4], pentru limba engleză, este acela al propozițiilor

*Flying planes **are** dangerous.*
*Flying planes **is** dangerous.*

dintre care prima se referă la faptul că avioanele sunt periculoase, iar cea de-a doua la faptul că pilotarea lor este periculoasă.

Diverşi autori detectează şi *ambiguități lexicale de ordin semantic*, generate de existența în majoritatea limbilor a cuvintelor polisemantice. Ambiguitatea lexicală de ordin semantic poate fi dublată sau nu de ambiguitate generată de existența mai multor părți de vorbire corespunzătoare aceluiaşi cuvânt. Ea este denumită, de către numeroşi autori, ambiguitate semantică.

- **ambiguitatea semantică**, care intervine atunci când un cuvânt are mai multe sensuri (este polisemantic), ca în cazul verbelor *a omorî* şi *a face* din următoarele grupuri de propoziții:

Andrei a omorât cățelul.
Andrei a omorât proiectul.
Andrei a omorât sticla cu vin.
Andrei a omorât speranța.

Kilogramul de roşii face 10.000 lei.
Ion face arhitectura.
Andrei face probleme.

Următoarea propoziție are *o unică structură sintactică*, dar *structura* ei *semantică este ambiguă*:

Fiecare copil iubește o prăjitură.

Propoziția nu prezintă ambiguitate structurală, dar este supusă ambiguității semantice (întrucât este posibil să existe un unic sortiment de prăjituri iubit de toți copiii, dar este la fel de posibil ca fiecărui copil să-i placă un alt sortiment de prăjituri). Astfel de probleme vor fi reluate în capitolul 6 al lucrării de față.

În mod evident, fiecărei limbi naturale îi sunt specifice anumite tipuri de ambiguitate, dintre care unele pot fi mai frecvente decât altele. Problema dezambiguizării rămâne una deschisă. Tehnicile de dezambiguizare puse la punct până în prezent se referă cu precădere la limba engleză și au fost testate și aplicate în cazul acestei limbi. Se urmărește adaptarea lor în cazul altor limbi naturale, precum și conceperea unor tehnici de dezambiguizare specifice fiecărei limbi.

CAPITOLUL 2

TEHNICI CU STĂRI FINITE

Automatele cu stări finite sunt printre cele mai simple *dispozitive computaționale* care pot fi imaginate. În cele ce urmează, ne vom concentra asupra **rețelelor de tranziție cu stări finite (RTSF)** ca dispozitive computaționale ce pot fi implementate pentru a fi utilizate în procesarea limbajului natural. Vom trata implementarea acestora în limbajul Prolog și vom arăta că ele reprezintă cea mai simplă abordare a tehnicilor de procesare a limbajului natural, dar că, atunci când sunt luate în considerare separat, nu pot rezolva majoritatea problemelor care apar în timpul procesării și, prin urmare, nu se dovedesc a fi suficient de utile. În cadrul sistemelor complexe de procesare a limbajului natural, tehnicile cu stări finite trebuie utilizate în combinație cu alte procedee specifice, asupra cărora ne vom opri pe tot parcursul lucrării de față.

O RTSF poate fi privită fie ca o descriere neutră a unui limbaj (o mulțime de șiruri de simboluri), fie ca o specificare a unui **automat cu stări finite (ASF)**. Este vorba despre o specificare ori a unui ASF care să **recunoască** elemente ale unui limbaj, ori a unui ASF care să **genereze** elemente ale limbajului. O extensie simplă a reprezentării generale de tip RTSF pe care o vom lua, de asemenea, în considerare este aceea care ne permite să privim rețelele ca pe niște **translatori cu stări finite (TSF)**, prin care înțelegem acele ASF-uri care pot recunoaște elemente ale unui limbaj în timp ce generează elemente ale altuia.

Cel mai simplu ASF este cel de tip **determinist**, definit în mod formal de A. Salomaa [92] după cum urmează:

Definiția 2.1

Un sistem de rescriere (V,F) se numește un *automat finit determinist* dacă și numai dacă sunt satisfăcute următoarele condiții:

(i) V este compus din alfabetele disjuncte S și V_T la care referirea se face prin *stare* și respectiv *alfabet terminal*;

(ii) sunt specificate un element $s_0 \in S$ și o submulțime $S_1 \subseteq S$ reprezentând așa-numitele *stare inițială* și respectiv *mulțimea stărilor finale*;

(iii) producțiile din F sunt de forma

$$s_i a_k \rightarrow s_j \ , \quad s_i, s_j \in S \ ; \quad a_k \in V_T \ . \tag{2.1}$$

Mai mult, pentru fiecare pereche (s_i, a_k) cu $s_i \in S$ și $a_k \in V_T$, există exact o producție de tip (2.1) în F.

La rândul său, un *ASF nedeterminist* este definit de către același A. Salomaa [92] ca fiind unul determinist, cu următoarele două modificări: în (ii), s_0 este înlocuit cu o submulțime $S_0 \subseteq S$, reprezentând mulțimea stărilor inițiale. În (iii) se omite a doua propoziție.

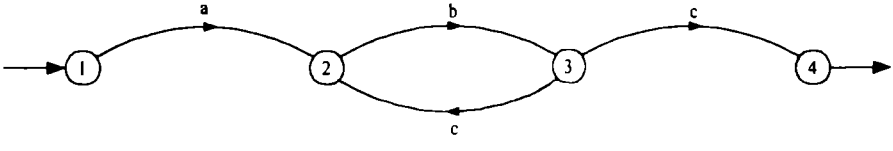
Ca urmare a celei de-a doua modificări este posibil ca, pentru o pereche (s_i,a_k), să nu existe nici o valoare a lui j ori să existe mai multe valori ale acestuia astfel încât producția (2.1) să aparțină lui F. Acest lucru poate fi interpretat în felul următor: atunci când se află în starea s_i și scanează a_k, automatul dispune de mai multe alegeri posibile pentru a trece în starea următoare. În această interpretare, un cuvânt P poate genera mai multe șiruri de stări de tranziție pornind de la o stare fixată. Cuvântul P este acceptat dacă și numai dacă el generează cel puțin un șir de stări de tranziție conducând de la o stare inițială la o stare finală. Astfel, dacă există un șir de stări de tranziție care are succes relativ la P, toate eșecurile cauzate de P vor fi ignorate.

2.1. Rețele de tranziție cu stări finite

Pentru a implementa o rețea de tranziție cu stări finite (RTSF) în limbajul Prolog este nevoie să realizăm o descriere a rețelei, precum și a modului în care se efectuează parcurgerea ei.

O descriere a unui RTSF constă din trei componente: numele rețelei, o mulțime de declarări și o mulțime de descrieri ale arcelor.

Un exemplu de rețea este dat de graful din figura următoare:



Acest ASF are patru stări, 1,2,3 și 4, starea 1 fiind cea inițială (semnalată prin simbolul "→"), iar starea 4 fiind cea finală (semnalată prin simbolul "→"). Cele patru vârfuri ale grafului reprezentând stările sunt conectate prin patru arce etichetate. În cazul automatelor de acest tip, arcele neetichetate vor fi considerate *arce de salt*. Ele constituie o importantă sursă de nedeterminare. În particular, automatul din figura anterioară este unul nedeterminist deoarece, atunci când se găsește în starea 3 și citește caracterul c, se poate fie întoarce în starea 2, fie poate avansa în starea 4.

Numele rețelei nu joacă un rol anume și nici o altă componentă nu face referire la el. El este introdus din motive de consistență și întrucât joacă un rol important în definirea rețelelor de tranziție recursive, care însă nu constituie

obiectul acestui curs. Cea de-a doua componentă constă dintr-o unică declarare obligatorie a unor stări inițiale și o unică declarare obligatorie a unor stări finale. Cea de-a treia componentă majoră a reprezentării constă dintr-o mulțime alcătuită din una sau mai multe descrieri ale arcelor, fiecare dintre acestea având aceeași formă. Câteva **reguli practice** pentru specificarea ASF-urilor prin RTSF-uri ar fi următoarele:

- Dacă se dorește trecerea din starea *i* în starea *j* fără a se parcurge un simbol din șirul de intrare sau a se adăuga un simbol la șirul de ieșire, atunci se conectează *i* și *j* cu un arc de salt.

- Dacă se dorește trecerea din starea *i* în starea *j* avându-se opțiunea de a parcurge sau scrie un simbol *s*, atunci se conectează *i* și *j* cu două arce, unul fiind etichetat cu simbolul *s*, iar celălalt reprezentând un arc de salt.

- Dacă ne aflăm în starea *i* și dorim să ne deplasăm peste ori să scriem zero sau mai multe apariții ale unui simbol *s*, atunci conectăm vârful *i* cu el însuși, printr-un arc etichetat cu *s*.

- Dacă ne aflăm în starea *i* și dorim să ne deplasăm peste ori să scriem una sau mai multe apariții ale unui simbol *s*, atunci conectăm vârful *i* cu un nou vârf *j* printr-un arc etichetat cu *s* și apoi conectăm *j* înapoi cu *i* prin intermediul unui arc de salt.

- Dacă ne aflăm în starea *i* și dorim să ne deplasăm peste ori să scriem una sau mai multe apariții ale unui șir S, care poate fi atins între stările *i* și *j*, atunci conectăm vârful *j* înapoi cu vârful *i* prin intermediul unui arc de salt.

- Dacă prezența chiar și a unei apariții a șirului este opțională în ultimul caz menționat, atunci trebuie ca *i* să fie conectat în plus la *j* prin intermediul unui arc de salt, care să țintească în direcția opusă celei deja introduse.

Acestea sunt simple reguli euristice, care s-ar putea să nu aibă efectul scontat dacă interacționează cu alte caracteristici ale rețelei și, mai ales, dacă unele dintre stările menționate reprezintă, în egală măsură, stări finale.

2.1.1. Reprezentarea RTSF-urilor în Prolog

În ceea ce privește **reprezentarea în Prolog a RTSF-urilor**, abordarea noastră va fi aceeași cu cea a autorilor G. Gazdar și C. Mellish [29], adică va fi una care reprezintă rețelele în mod **declarativ** ca pe niște structuri de date (fapte din baza de date Prolog), care pot fi examinate și manipulate. Având o astfel de descriere se vor putea scrie programe generale de recunoaștere și de generare. Reprezentarea RTSF-urilor ca structuri de date Prolog se va face specificând :

- nodurile (sau vârfurile) inițiale;
- nodurile finale;

- arcele, unde fiecare arc este definit prin: nodul de plecare, nodul destinație și eticheta arcului.

Aceste informații pot fi comunicate în Prolog utilizând predicate de forma:

initial(NOD).
final(NOD).
arc(NOD_PLECARE, NOD_DESTINATIE, eticheta).

Iată, spre exemplu, începutul reprezentării unei RTSF pentru limba română:

```
initial(1).
final(8).
arc(1,2,gn).
arc(2,3,conj).
arc(3,4,gn).
```

Evident, această descriere a rețelei trebuie completată prin specificarea semnificației abrevierii pe care o reprezintă cel de-al treilea argument al predicatului **arc**. Semnificația acestor abrevieri va fi indicată prin utilizarea predicatului

cuvant (Categorie, Cuvant).

Argumentul **Categorie** corespunde simbolului de abreviere, în timp ce argumentul **Cuvant** corespunde unuia dintre obiectele la care se referă abrevierea. În cazul exemplului considerat, din baza de date Prolog vor face parte următoarele fapte ce corespund declarărilor anterioare:

```
cuvant (gn, ioana).
cuvant (gn, maria).
cuvant (conj, si).
```

2.1.2. Parcurgerea RTSF-urilor în Prolog

Atunci când rețeaua este utilizată pentru **recunoaștere**, fiecare moment al parcurgerii ei este caracterizat de două elemente:

- R1. numele unui nod (locația curentă);
- R2. șirul de intrare rămas.

Atunci când rețeaua este utilizată pentru **generare**, aceste elemente vor fi înlocuite cu următoarele:

- G1. numele unui nod (locația curentă);

- G2. şirul de ieşire generat până la acel moment.

Se observă că, în modul recunoaştere, atât R1, cât şi R2 influenţează comportarea ulterioară a automatului, în timp ce G2 nu este relevant pentru aceasta în modul generare. În cele ce urmează, vom numi un cuplu de tip (R1, R2) şi/sau (G1, G2) **stare** şi vom avea în vedere faptul că, în timpul parcurgerii unei reţele, trebuie să ţinem evidenţa atât a **stării curente**, cât şi a **stărilor alternative** (care indică ce alte posibilităţi pot fi încercate în afara celei curente). Există două mari ***probleme*** care se ridică: *modul de organizare al căutării* pe care o presupune parcurgerea şi *modul de determinare a următoarelor stări valide* care decurg din starea curentă.

Fiecare stare curentă poate produce un număr de "stări următoare", care la rândul lor pot genera alte astfel de stări, dând naştere unui arbore al stărilor numit arbore de căutare. Căutarea în interiorul acestui arbore poate fi, în principiu, una de tip depth-first sau una de tip breadth-first. Implementarea Prolog pe care o vom prezenta în cazul acestor reţele realizează o căutare de tip **depth-first**, a cărei caracteristică de bază constă în faptul că o posibilitate dată este urmată atât timp cât este posibil, înainte ca o alternativă să fie luată în considerare. Facilitatea oferită de Prolog, aceea de a efectua procesul de backtracking, înlătură necesitatea de a reprezenta în mod explicit mulţimea stărilor alternative existente. Mulţimea alternativelor este reprezentată în mod implicit prin memorarea de către Prolog a locurilor spre care se poate merge înapoi, adică a poziţiilor către care se efectuează backtracking-ul.

Cea de a doua problemă care se ridică este aceea a determinării următoarelor stări valide care decurg dintr-o stare dată de tip <NOD, INPUT>. Regula de calcul este următoarea: pentru fiecare arc (având eticheta E şi destinaţia D), care pleacă din vârful NOD, trebuie incluse în mulţime noi stări, în funcţie de **natura etichetei**. Mai exact, regula este:

- *dacă E este un simbol* care coincide cu primul simbol din input, se include starea <D, ... restul de input>;
- *dacă E este o abreviere*, iar primul simbol din input se află sub incidenţa acestei abrevieri, se include starea <D, ... restul de input>;
- *dacă E este simbolul '#'*, se include starea <D, input>. (Se efectuează un salt care nu afectează inputul).

Vom considera, mai întâi, situaţia în care reţeaua este utilizată pentru *recunoaştere* şi vom descrie structura programului Prolog în acest caz. Astfel, predicatul **recunoaste (Nod, Sir)** va fi adevărat dacă şirul dat poate fi recunoscut de către RTSF plecându-se din nodul dat.

Acest predicat va fi utilizat de către două clauze Prolog, dintre care prima îşi propune să trateze cazul în care *nodul specificat este un nod final.*

Recunoaşterea şirului vid atunci când locaţia curentă este reprezentată de un nod final se exprimă prin următoarea clauză:

```
recunoaste(Nod,[ ]) :- final(Nod).
```

Dacă *nodul curent nu este unul final*, clauza Prolog corespunzătoare trebuie să prevadă şi posibilitatea continuării traversării. Următoarea clauză descrie parcurgerea unui arc pornindu-se din nodul curent, parcurgere urmată de o continuare a traversării din punctul în care s-a ajuns:

```
recunoaste(Nod_1, Sir) :-arc(Nod_1, Nod_2, Eticheta),
                         traverseaza(Eticheta,Sir,SirNou),
                         recunoaste(Nod_2,SirNou).
```

Predicatul **traverseaza** indică dacă un arc poate fi parcurs şi, în caz afirmativ, arată ce schimbări intervin în structura şirului de intrare. Argumentele acestui predicat sunt, în ordine, eticheta arcului, şirul iniţial şi respectiv noua valoare a şirului. Definiţia acestui predicat reflectă cele trei situaţii care pot să apară în parcurgerea reţelei:

- se traversează un arc etichetat cu următorul cuvânt din şir

```
traverseaza(Eticheta,[Eticheta|Cuvinte],Cuvinte):-
                         \+ special(Eticheta).
```

  (Această clauză exprimă faptul că un arc poate fi traversat dacă eticheta sa este identică cu primul cuvânt din input, cu condiţia ca eticheta să nu fie un simbol special, adică o abreviere sau simbolul #).

- traversarea implică continuarea parcurgerii unui şir al cărui prim cuvânt se înscrie în categoria indicată de eticheta arcului

```
traverseaza(Eticheta,[Cuvant|Cuvinte],Cuvinte) :-
                         cuvant(Eticheta,Cuvant).
```

- un şir poate fi recunoscut dintr-un nod dacă se poate efectua un salt din acel nod la un altul, caz în care şirul rămâne neschimbat

```
traverseaza ( '#', Sir, Sir ).
```

Pentru a completa definiţia este necesară definirea predicatului **special**, care detectează abrevierile şi simbolul de salt:

```
special('#').
special(Categorie) :- cuvant(Categorie, _ ).
```

În fine, atunci când se verifică recunoaşterea unui întreg şir de caractere, trebuie să ne asigurăm că se porneşte dintr-un nod iniţial. Această cerinţă este verificată de predicatul **test**, care apelează predicatul **recunoaşte**:

```
test(Cuvinte):- initial(Nod),recunoaste(Nod,Cuvinte).
```

În cazul în care se dorește adaptarea programului de recunoaștere descris anterior astfel încât el să *genereze* toate șirurile admise de rețea, este suficient să indicăm un scop de forma **test (X), write (X), nl, fail.**, fără a efectua vreo modificare în program. Un scop de această formă poate fi exprimat și prin introducerea în program a unui predicat **genereaza**, care apelează predicatul **test**:

```
genereaza :- test (X), write (X), nl, fail.
```

În cazul existenței în program a acestei clauze Prolog, scopul indicat de utilizator în execuție pentru realizarea generării va fi următorul:

```
? - genereaza.
```

Aceasta realizează o generare exhaustivă. Totuși, dacă rețeaua admite un număr infinit de șiruri de caractere diferite, execuția programului trebuie întreruptă de utilizator, după ce acesta și-a făcut o idee asupra tipurilor de șiruri care pot fi generate.

Un neajuns al acestei abordări este faptul că, datorită modului în care o apelare a predicatului **recunoaste** eșuează, și anume prin epuizarea tuturor posibilităților relative la această apelare înainte de a se analiza o alternativă, programul se va concentra întotdeauna asupra unei anumite alternative și a situațiilor care derivă din ea, înainte de a studia vreo altă alternativă posibilă. Acest comportament tipic căutărilor de tip depth-first nu deranjează atunci când mulțimea posibilităților de investigat este **finită**. În cazul în care această mulțime este infinită, rezultatul poate fi însă o foarte "îngustă" explorare a spațiului alternativelor ori, și mai grav, intrarea programului într-un ciclu infinit.

2.1.3. Realizări și limitări

Unul dintre avantajele pe care îl prezintă utilizarea unui asemenea dispozitiv computațional este dat de posibilitatea de a codifica cu ușurință secvențele de litere permise în diverse limbi. Spre exemplu, se știe că, **în limba engleză**, litera **q** trebuie să fie urmată de o literă **u**. Tot în limba engleză, în cadrul oricărui cuvânt care începe cu grupul de litere **st**, acesta nu poate fi urmat decât de o **vocală**, o literă **y** sau o literă **r**.

În limba română, grupul de litere **sp** poate fi urmat numai de o **vocală** (ex.: spate), de consoana **r** (ex.: sprijin) sau de consoana **l** (ex.: splai). Aceeași este situația în cazul grupului consonantic **st**, care poate fi urmat de o **vocală** (ex.: stea) sau de consoana **r** (ex.: strat). Grupul **st** nu poate fi urmat de consoana **l**. La rândul său, grupul **sc** poate fi urmat numai de **vocale** (ex.: scamă) ori de una dintre consoanele **r** (ex.: scriu) sau **l** (ex.: sclav).

Astfel de constrângeri sunt extrem de ușor de codificat cu ajutorul RTSF-urilor. Tot cu ajutorul acestora este, adesea, posibilă codificarea secvențelor de morfeme admise de către o limbă. RTSF-urile reprezintă cea mai simplă abordare a problemelor de procesare a limbajului natural, ale căror avantaje

constau, în principal, în posibilitatea de a fi uşor de implementat în Prolog şi de a codifica cu succes secvenţe admisibile de caractere şi, prin urmare, constrângeri de tipul celor comentate anterior.

Programul de recunoaştere descris în cadrul acestui capitol are rolul de a decide dacă un şir de caractere este sau nu acceptat de către un RTSF dat. Neajunsul major pe care îl prezintă programul este acela că, deşi în cadrul unei operaţii de recunoaştere încununate de succes el acumulează o serie de informaţii despre şirul investigat, toate aceste informaţii sunt pierdute ulterior. Singurul rezultat furnizat de program este o valoare de adevăr, raportată de Prolog sub forma unui răspuns de tipul `yes` sau `no`. Un ASF de tipul celui descris anterior are rol de recunoaştere şi nu prezintă nici una dintre caracteristicile unui translator sau ale unui analizor sintactic (parser). Singura decizie pe care o poate lua se referă la corectitudinea formării unui şir. Dacă sfârşitul unui şir de caractere poate fi atins într-o stare finală, atunci şirul este corect format. Dacă sfârşitul şirului nu poate fi atins, ori dacă nu este posibilă atingerea lui şi, în acelaşi timp, a unei stări finale, atunci şirul nu este corect format. Pentru a obţine mai multe informaţii asupra şirului reprezentând intrarea este necesară utilizarea unui parser ori a unui translator.

2.2. Translatori cu stări finite

Definiţia 2.2

Un **translator cu stări finite (TSF)** este acel tip de ASF care permite generarea unui şir de simboluri de ieşire concomitent cu recunoaşterea unui şir de intrare.

Conform acestei definiţii, putem privi un TSF ca fiind un dispozitiv computaţional ce verifică corespondenţa dintre două şiruri de caractere. Dacă, în aceeaşi măsură în care o RTSF poate fi privită ca având rol atât în recunoaşterea, cât şi în generarea şirurilor de caractere, o reţea reprezentând un TSF poate fi considerată fie un dispozitiv computaţional de verificare a corespondenţei şirurilor, fie unul care citeşte un şir de intrare în timp ce scrie un altul, de ieşire. Pentru a ne înscrie în această descriere este evident că putem specifica un TSF printr-o RTSF ale cărei arce sunt etichetate cu o pereche de simboluri. Spre exemplu, un arc etichetat **[i,y]** poate fi urmat numai dacă intrarea curentă este litera **i**, iar ieşirea o reprezintă litera **y**. Mai general, putem considera eticheta unui arc ca fiind o pereche de n-upluri de simboluri.

Un TSF poate fi utilizat pentru a reprezenta lexiconul sau pentru a transforma un cuvânt într-o secvenţă de morfeme. Întregul lexicon poate fi codificat sub forma unui TSF care codifică toate cuvintele corecte, pe care le transformă într-o secvenţă de morfeme. TSF-urile pentru diferitele sufixe se pot defini o singură dată şi toate rădăcinile de cuvinte care admit un anumit sufix

pot ținti către un același nod. Cuvintelor care au un prefix comun le pot, de asemenea, corespunde aceleași noduri, ceea ce reduce considerabil dimensiunea rețelei. Pe de altă parte, observăm că, o modalitate de reținere eficientă a cuvintelor cu aceeași rădăcină este construirea lor pe baza unor automate finite de acest tip. În exemplul care va urma va fi prezentată o altă aplicație a TSF-urilor, și anume utilizarea lor în realizarea traducerii automate.

Reprezentarea în Prolog a TSF-urilor, preluată, de asemenea, din [29], este similară cu cea descrisă în cazul RTSF-urilor. Diferența esențială constă în schimbarea descrierii arcelor, care acum trebuie să fie etichetate cu o pereche de caractere. Excepție fac situațiile în care eticheta se reduce la o abreviere (ce va fi definită în program), ori acelea în care se execută un salt atât în șirul de intrare, cât și în cel de ieșire, caz în care eticheta este dată de simbolul special amintit anterior. Un program de traversare a unui automat finit de acest tip va trebui să ia în considerare toate formele posibile ale unei etichete de arc.

Predicatul **trans** poate fi definit în mod similar cu predicatul **recunoaste**, la care ne-am referit anterior. Singura diferență notabilă este faptul că devine necesară o versiune modificată a predicatului **traverseaza**. Predicatul **trans** se definește după cum urmează:

```
trans(Nod,[ ],[ ]) :- final(Nod).
trans(Nod_1,Sir1,Sir2) :- arc(Nod_1,Nod_2,Eticheta),
        traverseaza(Eticheta,Sir1,SirNou1,Sir2,SirNou2),
        trans(Nod_2,SirNou1,SirNou2).
```

Predicatul **traverseaza** se definește în funcție de formele posibile ale etichetei, care în cazul TSF-urilor constă dintr-o pereche de caractere ori șiruri de caractere. Cazului în care eticheta este formată din cuplul [Cuvant1, Cuvant2] și nici unul dintre elementele acestui cuplu nu reprezintă o abreviere ori simbolul special de salt # îi corespunde în program următoarea clauză Prolog:

```
traverseaza([Cuvant1,Cuvant2],[Cuvant1|RestSir1],
    RestSir1,[Cuvant2|RestSir2],RestSir2):-
    \+ special(Cuvant1), \+ special(Cuvant2).
```

Cazului în care eticheta reprezintă o abreviere îi corespunde următoarea clauză Prolog (în care abrevierea este înlocuită de cuplul pe care îl reprezintă prin intermediul notației `EtichetaNoua`):

```
traverseaza(Abrev, Sir1,SirNou1,Sir2,SirNou2):-
    cuvant(Abrev,EtichetaNoua),
    traverseaza(EtichetaNoua,Sir1,SirNou1,Sir2,SirNou2).
```

În fine, trebuie luate în considerare cele trei situații de salt posibile, și anume acelea în care se execută saltul numai în șirul de intrare, numai în șirul de ieșire ori în ambele șiruri:

```
traverseaza(['#',Cuvant2],Sir1,Sir1,[Cuvant2|RestSir2],
            RestSir2).

traverseaza([Cuvant1,'#'],[Cuvant1|RestSir1],RestSir1,Sir2,
            Sir2).

traverseaza('#',Sir1,Sir1,Sir2,Sir2).
```

Evident, am presupus că Sir1 și respectiv Sir2 nu încep cu un caracter ce coincide cu simbolul de salt. Această presupunere este firească având în vedere faptul că tehnicile cu stări finite menționate se aplică în procesarea limbajului natural.

Ca și înainte, este nevoie de un predicat care să testeze faptul că se pleacă dintr-un nod inițial și care poate fi utilizat, în același timp, pentru tipărirea șirului de ieșire. Acesta este predicatul **test**, căruia îi corespunde, în program, următoarea clauză Prolog:

```
test(Sir_A):- initial(Nod),
              trans(Nod,Sir_A,Sir_B),
              write(Sir_B),
              nl.
```

Să mai observăm faptul că definiția predicatului **trans** este în întregime reversibilă, în sensul că putem privi TSF-ul ca realizând transferul de la Sir1 la Sir2 sau reciproc.

O primă concluzie care se impune este aceea că TSF-urile pot fi utilizate pentru a face corespondența între două șiruri de simboluri, precum și în procesarea morfologică. O altă utilizare posibilă a TSF-urilor o reprezintă traducerea automată, evident în acele domenii care utilizează un vocabular restrâns, cum ar fi domeniile tehnice și, mai ales, subdomenii ale lor. Un **exemplu** de utilizare a TSF-urilor în traducerea unui număr restrâns de propoziții din limba engleză în limba română este oferit de programul care urmează, scris în SICStus Prolog:

Programul 2.1

```
initial(1).
final(5).


arc(1,2,'UNDE').
arc(2,3,'FI').
```

```
arc(3,4,'FART').
arc(4,5,'FSUBST').
arc(3,6,'MART').
arc(6,5,'MSUBST').

cuvant('UNDE',[where, unde]).
cuvant('FI',[is,este]).
cuvant('FART',[a,o]).
cuvant('MART',[a,un]).
cuvant('FSUBST',[supermarket,alimentara]).
cuvant('FSUBST',[school,scoala]).
cuvant('FSUBST',[door,usa]).
cuvant('MSUBST',[teacher,profesor]).

trans(Nod,[],[]):-final(Nod).
trans(Nod_1,Sir1,Sir2):-arc(Nod_1,Nod_2,Eticheta),
      traverseaza(Eticheta,Sir1,SirNou1,Sir2,SirNou2),
      trans(Nod2,SirNou1,SirNou2).

traverseaza([Cuvant1,Cuvant2],[Cuvant1|RestSir1],RestSir1,
            [Cuvant2|RestSir2],RestSir2):-
            \+ special(Cuvant1),\+ special(Cuvant2).
traverseaza(Abrev,Sir1,SirNou1,Sir2,SirNou2):-
          cuvant(Abrev,EtichetaNoua),
          traverseaza(EtichetaNoua,Sir1,SirNou1,Sir2,SirNou2).
traverseaza(['#',Cuvant2],Sir1,Sir1,[Cuvant2|RestSir2],
            RestSir2).
traverseaza([Cuvant1,'#'],[Cuvant1|RestSir1],RestSir1,Sir2,
            Sir2).
traverseaza('#',Sir1,Sir1,Sir2,Sir2).

special('#').
special(Abrev):-cuvant(Abrev,_).
```

```
test(Sir_A):-initial(Nod),
             trans(Nod,Sir_A,Sir_B),
             write(Sir_B),
             nl.
```

Atunci când programul este testat specificându-se următorul scop

```
?-test([where,is,a,teacher]).
```

el răspunde:

```
[unde, este, un, profesor]

yes
```

2.3. Realizări și limitări ale automatelor cu stări finite

Una dintre utilizările cele mai frecvente ale automatelor cu stări finite constă în recunoașterea cu ajutorul acestora a *cuvintelor flexibile* în limbile având o morfologie flexionară. O astfel de limbă este, spre exemplu, finlandeza, în care verbele au aproximativ 12000 de forme. Totuși, majoritatea acestor forme rezultă prin aglutinare, adică prin simpla concatenare a morfemelor. S-a arătat că, în cazul acestei limbi, se poate defini o gramatică pentru cuvintele flexibile care, în esență, reprezintă o RTSF. G. Gazdar și C. Mellish comentează acest exemplu [29] remarcând faptul că, în lexiconul asociat, fiecărui morfem (inclusiv rădăcinii) îi corespund trei intrări, și anume: forma morfemului, proprietățile sale sintactico-semantice și un pointer către acele elemente (entități) care îi pot urma.

La rândul lor, TSF-urile prezintă avantajul de a fi bidirecționale, ceea ce le face atractive pentru un număr variat de aplicații, de la o posibilă verificare ortografică și până la traducerea automată în domenii restrânse.

Limitările automatelor cu stări finite în procesarea limbajului natural reprezintă, însă, o realitate care nu poate fi ignorată. Comentăm aici un singur exemplu constând în aceea că, deși RTSF-urile au capacitatea de a recunoaște limbaje non-finite (care conțin o mulțime infinită de șiruri de simboluri), există multe asemenea limbaje pe care dispozitivele computaționale de acest tip nu le pot recunoaște. Astfel, este relativ ușor de construit o RTSF care să recunoască limbajul a^n sau limbajul $a^m b^n$, dar nu se poate construi o asemenea rețea care să recunoască limbajul $a^n b^n$, în afara cazului în care se impune o limită superioară asupra mărimii lui n, caz în care limbajul devine unul finit. (Construcții de

forma $a^n b^n$ pot exista într-o limbă dacă aceasta permite introducerea unor şiruri de caractere în interiorul altora şi nu impune nici un fel de limitări asupra acestui proces. Un lexicon de forma celui amintit relativ la finlandeză nu ar putea fi utilizat în cazul unei limbi de acest tip). Limitarea valorii superioare a lui n şi deci transformarea limbajului $a^n b^n$ în unul finit nu constituie o soluţie acceptabilă întrucât, din ceea ce se cunoaşte până în prezent, nici un limbaj natural nu este unul finit.

CAPITOLUL 3

GRAMATICI

O **gramatică** reprezintă o specificare formală a regulilor ce definesc structurile "legale" (admise) ale unui limbaj.

Întrucât nici un limbaj natural nu este finit, se simte nevoia existenței unor sisteme formale (matematice) care să definească, în mod riguros, apartenența mulțimilor infinite de expresii lingvistice și care să atribuie o structură fiecărui element al acestor mulțimi. Astfel de sisteme formale sunt gramaticile. O *gramatică* poate fi, prin urmare, privită ca reprezentând *o definiție abstractă a unei mulțimi de obiecte structurate corect formate.*

Gramaticile vor fi gândite ca gestionând două procese total opuse: acela de **generare** și acela de **recunoaștere**. Astfel, o gramatică poate genera propoziții și fraze. Dar ea poate fi utilizată și pentru a recunoaște o propoziție dată. Un dispozitiv de recunoaștere decide dacă o propoziție dată aparține unui limbaj. Cu alte cuvinte, recunoaște dacă propoziția poate fi generată de gramatica corespunzătoare și, prin urmare, decide dacă propoziția este corectă în conformitate cu gramatica dată.

Gramaticile pe care le vom prezenta, în cele ce urmează, vor utiliza *recursivitatea* pentru a gestiona complexitatea structurală. Gramaticile discutate vor fi *declarative* și se vor baza, în majoritatea cazurilor, pe o descompunere a grupurilor sintactice în componente denumite *caracteristici.*

Din punctul de vedere al procesării limbajului natural, studiul gramaticilor este în general privit ca fiind o ramură a *reprezentării cunoștințelor.* Astfel, o gramatică poate fi pur și simplu privită ca o modalitate de a reprezenta anumite aspecte legate de ceea ce se cunoaște referitor la un limbaj, care este suficient de formal și de explicit pentru a fi înțeles de către o mașină.

Gramaticile utilizate în domeniul procesării limbajului natural trebuie să satisfacă atât criterii lingvistice, cât și computaționale. În esență, ele definesc mulțimi de șiruri de caractere și asociază acestora anumite *structuri.* Pentru reprezentarea structurii sintactice sunt, în general, utilizate diagramele care folosesc structuri arborescente.

Numeroase formalisme au fost alese, de-a lungul anilor, pentru a construi gramatici care să poată fi utilizate cu succes de către diverși analizori sintactici (parsers). Dintre numeroasele criterii generale care stau la baza proiectării unei gramatici, reamintim:

- naturalețea lingvistică;
- puterea matematică;

- eficiența computațională.

Toate tipurile de gramatici care au fost utilizate de lingvistica computațională au folosit, într-o formă sau alta, următoarele elemente:

- o reprezentare pentru grupurile sintactice (respectiv pentru părțile de vorbire care alcătuiesc aceste grupuri);
- un tip de dată pentru cuvinte (și, prin urmare, un lexicon asociat, un dicționar asociat sau o listă de cuvinte asociată);
- un tip de dată pentru reguli sintactice;
- un tip de dată pentru structuri sintactice.

Prin urmare, un formalism complet de specificare a unei gramatici trebuie să furnizeze cel puțin:

- un limbaj pentru specificarea grupurilor sintactice;
- un limbaj pentru reprezentarea intrărilor lexicale;
- un limbaj în care să poată fi scrise reguli (în general de mai multe tipuri);
- un limbaj de prezentare a structurilor sintactice.

Aceste limbaje pot fi distincte, dar este posibil și ca două sau mai multe dintre ele să se suprapună.

Formalismele DCG[1] și PATR[2] de reprezentare a gramaticilor, care vor fi utilizate, cu precădere, în cele ce urmează, au rolul de a reprezenta în special

---

[1] DCG este un formalism în care gramaticile sunt definite prin clauze. Caracteristica principală a acestui formalism, introdus de către Pereira și Warren (1980), este aceea că el îmbogățește regulile gramaticilor PS independente de context cu argumente și variabile. Multe implementări Prolog oferă extensia notației numită DCG (de la englezescul "definite clause grammars"). Clauzele care definesc gramatica sunt considerate definite deoarece nu reprezintă întrebări, ci clauze cu cap. Ele reprezintă, pur și simplu, reguli și fapte Prolog. O gramatică enunțată în DCG este executabilă în mod direct de către Prolog ca reprezentând un analizor sintactic. DCG facilitează, de asemenea, manevrarea semanticii unui limbaj, astfel încât sensul unei propoziții se întrețese cu sintaxa acesteia (a se vedea § 3.4.1).

[2] PATR este un formalism în care gramatica include caracteristici. Acest formalism se datorează lui Stuart Shieber și asociaților săi de la SRI International. El reprezintă un formalism tipic gramaticilor de unificare. Este suficient de expresiv pentru ca tot ceea ce se reprezintă prin intermediul unui alt formalism să poată fi tradus într-un formalism echivalent de tip PATR, ceea ce a dus la o utilizare a lui pe scară largă în domeniul procesării limbajului natural. PATR este un acronim pentru "parse and translate". Caracteristica principală a formalismului PATR este aceea că el combină regulile

așa-numitele **"phrase structure grammars"** (gramatici ale structurii grupului sintactic sau gramatici de structură sintagmatică) independente de context.

3.1. Noțiunea de constituent

Structura unei propoziții, dată de constituenții acesteia, reprezintă conceptul central al sintaxei.

Definiția 3.1

i) Un **șir** este orice secvență de două sau mai multe elemente adiacente.

ii) Un **constituent** este un șir care are o *coeziune internă*.

Este sarcina gramaticii să atribuie o analiză sintactică oricărei propoziții, adică să reprezinte structura sintactică a acesteia, prin care înțelegem structura propoziției dată de constituenții săi.

În ceea ce privește analiza sintactică, trebuie să remarcăm, încă de la început, că a ști cum să segmentăm o propoziție dată și a construi un sistem de reguli (generative) care să atribuie în mod explicit propozițiilor o structură de constituenți corectă sunt două lucruri complet diferite. Domeniul procesării limbajului natural utilizează, cu precădere, această a doua accepție a termenului de **analiză sintactică**.

Deși se pot construi multe sisteme diferite de reguli gramaticale care să determine o anumită analiză a structurii de constituenți a propozițiilor pe care le generează, ne vom limita la considerarea sistemelor de rescriere concatenative, introduse de Chomsky. Aceste sisteme vor fi numite **gramatici de structură sintagmatică**.

Pentru a determina structura unei propoziții prin constituenții acesteia, trebuie să se indice: care dintre șirurile propoziției analizate reprezintă constituenți și ce fel de constituent este fiecare, deci cărei *categorii* îi aparține. Astfel, **vocabularul unei gramatici** este alcătuit din *categorii, formative* și *caracteristici*, așa cum vom explicita în cele ce urmează.

Categoriile se împart în *categorii gramaticale* și *categorii lexicale*.

Categoriile gramaticale corespund **grupurilor sintactice** sau unor unități cu caracter mai larg decât acestea. Exemple de categorii gramaticale sunt:

---

gramaticilor PS independente de context cu un puternic sistem de caracteristici. (A se vedea § 5.2.3 și, în general, Capitolul 5).

- **S** (de la engl. "sentence") - reprezintă simbolul inițial al unei gramatici;

- **NP** (de la engl. "noun phrase") - reprezintă un grup sintactic al cărui singur element obligatoriu este un substantiv (grup substantival);

-**VP** (de la engl. "verb phrase") - reprezintă un grup sintactic al cărui element principal obligatoriu este un verb (grup verbal);

-**AP** sau **ADJP** (de la engl. "adjectival phrase") - reprezintă un grup sintactic al cărui singur element obligatoriu este un adjectiv (grup adjectival);

- **PP** (de la engl. "prepositional phrase") - reprezintă un grup sintactic al cărui singur element obligatoriu este o prepoziție (grup prepozițional);

- **AvP** sau **ADVP** (de la engl. "adverb phrase") - reprezintă un grup sintactic al cărui singur element obligatoriu este un adverb.

Categoriile lexicale corespund **părților de vorbire.** Ele poartă această denumire deoarece membrii acestor categorii figurează în *lexicon* (a se vedea și § 5.4): N (de la engl. "noun") -substantiv, V (de la engl. "verb") - verb, A (de la engl. "adjective") - adjectiv, Av (de la engl. "adverb") - adverb, P (de la engl. "preposition") - prepoziție, Det (de la engl. "determiner") - determinant (prin care se desemnează, în general, cuvinte precum articolele ori numeralele, care apar în fața substantivelor comune).

Categoriile sunt definite strict formal, adică numai din punctul de vedere al felului în care funcționează în cadrul regulilor.

Formativele sunt elemente minimale care au funcție sintactică. Ca și categoriile, ele pot fi *lexicale* și *gramaticale*.

Formativele lexicale includ elemente lexicale care figurează în lexicon (a se vedea §5.4) ca reprezentând diferite părți de vorbire: fată_N, băiat_N, pentru_P, frumos_A etc. (Precizăm faptul că, în accepția de aici, termenul de *lexicon* este sinonim cu cel de *vocabular*).

Formativele gramaticale sunt elemente menționate în mod individual în anumite reguli ale gramaticii. Ele se mai numesc și *cuvinte gramaticale* ("grammatical words"). În cazul limbii engleze se pot da ca exemple: *by*, care introduce agentul unei construcții pasive (*It was brought by him)* sau *there* ca subiect formal (*There is no one here*). În propoziția românească

Cartea este scrisă de către Ion.

de către reprezintă o prepoziție compusă cu ajutorul căreia se exprimă complementul de agent (persoana care realizează o acțiune). Fiecare dintre prepozițiile *de* și *către* luate în considerație separat reprezintă formative lexicale, în timp ce prepoziția compusă *de către* are funcțiune de *formativ gramatical*.

Caracteristicile, asupra cărora vom reveni, exprimă proprietăți ale categoriilor lexicale și pot fi fonologice, semantice sau sintactice. Fiecare *specificație a unei caracteristici* constă dintr-o caracteristică, spre exemplu *caz* și din o valoare a acelei caracteristici, spre exemplu *acuzativ*. Distincția care se face între categorii și caracteristici se bazează pe opoziția dintre funcția formativă a categoriilor care figurează în regulile gramaticii și funcția de caracterizare pe care o au caracteristicile cu privire la elemente date ale limbajului.

Operația care construiește obiecte complexe pornind de la cele elementare este **concatenarea**. Dându-se două obiecte, x și y, prin concatenare se poate construi fie obiectul xy, fie obiectul yx. **Regulile** prin care se formează aceste obiecte sunt numite, în literatura anglo-saxonă de specialitate, "**phrase structure rules**" (în accepția prezentă reguli ale structurii grupului sintactic sau, într-un sens mai larg, **reguli de structură sintagmatică**). Denumirea atribuită acestor reguli se bazează pe **relațiile sintagmatice** (sau **de succesiune**) care se stabilesc între cuvinte.

Relațiile de tip sintagmatic se stabilesc la oricare nivel lingvistic. Astfel, fonemele, morfemele, cuvintele, sintagmele, asociindu-se în combinații mai largi, stabilesc, în cadrul acestor combinații, relații sintagmatice. S-au propus mai multe criterii de clasificare a relațiilor sintagmatice, după tipul de gramatică în care se face analiza și după nivelul la care se manifestă relația. Conceptul de *sintagmă*, spre exemplu, în accepția tradițională, se poate defini ca reprezentând o unitate cu structură binară, alcătuită din asocierea a două componente aflate în relație sintagmatică. Relațiile sintagmatice (sau de succesiune) se mai numesc și relații contrastive (sau de contrast), combinatorii sau "in praesentia", pentru că se stabilesc între *unități co-ocurente* (care apar în același timp și împreună).

În contextul de față, aceste reguli indică structura grupurilor sintactice dată de constituenții lor. O regulă de tipul $S \rightarrow NP\ VP$ afirmă că orice propoziție conține un grup nominal (NP) și un grup verbal (VP), în această ordine. În cele ce urmează, vom numi regulile de structură sintagmatică sau de tip "phrase-structure" **reguli PS**. În conceperea regulilor PS lingviștii se bazează pe proprietățile formale ale constituenților. (Din punct de vedere intuitiv cuvintele, ca unități independente, reprezintă constituenți. Nu este însă la fel de evident că sintagmele, spre exemplu, reprezintă, și ele, constituenți).

În continuare, vom exemplifica cu două dintre situațiile în care studiul constituenților conduce la stabilirea unor reguli de structură sintagmatică sau reguli PS.

3.1.1. Factori distribuționali

Motivul esențial pentru care trebuie făcută referirea la constituenți atunci când se urmărește realizarea descrierii unui limbaj este acela că o referire de acest tip face posibilă găsirea și enunțarea unor generalizări ale tiparelor, schemelor, șabloanelor în care se încadrează structura propozițiilor limbajului.

Spre exemplu, s-a observat că, în limba engleză, distribuția numelor proprii și a substantivelor la plural este, în mare, aceeași. De asemenea, s-a observat că această distribuție este și cea a altor secvențe de cuvinte, cum ar fi: A+N, Det+N, Det+A+N, Det+N+S etc. Exemple de astfel de secvențe sunt (pentru limba engleză): *this boy* (Det+N), *lazy boys* (A+N), *the lazy boy* (Det+A+N). S-a constatat că, în limba engleză, toate aceste secvențe de cuvinte au aceeași distribuție, ele apărând înaintea unui grup verbal (VP) și după o prepoziție. De aceea s-a presupus că ele mai au ceva în comun, și anume faptul că instanțiază o aceeași categorie. După o analiză de tip statistic a exemplelor găsite, s-a constatat că această categorie are un substantiv ca unic constituent obligatoriu. În urma acestui studiu a fost propusă următoarea regulă pentru structura unei propoziții:

$$S \rightarrow NP\ VP$$

Aceasta este doar o exemplificare, preluată din [17], a modului în care factorii distribuționali pot duce la elaborarea de reguli PS ale unei gramatici. Reguli similare pentru structura unei propoziții pot fi deduse, pe baza unor considerente de aceeași natură, pentru alte limbi.

3.1.2. Coordonarea

Așa cum se arată în [17], distribuția conjuncțiilor *și/sau* în limba engleză, dar și în cazul majorității celorlalte limbi, oferă un alt criteriu pentru studiul constituenților. Astfel, conjuncțiile *și/sau* leagă numai constituenți, mai mult, numai constituenți de același tip(ex.: două NP, două ADJP etc.).

În general, dacă atât șirul X, cât și șirul Y, intervin într-un anumit context $Z_1..Z_2$ (i.e. sunt admise atât șirul Z_1XZ_2 cât și șirul Z_1YZ_2), se poate determina dacă ele reprezintă constituenți testând dacă șirul "X și Y" poate, de asemenea, să apară în același context (i.e. dacă există în limbaj șirul "Z_1X și YZ_2").

Teste de acest tip au fost stabilite de către lingviști pentru ca, împreună cu unele fapte empirice bine cunoscute, să poată fi utilizate pentru a se determina structura dată prin constituenți a unei propoziții. Aplicarea testului amintit poate arăta că șirurile *studenți înțeleg* și *studenți vor* nu pot interveni în calitate de constituenți în contextul *puțini .. cursul*:

a) Puțini ***studenți înțeleg*** cursul, *iar și mai puțini* ***studenți vor*** *să îl urmeze.*

b) Puțini ***studenți înțeleg și studenți vor*** cursul.

Cea de-a doua frază (b) nu este corectă în limba română. Ea ar trebui reformulată astfel:

b') *Puțini studenți înțeleg și vor cursul.*

Același test ne arată că, în cadrul unui grup verbal, atât verbul (*iubește, admiră* din contextul de mai jos), cât și grupul nominal (*cursul de algebră, cursul de geometrie* din același context) intervin în calitate de constituenți:

Andrei iubește cursul de algebră.

Andrei iubește cursul de geometrie.

Andrei iubește cursul de algebră și cursul de geometrie.

Andrei iubește profesorul de informatică.

Andrei admiră profesorul de informatică.

Andrei iubește și admiră profesorul de informatică.

Utilizând concluzia că atât verbul, cât și grupul nominal din structura grupului verbal reprezintă constituenți, concluzie trasă pe baza datelor de tipul celor de mai sus, grupului verbal i s-a putut atribui următoarea structură:

$$VP \rightarrow V\ NP$$

Alte reguli PS au putut fi deduse, în egală măsură, în urma studierii constituenților și a proprietăților formale ale acestora.

Să mai notăm faptul că elementele unui constituent sunt, în general, adiacente. Lingvistica acceptă însă și noțiunea de *constituent discontinuu*, alcătuit din elemente aflate la distanță unele de altele, în interiorul propoziției, dar despre care se poate arăta, prin aplicarea unor teste de tip formal, că sunt legate între ele. Următoarele propoziții preluate din [17] oferă două astfel de exemple pentru limba engleză:

What *do you always* ***take*** *your shoes* ***off*** **for** *in my class?*

Why *do you always* ***take*** *your shoes* ***off*** *in my class?*

Why *do you always* ***remove*** *your shoes in my class?*

În aceste exemple **what..for** reprezintă un constituent discontinuu care poate fi înlocuit prin **why**, iar ***take..off*** reprezintă un constituent discontinuu care poate fi înlocuit prin ***remove***.

Exemple similare pentru limba română ar putea fi:

Studentul își ***dă*** *cu multă ușurință pantofii* ***jos*** *în timpul cursului.*

Studentul își ***scoate*** *cu multă ușurință pantofii în timpul cursului.*

(unde ***dă..jos*** reprezintă un constituent discontinuu care poate fi înlocuit prin ***scoate***)

și

*Îmi **aduc** cu plăcere **aminte** de anii copilăriei.*

*Îmi **amintesc** cu plăcere de anii copilăriei.*

(unde ***aduc..aminte*** reprezintă un constituent discontinuu care poate fi înlocuit prin ***amintesc***).

3.2. Reguli PS și gramatici PS. Gramatici generative

Utilizând reguli PS o gramatică PS poate genera propoziții cărora le atribuie o anumită structură dată prin constituenți și, prin urmare, o anumită *analiză sintactică*. Determinarea structurii propoziției facilitează înțelegerea sensului ei (interpretarea semantică).

Iată câteva dintre cele mai importante **proprietăți ale regulilor PS** ca **reguli de structură**:

1) Regulile PS prezintă **structura** prin specificarea constituenților obligatorii și a celor opționali. Spre exemplu, în cazul limbii engleze, s-a constatat [17] că cele mai frecvente reguli PS referitoare la grupul verbal sunt:

$VP \rightarrow V \ \ NP$

$VP \rightarrow V$

$VP \rightarrow V \ \ PP$

$VP \rightarrow V \ \ NP \ PP$

Acestea reprezintă instanțieri ale unei reguli PS generale, care spune că verbul reprezintă un constituent obligatoriu, în timp ce grupul nominal și grupul prepozițional reprezintă constituenți opționali, regulă PS de forma:

$VP \rightarrow V \ \ (NP) \ (PP)$

(În această notație parantezele rotunde desemnează prezența facultativă a constituenților incluși între paranteze).

2) Regulile PS reprezintă așa-numite **reguli de rescriere**, întrucât ele înlocuiesc categoria pe care o analizează prin constituenții săi, care sunt concatenați. Astfel, spre exemplu, regula PS

$S \rightarrow NP \ \ VP$

reprezintă instrucțiunea de **a rescrie** (sau de a înlocui) simbolul categorial S prin succesiunea de categorii NP VP.

3) Regulile PS sunt **reguli independente de context**. Aceasta înseamnă că, în cazul existenței unei reguli PS de forma $A \rightarrow Z$, rescrierea lui A ca fiind Z nu depinde de contextul în care intervine A.

4) Regulile PS **nu sunt ordonate**. Categoriile pot fi rescrise în orice moment al analizei unei propoziții în care ele intervin. Totuși, la un anumit moment, poate fi analizată o singură categorie.

5) Regulile PS furnizează **proceduri de decizie** referitoare la șirurile de cuvinte în sensul că ele selectează, din totalitatea șirurilor construite cu elemente ale lexiconului, pe acelea care reprezintă constituenți de un anumit tip.

O **derivare** este o secvență de șiruri de simboluri, fiecare dintre acestea fiind obținut din anteriorul, prin aplicarea unei reguli a gramaticii. Derivările încep cu simbolul inițial. Ordinea în care se aplică regulile nu joacă nici un rol. Prin urmare, pot exista mai multe derivări echivalente ale aceleiași propoziții.

Fiecare dintre șirurile aparținând unei derivări reprezintă propoziția analizată într-un mod care este relevant pentru proprietățile formale și *sensul* propoziției. Analiza derivării poate arăta, spre exemplu, dacă un șir dat reprezintă un constituent și, în caz afirmativ, care este tipul acestuia.

Definiția 3.2

O gramatică care se bazează pe reguli PS va fi numită **gramatică PS**. Ea este o gramatică dată de o mulțime *finită* de reguli care operează asupra *categoriilor* și *lexiconului* și generează un *limbaj*, adică o mulțime *infinită* de propoziții.

Preluând terminologia existentă în studiul gramaticilor formale vom nota că simbolurile gramaticii care nu mai pot fi descompuse în continuare se numesc *simboluri terminale*. În cazul limbilor naturale simbolurile terminale sunt chiar cuvintele unei limbi. Celelalte simboluri, cum ar fi NP, VP, S etc. se numesc *simboluri neterminale*. Simbolurile gramaticale cum ar fi N sau V, care se referă la părți de vorbire, se numesc *simboluri lexicale*. Se spune că o astfel de gramatică *derivă* o propoziție dacă există o secvență de reguli care permit rescrierea simbolului inițial sub forma întregii propoziții. Două procese extrem de importante se bazează pe derivare: unul este generarea propozițiilor corecte și deci a limbajului, celălalt constă în identificarea structurii propozițiilor (parsing).

Întrucât regulile PS sunt independente de context, gramaticile PS reprezintă gramatici independente de context și ele generează **limbaje independente de context**.

Vom mai nota faptul că gramaticile PS pot conține *reguli recursive*.

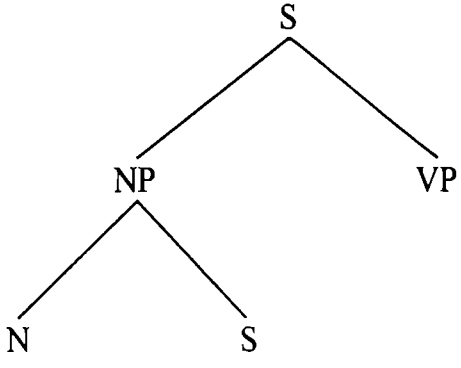
Definiția 3.3

Un simbol neterminal (S, NP etc.) este recursiv dacă poate constitui rădăcina unui arbore sau subarbore care îl conține.

O astfel de situație se întâlnește în următorul exemplu:

[Profesorul, [care cunoaște limba engleză]$_S$]$_{NP}$, a tradus textul.

Arborele de derivare corespunzător este:



Întrucât însuși S este un simbol recursiv, așa cum se vede și în exemplul anterior, o gramatică PS are capacitatea de a genera propoziții și fraze de o mare complexitate.

Există și *neajunsuri* ale analizei sintactice bazate pe gramatici PS. Sunt cunoscute, în aproape toate limbile, diferite tipuri de construcții a căror structură dată prin constituenți nu poate fi în mod corect și riguros pusă în evidență cu ajutorul acestui tip de gamatici. Cazul *constituenților discontinui*[3], la care ne-am referit anterior, reprezintă o astfel de situație. Există, de asemenea, cazuri de *ambiguitate* sau de *propoziții eliptice*, care nu pot fi tratate în mod satisfăcător de riguros de către gramaticile PS. Astfel, în cazul propoziției eliptice

Profesorul s-a bazat întotdeauna pe student și studentul pe el.

[3] Grupurile verbale sunt adesea și în foarte multe limbi discontinue, iar numeroase teorii gramaticale consideră discontinuitatea grupurilor verbale ca reprezentând un obiect de studiu major (Emonds, 1976, 1979; Hepple, 1990; Hoekstra, 1990; McCawley, 1982; Reape, 1990; Ross, 1973). Constituenții discontinui în general sunt suficient de frecvenți pentru a suscita interesul crescând al cercetătorilor. Problema descrierii structurilor alcătuite din elemente neadiacente prezintă numeroase dificultăți și încercările de rezolvare a ei au dat naștere unor tehnici de analiză sintactică specifice "Gramaticilor PS discontinue" și diferite de cele clasice (a căror prezentare se va face în capitolul 4 al lucrării de față).

este dificil de atribuit o **analiză sintactică de tip PS** şirului *"studentul pe el"*. Pentru a justifica utilizarea prepoziţiei *pe* în acest context, este necesar să ne raportăm la un nivel mai abstract al analizei, în care există formularea completă *studentul s-a bazat întotdeauna pe el*. Tocmai de aceea s-au dezvoltat, de-a lungul anilor, o multitudine de teorii lingvistice diferite asupra analizei sintactice şi a tipurilor de gramatici. Unele dintre acestea, asupra cărora vom reveni, întrucât ele au fost cu succes implementate pe calculator, inclusiv pentru limba română, concep un model de gramatică care **nu** se bazează pe noţiunea de constituent, ci pe relaţiile directe existente între cuvinte, indiferent dacă prin acestea înţelegem funcţii gramaticale, ori legături care organizează cuvintele în unităţi mai largi (cum ar fi grupurile sintactice). O importantă clasă de gramatici de acest fel este cea a *gramaticilor de dependenţă*, asupra cărora vom reveni în § 3.5. Anumite clase de *gramatici contextuale* (vezi § 3.6) s-au dovedit a fi un model extrem de adecvat în special pentru sintaxa limbajelor naturale.

Revenind la gramaticile PS vom observa că ele reprezintă cadrul mai larg în care ne-am propus să ne referim la **gramaticile generative**.

Este cunoscut faptul că obiectivul unei sintaxe tradiţionale îl constituie descompunerea frazelor în propoziţii componente şi descompunerea propoziţiilor în părţi de propoziţie. Pentru acest motiv, sintaxa tradiţională poate fi considerată o **sintaxă analitică**.

Gramaticile generative sunt modele lingvistice propuse de N. Chomsky în lucrările din perioada 1957-1965, ca reacţie în primul rând la aspectul analitic excesiv existent până la acea oră. Aşa cum sunt definite în [9], gramaticile generative reprezintă modele formalizate de tip **sintetic**, concepute ca mecanisme formale capabile să capteze şi să explice procesele creative ale limbii. În procesul de adecvare progresivă a modelului generativ[4] la descrierea lingvistică, N. Chomsky a propus, succesiv, trei tipuri de modele generative: modelul gramaticii cu un număr finit de stări; al gramaticii de constituenţi imediaţi; al gramaticii generativ-transformaţionale.

Automatele cu număr finit de stări, la care ne-am referit în capitolul 2, reprezintă cele mai simple sintaxe generative. Maşina memorează un număr de stări şi este programată să treacă de la o stare la alta, într-o ordine dată, fiecare tranziţie având ca efect producerea unui cuvânt. Orice traseu parcurs de maşină reprezintă, de fapt, *procesul de construcţie a unei propoziţii*. O propoziţie este o secvenţă de tranziţii în care ultima tranziţie are ca urmare atingerea unei stări finale.

Ultimul model prezentat de N. Chomsky în 1965 este considerat ca având puterea generativă, descriptivă şi explicativă cea mai mare şi este desemnat astăzi ca reprezentând **varianta standard a gramaticii generative sau varianta generativ-transformaţională**. Obiectivul fundamental al acestui

[4] Pentru care vezi şi [9], s.v. *gramatică* (mai ales tipurile **8** şi **10**), pp. 227-229.

model a fost prezentarea organizării și a funcționării așa-numitei **competențe lingvistice**. Astfel, așa cum se arată în [41], cunoașterea unui număr limitat de simboluri și de reguli de combinare a acestora reprezintă tocmai abilitatea (sau mulțimea abilităților) de a folosi o limbă și se numește **competență.** Actele concrete de producere a propozițiilor constituie domeniul teoretic nelimitat al **performanței**.

Obiectivul modelului din 1965 a fost, prin urmare, prezentarea organizării și a funcționării competenței lingvistice, precum și încercarea de a oferi explicații pentru fiecare dintre trăsăturile acesteia: pentru "creativitatea" competenței; pentru capacitatea de a distinge echivalențele a două sau mai multe fraze dintr-o limbă, precum și de a surprinde ambiguitățile; pentru capacitatea de a distinge frazele *bine-formate* de cele *rău-formate*. În esență, modelul este conceput ca un mecanism formal de tip generativ, care, prin *regulile recursive de structură (reguli PS)*, produce infinitatea propozițiilor și a frazelor corecte dintr-o limbă.

O *variantă a gramaticii generative* formulată de N. Chomsky după 1980 este cea dată de așa-numita teorie "Government and Binding" (guvernare și legare), care aduce modificări numeroase și substanțiale în raport cu varianta standard. Aceasta din urmă rămâne însă cadrul general în care ne plasăm și în care ulterior s-a dezvoltat sintaxa transformațională românească.

3.3. Clase de gramatici și limbaje

Formalismele de reprezentare a gramaticilor bazate pe reguli de rescriere pot fi comparate în funcție de **capacitatea** lor **generativă**, reprezentată prin gama de limbaje pe care o poate descrie fiecare formalism în parte. O gramatică formală este un sistem pentru rescrierea șirurilor peste un alfabet V. În contextul actual suntem interesați exclusiv de studiul limbilor naturale, dar se pare că nici un limbaj natural nu poate fi caracterizat în mod suficient de exact pentru a se defini capacitatea lui generativă. Limbajele formale, spre deosebire de limbile naturale, permit o descriere matematică exactă. Cele mai importante rezultate în studiul limbajelor formale sunt cele datorate lui N. Chomsky, începând din anul 1956.

Chomsky identifică patru clase de gramatici:

Clasa 0: gramatici PS nerestricționate

În cadrul acestei clase nu există restricții asupra tipurilor de reguli și, prin urmare, sunt permise reguli de rescriere arbitrare.

Clasa 1: gramatici dependente de context (sau senzitive de context)

Regulile de rescriere sunt de forma

$$x \rightarrow y$$

unde x şi y sunt secvenţe de simboluri terminale şi/sau neterminale. Lungimea lui y este mai mare sau egală cu lungimea lui x. Precizăm că, în cazul limbilor naturale, *simbolurile terminale* sunt chiar cuvintele unei anumite limbi sau - dacă se doreşte şi integrarea morfologiei - morfeme, iar *simbolurile neterminale* sunt cele de tipul S, NP, VP etc.

Clasa 2: gramatici independente de context

Regulile de rescriere sunt de forma

$$A \rightarrow x$$

unde: A este un simbol neterminal;
x este o secvenţă de simboluri terminale şi/sau neterminale.

Clasa 3: gramatici regulate

Regulile de rescriere sunt de forma

$$A \rightarrow tB$$

sau

$$A \rightarrow t$$

unde: A, B sunt simboluri neterminale;
t este simbol terminal.

Observaţia 3.1

Cu cât clasa are un număr de ordine mai mare, cu atât ea este mai restrictivă. Întrucât limbajele generate de gramaticile regulate reprezintă o submulţime a celor generate de gramaticile independente de context, care constituie o submulţime a celor generate de gramaticile dependente de context, care, la rândul lor, formează o submulţime a celor generate de gramaticile de tipul 0, ele constituie o ierarhie de limbaje, numită **ierarhia Chomsky**.

Dintre limbajele ce constituie ierarhia Chomsky, un interes deosebit suscită cele *independente de context*, întrucât s-a arătat că *aproape toate construcţiile din aproape toate limbile naturale pot fi analizate folosind tehnici care limitează sistemul la analiza limbajelor de acest tip*. Limbajele independente de context sunt cele care pot fi descrise prin intermediul unei gramatici PS independente de context. Formalismul oferit de gramaticile independente de context este suficient de puternic pentru a descrie majoritatea structurilor care apar în limbile naturale, dar, în acelaşi timp, este suficient de restricţionat pentru a permite construcţia unor analizori sintactici eficienţi.

Gramaticile independente de context sunt relativ simplu de scris şi de modificat. Există deja o tradiţie în ceea ce priveşte tehnicile de traducere, compilare şi analiză sintactică (parsing) legate de aceste gramatici. Ele sunt considerate deosebit de importante datorită existenţei unor algoritmi considerabil de eficienţi pentru recunoaşterea şi analiza sintactică a limbajelor independente de context. Tehnici de parsing suficient de evoluate au fost deja puse la punct relativ la aceste limbaje, iar parsingul independent de context rămâne unul dintre instrumentele de bază ale procesării limbajului natural. În prezent se cunoaşte faptul că majoritatea covârşitoare a structurilor oricărui limbaj natural pot fi analizate sintactic în mod eficient utilizând tehnici de parsing independente de context. Aceste tehnici vor fi descrise pe larg în capitolul 4 al lucrării de faţă.

O altă problemă care s-a pus în lingvistica computaţională a fost aceea de a găsi cea mai mică clasă de limbaje formale despre care se poate considera că include toate limbajele naturale. Cea mai mică astfel de clasă care a fost luată în mod serios în consideraţie a fost cea a limbajelor de tipul 3, numite *şi limbaje cu stări finite* sau *limbaje regulate*. Acestea sunt si limbajele care pot fi recunoscute de către RTSF-urile descrise în capitolul 2 al lucrării de faţă. Automatele cu stări finite sunt considerate a fi echivalente, în ceea ce priveşte puterea de expresie, cu gramaticile regulate. Prin urmare, ele nu sunt suficient de puternice pentru a descrie toate limbajele care pot fi descrise de către o gramatică independentă de context. Pentru a obţine puterea descriptivă a acesteia din urmă este necesară introducerea noţiunii de *recursivitate* şi utilizarea aşa-numitelor **reţele de tranziţie recursive**, care nu constituie însă obiectul lucrării de faţă. Aceasta şi deoarece, în prezent, gramaticile PS independente de context sunt fără dubiu preferate reţelelor de tranziţie recursive atât în descrierea limbajelor naturale, cât şi a celor formale. Gramaticile PS independente de context şi-au dobândit supremaţia în primul rând prin faptul că ele lasă deschise toate chestiunile legate de procesare, oferind atât lingvistului, cât şi programatorului, o gamă largă de alegeri şi de posibilităţi. În timp ce reţelele de tranziţie recursive implică, spre exemplu, o ontologie a stărilor, împreună cu proprietăţi ale acestora - iniţial, final - gramaticile PS independente de context nu sunt supuse nici unei constrângeri de acest fel.

Cea mai mică clasă de limbaje formale care a fost luată în consideraţie ca incluzând toate limbajele naturale este, prin urmare, cea a limbajelor regulate. Există însă argumente valide [29] conform cărora nu toate limbajele naturale sunt limbaje regulate, ci dimpotrivă. Unul dintre aceste argumente se bazează, spre exemplu, pe faptul că un şir de forma

Un doctor [pe care un doctor]^m [l-a angajat]^n a angajat o asistentă.

constituie o frază legală (sau admisă, deci corectă) în limba română numai dacă m şi n coincid.

Faptul că limbile naturale nu sunt regulate nu înseamnă, neapărat, că *tehnicile cu stări finite* utilizate în analiza sintactică a limbajelor regulate nu sunt relevante pentru domeniul procesării limbajului natural. Teoria gramaticilor regulate și a automatelor finite continuă să joace un anumit rol în studiul limbajelor naturale, chiar dacă acestea nu sunt regulate. Anumite utilizări și aplicații au fost deja menționate în capitolul 2 al lucrării de față.

3.3.1. Gramatici independente de context probabiliste și nonprobabiliste

Ideea care stă la baza **gramaticilor probabiliste sau stocastice** este aceea a augmentării unei teorii gramaticale cu probabilități, prin atribuirea de probabilități *șirurilor*, *arborilor de derivare* și *derivărilor*.

Există diverse motivații în sprijinul realizării acestei augmentări. Un prim motiv ar putea fi dat de dorința clasificării diferitelor șiruri de cuvinte conform probabilităților asociate pentru a-l putea selecta pe cel mai probabil dintr-o mulțime de șiruri generate, spre exemplu, de un sistem de recunoaștere a vorbirii. În acest caz gramatica stocastică este utilizată ca un *model* al limbii. Un alt motiv ar fi acela de a putea selecta o anumită analiză sintactică dintre cele atribuite de către o teorie lingvistică unui șir dat de cuvinte, și anume acea analiză care este cea mai probabilă. În acest caz, gramatica stocastică este utilizată pentru *dezambiguizare*. În fine, utilizarea unei astfel de gramatici poate să accelereze procesul de *parsing* prin limitarea spațiului de căutare. Procesul de analiză sintactică devine, în acest caz, un așa-numit "parsing bazat pe preferințe" [Kimball 1973] și [Marcus 1980].

Cel mai simplu exemplu legat de modul în care o gramatică poate fi augmentată printr-o teorie probabilistă îl constituie **gramaticile independente de context stocastice**. Pentru a înțelege mai bine cum se face această augmentare, ne vom referi, mai întâi, la gramaticile independente de context așa cum ne sunt ele cunoscute din teoria limbajelor formale.

3.3.1.1. Gramatici independente de context

O *gramatică formală* reprezintă un sistem de rescriere a șirurilor peste un alfabet dat. Gramaticile formale sunt utilizate în definirea limbajelor de programare, dar și a unor părți din limbajele naturale.

Definiția 3.4

O **gramatică independentă de context** G este un cvadruplu (V_N, V_T, S, R), unde:

V_N este o mulțime finită de simboluri neterminale;

V_T este o mulțime finită de simboluri terminale; vom nota prin V reuniunea $V_N \cup V_T$;

$S \in V_N$ este un simbol de start distins sau o axiomă;

R este o mulțime finită de producții $X \to \beta$, unde $X \in V_N$ și $\beta \in V^*$.

Șirul $\alpha X \gamma \in V^*$ poate fi *rescris* într-un singur pas ca $\alpha\beta\gamma \in V^*$ dacă și numai dacă $X \to \beta$ aparține lui R. Aceasta înseamnă că o apariție a simbolului neterminal X într-un șir este înlocuită de șirul β, ceea ce se notează în felul următor: $\alpha X \gamma \Rightarrow \alpha\beta\gamma$. În general, dacă un șir $\phi \in V^*$ poate fi rescris, într-un număr finit de pași, ca fiind șirul φ, aceasta se notează

$$\phi \Rightarrow^* \varphi .$$

Prin urmare, $\Rightarrow^*$ este închiderea tranzitivă a relației $\Rightarrow$. Numărul regulilor de rescriere aplicate poate fi nul, fiind permisă situația $\phi \Rightarrow^* \phi$, ceea ce arată că $\Rightarrow^*$ este reflexivă. Dacă are loc $S \Rightarrow^* \phi$, adică ϕ poate fi derivat din simbolul de start al gramaticii S, atunci ϕ este numită *formă sentențială*.

L(G) denotă mulțimea de șiruri peste V_T generate de G și este definit ca reprezentând mulțimea

$$\left\{ w \in V_T^* \mid S \Rightarrow^* w \right\}.$$

O gramatică independentă de context se spune că este în *forma normală Chomsky (FNC)* dacă producțiile ei sunt fie de tipul $X_i \to X_j X_k$, producție binară care rescrie un neterminal sub forma a două simboluri neterminale, fie de tipul $X_i \to w$ cu $w \in V_T$, producție unară care rescrie un simbol neterminal ca un unic simbol terminal. Gramaticile independente de context care, în plus, permit producții de forma $X_i \to X_j$, adică producții unare care rescriu un simbol neterminal sub forma unui alt simbol neterminal, reprezintă gramatici în *forma normală Chomsky extinsă (FNCE)*.

O *derivare* a unui șir ω de simboluri terminale este o succesiune de rescrieri ale unor șiruri peste V de forma

$$S = \phi_0 \Rightarrow \ldots \Rightarrow \phi_M = \omega ,$$

în care prima formă sentențială ϕ_0 constă numai din axioma S, iar ultima, ϕ_M, este chiar ω. O astfel de derivare se numește *derivare stângă* dacă și numai dacă, la fiecare pas, este rescris simbolul neterminal al formei sentențiale aflat *cel mai la stânga*. Aceasta presupune că fiecare formă sentențială este

descompusă în mod unic într-un şir iniţial - posibil vid - de simboluri terminale, urmat de un simbol neterminal, urmat de restul şirului. Întrucât aceasta specifică care simbol neterminal al şirului este rescris de fiecare dată, o derivare stângă poate fi reprezentată prin succesiunea $r_1,...,r_M$ de reguli aplicate la fiecare pas. Un pas al unei derivări stângi va fi notat $\Rightarrow_s$, iar închiderea tranzitivă corespunzătoare se va nota $\Rightarrow_s^*$. În cele ce urmează, vom fi interesaţi de stabilirea *legăturii dintre derivări şi arbori de derivare.*

Un *arbore* τ este un graf conex, orientat şi aciclic. Mulţimea vârfurilor arborelui va fi notată *N*. Vom considera că există o funcţie *l*, care face corespondenţa dintre vârfurile arborelui şi o mulţime de simboluri, aşa-numita *funcţie de etichetare*. Arcele arborelui descriu *relaţia de dominare imediată DI.* (Dacă vârful n "domină imediat" vârful n', atunci n se numeşte tatăl lui n', iar n' se numeşte fiul lui n). Relaţia de dominare imediată nu este tranzitivă şi nici reflexivă. Închiderea ei tranzitivă se numeşte *relaţie de dominare* şi se notează cu *D*.

Relativ la arbori există, de asemenea, o ordine parţială dată de poziţia orizontală a fiecărui vârf - *relaţia de precedenţă P*. Dacă un vârf domină un alt vârf, atunci nici unul dintre ele nu îl precede pe celălalt. Prin urmare, o pereche de vârfuri (n,n') poate fi un element al lui *P* numai dacă nici perechea (n,n') şi nici perechea (n',n) nu reprezintă elemente ale lui *D*.

Arcele unui arbore nu au voie să se intersecteze. Aceasta înseamnă că avem:

$$\forall n,n',n'' \quad P(n,n') \wedge D(n,n'') \Rightarrow P(n'',n')$$

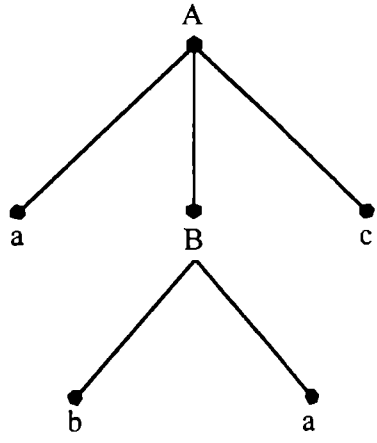
şi

$$\forall n,n',n'' \quad P(n,n') \wedge D(n',n'') \Rightarrow P(n,n'').$$

În plus, oricare două vârfuri distincte ale arborelui, n şi n', vor fi legate fie prin relaţia *D*, fie prin relaţia *P*, i.e. este adevărată numai una dintre

$$D(n,n'),\ D(n',n),\ P(n,n')\ sau\ P(n',n).$$

Un arbore are o unică rădăcină, prin urmare există un unic element minimal relativ la *D*. Dacă τ este un arbore, atunci $R(\tau)$ desemnează rădăcina lui τ. Elementele maximale relativ la *D* se numesc *frunzele* arborelui. Acestea trebuie să fie ordonate de către relaţia *P*. Secvenţa ordonată de către *P* a etichetelor frunzelor formează *coroana arborelui*. $Y(\tau)$ desemnează coroana arborelui τ. Spre exemplu, coroana arborelui τ din figura următoare



este $Y(\tau)$ = *abac*. Elementele care nu sunt maximale relativ la relația D se numesc *vârfuri interne*.

Un *arbore de derivare ("parse tree")* al unui șir w de simboluri terminale generat de o gramatică independentă de context G, trebuie să îndeplinească următoarele condiții:

- rădăcina arborelui este etichetată cu simbolul de start (axioma) S;
- toate frunzele arborelui sunt etichetate cu elemente din V_T; mai exact, produsul arborelui trebuie să fie ω;
- toate vârfurile interne sunt etichetate cu elemente din V_N
- dacă în arborele de derivare există un vârf etichetat cu X și care domină imediat vârfurile $n_1, \ldots, n_k$ etichetate cu $X_1, \ldots, X_k$ (unde n_i precede pe n_j pentru $i < j$, i.e. $P(n_i, n_j)$), atunci există în G o producție de forma $X \rightarrow X_1, \ldots, X_k$.

τ (G) desemnează mulțimea arborilor de derivare generați de gramatica G și se definește ca fiind $\{\tau \mid Y(\tau) \in L(G)\}$.

O gramatică G este *finit ambiguă* dacă și numai dacă există doar un număr finit de arbori de derivare corespunzător oricărui șir finit din L(G), adică dacă

$$\forall \omega \in L(G) \quad |\omega| < \infty \rightarrow |\{\tau \mid Y(\tau) = \omega\}| < \infty .$$

Această cerință este echivalentă cu a cere ca nici un simbol neterminal să nu poată fi rescris în el însuși în unul sau mai mulți pași. Aceasta înseamnă că este imposibil să avem $X \Rightarrow^+ X$. În cele ce urmează, vom presupune că orice

gramatică este finit ambiguă. (Chiar şi în aceste condiţii însă numărul de arbori de derivare creşte exponenţial cu lungimea şirurilor de caractere).

Un arbore de derivare parţial este un arbore de derivare pentru care se renunţă la condiţia ca rădăcina să fie etichetată cu axioma S şi la condiţia ca frunzele să fie etichetate cu simboluri terminale. Arborii de derivare parţiali pot fi combinaţi prin operaţia de *substituţie a arborilor*. Definim, în continuare, *substituţia arborilor la stânga*.

Substituţia arborilor la stânga:

Fie τ şi τ' arbori de derivare parţiali. Vom nota prin $\tau \circ \tau'$ arborele obţinut prin extinderea lui τ în urma identificării frunzei sale aflate cel mai la stânga (i.e. a frunzei minimale relativ la *P*) şi etichetate cu un simbol neterminal cu rădăcina arborelui τ'. Frunza selectată a lui τ se numeşte *poziţia de substituţie*. Eticheta corespunzătoare poziţiei de substituţie trebuie să fie aceeaşi cu eticheta rădăcinii arborelui τ'. Relaţiile de dominanţă şi de precedenţă, *D* şi respectiv *P*, se extind în mod analog. Deşi operaţia de substituţie a arborilor nu este asociativă, vom nota prin $\tau \circ \tau' \circ \tau''$ succesiunea de operaţii $((\tau \circ \tau') \circ \tau'')$.

3.3.1.2. Gramatici independente de context stocastice

Gramaticile independente de context stocastice reprezintă cel mai simplu exemplu legat de modul în care o gramatică poate fi augmentată cu o teorie probabilistă. În esenţă, pentru a realiza o asemenea augmentare, se adaugă o repartiţie de probabilitate P a mulţimii producţiilor R.

Definiţia 3.5

O **gramatică independentă de context stocastică** este un cvintuplu (V_N, V_T, S, R, P), unde:

V_N este o mulţime finită de simboluri neterminale;

V_T este o mulţime finită de simboluri terminale; din nou V semnifică $V_N \cup V_T$;

$S \in V_N$ este un simbol de start distins sau axiomă;

R este o mulţime finită de producţii $X \to \beta$, unde $X \in V_N$ şi $\beta \in V^*$;

P este o funcţie de la R în [0,1], astfel încât $\forall X \in V_N \quad \sum_{\beta \in V^*} P(X \to \beta) = 1$.

Probabilitățile șirurilor vor fi definite în termeni de probabilități ale arborilor, care, la rândul lor, vor fi definite în termeni de probabilități ale derivărilor. (Probabilitățile șirurilor pot fi definite și direct în termeni de probabilități de derivare).

Definiția 3.6

Probabilitatea unui șir este suma probabilităților arborilor de derivare pentru care acest șir reprezintă coroana arborelui.

Definiția 3.7

Probabilitatea unui arbore de derivare este probabilitatea derivării stângi corespunzătoare.

Fie derivarea stângă corespunzătoare o derivare reprezentată prin șirul producțiilor utilizate la fiecare pas al procesului de rescriere și fie variabila aleatoare ξ_m producția utilizată în rescriere la pasul m. În aceste condiții putem privi o *derivare stângă* ca fiind un *proces stocastic* $\xi_1, \ldots, \xi_M$ în care mulțimea stărilor este reprezentată de mulțimea producțiilor gramaticii. Aceasta ne permite să definim *probabilitatea unei derivări stângi* în mod recursiv, după cum urmează:

$$P(\xi_1 = r_{i_1}, \ldots, \xi_{M-1} = r_{i_{M-1}}, \xi_M = r_{i_M}) =$$

$$= P(\xi_M = r_{i_M} \mid \xi_1 = r_{i_1}, \ldots, \xi_{M-1} = r_{i_{M-1}}) \cdot P(\xi_1 = r_{i_1}, \ldots, \xi_{M-1} = r_{i_{M-1}}) =$$

$$= \prod_{m=1}^{M} P(\xi_m = r_{i_m} \mid \xi_1 = r_{i_1}, \ldots, \xi_{m-1} = r_{i_{m-1}}).$$

Pentru a caracteriza în întregime acest proces stocastic este necesar să specificăm probabilitățile

$$P(\xi_m = r_{i_m} \mid \xi_1 = r_{i_1}, \ldots, \xi_{m-1} = r_{i_{m-1}})$$

pentru toate valorile posibile ale lui *m (1,...,M)* și toți $i_1, \ldots, i_M$. Presupunerea de independență

$$P(\xi_m = r_{i_m} \mid \xi_1 = r_{i_1}, \ldots, \xi_{m-1} = r_{i_{m-1}}) = P(r_{i_m})$$

este caracteristică pentru gramaticile independente de context stocastice. Aici $P(r_i)$ reprezintă probabilitatea producției r_i dată de către gramatica stocastică. Aceasta înseamnă că probabilitatea de a rescrie un simbol neterminal X utilizând o producție din R este independentă de secvențele de rescriere anterioare. Prin urmare, avem:

$$P(\xi_1 = r_{i_1}, \ldots, \xi_M = r_{i_M}) = \prod_{m=1}^{M} P(r_{i_m}).$$

Oricât de riguroasă ar părea însă, din punct de vedere matematic, această abordare, ea lasă neobservată relația destul de complicată dintre secvența de pași ai derivării și vârful neterminal curent cel mai din stânga al arborilor de derivare parțiali care rezultă la fiecare pas al acesteia. Fiecare producție $X \to X_1, \ldots, X_k$ din R corespunde unui arbore de derivare parțial în care rădăcina, etichetată cu X, domină imediat o secvență de noduri etichetate $X_1, \ldots, X_k$, nemaiexistând și alte noduri.

Fie τ_m arborele de derivare parțial corespunzător lui r_m într-o derivare stângă $r_1, \ldots, r_M$. Atunci $\tau_1 \circ \ldots \circ \tau_M$ este arborele de derivare corespunzător acestei derivări stângi.

Fie $t_m = \tau_1 \circ \ldots \circ \tau_m$ arborele de derivare parțial rezultat în urma aplicării primelor m reguli de rescriere și fie secvența de arbori de derivare parțiali t_m obținută pentru $m = 1, \ldots, M$. Se observă că arborele de derivare final este ultimul element al acestei secvențe, t_M. Pare mai natural ca probabilitatea arborelui de derivare rezultat să fie discutată în termenii acestei secvențe de arbori de derivare parțiali, decât raportat la secvența de reguli de rescriere corespunzătoare, deși noțiunile sunt izomorfe. Aceasta conduce la următoarea formulare pentru *probabilitatea unui arbore de derivare* τ:

$$P(\tau) = P(t_M) = \prod_{m=1}^{M} P(\tau_m \mid t_{m-1}) \tag{3.1}$$

unde $t_m = \tau_1 \circ \ldots \circ \tau_m$, $1 \le m \le M$

$$t_0 = \begin{bmatrix} N = \{n_0\} \\ D = \{(n_0, n_0)\} \\ P = \phi \\ l(n_0) = S \end{bmatrix}$$

iar $P(\tau_m \mid \tau_1 \circ \ldots \circ \tau_{m-1}) = P(r_m \mid r_1, \ldots, r_{m-1})$.

În cele ce urmează, se va folosi o funcție de extracție $g(t_m)$ pentru a extrage informația relevantă din arborele de derivare parțial t_m, atunci când se estimează probabilitățile $P(\tau_{m+1} \mid t_m)$. Aceasta înseamnă că ne vom uita numai

la acele porțiuni ale arborelui pe care le considerăm relevante pentru estimarea probabilității:

$$P(\tau_{k+1} \mid t_m) \approx P(\tau_{k+1} \mid g(t_m)). \tag{3.2}$$

În cazul gramaticilor independente de context stocastice informația "interesantă" și deci relevantă este dată de eticheta celui mai din stânga neterminal al coroanei arborelui. Această etichetă trebuie să coincidă cu membrul stâng al producției utilizate în următorul pas al rescrierii. Astfel, funcția de extracție $g(\tau)$ este $L(Y(\tau))$, unde $Y(\tau)$ desemnează coroana arborelui τ, iar $L(\phi)$ întoarce simbolul neterminal cel mai din stânga al șirului ϕ.

Probabilitățile atribuite analizelor sunt păstrate în timpul transformărilor la și de la forma normală Chomsky (FNC):

- producțiile de forma $A \to BCDE\,(p)$ cu A, B, C, D, E $\in V_N$ sunt înlocuite cu o mulțime de producții de forma $A \to BC'\,(p)$, $C' \to CD'\,(1.0)$ și $D' \to DE\,(1.0)$. Aceasta introduce neterminalele noi C', D', E', care nu mai figurează nicăieri altundeva în gramatica transformată;
- producțiile de forma $A \to B\,(p_1)$ sunt înlăturate (în afara cazului în care se utilizează FNCE); pentru fiecare producție de forma $X \to \alpha A \beta\,(p_2)$ cu $\alpha, \beta \in V^*$, se introduce o nouă producție $X \to \alpha B \beta\,(p_1 \cdot p_2)$;
- producțiile de forma $A \to \varepsilon\,(p_1)$, unde ε este șirul vid sunt înlăturate; pentru fiecare producție de forma $X \to \alpha A \beta\,(p_2)$, se introduce o nouă producție $X \to \alpha\beta\,(p_1 \cdot p_2)$;
- producțiile de forma $X \to \alpha a \beta\,(p)$ cu $a \in V_T$ sunt înlocuite cu perechea de producții $X \to \alpha A' \beta\,(p)$ și $A' \to a\,(1.0)$.

Ca o concluzie a celor prezentate până acum, fără a dori să intrăm în detaliile abordării statistice a gramaticilor independente de context, vom remarca faptul că, în principiu, este necesar *să cunoaștem sau să estimăm probabilitatea utilizării fiecărei reguli de rescriere.* Cea mai simplă abordare este aceea de a număra de câte ori este utilizată fiecare regulă în cadrul unui *corpus* ce conține propoziții gata analizate sintactic și de a folosi acest număr în estimarea probabilității de utilizare a fiecărei reguli. Spre exemplu, dacă luăm în

considerație o categorie C și o gramatică ce conține m reguli $R_1, \ldots, R_m$ al căror membru stâng este dat de C, atunci se poate estima în felul următor probabilitatea de a utiliza regula R_j pentru a-l deriva pe C:

$$P(R_j \mid C) \cong \text{Numără}(\#\text{ utilizări } R_j) / \sum_{i=1}^{m} (\#\text{ utilizări } R_i).$$

Având astfel de estimări ale utilizării producțiilor se pot dezvolta algoritmi care, pentru o propoziție dată, să determine arborele de derivare cel mai probabil care ar fi putut genera acea propoziție. Această tehnică necesită efectuarea anumitor presupuneri de independență în legătură cu utilizarea regulilor (a se vedea și § 4.5.9). În particular, trebuie presupus că probabilitatea ca un constituent să fie derivat de către o regulă R_j este independentă de modul în care acest constituent va fi utilizat ca subconstituent. Spre exemplu, această presupunere ar implica faptul că probabilitățile regulilor având în membrul stâng un grup nominal sunt aceleași, indiferent dacă grupul nominal este pe post de subiect sau de obiect al unui verb.

Făcând o astfel de presupunere se poate dezvolta, conform [4], un formalism bazat pe probabilitatea ca un constituent C să genereze o secvență de cuvinte $w_i, w_{i+1}, \ldots, w_j$, secvență notată $w_{i,j}$. Acest tip de probabilitate se numește **probabilitate interioară** deoarece atribuie o probabilitate secvenței de cuvinte în interiorul constituentului. Ea va fi notată $P(w_{i,j} \mid C)$. O problemă importantă care se ridică este aceea a estimării probabilităților interioare. Cazul categoriilor lexicale este simplu. Așa-numitele **probabilități de generare lexicală**, $P(w_i \mid C_i)$, prin care înțelegem probabilitatea ca o anumită categorie C_i să fie realizată prin cuvântul w_i, pot fi estimate, pur și simplu, numărând aparițiile fiecărui cuvânt în cadrul fiecărei categorii. Trebuie remarcat faptul că o astfel de probabilitate reprezintă probabilitatea ca o categorie dată să fie realizată printr-un anumit cuvânt, iar nu probabilitatea ca un cuvânt dat să aparțină unei anumite categorii. Spre exemplu, P(*măr* | N) reprezintă probabilitatea interioară ca constituentul N să se realizeze sub forma cuvântului *măr*.

Utilizând astfel de probabilități de generare lexicală se pot găsi probabilitățile ca anumiți constituenți să genereze secvențe mai lungi de cuvinte. Presupunând, spre exemplu, că este dată gramatica probabilistă cu următoarele producții

	Reguli	Probabilități
1	S → NP VP	1
2	VP → V	.386
3	VP → V NP	.393
4	VP → V NP PP	.22
5	NP → NP PP	.24
6	NP → N N	.09
7	NP → N	.14
8	NP → ART N	.55
9	PP → P NP	1

Probabilități de generare lexicală

P(*un* |ART)	.36
P(*insecte* | N)	.025
P(*zboară* | V)	.076
P(*place* | V)	.1
P(*ca* | P)	.068
P(*un* | N)	.001
P(*măr* | N)	.063
P(*măr* |V)	.05
P(*păsări* | N)	.076
P(*floare* | N)	.063
P(*floare* | V)	.05

putem determina probabilitatea ca un constituent NP să genereze secvența *un măr* după cum urmează: în gramatica dată există numai două reguli având în membrul stâng categoria NP care ar putea genera o secvență de două cuvinte. Probabilitatea ca NP să genereze cuvintele *un măr* va fi suma probabilităților celor două moduri în care această secvență poate fi derivată:

P(*un măr* | NP) = P(Regula 8 | NP)*P(*un* | ART)*P(*măr* | N) +

+P(Regula 6 | NP)*P(*un* | N) *P(*măr* | N)=

=.55*.36*.06+.09*.001*.06=.012

Această probabilitate poate fi în continuare utilizată pentru a calcula, în mod analog, probabilitățile unor constituenți mai ampli. Utilizând această metodă ar putea fi calculată în mod eficient probabilitatea ca o propoziție dată să fie generată de o gramatică dată. Totuși, studiile nu au continuat în această direcție deoarece, în procesul de analiză sintactică (parsing), suntem interesați în determinarea *celei mai probabile structuri sintactice* a unei propoziții și nu de probabilitatea globală a unei propoziții date. În particular, pentru exemplul dat, contează dacă interpretarea pentru structura grupului nominal este cea dată de regula 6 sau de regula 8. Reținem faptul că probabilitatea fiecărui constituent se calculează luând în considerație probabilitatea subconstituenților săi și probabilitatea regulii folosite (a se vedea și § 4.5.9). În mod concret, probabilitatea intrării E corespunzând unei categorii C se calculează, utilizând o regulă *i* cu *n* subconstituenți corespunzând intrărilor $E_1, \ldots, E_n$, după formula

$$P(E) = P(\textbf{Regula } i \mid C) * P(E_1) * \ldots * P(E_n)$$

S-a arătat [4] că, în cazul categoriilor lexicale, este recomandabil să se utilizeze așa-numitele **probabilități înainte** în locul probabilităților de generare lexicală. Probabilitatea înainte $\alpha_i(t)$ se definește ca fiind probabilitatea de a produce cuvintele $w_1, \ldots, w_t$ și de a încheia în starea w_t / L_i (stare în care cuvântul w_t aparține categoriei lexicale L_i):

$$\alpha_i(t) = P(w_t / L_i, w_1, \ldots, w_t)$$

Utilizând definiția probabilității condiționate se poate determina probabilitatea ca un cuvânt w_t să reprezinte o instanțiere a categoriei lexicale L_i după cum urmează:

$$P(w_t / L_i \mid w_1, \ldots, w_t) = P(w_t / L_i, w_1, \ldots, w_t) / P(w_1, \ldots, w_t).$$

Valoarea lui $P(w_1, \ldots, w_t)$ poate fi estimată ca fiind $\sum_{j=1}^{N} \alpha_j(t)$. Prin urmare, obținem:

$$P(w_t / L_i \mid w_1, \ldots, w_t) \cong \alpha_i(t) / \sum_{j=1}^{N} \alpha_j(t).$$

Utilizarea probabilității $\alpha_i(t)$ în locul probabilităților de generare lexicală, în cazul categoriilor lexicale, va produce estimații mai bune, întrucât ea oferă implicit explicații pentru o parte a *contextului* propoziției.

Să mai notăm și faptul că se pot utiliza, în egală măsură, așa-numitele **probabilități înapoi**, $\beta_i(t)$, prin care înțelegem probabilitatea de a produce secvența de cuvinte $w_t, \ldots, w_T$, începând din starea w_t / L_i. Aceste valori pot fi calculate în mod similar cu valorile $\alpha_i(t)$, dar începând procesul de calcul la sfârșitul propoziției și trecând în revistă stările în sensul "înapoi". Astfel, o metodă mai bună de a estima probabilitățile lexicale pentru cuvântul w_t pare a fi aceea de a lua în considerație întreaga propoziție în loc de o secvență de cuvinte până la t. În acest caz, estimația ar fi:

$$P(w_t / L_i) = (\alpha_i(t) * \beta_i(t)) / \sum_{j=1}^{N} (\alpha_j(t) * \beta_j(t)) \ .$$

Un algoritm de analiză sintactică care utilizează gramaticile independente de context stocastice va fi prezentat în § 4.5.9.

Din păcate, analizorii sintactici care au fost construiți utilizând aceste tehnici nu au lucrat atât de performant pe cât s-a așteptat. Performanța nu a putut fi suficient de ridicată întrucât presupunerile de independență care au fost făcute s-au dovedit a fi prea radicale. Unul dintre aspectele critice l-a constituit manevrarea unităților lexicale. Spre exemplu, modelul independent de context presupune că probabilitatea unui anumit verb care este utilizat în cadrul unei reguli referitoare la grupul verbal este independentă de care anume regulă se ia în considerație. Aceasta înseamnă că preferințele lexicale pentru anumite reguli nu pot fi modelate în acest cadru, ceea ce influențează și multe alte aspecte ale analizei. Toate acestea reprezintă un motiv în plus pentru care, în studiul procesului de parsing, nu ne vom opri în detaliu asupra analizei sintactice de tip stocastic.

3.4. Reprezentarea gramaticilor în Prolog

Caracterul regulilor utilizate de o gramatică variază considerabil în funcție de tipul acesteia, adică de teoria referitoare la gramatici care se implementează. Tipul de reguli cel mai frecvent folosit în procesarea limbajului natural și în lingvistica computațională rămâne însă cel al regulilor PS. În cele ce urmează, ne vom concentra atenția asupra reprezentării în Prolog a gramaticilor PS independente de context, care utilizează o mulțime finită de categorii gramaticale și o mulțime finită de reguli pentru a specifica modul în care

categoriile din membrul stâng pot fi realizate ca o succesiune de elemente ale membrului drept.

În scrierea unei gramatici, pe lângă specificarea *categoriilor gramaticale* și a *regulilor sau producțiilor*, este necesară și furnizarea unui *lexicon*, care să arate *modul de realizare a unor categorii gramaticale și, mai ales, lexicale.* Conform unei definiții minimale, un **lexicon** reprezintă o listă de cuvinte care asociază fiecărui cuvânt o serie de proprietăți sintactice și morfologice. Cea mai importantă dintre acestea este dată de categoria căreia îi aparține cuvântul. În plus, un lexicon mai poate conține informații referitoare la subcategoria unui cuvânt (dacă un verb, spre exemplu, este tranzitiv sau intranzitiv), ori alte informații, de natură morfologică (cum ar fi genul substantivelor în cazul limbilor care fac distincția de gen) sau de natură semantică. În cea mai simplă gramatică pe care o putem lua în considerație, utilizând notația de tip PATR, o *intrare lexicală* ar putea avea următoarea formă:

Cuvânt iubește:

<cat> = V.

Aceasta ne spune, pur și simplu, că "iubește" este un cuvânt a cărui categorie (lexicală) este aceea de verb. O intrare lexicală puțin mai complexă, corespunzătoare aceluiași cuvânt, ar putea fi:

Cuvânt iubește:

<cat>=V

<timp>=prezent

<arg1>=NP.

Aceasta ne spune, în plus, că verbul este la timpul prezent și că "iubește" reprezintă un verb tranzitiv (deoarece admite un complement direct, întrucât este urmat de un grup nominal).

Reprezentarea gramaticilor PS independente de context în Prolog se poate face într-un mod extrem de natural și de direct. Astfel, fiecărei reguli PS de tipul

$$S \rightarrow NP\ VP$$

îi va corespunde, în programul Prolog de implementare a gramaticii, clauza

```
s(L1,L):-np(L1,L2),vp(L2,L).
```

în care: L1 reprezintă șirul de intrare inițial (spre exemplu, șirul ['Colentina', angajează, surori]);

L2 reprezintă șirul de intrare fără grupul nominal inițial (în exemplul dat, [angajează, surori]);

L reprezintă șirul de intrare fără grupul nominal și grupul verbal (în acest caz, [ ]).

Această clauză Prolog ar mai putea fi descrisă astfel:

Înlăturarea (separarea, izolarea) unui S din L1 produce L
dacă
înlăturarea unui NP din L1 produce L2
şi
înlăturarea unui VP din L2 produce L.

Regulile care introduc simboluri terminale (cuvinte) vor fi de forma:

```
n([pacient|L],L).
```

O astfel de clauză Prolog s-ar putea traduce prin: "pentru a izola un substantiv, înlătură cuvântul *pacient* de la începutul şirului de intrare". Ea corespunde unei intrări lexicale de cea mai simplă formă cu putinţă, adică una care specifică numai categoria lexicală a cuvântului dat.

Menţionăm faptul că, o asemenea formă a clauzelor Prolog presupune ca propoziţiile să fie reprezentate sub forma *diferenţei de liste* sau, mai corect spus, a *listei diferenţă*. Aceasta este o reprezentare a listelor care doreşte să pună în evidenţă *sfârşitul* unei liste. O soluţie pentru a realiza acest lucru este reprezentarea unei liste printr-o *pereche de liste*. Spre exemplu, lista

```
[a,b,c]
```

poate fi reprezentată prin intermediul următoarelor două liste:

```
L1=[a,b,c,d,e]

L2=[d,e]
```

O asemenea pereche de liste, pe care o vom *nota* L1-L2, reprezintă "diferenţa" dintre listele L1 şi L2. Condiţia care trebuie satisfăcută pentru a utiliza o asemenea reprezentare şi notaţia aferentă este ca lista L2 să reprezinte un sufix al listei L1. Se observă că o aceeaşi listă poate fi reprezentată prin mai multe perechi de liste. Spre exemplu, aceeaşi listă `[a,b,c]` mai poate fi reprezentată în modurile următoare:

```
[a,b,c]-[]
```

sau

```
[a,b,c,d,e|T]-[d,e|T]
```

sau

```
[a,b,c|T]-T
```

unde T este orice listă.

Întrucât cel de-al doilea membru al perechii indică sfârșitul listei pe care o reprezentăm, acest sfârșit de listă poate fi accesat în mod direct. Această reprezentare poate fi, prin urmare, utilizată pentru o implementare eficientă a *concatenării*. Lista reprezentată prin perechea A1-Z1, concatenată cu lista reprezentată prin perechea Z1-Z2, produce o listă reprezentată prin perechea A1-Z2. Relația de concatenare corespunzătoare poate fi tradusă prin următoarea faptă Prolog:

```
concat(A1-Z1,Z1-Z2,A1-Z2).
```

În acest caz, pentru concatenarea listei `[a,b,c]` reprezentată prin perechea `[a,b,c|T1]-T1` cu lista `[d,e]` reprezentată prin perechea `[d,e|T2]-T2` se specifică scopul

```
?-concat([a,b,c|T1]-T1,[d,e|T2]-T2,L).
```

Concatenarea se face prin împerecherea acestui scop cu clauza **concat**, definită anterior, iar programul răspunde:

```
L=[a,b,c,d,e|T2]-T2
T1=[d,e|T2]
```

Așa cum se arată în [11], datorită eficienței sale, această tehnică este mult mai populară decât utilizarea lui **append** pentru realizarea concatenării.

Revenind la o regulă PS de tipul

$$S \rightarrow NP\ VP$$

căreia îi va corespunde, în implementarea Prolog a gramaticii, o clauză de forma

```
s(L1,L):-np(L1,L2),vp(L2,L).
```

vom observa că operația care stă la baza acestei clauze este aceea a *concatenării* unui șir de cuvinte ce formează un grup nominal și pe care îl privim ca pe o listă reprezentată prin perechea de liste L1-L2 cu un alt șir de cuvinte ce formează un grup verbal, privit ca o listă reprezentată prin perechea de liste L2-L. Rezultatul concatenării îl constituie propoziția dată de lista reprezentată prin perechea de liste L1-L.

Utilizând o reprezentare Prolog de tipul descris anterior (și care, la rândul ei, folosește reprezentarea propozițiilor sub forma listei diferență) vom indica, în continuare, un prim exemplu de gramatică reprezentată în Prolog:

```
s(X,Z):-np(X,Y),vp(Y,Z).
vp(X,Z):-v(X,Z).
vp(X,Z):-v(X,Y),np(Y,Z).
np(['Dr. Popescu'|X],X).
np(['Colentina'|X],X).
np([surori|X],X).
np([pacienti|X],X).
v([omoara|X],X).
v([angajeaza|X],X).
```

Gramatica nr.1

O astfel de gramatică **generează** propoziții. Ea poate fi însă utilizată și "în sens invers", adică pentru a **recunoaște** o propoziție dată. Un dispozitiv de recunoaștere ca cel programat mai sus decide dacă o propoziție dată aparține unui anumit limbaj. Cu alte cuvinte, recunoaște dacă propoziția poate fi generată de gramatica corespunzătoare.

Se spune că o gramatică *supragenerează* dacă ea legitimează șiruri de cuvinte care nu pot fi construite ca expresii gramaticale (corecte) ale limbajului studiat. Într-un context computațional, supragenerarea de către o gramatică nu reprezintă o problemă reală atunci când demersul este acela de a *recunoaște* sau înțelege limbajul corect format. Pe de altă parte, fenomenul de supragenerare s-ar putea dovedi fatal dacă scopul este acela de a *produce* șiruri corect formate.

Se spune că o gramatică *subgenerează* atunci când există șiruri de cuvinte ale limbajului studiat corect formate din punct de vedere sintactic și cărora gramatica nu le atribuie nici o structură. În context computațional, subgenerarea de către o gramatică nu reprezintă o problemă reală dacă scopul este acela de a *produce* limbaj adecvat. Pe de altă parte, fenomenul se poate dovedi fatal dacă țelul este acela de a *recunoaște* sau de a înțelege.

În cazul exemplului de gramatică prezentat (Gramatica nr.1), atunci când dorim să testăm, de pildă, dacă propoziția

Colentina angajează surori.

este una corect formată, adică dacă ea este *recunoscută* de către gramatică, vom interoga Prologul astfel:

```
?-s(['Colentina',angajeaza,surori],[]).
```

Programul va răspunde afirmativ. Pentru a utiliza aceeași gramatică în *generare*, interogarea se va face astfel:

```
?-s(Propozitie,[]).
```

Printre șirurile generate de gramatică se numără

```
Propozitie=['Dr. Popescu',omoara,pacienti]
```

```
Propozitie=['Dr. Popescu',angajeaza,surori]
```

dar și

```
Propozitie=[surori,omoara,'Colentina']
```

Dacă utilizăm Gramatica nr.1 în generare și cerem generarea tuturor șirurilor corect formate, vom obține 40 de propoziții, care sunt fie corecte, fie aproape corecte, atât din punct de vedere sintactic, cât și semantic, mai ales dacă se au în vedere diverse posibile sensuri figurate ale cuvintelor.

Un alt exemplu de gramatică, care utilizează regulile de rescriere

$$S \rightarrow NP\ VP$$

$$VP \rightarrow V\ NP$$

$$NP \rightarrow Det\ N$$

și dispune de intrări lexicale diferite este:

```
s(X,Z):-np(X,Y),vp(Y,Z).
vp(X,Z):-v(X,Y),np(Y,Z).
np(X,Z):-det(X,Y),n(Y,Z).
det([un|X],X).
det([niste|X],X).
det([o|X],X).
n([elev|X],X).
n([elevi|X],X).
n([carte|X],X).
v([iubeste|X],X).
v([iubesc|X],X).
```

Gramatica nr.2

Printre propozițiile generate de această gramatică se numără propoziția corectă

Un elev iubește o carte.

dar și

Un elev iubește un elevi.
Un elev iubește niște elev.
Un elev iubește un carte.
Un elev iubesc un carte.

și altele (atât corecte, cât și incorecte din punct de vedere gramatical).

3.4.1. Gramatici DC

Conversia regulilor PS în clauze Prolog este atât de simplă, încât ea se poate efectua și în mod automat. De altfel, Prologul include o facilitate pentru a realiza acest lucru, și anume o extindere a notației numită *notație DCG* (de la englezescul *definite-clause grammar*). Aceasta reprezintă o notație specială pentru regulile unei gramatici. Exemple de clauze scrise utilizând **notația DCG** sunt:

```
s --> np , vp.
np --> det , n.
n --> [elev].
```

Aceste clauze vor fi automat convertite, în faza de consultare, la:

```
s(X,Z):-np(X,Y),vp(Y,Z).
np(X,Z):-det(X,Y),n(Y,Z).
n([elev|X],X).
```

Sistemul DCG reprezintă un *compilator* pentru regulile gramaticii, pe care le traduce direct în clauze Prolog executabile.

Definiția 3.8

Prin **gramatici DC** vom înțelege acele gramatici ale căror reguli de rescriere sunt exprimate în notația DCG.

Așa cum s-a mai spus, o gramatică generează propoziții. Dar ea poate fi, în egală măsură, utilizată pentru a recunoaște o propoziție dată. În timpul unui asemenea proces de recunoaștere propoziția dată este practic descompusă în constituenții săi. Aceasta reprezintă tocmai procesul de analiză sintactică (sau parsing), la care ne-am mai referit. Pentru a implementa o gramatică este necesar să se scrie, de fapt, un program de analiză sintactică corespunzător acelei gramatici. O gramatică scrisă în DCG reprezintă deja un program de parsing pentru ea însăși.

Un program Prolog poate conține atât reguli DCG, cât și clauze Prolog obișnuite. Translatorul DCG afectează numai clauzele ce conțin functorul '-->'. Toate celelalte clauze sunt presupuse a fi clauze Prolog obișnuite și sunt lăsate neschimbate. Translatorul transformă regulile DCG în clauze Prolog prin adăugarea a două argumente suplimentare corespunzător fiecărui simbol care nu se află inclus între paranteze sau acolade. Argumentele sunt în mod automat aranjate astfel încât să poată fi corect utilizate în procesul de analiză sintactică.

O **regulă DCG** este de forma

Simbol neterminal --> extindere

unde *extindere* constă în unul dintre următoarele elemente:

- un simbol neterminal (ex.: np);
- o listă de simboluri terminale (ex.: [elev] sau [fiecare,elev]);
- un constituent vid reprezentat prin [ ];
- un scop Prolog inclus între acolade, cum ar fi `{write('Gasit NP')}`;
- o serie de oricare dintre aceste elemente, separate prin virgule.

Ca şi în Prolog, simbolul "punct şi virgulă" are semnificaţia lui "sau", ceea ce permite scrierea unor reguli DCG de forma

```
n-->[elev];[elevi];[carte].
```

În ceea ce priveşte sintaxa DCG, să mai notăm faptul că simbolurile neterminale nu se mai află între paranteze, în timp ce simbolurile terminale sunt incluse între paranteze drepte, ceea ce le transformă în liste Prolog. Simbolurile sunt separate prin virgule şi fiecare regulă se încheie prin punct, ca orice clauză Prolog.

Spre deosebire de notaţia PATR, care nu impune o ordine şi o aritate fixate, notaţia DCG presupune în mod esenţial ca numărul şi ordinea argumentelor să fie fixate. De asemenea, DCG permite imbricarea termenilor şi a variabilelor până la adâncimi arbitrare.

În cazul implementărilor Prolog care acceptă notaţia DCG, gramaticile transformate corespunzătoare pot fi imediat utilizate ca dispozitive de recunoaştere a propoziţiilor. Un asemenea dispozitiv de recunoaştere presupune însă, în mod esenţial, ca propoziţiile să fie reprezentate sub forma listei diferenţă de simboluri terminale. Acest fapt a fost deja justificat atunci când s-a arătat că transcrierea regulilor PS în Prolog se bazează pe reprezentarea operaţiei de concatenare, iar aceasta din urmă se realizează mult mai eficient utilizând lista diferenţă în detrimentul lui **append**. Luând în consideraţie această reprezentare a propoziţiilor, gramatica DC

```
s --> [a],[b].
s --> [a],s,[b].
```

poate fi utilizată în recunoaşterea unor propoziţii prin interogări de tipul:

```
?-s([a,a,b,b],[]).
yes
```

Prin operaţia de conversie pe care o efectuează, Prologul transformă automat regulile DCG în clauze Prolog obişnuite şi, în acest mod, converteşte regulile de rescriere date ale gramaticii într-un *program* de recunoaştere a propoziţiilor generate de gramatica respectivă.

Iată, spre exemplu, transcrierea în notaţia DCG a Gramaticii nr.2:

```
s-->np,vp.
vp-->v,np.
np-->det,n.
det-->[un].
det-->[niste].
det-->[o].
n-->[elev].
n-->[elevi].
n-->[carte].
v-->[iubeste].
v-->[iubesc].
```

Gramatica nr.3 (notație DCG)

Gramaticile DC *extind* gramaticile independente de context în următoarele moduri (cu referire la SICStus Prolog):

- Un simbol neterminal poate fi orice termen Prolog (altul decât o variabilă sau un număr).
- Un simbol terminal poate fi orice termen Prolog. Pentru a se face distincția între terminale și neterminale, un șir de unul sau mai multe simboluri terminale este scris, în cadrul unei reguli a gramaticii, ca o listă Prolog. Un șir vid este scris sub forma listei vide [ ]. Dacă simbolurile terminale sunt coduri de caractere, astfel de liste pot fi scrise (ca și în alte situații) ca șiruri de caractere (tipul string). Un șir vid este scris sub forma listei vide, [ ] sau " ".
- Se pot include în membrul drept al unei reguli a gramaticii condiții suplimentare, sub forma unor apelări Prolog de proceduri. Astfel de apelări de proceduri sunt incluse între acolade { }.
- Membrul stâng al unei reguli a gramaticii constă dintr-un neterminal urmat, în mod opțional, de un șir de terminale (scrise ca listă Prolog).
- Disjuncția, if-then, if-then-else și nedemonstrabilul pot fi declarate în mod explicit în membrul drept al unei reguli a gramaticii, prin folosirea operatorilor ; (|), -> și respectiv \+, ca și în cazul unei clauze Prolog.
- Simbolul **cut** poate fi inclus în membrul drept al unei reguli a gramaticii, ca și în cadrul unei clauze Prolog. Simbolul **cut** nu trebuie inclus între acolade { }.

Pentru a rezuma, vom formula în mod mai general felul în care se face *traducerea între DCG și Prologul standard*. Astfel, fiecare *regulă DCG* este tradusă într-o *clauză Prolog* conform următoarei scheme generale:

- Dacă regula DCG conține în membrul drept numai simboluri neterminale, adică este de forma

```
n-->n1,n2,...,nn.
```

cu n1, n2,...,nn simboluri neterminale, atunci ea este tradusă în clauza Prolog

```
n(X,Y):-n1(X,Y1),n2(Y1,Y2),...,nn(Yn-1,Y).
```

- Dacă regula DCG conține în membrul drept atât simboluri neterminale, cât și terminale, adică este de forma

```
n-->n1,[t2],n3,[t4].
```

cu n1, n3 simboluri neterminale și t2, t4 simboluri terminale, atunci ea este tradusă în clauza Prolog

```
n(X,Y):-n1(X,[t2|Y1]),n3(Y1,[t4|Y]).
```

(Se observă că simbolurile terminale sunt tratate în mod diferit, în sensul că un simbol terminal nu apare ca un scop al clauzei, ci este inserat direct în lista corespunzătoare).

Pentru a vedea una dintre problemele frecvente care se pot ridica *după traducerea regulilor DCG în clauze Prolog*, să luăm, din nou, în considerație Gramatica nr.1, ușor modificată și scrisă utilizând notația DCG:

```
s-->np,vp.
vp-->v.
vp-->v,np.
np-->['Dr. Popescu'].
np-->['Colentina'].
np-->[surori].
np-->[pacienti].
v-->[omoara].
v-->[angajase].
```

Gramatica nr.4

În timp ce Gramatica nr.1 genera 40 de propoziții, mai mult sau mai puțin corecte din punct de vedere sintactic, datorită schimbării timpurilor verbale, Gramatica nr.4 va genera, pe lângă propoziții corecte, și propoziții cum ar fi:

surori angajase

(în loc de *angajaseră*)

surori omoară pacienți

(în loc de *omorau*)

Dezacordul dintre subiect şi predicat se datorează existenţei regulii

```
s-->np,vp.
```

care afirmă că, practic, *orice* grup nominal şi *orice* grup verbal pot fi concatenate pentru a forma o propoziţie. Dar în limba română ca, de altfel, în majoritatea limbilor, grupul verbal şi grupul nominal nu sunt independente, ci trebuie să concorde în număr. Acesta este un exemplu de *dependenţă de context* - i.e. dependenţa grupului sintactic de contextul în care el intervine. Astfel de fenomene pot fi uşor tratate de către gramaticile DC, întrucât acestea admit adăugarea de *argumente* simbolurilor neterminale ale gramaticii. În exemplul anterior, prin adăugarea argumentului `Numar`, prima regulă DCG devine:

```
s(Numar)-->np(Numar),vp(Numar).
```

şi ea asigură concordanţa de număr între grupul nominal şi grupul verbal care se concatenează pentru a forma întreaga propoziţie.

Atunci când regulile DCG sunt citite de către Prolog şi convertite în clauze Prolog, argumentele simbolurilor neterminale sunt, pur şi simplu, adăugate argumentelor (de tip listă) obişnuite, respectându-se convenţia că cele două liste apar ultimele. Astfel, regula DCG

```
s(Numar)-->np(Numar),vp(Numar).
```

este convertită în următoarea clauză Prolog:

```
s(Numar,X,Y):-np(Numar,X,Y1),vp(Numar,Y1,Y).
```

Interogarea Prologului se modifică în mod corespunzător, pentru a fi incluse noile argumente. Spre exemplu, pentru a face ca Gramatica nr.4, la care s-a adăugat *caracteristica de număr* prin introducerea parametrului `Numar`, să genereze toate propoziţiile posibile, interogarea se va face în felul următor:

```
?-s(Numar,Propozitie,[]).
```

Gramatica nr.5 modifică, în acest sens, Gramatica nr.3 prin adăugarea atât a unei caracteristici de număr, cât şi a uneia de gen.

Pentru a vedea toate propoziţiile care pot fi generate de Gramatica nr.5 se interoghează Prologul în acelaşi mod

```
?-s(Numar,Propozitie,[]).
```

şi se obţine răspunsul:

```
Numar=sing,
Propozitie=[un,elev,iubeste,un,elev]

Numar=sing
Propozitie=[un,elev,iubeste,o,carte]
```

```
Numar=plural
Propozitie=[niste,elevi,iubesc,niste,elevi]

Numar=sing
Propozitie=[o,carte,iubeste,un,elev]

Numar=sing
Propozitie=[o,carte,iubeste,o,carte]

no
```

În total există 5 soluții, toate corecte din punct de vedere sintactic (dar nu și din punct de vedere semantic). Întrucât, pentru moment, nu suntem interesați și de aspectele semantice, vom spune că gramatica generează numai propoziții corecte.

```
s(Numar)-->np(Numar),vp(Numar).
vp(Numar)-->v(Numar),np(Numar).
np(Numar)-->det(Numar,Gen),n(Numar,Gen).
det(sing,masc)-->[un].
det(plural,masc)-->[niste].
det(sing,fem)-->[o].
n(sing,masc)-->[elev].
n(plural,masc)-->[elevi].
n(sing,fem)-->[carte].
v(sing)-->[iubeste].
v(plural)-->[iubesc].
```

Gramatica nr.5

Gramatica nr.5 este chiar prea restrictivă, întrucât ea nu generează, spre exemplu, propoziția corectă

Niște elevi iubesc o carte.

Pentru a vedea dacă această propoziție este generată de gramatica dată, interogarea Prologului se face astfel

```
?-s(plural,[niste,elevi,iubesc,o,carte],[]).
```

iar răspunsul este

```
no
```

Gramatica generează, în schimb, aceeași propoziție la singular:

```
?-s(sing,[un,elev,iubeste,o,carte],[]).

yes
```

Neajunsul menționat este remediat de Gramatica nr.6. Interogarea

```
?-s(Numar,Propozitie,[]).
```

primește ca răspuns 9 soluții ale căror propoziții sunt toate corecte din punct de vedere sintactic (dar mai puțin din punct de vedere semantic). Printre propozițiile generate se numără și propoziția pe care Gramatica nr.5 nu o recunoștea:

Niște elevi iubesc o carte.

```
s(Numar)-->np(Numar),vp(Numar).
vp(Numar)-->v(Numar),np(Numar1).
np(Numar)-->det(Numar,Gen),n(Numar,Gen).
det(sing,masc)-->[un].
det(plural,masc)-->[niste].
det(sing,fem)-->[o].
n(sing,masc)-->[elev].
n(plural,masc)-->[elevi].
n(sing,fem)-->[carte].
v(sing)-->[iubeste].
v(plural)-->[iubesc].
```

Gramatica nr.6

Schimbarea efectuată, care determină recunoașterea acestei propoziții de către Gramatica nr.6, constă în utilizarea parametrului `Numar1` în cea de-a doua regulă DCG, în locul aceluiași parametru `Numar`. Această schimbare de parametru nu mai impune concordanța de număr și în *interiorul* grupului verbal, rămânând în vigoare doar concordanța de număr între grupul nominal și grupul verbal care, prin concatenare, alcătuiesc întreaga propoziție.

3.4.2. Gramatici cu caracteristici și reprezentarea lor în notația DCG

Majoritatea analizorilor sintactici se bazează pe gramatici independente de context. În mod evident însă acestea nu pot cuprinde toată bogăția unui limbaj natural. De aceea se obișnuiește să se utilizeze în procesare o extensie a acestor gramatici, realizată prin asocierea unei mulțimi de caracteristici. Astfel de gramatici augmentate sunt deosebit de utile în modelarea cu precădere a fenomenelor specifice limbajului natural.

Gramatica nr.5 și Gramatica nr.6 reprezintă niște **gramatici cu caracteristici,** întrucât ele folosesc caracteristicile de număr și de gen. Sintaxa teoretică modernă se bazează în mod esențial pe utilizarea caracteristicilor, întregi categorii atomice cum ar fi NP sau V fiind înlocuite prin mulțimi de specificații ale unor caracteristici. Vom reveni pe larg asupra acestui subiect în capitolul 5 al lucrării de față. Pentru moment, reținem:

Definiția 3.9

O **specificație a unei caracteristici** constă dintr-o **caracteristică** (cum ar fi *cazul*) și o **valoare** pentru acea caracteristică (cum ar fi *acuzativ*).

Există teorii sintactice bazate pe caracteristici care reintroduc denumirile cunoscute ale categoriilor, cum ar fi NP sau V, ca **valori** ale unei anumite caracteristici pe care o numesc **cat**. (Această caracteristică a fost deja utilizată de noi în definirea intrărilor lexicale). Întrucât, în felul acesta, se poate face trecerea de la categoriile atomice (de tip S, NP, V etc.) la mulțimi de informații exprimate prin caracteristici, se pune problema exprimării regulilor de rescriere ale gramaticilor cu caracteristici. *În mod formal*, putem exprima o regulă de tipul

$$S \rightarrow NP\ VP$$

sub forma:

Regulă {formarea propozițiilor}

$X0 \rightarrow X1X2$:

<X0 cat> = S

<X1 cat> = NP

<X2 cat> = VP.

Într-o notație de acest tip, regula în sine conține niște *marcaje* (X0,X1 și X2) însoțite de o mulțime de ecuații precum <X0 cat>=S, ecuație care trebuie interpretată ca stipulând: "valoarea caracteristicii de categorie a lui X0 este S".

Formalismul DCG este extrem de eficient în formularea gramaticilor cu caracteristici. Acest formalism permite atribuirea de termeni și/sau de variabile logice, în număr nelimitat, ca argumente ale categoriilor, o facilitate inexistentă în contextul formalismului definit de gramaticile PS. Astfel, următoarei *reguli a unei gramatici cu caracteristici* (exprimate într-o *notație de tip PATR*)

Regulă

$S \rightarrow NP\ VP$:

<NP persoană>=<VP persoană>

<NP număr>=<VP număr>

<NP caz>=nominativ

<VP vforma>=noninfinitival.

îi corespunde *regula DCG*

```
s-->np(Persoana,Numar,nominativ),
    vp(Persoana,Numar,noninfinitival).
```

O *intrare lexicală* într-o astfel de gramatică ar putea fi de forma:

```
np(3,singular,_)-->[anca].
```

În acest caz, gramatica următoare

```
s-->np(Persoana,Numar,nominativ),
    vp(Persoana,Numar,noninfinitival).
np(3,singular,_)-->[anca].
vp(3,singular,noninfinitival)-->[mananca].
```

generează unica propoziție

anca mănâncă

Să mai notăm faptul că regulile DCG pot avea forme extrem de complexe, întrucât acest formalism permite imbricarea termenilor până la niveluri arbitrare.

Așa cum am mai remarcat, caracteristica principală a formalismului DCG este aceea că el îmbogățește regulile gramaticilor PS independente de context cu *argumente și variabile*. Printre cele mai importante utilizări ale argumentelor introduse se numără *specificarea acordului între subiect și predicat*. Cu ajutorul caracteristicilor se poate semnala acordul în număr între verb (cu valoare predicativă) și subiectul său:

***Copiii** își **fac** lecțiile.*

***Copilul** își **face** lecțiile.*

Se observă că, atât în limba engleză, cât și în limba română, obiectul verbului (complementul direct) nu se supune aceleiași reguli de acord în număr. Astfel, este corectă atât propoziția

Copiii își fac **lecțiile**.

cât și propoziția

Copiii își fac ***lecția***.

Pentru a face ca regulile DCG să ia în considerație și să transcrie acordul, se va introduce caracteristica `Numar` având valorile `singular` și `plural` și se vor adăuga argumente atât categoriei NP, cât și categoriei VP, pentru a indica numărul. Un grup nominal (NP) are același număr (singular sau plural) ca și substantivul în jurul căruia se organizează, în timp ce un grup verbal (VP) are numărul verbului corespunzător. Iată transcrierea unora dintre regulile DCG

deja utilizate, care respectă regula acordului, prin intermediul caracteristicii de număr:

```
np(Numar)-->det,n(Numar).
vp(Numar)-->v(Numar),np(_).
```

Grupul nominal şi grupul verbal prin concatenarea cărora se formează propoziţia trebuie să aibă acelaşi număr:

```
s-->np(Numar),vp(Numar).
```

Un analizor sintactic (parser) care utilizează aceste reguli va accepta numai acele propoziţii în care verbul (cu valoare predicativă) şi subiectul său respectă acordul în număr. La rândul lor, intrările lexicale care vor fi utilizate trebuie să specifice numărul fiecărui substantiv şi respectiv verb, ca în exemplele:

```
n(singular)-->[copil];[elev];[profesor];[lectia].
n(plural)-->[copii];[elevi];[profesori];[lectiile].
v(singular)-->[face];[asista];[citeste];[scrie].
v(plural)-->[fac];[asista];[citesc];[scriu].
```

Structura grupului verbal, atât în limba română, cât şi în limba engleză, depinde de verbul în jurul căruia se organizează acest grup sintactic. În cazul verbelor *tranzitive*, spre exemplu, adică a acelor verbe care (în sintaxa tradiţională, analitică) acceptă un complement direct, în funcţie de forma (lărgită) a complementului se pot determina structuri diferite ale grupului verbal, care generează reguli PS diferite, după cum urmează:

VERB	COMPLEMENT	EXEMPLU
a lucra	-	Eu lucram.
a mânca	-	Eu mâncam.
a lucra	un NP	Eu lucram *pământul de la ţară*.
a mânca	un NP	Eu mănânc *o prăjitură gustoasă*.
a lucra	două NP	El lucrează **vecinilor săi** *pământul de la ţară*.
a vinde	două NP	Negustorul vinde **mărfuri de calitate** *clienţilor săi*.
a lucra	propoziţie	El lucrează *ce ţi-a promis*.
a afirma	propoziţie	El afirmă *că soluţia problemei nu este corectă*.
a lucra	NP propoziţie	El lucrează **pentru vecinii săi** *ce le-a promis*.
a trimite	NP propoziţie	El trimite **prietenilor săi** *ceea ce este necesar*.

Acest tabel ne sugerează pentru structura grupului verbal o regulă PS de forma

$$VP \rightarrow V(NP)\ (NP)\ (S)$$

Verbul *a lucra*, spre exemplu, este un verb tranzitiv care se poate afla în oricare dintre situațiile prevăzute în cadrul tabelului anterior. Există însă și verbe tranzitive care nu se pot afla în toate aceste situații și care impun să fie urmate numai de anumite structuri. Verbul *a trăi*, de pildă, nu poate apărea ca verb tranzitiv decât într-un context de tipul

El trăiește clipe fericite.

deci nu poate fi urmat decât de un unic grup nominal. (Verbul a trăi intervine în alte contexte ca verb intranzitiv). Aceasta arată că, pentru a stabili *structura grupului verbal* nu este suficient să invocăm o regulă de forma

$$VP \rightarrow V(NP)\ (NP)\ (S)$$

Pentru aceasta sunt necesare cel puțin următoarele cinci reguli,

$$VP \rightarrow V$$
$$VP \rightarrow V\ NP$$
$$VP \rightarrow V\ NP\ NP$$
$$VP \rightarrow V\ S$$
$$VP \rightarrow V\ NP\ S$$

având în vedere că am studiat numai subcategoria verbelor tranzitive, plus stabilirea unei modalități de a asocia fiecărui verb regula corectă.

O posibilitate de a rezolva acest lucru ar fi aceea de a elimina conceptul de "verb" din gramatică și de a utiliza, în locul acestuia, cinci categorii diferite, V1, V2, V3, V4 și V5, după cum urmează:

$$VP \rightarrow V1$$
$$VP \rightarrow V2\ NP$$
$$VP \rightarrow V3\ NP\ NP$$
$$VP \rightarrow V4\ S$$
$$VP \rightarrow V5\ NP\ S$$

O abordare de acest tip nu ne poate satisface, întrucât ea sugerează faptul că nu există nici o legătură între V1, V2, V3, V4, V5, care pot aparține unor categorii gramaticale sau lexicale total diferite. Este, în mod evident, necesar să putem trata diferitele tipuri de verbe în mod similar, dar, în același timp, în mod diferit. Este, prin urmare, necesară existența unei categorii unice a verbelor (V), despărțită în subcategorii. O modalitate de a realiza acest lucru este aceea de a adăuga lui V un argument. Regulile care trebuie să ia în considerație această caracteristică îi pot specifica valoarea, în timp ce regulile care nu au nevoie să o

ia în considerație pot utiliza o variabilă anonimă în locul ei. Iată regulile DCG corespunzătoare, precum și câteva intrări lexicale ale gramaticii:

```
vp-->v(1).
vp-->v(2),np.
vp-->v(3),np,np.
vp-->v(4),s.

v(1)-->[manca].
v(2)-->[traia].
v(3)-->[vindea].
v(4)-->[afirma].
v(5)-->[trimitea].
```

Cea de-a treia intrare lexicală, spre exemplu, ne spune că forma *vindea* a verbului *a vinde* trebuie să fie urmată de două grupuri nominale. Un verb aparținând oricăreia dintre clasele reprezentând subcategoriile posibile va fi desemnat prin `v(_)`. În scrierea fragmentului de gramatică anterior am ignorat problema acordului, renunțând la caracteristica de număr, dar aceasta poate fi oricând introdusă.

Reprezentarea gramaticilor cu caracteristici în DCG se bazează pe o proprietate fundamentală a acestor gramatici: aceea că există o caracteristică (pe care am numit-o **cat**) a cărei valoare este atomică și întotdeauna furnizată în cadrul oricărei descrieri a unei categorii date a gramaticii. Această proprietate este necesară deoarece am utilizat, în permanență, valoarea lui **cat** în rolul predicatului Prolog. Iar în Prolog este imposibilă specificarea unui scop al cărui predicat este necunoscut (o variabilă).

În concluzie, putem spune că **formalismul DCG**, care îmbogățește regulile gramaticilor PS independente de context cu argumente și variabile, este extrem de util și că gramaticile DC reprezintă o extensie relativ naturală a gramaticilor PS independente de context, în cadrul oferit de limbajul Prolog. Așa cum s-a văzut, caracteristica de bază a unei reguli independente de context este faptul că ea nu impune nici un fel de restricții asupra contextului în care membrul său stâng se poate realiza sub forma membrului său drept. Ne-am concentrat, cu precădere, asupra formalismului oferit de gramaticile independente de context, întrucât, așa cum s-a menționat deja, acesta este suficient de puternic pentru a descrie majoritatea structurilor care apar în limbile naturale. Gramaticile DC pot lua însă în considerație și *contextul,* prin înregistrarea de informații în argumente suplimentare ale predicatelor. În acest mod, ele sunt capabile să descrie limbaje care nu pot fi descrise prin intermediul gramaticilor PS independente de context. Pentru un comentariu asupra modului în care se face

scrierea gramaticilor DC corespunzător limbajelor care nu sunt independente de context, vezi [29].

Reprezentarea gramaticilor în Prolog s-a referit cu precădere la gramaticile PS independente de context, întrucât majoritatea algoritmilor de analiză sintactică existenți utilizează acest tip de gramatici (a se vedea capitolul 4). În continuare, vom prezenta două clase fundamental diferite de gramatici (gramatici de dependență, gramatici contextuale), care nu se bazează pe constituenți și care nu privesc structura propoziției ca fiind alcătuită din constituenți în accepția folosită până în prezent.

3.5. Gramatici de dependență și gramatici WG

În ciuda diverselor teorii lingvistice existente, care au ca rezultat modalități total diferite de a privi structura propoziției și, prin urmare, procesul de analiză sintactică, majoritatea lingviștilor sunt astăzi de acord cu faptul că în centrul conceptului de **structură a propoziției** se află **relațiile dintre cuvinte**. Aceste relații pot fi de diverse naturi, referindu-se ori la funcții gramaticale (subiect, complement etc.), ori la acele legături care îmbină cuvintele în unități mai largi, cum ar fi grupurile sintactice sau chiar propozițiile în ansamblul lor.

Spre deosebire de gramaticile generative, **gramaticile de dependență** (Mel'čuk 1962, 1963, 1964 până la Mel'čuk 1979, 1987 și 1988), precum și **gramaticile WG** (abreviere de la *"word grammar"*, Hudson 1984, 1990 și 1998) **nu** se bazează pe noțiunea de constituent ci pe **relațiile directe existente între cuvinte**, privite fiind ca niște relații de dependență. Gramaticile WG reprezintă o variantă a gramaticilor de dependență și, în același timp, o teorie care a fost cu succes adaptată și aplicată specificului limbii române [33], [34], așa cum se va vedea și în cele ce urmează (§3.5.2, §3.5.3). Caracteristica distinctivă principală a gramaticilor WG o constituie utilizarea de către acestea a dependențelor în defavoarea structurii bazate pe constituenți.

Teoria gramaticilor de dependență necesită punerea în evidență a unei *structuri de dependență*, care poate fi privită, printre altele, ca opunându-se structurii alcătuite din constituenți[5]. Așa cum caracterizează Richard Hudson

[5] Gramaticile de dependență dau naștere, în ultimă instanță, unui formalism pentru descrierea la nivel sintactic a propozițiilor aparținând limbajului natural. Popularitatea dependențelor, ca mijloc formal de reprezentare a structurii sintactice a propozițiilor, a fost mereu în creștere și a culminat cu opera lui Lucien Tesnière (Tesnière 1959). Cu toate acestea, în America de Nord, la începutul anilor '30, "sintaxa de dependență" a fost eclipsată de ceea ce s-a numit, la acea vreme, "sintaxa constituenților imediați" (sau "analiza de tip IC" - de la "immediate constituency"). Aceasta s-a transformat mai târziu

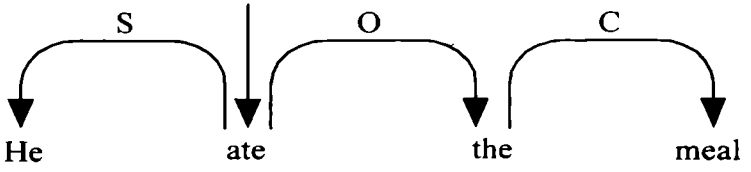
aceste structuri, în timp ce structura dată prin constituenți a unei propoziții "constă din relații de la parte la întreg între cuvinte și grupurile sintactice", structura de dependență a unei propoziții "constă din relații de la cuvânt la cuvânt între cuvinte individuale". Ideea centrală pe care se bazează noțiunea de dependență este aceea că fiecare cuvânt este privit ca depinzând de cuvântul care îl leagă de restul propoziției, practic explicând de ce este utilizat. Acesta din urmă este numit *capul* celuilalt. Structura sintactică a propoziției este reprezentată prin relații sintactice binare între cuvinte. În cadrul fiecărei perechi de cuvinte aflate într-o relație de acest tip, unul dintre cuvinte (cel *dependent*) depinde de celălalt (*capul*), care îl susține atât din punct de vedere sintactic, cât și semantic. În același timp, *valența* unui cuvânt este considerată a fi mulțimea dependențelor în care acesta poate să fie implicat: subiectul său, complementele sale etc.

La baza gramaticilor de dependență se află relația dintre cuvântul cap și cuvântul dependent. **Analiza sintactică a unei propoziții** înseamnă, din punctul de vedere al gramaticilor de dependență, descrierea tuturor relațiilor de dependență care intervin între toate cuvintele unei propoziții. O caracteristică importantă a acestui tip de analiză sintactică este aceea că ea *tratează grupurile sintactice ca reprezentând produse secundare ale dependențelor stabilite*. O analiză sintactică de acest tip a unei propoziții poate fi descrisă prin intermediul unei diagrame relativ simple care pune în evidență unitățile componente (fiecare cuvânt în parte) și relațiile ce se stabilesc între ele. Pe scurt, teoria generală consideră toate unitățile sintaxei ca fiind cuvinte individuale.

Întrucât analiza sintactică bazată pe constituenți, adică utilizând gramatici PS, cu care încercăm să facem o comparație, este tipică limbii engleze, să considerăm analiza sintactică de tip WG (care utilizează gramatici WG sau gramatici de dependență) a următoarei propoziții englezești:

în "analiză PS", care determină PS-structura unei propoziții. Formulată în mod riguros de Leonard Bloomfield (Bloomfield 1933), dar și de către Wells în 1947 și Percival în 1976, reprezentarea de tip PS în sintaxă a fost promovată cu multă energie de școala structuralistă în anii '30, '40 și '50. Ea a devenit unica reprezentare sintactică luată în mod serios în considerație de către Noam Chomsky și școala generativ-transformațională pe care acesta a fondat-o la sfârșitul anilor '50.

Principiul de lucru pe care se bazează PS-sintaxa este concentrarea atenției în jurul constituenților și al categoriilor acestora, ceea ce conduce spre taxonomie, adică spre clasificare și distribuție. Sintaxa de dependență sau D-sintaxa se bazează, la rândul ei, pe relații existente între unități sintactice finale și, prin urmare, tinde să fie preocupată de legături cărora li se conferă sens, adică de semantică.



Propoziția (*El a luat masa.*) conține patru cuvinte, astfel încât diagrama corespunzătoare ia în considerație patru unități și pune în evidență relațiile care se stabilesc între ele. Relațiile sunt reprezentate prin săgeți, fiecare dintre acestea ținând spre cuvântul care are funcția gramaticală indicată de către etichetă ("s" pentru subiect, "o" pentru obiect și "c" pentru complement, altul decât cel direct). Etichetele indică, în același timp, tipul relației. Săgeata verticală indică unicul cuvânt care nu depinde de nici un altul, atât din punct de vedere sintactic, cât și semantic (verbul "ate"). Faptul că "the meal" constituie un grup sintactic (alcătuit dintr-un determinant și un substantiv) este indicat prin săgeata care leagă cuvântul "the" direct de cuvântul "meal".

Se remarcă faptul că, într-o analiză sintactică bazată pe constituenți, grupul sintactic "the meal" ar reprezenta o realizare a regulii de rescriere

$$NP \rightarrow Det\ N$$

și, prin urmare, ar reprezenta un grup nominal organizat în jurul substantivului "meal" cu rol de cap sau centru al grupului sintactic. Într-o analiză sintactică bazată pe gramatici de dependență, în schimb, cuvântul "the" reprezintă capul, în timp ce substantivul "meal" este cuvântul dependent.

În efectuarea unei **analize sintactice de dependență (sau de tip WG)** prin intermediul unei diagrame de acest tip, trebuie avut permanent în vedere faptul că funcția gramaticală este întotdeauna funcția cuvântului dependent în relația sa cu capul, astfel încât săgeata ține întotdeauna înspre cuvântul dependent. Așa cum s-a menționat deja, ideea care stă la baza acestei diagrame este aceea că fiecare cuvânt depinde de cuvântul care îl leagă de restul propoziției, explicând în mod implicit de ce el este utilizat. Conceptul central pe care se bazează gramaticile de dependență, cât și cele de tip WG este acela al *relațiilor de dependență existente între cuvinte individuale*. În acest cadru, **analiza sintactică** înseamnă descrierea structurii unei întregi propoziții ca fiind o mulțime de relații de dependență existente între perechi de cuvinte individuale ale acelei propoziții. Paragraful următor va oferi o definiție mai riguroasă a *structurii sintactice a unei propoziții*, prin care vom înțelege o structură (S,D), unde S va desemna *structura de dependență*, iar D va desemna *tipul dependențelor*.

O întrebare care se ridică în mod natural este cea care încearcă să elucideze de ce și când ar putea fi preferat acest formalism pentru descrierea structurii sintactice a propozițiilor limbajului natural. Dintre numeroasele avantaje oferite

de formalismul generat de gramaticile de dependență amintim aici doar faptul că acesta poate descrie fenomene lingvistice cum ar fi *existența constituenților discontinui* sau *variația ordinii cuvintelor* în cadrul propoziției. Astfel, în timp ce limbi cum ar fi rusa sau latina, precum și majoritatea limbilor romanice, prezintă o ordine a cuvintelor extrem de flexibilă (chiar dacă nu arbitrară), care însă poate fi surprinsă și modelată cu ajutorul acestui formalism, limba engleză se caracterizează printr-o ordine a cuvintelor mai degrabă rigidă. Nu este, prin urmare, întâmplător faptul că analiza sintactică de tip PS își are originile în Statele Unite și că ea s-a dezvoltat cu specială referire la limba engleză.

3.5.1. Relații de dependență

Atunci când nu sunt specificate nici un fel de restricții, între cuvintele unei propoziții poate exista o întreagă varietate de relații de dependență. Rolul gramaticilor de dependență este în primul rând acela de a specifica restricțiile pe care relațiile de dependență trebuie să le satisfacă astfel încât structura pe care ele o definesc să fie corectă din punct de vedere lingvistic.

Indiferent de limba la care se referă, orice gramatică de dependență trebuie să ia în considerație următoarele trei *principii generale*, motivate din punct de vedere lingvistic:

- orice cuvânt trebuie să depindă de un singur alt cuvânt (capul), cu excepția predicatului propoziției, care nu depinde de nici un alt cuvânt;
- mai multe cuvinte pot depinde de același cap;
- dacă relațiile dintre cuvântul dependent și cap sunt reprezentate prin arce orientate de la cap spre dependent, atunci aceste arce nu trebuie să se intersecteze, iar graful orientat astfel format trebuie să fie aciclic.

Dincolo de respectarea acestor principii generale, o gramatică de dependență mai poate, de asemenea, specifica ce relații sunt admise între diverse cuvinte, conform părții de vorbire pe care acestea o reprezintă sau conform altor criterii. Spre exemplu, o gramatică de dependență ar putea stipula faptul că un verb nu poate depinde de un substantiv sau ar putea indica de care părți de vorbire poate depinde un adjectiv etc.

Relațiile de dependență care definesc structura unei propoziții pot fi descrise prin intermediul *structurii de dependență* și al *tipului* dependențelor. *Structura de dependență* va specifica, în cazul fiecărui cuvânt, care sunt celelalte cuvinte de care acesta depinde. *Tipul dependențelor* va specifica, în cazul fiecărei dependențe, tipul acesteia.

Din punct de vedere formal, relațiile de dependență pot fi descrise după cum urmează:

Fie W o mulțime finită de *cuvinte* (vocabularul) cu ajutorul cărora se formează propozițiile. O propoziție va fi desemnată prin secvența de cuvinte

$$w_1, w_2, \ldots, w_n$$

Vom nota prin w_0 un cuvânt special numit *BOS* (de la "beginning of sentence"), care denotă începutul propoziției și prin w_{n+1} cuvântul special *EOS* (de la "end of sentence"), care denotă sfârșitul propoziției.

Fie T o mulțime finită de *etichete* numite părți de vorbire. Acestea vor desemna părțile de vorbire (substantiv, verb, adjectiv etc.) cărora le pot aparține cuvintele vocabularului W.

Fie D o mulțime finită de *tipuri ale dependențelor* (subiect, atribut, determinant etc.).

Structura sintactică a unei propoziții

$$w_1, w_2, \ldots, w_n$$

va fi o structură (S,D), unde prin S vom desemna *structura de dependență*, iar prin D vom desemna *tipul dependențelor*.

Structura de dependență S este, la rândul ei, o structură (T,P), unde $T = t_1, t_2, \ldots, t_n; t_i \in T \quad \forall 1 \le i \le n$. T este o secvență de etichete t_i ce desemnează părțile de vorbire cărora le aparține fiecare cuvânt w_i. În cadrul aceleiași structuri de dependență, $P = p_1, p_2, \ldots, p_n; \quad p_i \in \{1, 2, \ldots, n+1\}$ $\forall 1 \le i \le n$ este o secvență de numere care specifică, corespunzător fiecărui cuvânt w_i, capul său (cuvântul de care acesta depinde). Capul sau părintele cuvântului w_i va fi cuvântul w_{p_i}. Întrucât există un cuvânt w_h care nu depinde de nici un altul, se va presupune că părintele acestui cuvânt este $w_{n+1} = EOS$ și că, prin urmare, $p_h = n+1$.

Tipul dependențelor D este o funcție $D: \{1, 2, \ldots, n\} \to$ D, unde $D(i) = d$ reprezintă tipul dependenței dintre cuvintele w_i și w_{p_i}.

Principiile generale de care relațiile de dependență trebuie să țină cont și pe care trebuie să le îndeplinească pot fi "traduse", conform acestui formalism, după cum urmează: pentru orice propoziție

$$w_1, w_2, \ldots, w_n$$

- $\exists h, 1 \le h \le n$ a.î. $p_h = n+1$ și $\forall i, 1 \le i \le n, i \ne h$, $p_i \in \{1, 2, \ldots, n\}$ și $p_i \ne i$.
- $\forall 1 \le i < j \le n$

  - dacă $p_i < i$, atunci $p_j \le p_i$ sau $p_j \ge i$;

- dacă $i < p_i \leq j$, atunci $p_j \leq i$ sau $p_j \geq p_i$;
- dacă $p_i > j$, atunci $i \leq p_j \leq p_i$.

(Condițiile mai sus amintite reprezintă exprimarea formală a faptului că arcele - sau săgețile - definite de relațiile de dependență nu trebuie să se intersecteze).

- Structura de dependență *(T,P)* definește un graf *G=(V,E)*, unde $V = \{1, 2, \ldots, n, n+1\}$ și $E = \{(i, p_i) \mid 1 \leq i \leq n\}$. Acest graf trebuie să fie aciclic.

Spre exemplu, atunci când se ia în considerație mulțimea de etichete T=*{NN, VBD, DT, IN}* și mulțimea de tipuri ale dependențelor D={sub, atr, det, pred}, *relațiile de dependență* pentru propoziția

i	1	2	3	4	5	6	7
w_i	*the*	*price*	*of*	*the*	*stock*	*fell*	*EOS*

sunt descrise de structura *(S,D)*; *S=(T,P)*, unde

T = DT, NN, IN, DT, NN, VBD
$P = 2, 6, 2, 5, 3, 7$
$D(1)$ = det, $D(2)$ = sub, $D(3)$ = atr, $D(4)$ = det, $D(5)$ = det, $D(6)$ = pred

sau, mai compact:

i	1	2	3	4	5	6	7
w_i	*the*	*price*	*of*	*the*	*stock*	*fell*	*EOS*
t_i	DT	NN	IN	DT	NN	VBD	*EOS*
p_i	2	6	2	5	3	7	0
d_i	det	sub	atr	det	det	pred	*EOS*

Se observă că relațiile de dependență astfel definite sunt în concordanță cu principiile generale enunțate anterior. Alte exemple de propoziții, în limba română, vor fi discutate în § 3.5.3.

Având această definire formală a relațiilor de dependență, putem spune că o **gramatică de dependență** sau **gramatică WG** este o structură *(R,C)*, unde R este o mulțime de restricții $R \subset T \times T \times D \cup T \times W \times D \cup W \times T \times D \cup W \times \quad W \times D$, iar C este o mulțime de cerințe $C \subset T \times T \times D \cup T \times \quad W \times D \cup W \times T \times D \cup W \times W \times D$.

O **structură sintactică** *(S,D)*; $S = (T,P)$ relativ la o propoziție $w_1, w_2, \ldots, w_n$ este *corectă* din punctul de vedere al gramaticii *(R , C)* dacă

- $\forall 1 \le i \le n$,
  $(t_i, t_{p_i}, D(i)) \in R \vee (t_i, w_{p_i}, D(i)) \in R \vee (w_i, t_{p_i}, D(i)) \in R \vee$
  $(w_i, w_{p_i}, D(i)) \in R$

(Restricțiile sunt îndeplinite.)

- $\forall 1 \le i \le n$
  - dacă $\exists (t_i, t, d) \in C$, atunci $\exists 1 \le j \le n$ a.î. $p_j = i$, $t_j = t$, $D(j) = d$;
  - dacă $\exists (t_i, w, d) \in C$, atunci $\exists 1 \le j \le n$ a.î. $p_j = i$, $w_j = w$, $D(j) = d$;
  - dacă $\exists (w_i, t, d) \in C$, atunci $\exists 1 \le j \le n$ a.î. $p_j = i$, $t_j = t$, $D(j) = d$;
  - dacă $\exists (w_i, w, d) \in C$, atunci $\exists 1 \le j \le n$ a.î. $p_j = i$, $w_j = w$, $D(j) = d$.

(Cerințele sunt îndeplinite.)

Restricțiile impun ca numai anumite relații dintre cuvinte să fie considerate valide, în timp ce cerințele permit anumitor cuvinte sau tipuri de cuvinte să impună existența în cadrul propoziției a altor cuvinte sau tipuri de cuvinte care trebuie să îndeplinească anumite relații.

În acest cadru se recomandă ca procesul de analiză sintactică să se desfășoare în două etape. Astfel, într-o primă etapă, ar trebui determinată structura de dependență *(T,P)*, după care, în etapa următoare, ar trebui stabilit tipul dependențelor *D*. După ce structura de dependență este cunoscută, pentru a stabili tipul relației de dependență corespunzătoare fiecărei perechi de cuvinte (w_i, w_{p_i}), trebuie avută în vedere, între altele, funcția gramaticală, precum și faptul că aceasta este întotdeauna privită ca fiind funcția cuvântului dependent în relația sa cu capul.

Principiul conform căruia fiecare propoziție are o structură de dependență în cadrul căreia nici un arc (săgeată) nu se intersectează cu un altul și există câte o săgeată care ține spre fiecare cuvânt este crucial în analiza sintactică de dependență și a fost numit de Richard Hudson "the No-tangling Principle" (**principiul neîncâlcirii**). Acest principiu se bazează pe *tendința grupurilor sintactice de a fi continue*. Totuși, trebuie observat faptul că aceasta este numai o tendință, întrucât sunt cunoscute numeroase excepții. Acest principiu ar trebui, prin urmare, interpretat ca stipulând faptul că există o structură de tip schelet a propoziției, care ține laolaltă toate cuvintele acesteia fără apariția fenomenului de încâlcire sau intersectare, alte dependențe putând fi însă adăugate acestei structuri. Cea mai importantă *excepție* la acest principiu este *coordonarea*, pe

care R. Hudson propune să o tratăm prin recunoaşterea unor şiruri sau secvenţe de cuvinte, cum ar fi "Ion şi Maria" din exemplul care urmează, alcătuite din alte şiruri mai mici, numite "conjuncţi". Diagramele corespunzătoare ar putea fi de tipul

{[*Ion*] *şi* [*Maria*]} *au venit.*

unde *conjuncţii* sunt *Ion* şi respectiv *Maria*. Un alt exemplu de coordonare este cel din fraza

Noi {[*am făcut un duş*] *şi* [*am împachetat bagajele*]} *înainte de a pleca.*

O coordonare este în mod normal semnalată printr-o conjuncţie, aceasta fiind plasată de obicei la începutul ultimului conjunct. Conjuncţii pot fi şiruri de cuvinte, ca în cel de-al doilea exemplu, nu neapărat cuvinte individuale. Este evident faptul că fenomenul coordonării reprezintă o excepţie la teoria relaţiilor de dependenţă, întrucât dependenţa implică subordonare (a cuvântului dependent faţă de cap), cel puţin din punct de vedere sintactic dacă nu şi semantic, în timp ce în cazul coordonării conjuncţii au roluri sensibil egale. Tocmai de aceea, în aplicarea principiului neîncâlcirii, fiecare conjunct trebuie tratat în mod separat.

3.5.2. Relaţii de dependenţă în limba română

În stabilirea celor mai frecvente relaţii de dependenţă în limba română s-a luat în consideraţie [33], [34], de cele mai multe ori, funcţia sintactică a cuvântului dependent. Astfel, au fost urmate o serie de reguli generale, aceasta fiind cel mai frecvent utilizată dintre ele.

Regulile generale de bază[6] care au fost concepute şi aplicate pentru stabilirea celor mai frecvente relaţii de dependenţă în limba română sunt următoarele:

- considerarea funcţiei sintactice (date de analiza sintactică clasică) a cuvântului dependent;
- în acele cazuri în care cuvântul dependent este o prepoziţie sau o conjuncţie coordonatoare, stabilirea tipului dependenţei se face în concordanţă cu funcţia sintactică (clasică) a elementului (cuvântului) introdus în propoziţie de către acea prepoziţie sau conjuncţie;

---

[6] Acest studiu a fost realizat în cadrul unui proiect de cercetare-dezvoltare finanţat de către Fundaţia Volkswagen pe o perioadă de doi ani (1996-1998). Adaptarea teoriei gramaticilor de dependenţă limbii române s-a făcut ca parte integrantă a conceperii sistemului german-bulgar-român de traducere asistată de calculator numit DBR-MAT. Scopul declarat al DBR-MAT a fost implementarea pilot a unui sistem de traducere asistată de calculator care să combine o abordare bazată pe procesarea cunoştinţelor cu metode statistice în prelucrarea limbajului natural.

- caracteristicile morfologice ale cuvântului dependent sunt uneori luate în considerație (exemplu: *relația nehotărâtă*, stabilită atunci când cuvântul dependent este un articol nehotărât);
- tipul dependenței este dat de cuvântul cap numai în acele cazuri în care capul este reprezentat de o prepoziție sau o conjuncție coordonatoare (exemplu: *relația prepozițională*, *relația conjuncțională*);

Relațiile de dependență cel mai frecvent întâlnite în cazul limbii române și stabilite conform acestor reguli generale [33], [34] sunt prezentate în Tabelul 3.1. În mod evident, aceste dependențe ar putea fi în continuare rafinate în funcție de categoria gramaticală a cuvântului dependent (care poate fi substantiv, pronume, verb, adjectiv etc.), luând astfel naștere relații ca "relația prepozițional-adjectivală" (în care cuvântul cap este o prepoziție, iar cuvântul dependent este un adjectiv). O asemenea rafinare nu se recomandă însă atunci când nu este disponibil un corpus de dimensiuni semnificativ de mari.

Tabelul 3.1 include cele mai frecvente relații de dependență găsite, în cadrul de lucru oferit de către proiectul DBR-MAT, corespunzător limbii române. Menționăm faptul că, în acest cadru, au fost studiate numai texte științifice din domeniul chimiei și al tehnologiilor chimice. Tipurile de dependențe au fost stabilite conform regulilor generale menționate anterior, iar tabelul clasifică relațiile de dependență corespunzătoare în funcție de cuvântul cap. Precizăm, de asemenea, faptul că, în cadrul Tabelului 3.1, ori de câte ori se face referire la un cuvânt cap de tip verb, în afara cazului în care se specifică altceva, verbul respectiv poate fi atât predicativ, cât și nepredicativ.

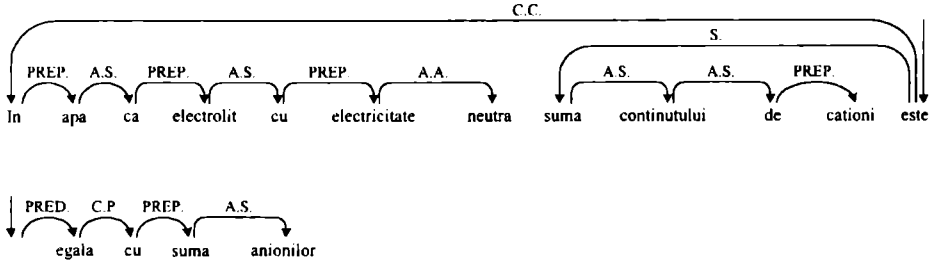
TABELUL 3.1

CUVÂNT CAP	CUVÂNT DEPENDENT	TIPUL RELAȚIEI DE DEPENDENȚĂ	ABREVIERE
verb	prepoziție	complement circumstanțial	C.C.
verb	prepoziție	complement prepozițional	C.P.
verb	prepoziție	complement de agent	C.A.
verb	prepoziție	complement direct	C.D.
verb	verb (participiu)	complement direct	C.D.
verb (participiu)	verb auxiliar	relație auxiliară	AUX.
verb (infinitiv)	"a"	relație infinitivală	INF.
verb	conjuncție coordonatoare	complement direct	C.D.

verb	adverb	complement circumstanțial	C.C.
verb	pronume reflexiv	relație reflexivă	REF.
verb	substantiv	subiect	S.
verb	pronume	subiect	S.
verb	numeral	subiect	S.
verb	verb nepredicativ	subiect	S.
verb	substantiv	complement direct	C.D.
verb	pronume	complement direct	C.D.
verb	numeral	complement direct	C.D.
verb	verb nepredicativ	complement direct	C.D.
verb	substantiv	complement indirect	C.I.
verb	pronume	complement indirect	C.I.
verb	substantiv	nume predicativ	PRED.
verb	adjectiv	nume predicativ	PRED.
verb	adverb	nume predicativ	PRED.
verb	pronume	nume predicativ	PRED.
verb	numeral	nume predicativ	PRED.
verb	verb nepredicativ	nume predicativ	PRED.
prepoziție	substantiv	relație prepozițională	PREP.
prepoziție	conjuncție coordonatoare	relație prepozițională	PREP.
prepoziție	pronume demonstrativ	relație prepozițională	PREP.
prepoziție	verb (infinitiv)	relație prepozițională	PREP.
conjuncție coordonatoare	substantiv	relație conjuncțională	CONJ.
conjuncție coordonatoare	verb	relație conjuncțională	CONJ.
conjuncție coordonatoare	adjectiv participial	relație conjuncțională	CONJ.

adjectiv participial	prepoziție	complement de agent	C.A.
adjectiv participial	substantiv	complement indirect	C.I.
adjectiv participial	prepoziție	complement circumstanțial	C.C.
adjectiv participial	conjuncție coordonatoare	complement indirect	C.I.
adjectiv participial	adverb	complement circumstanțial	C.C.
adjectiv	prepoziție	complement prepozițional	C.P.
adjectiv	adverb	relație comparativă	COMP.
adjectiv	articol demonstrativ	relație comparativă	COMP.
substantiv	articol nehotărât	relație nehotărâtă	NEHOT.
Substantiv	prepoziție	atribut substantival	A.S.
Substantiv	conjuncție coordonatoare	atribut substantival	A.S.
Substantiv	substantiv	atribut substantival apozițional	A.S.AP.
Substantiv	substantiv	atribut substantival	A.S
Substantiv	adjectiv	atribut adjectival	A.A.
Substantiv	pronume	atribut pronominal	A.P.
Adverb	adverb	relație comparativă	COMP.
Adverb	articol demonstrativ	relație comparativă	COMP.

Un exemplu de **analiză sintactică** a unei propoziții românești adnotate conform Tabelului 3.1 este următorul:



O analiză sintactică[7] de acest tip a propozițiilor românești a fost realizată în cadrul proiectului internațional de cercetare DBR-MAT.

---

[7] În efectuarea analizei sintactice de dependență (sau de tip WG) s-a început prin a se face observația că un număr relativ mare de relații de dependență care respectă principiile generale enunțate anterior poate fi conceput, fără ca aceste relații să aibă neapărat ceea ce am putea numi semnificație lingvistică. Țelul cercetătorului trebuie, prin urmare, să fie acela de a găsi structuri care pot fi considerate corecte nu numai din punct de vedere formal, ci și din punctul de vedere al realității lingvistice și implicit al vorbitorului nativ. Acest țel poate fi, în principiu, atins în două moduri. O posibilă soluție este aceea a definirii unei gramatici adecvate. În cadrul de lucru oferit de această teorie lingvistică, specificarea unei asemenea gramatici înseamnă găsirea unei mulțimi de constrângeri care să ajute la stabilirea faptului că anumite structuri sintactice sunt corecte, iar altele nu. Spre exemplu, în virtutea unor asemenea constrângeri se poate decide faptul că anumite cuvinte ale unei propoziții pot avea rolul de cuvânt cap, în timp ce altele nu pot deține acest rol. Definirea unor astfel de constrângeri pentru o anumită limbă și respectiv specificarea unei gramatici adecvate reprezintă însă o sarcină extrem de grea și de o deosebită amploare. O a doua soluție a problemei enunțate, care a fost cu succes aplicată limbii române în cadrul proiectului DBR-MAT, este de natură stocastică și se referă la asocierea unei probabilități fiecărei structuri sintactice. În acest caz, presupunând că probabilitățile au fost atribuite structurilor astfel încât structura sintactică având probabilitatea cea mai mare este cea corectă, pentru o propoziție dată trebuie aleasă acea structură sintactică a cărei probabilitate asociată are valoarea maximă. Atribuirea unor asemenea probabilități înseamnă găsirea unui *model stocastic*, și anume a acelui model stocastic care este cel mai adecvat. În această abordare, pentru găsirea structurii sintactice de dependență a unei propoziții nu este necesară specificarea explicită a unei gramatici de dependență (sau de tip WG). Gramatica va fi în mod implicit inclusă în parametrii modelului stocastic, care, la rândul lor, vor fi estimați pe baza datelor lingvistice (adică a unui corpus).

Pe baza acestor considerații putem spune că a găsi un *algoritm de analiză sintactică* înseamnă a găsi un algoritm care are ca *input* o propoziție și ca *output* structura sintactică *(S,D)* a acelei propoziții, unde *S=(T,P)* și *D* au aceleași semnificații din §3.5.1. Etapele în derularea unui asemenea algoritm sunt:

- găsirea mulțimii *T* ("part of speech tagging");
- găsirea mulțimii *P* (adică a relațiilor de dependență);

3.5.3. Concluzii

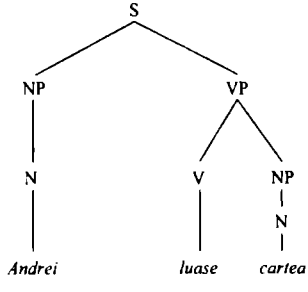
Așa cum se cunoaște deja, există două metode diametral opuse de a descrie structura sintactică a propozițiilor aparținând limbajului natural: utilizarea D-arborilor și respectiv a PS-arborilor. În mod evident, sunt posibile combinații ale celor două metode, care presupun anumite linii de compromis în diverse momente ale analizei, dar nu există nici o a treia posibilitate complet distinctă.

După cum se arată în [71], există câteva diferențe majore între **D-limbaj** (limbajul gramaticilor de dependență) și **PS-limbaj** (limbajul gramaticilor PS), pe care vom încerca să le menționăm, pe scurt, în continuare. Precizăm că, în cele ce urmează, arborele de derivare rezultat în urma efectuării analizei sintactice care utilizează o gramatică PS va fi denumit **PS-arbore**, în timp ce arborele corespunzător rezultat în urma utilizării în analiza sintactică a unei gramatici de dependență va fi numit **D-arbore**.

PS-structura propoziției

Andrei luase cartea.

este reflectată de următorul **PS-arbore**

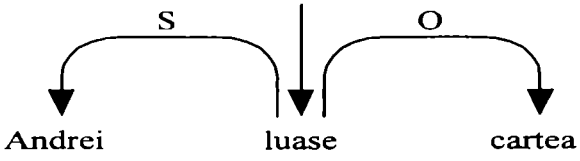


- găsirea mulțimii *D* (adică a tipului dependențelor).

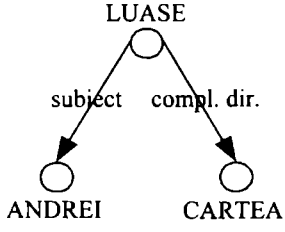
În cadrul proiectului DBR-MAT, găsirea mulțimii *T* s-a făcut utilizându-se un algoritm propus de Ratnaparkhi în 1996 [89]. Acest algoritm este de natură stocastică și utilizează entropia maximă. Găsirea mulțimii *P*, adică a relațiilor de dependență, s-a făcut, de asemenea, prin utilizarea unui algoritm stocastic, și anume a algoritmului lui Eisner [24] propus în același an. Acest algoritm a fost modificat prin schimbarea modelului stocastic, cu utilizarea din nou a entropiei maxime. Algoritmul de găsire a mulțimii P reprezintă o implementare a metodei programării dinamice cu scopul de a găsi cea mai probabilă analiză în manieră "bottom up" (de jos în sus). După determinarea mulțimilor *T* și *P*, găsirea mulțimii *D* nu mai ridică nici un fel de probleme.

Principala concluzie care s-a desprins, independent de limbă, în urma acestui studiu, este aceea că formalismul gramaticilor de dependență este extrem de adecvat și poate fi utilizat cu succes în efectuarea analizei sintactice de tip stocastic.

în timp ce **D-structura** aceleiași propoziții



este reflectată de următorul **D-arbore:**



O primă diferență semnificativă între D-limbaj și PS-limbaj constă în aceea că un PS-arbore corespunzător unei expresii aparținând limbajului natural arată care elemente ale acesteia (cuvinte sau chiar grupuri sintactice) se pot combina cu alte elemente pentru a forma niște unități de ordin mai mare. Un PS-arbore dezvăluie structura unei propoziții în termeni de grupări ale elementelor sale: blocuri maximale care constau din blocuri mai mici, care, la rândul lor, constau din blocuri și mai mici etc.. PS-structura se exprimă în termeni de constituenți, operația logică aflată la baza acestei abordări fiind aceea a incluziunii de mulțimi, cu ajutorul căreia se exprimă apartenența la un grup sintactic, la o categorie etc.. Această abordare favorizează punctul de vedere *analitic*. Un D-arbore, pe de altă parte, arată ce elemente se află în relație cu alte elemente și în ce mod. D-structura propoziției reflectă relațiile existente între unități sintactice indivizibile, lucrând direct cu forme lexicale. În această abordare, operația logică de bază este aceea a stabilirii de relații binare. Propoziția nu mai este alcătuită din grupuri sintactice, categorii, ci din cuvinte legate între ele prin relații de dependență. Această abordare favorizează, prin urmare, punctul de vedere *sintetic*.

O altă diferență între PS-limbaj și D-limbaj este dată de faptul că, în cadrul unui PS-arbore, apartenența la o anumită categorie este specificată ca parte a reprezentării sintactice. Simboluri ca NP, VP, N etc. intervin în PS-arbori ca etichete ale unor vârfuri. Cu alte cuvinte, unele caracteristici sintactice date de operații precum categorizarea și subcategorizarea sunt folosite ca instrument principal în exprimarea rolului sintactic. În cadrul unui D-arbore, pe de altă parte, simbolurile reprezentând apartenența la o categorie, precum și alte

proprietăți sintactice nu sunt admise ca elemente imediate ale structurii sintactice. (Astfel de informații sunt incluse în dicționar, lexicon etc.).

O a treia diferență esențială constă în faptul că, într-un PS-arbore, majoritatea nodurilor corespund unor simboluri neterminale. Ele reprezintă grupuri sintactice și nu corespund formelor lexicale efective care intervin în propoziția analizată. Prin contrast, un D-arbore conține numai noduri terminale, nefiind necesară nici o reprezentare abstractă a grupurilor sintactice.

PS-limbajul este, în esență, un limbaj linear, în timp ce D-limbajul este unul bidimensional, aceasta generând o altă deosebire fundamentală între cele două tipuri de reprezentări sintactice discutate aici. Astfel, în cadrul unui PS-arbore, vârfurile arborelui trebuie să fie ordonate linear, ordinea nefiind neapărat cea a formelor lexicale care intervin în propoziție. În cadrul unui D-arbore, pe de altă parte, vârfurile nu se află într-o astfel de ordine. Ordinea liniară a formelor lexicale din interiorul propoziției este un mijloc folosit de limbile naturale pentru a codifica relații sintactice și, prin urmare, ordinea liniară nu trebuie să intervină în structurile sintactice.

În fine, o ultimă deosebire importantă între cele două tipuri de reprezentări constă în aceea că, în timp ce un PS-arbore nu specifică tipul legăturii sintactice existente între doi constituenți, un D-arbore pune în mod special accentul pe specificarea în detaliu a tipului legăturii dintre oricare două elemente aflate în relație de dependență.

3.6. Gramatici contextuale

O altă clasă de gramatici care generează limbaje ce nu au o legătură directă cu ierarhia Chomsky (neputând fi comparate cu familiile de bază ale acestei ierarhii), dar despre care s-a arătat că modelează foarte bine limbajul natural este aceea a **gramaticilor contextuale**.

Introduse de către Solomon Marcus în 1969, gramaticile contextuale reprezintă fără îndoială cea mai importantă contribuție a școlii românești în domeniu, ele suscitând și astăzi un viu interes, așa cum se va vedea și în cele ce urmează.

În ceea ce privește rădăcinile lor strict lingvistice, gramaticile contextuale sunt înrudite cu lingvistica distribuțională americană, al cărei potențial încearcă să îl exploateze. Ele apar într-un moment în care devenise deja clar un fapt formulat ca atare mult mai târziu, și anume acela că structura bazată pe existența grupurilor sintactice (a constituenților) era inadecvată din punctul de vedere al capacității și al dispozitivelor sale descriptive. În cazul unei gramatici contextuale procesul generativ se bazează pe două operații lingvistice duale, care se numără printre cele mai importante atât în limbajele naturale, cât și în cele formale: inserarea unui șir într-un context dat și adăugarea unui context la

un şir dat. Lingvistica distribuţională descriptivă dezvoltată în S.U.A. în anii '40 şi '50 se bazează în întregime pe aceste idei. Orice derivare într-o gramatică contextuală este o secvenţă finită de operaţii de acest fel, începând de la un stoc iniţial şi finit de şiruri, suficient de simple pentru a putea fi considerate şiruri primitive corect formate sau axiome.

S. Marcus introduce [61] gramaticile contextuale ca pe nişte "gramatici intrinseci", fără simboluri auxiliare, bazate numai pe operaţia lingvistică fundamentală de inserare a cuvintelor în structuri date, în conformitate cu anumite dependenţe contextuale. Gramaticile contextuale includ *contexte* (sau perechi de cuvinte) asociate unor *selectori* (mulţimi de cuvinte). Un context poate fi alăturat oricărui element selector asociat. În acest fel, pornindu-se de la o mulţime finită de cuvinte (axiome), este generat limbajul.

O întreagă varietate de proprietăţi ale gramaticilor contextuale a fost studiată de-a lungul timpului, o sursă completă de informaţie în acest sens constituind-o monografia din 1997 a lui G. Păun [88]. O realitate de necontestat este aceea că numeroasele variante ale gramaticilor contextuale luate în consideraţie de către literatura de specialitate au fost investigate în special din punct de vedere matematic. În prezent se fac încă eforturi pentru a demonstra relevanţa acestor gramatici în chiar domeniul care motivează cel mai puternic existenţa lor: lingvistica, în general studiul limbajului natural şi al procesării acestuia.

O clasă de gramatici contextuale introdusă relativ recent [69] pare a fi foarte promiţătoare din acest punct de vedere: *gramaticile cu o utilizare maximală a selectorilor*. În aceste gramatici, un context este alăturat unui *cuvânt-selector* dacă acest selector este cel mai mare în acea poziţie (i.e. nici un alt cuvânt care îl conţine ca subcuvânt nu poate fi selector). Chiar dacă, din punctul de vedere al teoriei limbajelor formale, aceste gramatici nu au un comportament spectaculos, s-a demonstrat [68] că ele au o proprietate surprinzătoare şi extrem de importantă din punct de vedere lingvistic: toate cele trei caracteristici de bază ale limbajelor naturale, care conduc la *nonindependenţa* lor *de context* (reduplicarea, dependenţele încrucişate şi acordurile multiple), pot fi modelate de aceste gramatici (şi de nici o altă clasă de gramatici contextuale). Caracteristicile menţionate reprezintă restricţiile obişnuite care apar în limbajele naturale (şi artificiale), ducând la *nonindependenţa de context* a acestora. Ele conduc la nişte limbaje formale de forma:

$\{xcx \mid x \in \{a,b\}^*\}$ (duplicarea unor cuvinte de lungimi arbitrare)

$\{a^n b^m c^n d^m \mid n,m \geq 1\}$ (două dependenţe încrucişate)

şi $\{a^n b^n c^n \mid n \geq 1\}$ ([cel puţin] trei poziţii corelate).

Toate aceste limbaje sunt nonindependente de context şi toate pot fi generate în mod relativ simplu de către gramatici contextuale cu o utilizare maximală a selectorilor (a se vedea §3.6.2).

Pentru a demonstra faptul că gramaticile contextuale reprezintă un formalism adecvat în analiza limbajului natural, este necesară şi furnizarea unui *analizor sintactic* eficient care să lucreze cu aceste gramatici. Preocupări dintre cele mai recente în domeniul procesării limbajului natural se referă la acest subiect ([31], [104]).

3.6.1. Formalismul gramaticilor contextuale

Pentru a defini, din punct de vedere formal, gramaticile contextuale, vom lua în consideraţie un alfabet V. Fiind dat alfabetul V (pe care îl numim şi vocabular), vom nota prin V^* mulţimea tuturor cuvintelor (şirurilor) peste V, inclusiv cuvântul vid, care este notat cu λ. Mulţimea tuturor cuvintelor nevide peste V, i.e. mulţimea $V^*-\{\lambda\}$, va fi notată V^+. Lungimea lui $x \in V^*$ se notează $|x|$, iar imaginea lui în oglindă (numită şi inversarea lui x) va fi notată $mi(x)$. Familiile limbajelor finite, regulate şi independente de context vor fi desemnate prin notaţiile FIN, REG şi respectiv CF.

O **gramatică contextuală (GC) (cu selecţie)** este o construcţie

$$G = (V, A, \{(S_1, C_1), \ldots, (S_n, C_n)\}) \ , \ n \geq 1 \, ,$$

unde V este un alfabet, A este un limbaj finit peste V, $S_1, \ldots, S_n$ sunt limbaje peste V, iar $C_1, \ldots, C_n$ sunt submulţimi finite ale lui $V^* \times V^*$.

Elementele lui A sunt numite *axiome* sau *cuvinte de pornire*, mulţimile S_i sunt numite *selectori*, iar elementele mulţimilor C_i, scrise sub forma (u, v), sunt numite *contexte*. Perechile (S_i, C_i) se numesc *producţii* sau *perechi context-selector*. În mod intuitiv, această construcţie se caracterizează prin alăturarea contextelor din C_i cuvintelor din mulţimea asociată S_i. Astfel, operaţia recursivă este aceea care extinde un element s_i al unui selector S_i la *stânga* prin u şi la *dreapta* prin v i.e. unul dintre contextele mulţimii C_i este alăturat cuvântului s_i din mulţimea asociată S_i.

Un **exemplu** de gramatică contextuală este următorul:

$$G_1 = \left(\{a, b\}, \{ab, a^2b^2\}, \{ (\{ab\}, \{(a, \lambda)\}), (\{a^2b^2\}, \{(a, b)\}) \} \right).$$

În cadrul acestui triplet, pe prima poziție se definește alfabetul gramaticii: $V_{gc} = \{a, b\}$. Pe poziția a doua figurează mulțimea finită a axiomelor: $A_{gc} = \{ab, a^2b^2\}$. Pe cea de-a treia poziție se află mulțimea finită a perechilor context-selector: $(S_1, C_1) = (\{ab\}, \{(a, \lambda)\})$, $(S_2, C_2) = (\{a^2b^2\}, \{(a, b)\})$. Spre exemplu, în $(\{ab\}, \{(a, \lambda)\})$, *ab* este singurul *element selector* sau *cuvânt-selector* al *mulțimii selector* $\{a, b\}$, iar perechea (a, λ) desemnează unicul *context* de aici i.e. *a* se alătură elementului selector *ab* la stânga, iar λ, șirul vid, se alătură elementului selector *ab* la dreapta. În mod corespunzător, *a* se numește *context stâng (s)*, iar λ se numește *context drept (d)* al elementului selector *ab*. O asemenea pereche $(ab, (a, \lambda))$ este numită *element-pereche context-selector*.

Din punct de vedere formal, **relația de derivare directă** pe V^* se definește după cum urmează:

$$x \Rightarrow_{in} y$$

iff $x = x_1x_2x_3$, $y = x_1ux_2vx_3$, unde $x_2 \in S_i$, $(u, v) \in C_i$, pentru un i, $1 \le i \le n$.

Dacă se notează prin $\Rightarrow_{in}^*$ închiderea reflexivă și tranzitivă a relației[8] $\Rightarrow_{in}$, atunci limbajul generat de gramatica G este:

$$L_{in}(G) = \{z \in V^* \mid w \Rightarrow_{in}^* z, \text{ pentru un } w \in A\}.$$

În consecință, $L_{in}(G)$ conține toate cuvintele lui *A*, precum și cuvinte care pot fi obținute din acestea prin alăturarea (finită) a contextelor, în conformitate cu selecția impusă de împerecherea de tip (S_i, C_i).

---

[8] Indicele *in* sugerează o derivare internă și deosebește această operație de operația notată $\Rightarrow_{ex}$ i.e. *derivarea externă* definită pentru *G*. Conform acesteia din urmă, contextul este alăturat la capetele cuvintelor care rezultă din derivare: $x \Rightarrow_{ex} y$ *iff* $y = uxv$ cu $(u, v) \in C_i$, $x \in S_i$ pentru un i, $1 \le i \le n$. În Marcus (1969) este luată în considerație numai derivarea externă, în timp ce gramaticile contextuale cu derivare internă sunt introduse de către Păun și Nguyen (1980). În cele ce urmează nu vom investiga derivarea externă, ci ne vom referi numai la cea internă, singura care a fost luată în considerație pentru conceperea unor analizori sintactici.

Observaţia 3.2

Definiţia anterioară a gramaticilor contextuale este numită **modulară**. Adesea (şi în special în cadrul eforturilor de a concepe un analizor sintactic care să opereze cu aceste gramatici) este utilă prezentarea unei gramatici contextuale în aşa-numita **formă funcţională**.

Forma funcţională a unei GC este definită după cum urmează: gramatica este de forma

$$G = (V, A, C, \varphi),$$

unde V şi A au aceleaşi semnificaţii de până acum, C este o mulţime finită de contexte peste V (i.e. $C = \bigcup_{i=1}^{n} C_i$), iar funcţia $\varphi : V^* \to 2^C$ asociază mulţimi de contexte din C şirurilor din V^*. În acest caz, putem scrie:

$$x \Rightarrow_{in} y$$

iff $x = x_1 x_2 x_3$, $y = x_1 u x_2 v x_3$, pentru un $(u, v) \in \varphi(x_2)$, $x_1, x_2, x_3 \in V^*$.

Aşa cum se remarcă în [68], este uşor de constatat faptul că, pornindu-se de la o gramatică contextuală dată în forma modulară

$$G = (V, A, \{(S_1, C_1), \ldots, (S_n, C_n)\})$$

poate fi luată în consideraţie perechea ei funcţională

$$G' = (V, A, C, \varphi)$$

cu:

$$C = \bigcup_{i=1}^{n} C_i,$$

$$\varphi(x) = \{(u, v) \mid (u, v) \in C_i, x \in S_i, 1 \le i \le n\}, x \in V^*.$$

Reciproc, de la o gramatică dată sub forma $G = (V, A, C, \varphi)$ cu

$$C = \{(u_1, v_1), \ldots, (u_n, v_n)\},$$

se poate trece, spre exemplu, la $G' = (V, A, \{(S_1, C_1), \ldots, (S_n, C_n)\})$, luând, pentru fiecare $i, 1 \le i \le n$:

$$C_i = \{(u_i, v_i)\}$$

și S_i mulțimea șirurilor din V^* cărora li se poate alătura contextul (u_i, v_i), adică:

$$S_i = \{x \in V^* \mid (u_i, v_i) \in \varphi(x)\}.$$

În mod evident, în ambele cazuri, cele două gramatici G și G' sunt echivalente.

Revenind la relația de derivare internă $\Rightarrow_{in}$, definită anterior, vom remarca faptul că literatura de specialitate [69] ia în considerație două variante naturale ale acesteia:

$$x \Rightarrow_{Ml} y$$

iff $x = x_1x_2x_3$, $y = x_1ux_2vx_3$, pentru $x_2 \in S_i$, $(u,v) \in C_i$ pentru un $1 \le i \le n$ și nu există nici un $x_1', x_2', x_3' \in V^*$ astfel încât $x = x_1'x_2'x_3'$, $x_2' \in S_i$ și $|x_1'| \le |x_1|$, $|x_3'| \le |x_3|$, $|x_2'| > |x_2|$;

$$x \Rightarrow_{Mg} y$$

iff $x = x_1x_2x_3$, $y = x_1ux_2vx_3$, pentru $x_2 \in S_i$, $(u,v) \in C_i$ pentru un $1 \le i \le n$ și nu există nici un $x_1', x_2', x_3' \in V^*$ astfel încât $x = x_1'x_2'x_3'$, $x_2' \in S_j$ pentru un $1 \le j \le n$ și $|x_1'| \le |x_1|$, $|x_3'| \le |x_3|$, $|x_2'| > |x_2|$.

Vom spune că $\Rightarrow_{Ml}$ este o derivare în *modul maximal local* (cuvântul-selector x_2 este maximal în S_i) și că $\Rightarrow_{Mg}$ este o derivare în *modul maximal global* (cuvântul-selector x_2 este maximal cu privire la toți selectorii $S_1, \ldots, S_n$).

Pentru $\alpha \in \{Ml, Mg\}$ se notează:

$$L_\alpha(G) = \{z \in V^* \mid w \Rightarrow_\alpha^* z, \text{ pentru un } w \in A\}.$$

Revenind la exemplul de gramatică luat în considerație anterior,

$$G_1 = \left(\{a,b\}, \{ab, a^2b^2\}, \{ (\{ab\}, \{(a,\lambda)\}), (\{a^2b^2\}, \{(a,b)\}) \} \right),$$

vom menționa faptul că limbajul generat de această gramatică este

$$L_1 = \{a^n b \mid n \geq 1\} \cup \{a^n b^n \mid n \geq 1\}$$

dacă se ia în considerație maximalitatea cuvântului-selector în raport cu toți selectorii (operația Mg).

Dacă, într-o gramatică $G = (V, A, \{(S_1, C_1), \ldots, (S_n, C_n)\})$, toți selectorii $S_1, \ldots, S_n$ sunt limbaje aparținând unei familii date F, spunem că G este o *gramatică contextuală cu F selecție*. Familiile de limbaje $L_\alpha(G)$, pentru G o gramatică contextuală cu F selecție, sunt notate prin $CL_\alpha(F)$, unde $\alpha \in \{in, Ml, Mg\}$. Problema conceperii unor analizori sintactici care să utilizeze aceste gramatici și care să lucreze în timp polinomial a fost, spre exemplu, abordată în literatura de specialitate [31] pentru F reprezentând una dintre familiile *FIN*, *REG* și CF[9].

---

[9] Analiza sintactică a limbajelor generate de gramaticile contextuale de orice tip reprezintă un domeniu de cercetare care a rămas deschis. Cele mai importante rezultate în această privință sunt cele obținute de K. Harbusch [31], care propune un analizor sintactic (parser) pentru gramatici contextuale cu selectori lineari, regulați și independenți de context pe baza celor trei definiții ale derivării *in, Mg și Ml* i.e. pe baza înlocuirii nerestricționate de contexte în conformitate cu un selector sau pe baza înlocuirii numai a selectorilor maximali, în conformitate cu toți selectorii și respectiv cu același selector. Analizorul sintactic este bazat pe metoda propusă de Earley și, prin urmare, acționează de sus în jos și de la stânga la dreapta (a se vedea capitolul 4). Este vorba de un parser cu două nivele, cele două treceri executând definițiile derivărilor *in, Mg și Ml*. Mai mult, în timp polinomial, pot fi enumerate într-o manieră condensată toate derivările. Costurile totale de procesare sunt de ordinele $O(n^6)$ în ceea ce privește timpul și $O(n^4)$ în ceea ce privește spațiul, în conformitate cu definiția *in*. Conform definițiilor *Mg* și *Ml* aceleași costuri sunt de ordinele $O(n^9)$ în ceea ce privește timpul și respectiv $O(n^6)$ în ceea ce privește spațiul. Aceste din urmă ordine se mențin atunci când este cerută reprezentarea condensată a tuturor derivărilor în cazul efectuării calculelor conform definiției *in*. Implementarea acestui analizor sintactic se află încă într-un stadiu de început, fiind necesară aprofundarea studiului prin folosirea unor fragmente cât mai

3.6.2. Asupra relevanței lingvistice a gramaticilor contextuale cu utilizare maximală a selectorilor

Gramaticile contextuale cu utilizare maximală a selectorilor, introduse în [69], s-au dovedit a fi în mod special extrem de adecvate pentru descrierea sintaxei limbajelor naturale. Deși, așa cum se arată în [68], capacitatea generativă a acestor gramatici nu este "prea mare", existând limbaje independente de context pe care ele nu le pot genera[10], puterea lor, pe de altă parte, nu este nicidecum "prea mică". O dovadă în acest sens o constituie și următoarea teoremă a cărei demonstrație poate fi găsită în [68]:

Teorema 3.1

Familia de limbaje $CL_{Mg}(FIN)$ conține limbaje nonindependente de context.

Aceste gramatici având capacitate generativă relativ mică s-au dovedit a fi un foarte bun model pentru sintaxa limbajelor naturale, ele putând genera limbaje care prezintă toate cele trei caracteristici de bază ale limbilor naturale (dar și artificiale) ce conduc la nonindependența lor de context (reduplicarea, dependențele încrucișate și acordurile multiple).

Caracteristicile de bază ale limbajelor de programare ce conduc la existența unor asemenea dependențe sunt necesitatea declarării identificatorilor și a numelor de proceduri, precum și a definirii etichetelor. În limbile naturale, astfel de copieri și dependențe pot să apară fie la nivelul vocabularului, fie la cel al

---

mari de descrieri ale limbajului natural. Rezultatele promițătoare obținute până în prezent [31] i-au determinat pe cercetători să își propună ca, în viitor, să se ocupe de construirea unei gramatici contextuale a limbii engleze. O implementare în Java a acestui parser este disponibilă via Internet la adresa:

http://www.uni-koblenz.de/~harbusch/CG-PARSER/welcome-cg.html

Un alt parser pentru gramatici contextuale cu selectori independenți de context având complexitatea $O(n^2)$ în cazul determinist a fost dezvoltat de R. Gramatovici în [104].

[10] Există astfel de limbaje independente de context, chiar și unele lineare, cum ar fi $L = \{a^n cb^m cb^m ca^n \mid n, m \geq 1\}$, limbaj despre care se demonstrează în [68] că nu aparține familiei $CL_{Mg}(REG)$. Incapacitatea gramaticilor contextuale cu o utililizare maximală globală a selectorilor de a genera astfel de structuri centrat imbricate, întâlnite în limbajele naturale, chiar și atunci când sunt folosiți selectori regulați, reprezintă o limitare surprinzătoare a acestui tip de gramatici.

propozițiilor unei limbi date. Problema nu este una simplă, întrucât nu va fi întotdeauna evident ce anume reprezintă o construcție corectă din punct de vedere gramatical și ce nu relativ la o anumită limbă naturală. Nonindependența de context atât a limbajelor naturale, cât și a celor de programare a fost investigată încă de la începutul anilor '60 (Bar-Hillel și Shamir [1964]; Floyd [1962], printre alții). În timp ce pentru Algol 60 și pentru toate limbajele de programare evoluate, chestiunea a fost rezolvată încă de la început - aceste limbaje nu sunt independente de context - o lungă dezbatere a fost necesară în ceea ce privește limbile naturale. În prezent au fost însă găsite exemple convingătoare de construcții nonindependente de context aparținând mai multor limbaje naturale.

La nivelul vocabularului, un exemplu cunoscut (Culy 1985) și preluat de noi din [68] este acela al unei limbi, Bambara, din familia africană Mande. Această limbă admite cuvinte compuse de forma *șir-de-cuvinte-o-șir-de-cuvinte*, astfel încât limbajul formal corespunzător constă din cuvinte de forma xcx, unde x este un cuvânt de lungime arbitrară peste un alfabet care nu conține simbolul c. (Acest simbol corespunde operatorului o din construcția în limba Bambara menționată anterior). Întrucât cuvintele pot fi întotdeauna codificate folosind două simboluri, în acest caz putem lucra cu limbajul:

$$M_1 = \left\{ xcx \mid x \in \{a,b\}^* \right\}.$$

O altă construcție nonindependentă de context a fost găsită într-un dialect al limbii germane vorbit în Elveția, în jurul orașului Zürich (Shieber 1985; Pullum 1985). Acest dialect permite existența unor construcții de forma $NP_a^m NP_d^n V_a^{m'} V_d^{n'}$, unde NP_a reprezintă grupuri nominale în cazul acuzativ, NP_d sunt grupuri nominale în dativ, V_a sunt verbe care cer acuzativul, iar V_d sunt verbe care cer dativul. În același timp, este îndeplinită condiția $m=m'$, $n=n'$. Aceasta conduce la limbaje de forma

$$M_2 = \left\{ a^m b^n c^m d^n \mid n,m \geq 1 \right\}.$$

Limbajele M_1 și M_2 ilustrează fenomenele de reduplicare și de existență a dependențelor încrucișate, pe care le-am amintit și la începutul acestui paragraf. O altă caracteristică esențială a limbajelor naturale, care, de asemenea, conduce la nonindependența lor de context, o constituie, așa cum am mai arătat, acordurile multiple. Acestea se traduc în limbaje de forma:

$$M_3 = \left\{ a^n b^n c^n \mid n \geq 1 \right\}.$$

În ceea ce privește versiunea din limbajul natural a acestei forme, Manaster Ramer (1993, 1994) remarcă faptul că nu este cunoscută nici o construcție a nici unei limbi naturale care să fie de această formă, dar că astfel de construcții pot să apară ca urmare a interacțiunii de diverse tipuri (spre exemplu de tip coordonare) a două construcții diferite sau ca urmare a compunerii a două construcții distincte. Toate acestea îi servesc lui Manaster Ramer ca argumente pentru a putea afirma că "în acești termeni, limbile naturale posedă o proprietate importantă diferită de limbajele formale obișnuite. Anume, în limbile naturale, construcțiile individuale adesea au forme pe care nici un limbaj natural, considerat ca un întreg, nu le poate avea. Astfel, reduplicarea este comună (probabil universală), dar nu există nici un limbaj natural care să fie alcătuit, în întregul său, din reduplicări."

Din punctul de vedere al teoriei limbajelor formale se demonstrează cu ușurință faptul că nici unul dintre limbajele M_1, M_2, M_3 nu este independent de context. Mai mult, așa cum se remarcă în [68], a cărei linie de argumentație o urmăm aici, M_1 și M_3 nu aparțin nici unei familii $CL_{in}(F)$, pentru F arbitrar (chiar mai general decât *FIN* și *REG*). Toate cele trei limbaje menționate pot fi însă generate folosind selectorii în modul maximal, atât în manieră locală, cât și globală. Iată gramaticile contextuale care generează aceste limbaje:

$$G_1 = \left(\{a,b,c\}, \{c\}, \left\{\left(\{c\}\{a,b\}^*, \{(a,a),(b,b)\}\right)\right\}\right)$$
$$G_2 = \left(\{a,b,c,d\}, \{abcd\}, \left\{\left(ab^+c, \{(a,c)\}\right), \left(bc^+d, \{(b,d)\}\right)\right\}\right)$$
$$G_3 = \left(\{a,b,c\}, \{abc\}, \left\{\left(b^+, \{(a,bc)\}\right)\right\}\right).$$

Se pot verifica ușor egalitățile $L_{Ml}(G_i) = L_{Mg}(G_i) = M_i$, $i = 1,2,3$.

Așa cum se remarcă în [68], semnificativ este aici faptul că toate aceste limbaje și, prin urmare, toate relațiile sintactice subiacente pot fi tratate de către gramatici contextuale cu o utilizare maximală atât locală, cât și globală a selectorilor, deși, așa cum s-a mai menționat, puterea generativă a acestor gramatici nu este "prea mare". Dacă pentru gramatica G_2 avem $L_{in}(G_2) = L_{Mg}(G_2) = L_{Ml}(G_2)$ și, prin urmare, $M_2 \in CL_{in}(REG)$ [68], caracteristica de maximalitate este însă esențială pentru G_1 și G_3, întrucât, așa cum s-a mai arătat, limbajele M_1 și M_3 nu pot fi generate de gramatici contextuale care lucrează în modul *in*.

3.6.3. Concluzii

Gramaticile contextuale în general şi cele cu utilizare maximală (locală sau globală) a selectorilor în mod special sunt prezentate în [68] ca un model alternativ (în raport cu gramaticile lui Chomsky) pentru sintaxa limbajelor naturale. Iată doar câteva dintre principalele argumente folosite de autori [68] în acest sens:

- Familiile limbajelor contextuale nu sunt comparabile cu unele dintre familiile de bază ale ierarhiei Chomsky (cu LIN şi CF) sau ale rafinărilor acestei ierarhii (limbaje programate, limbaje indexate, limbaje generate de diverse clase de sisteme Lindenmayer).
- Gramaticile contextuale cu utilizare maximală globală a selectorilor nu pot genera toate limbajele bazate pe structuri centrat imbricate, operaţie realizată de gramaticile lineare Chomsky. Asemenea construcţii nu par foarte frecvente în limbile naturale.
- Gramaticile contextuale cu utilizare maximală (locală sau globală) a selectorilor pot genera, într-un mod foarte simplu, cele trei construcţii de bază nonindependente de context din limbile naturale: reduplicarea, dependenţele încrucişate şi acordurile multiple.
- Prin definiţie, gramaticile contextuale sunt nişte gramatici lexicalizate, fiecare dintre producţiile lor (pereche selector-context) constând numai din simboluri terminale. Acest fapt este în concordanţă cu multe tendinţe actuale ale sintaxei formale.
- Limbajele generate de gramaticile contextuale au proprietatea creşterii mărginite, care este specifică limbilor naturale: fiecare cuvânt generat de o gramatică contextuală - exceptând axiomele - este obţinut prin alăturarea unui context dintr-o mulţime finită.
- Din punctul de vedere al sensului cognitiv putem spune că un model bazat pe gramatici contextuale este mult mai apropiat de modul în care lucrează creierul uman. Aceasta deoarece operaţia de *alăturare* este probabil mai apropiată de felul în care operează creierul atunci când construieşte o propoziţie decât aceea de *rescriere*. Evident, astfel de consideraţii se înscriu în domeniul speculaţiilor, dar creierul uman este greu de imaginat ca folosind propoziţii intermediare auxiliare de tip

neterminal. În schimb, pare mult mai *natural*, în sensul propriu al cuvântului, să începem cu o colecție de propoziții considerate corect formate, acumulate, probabil, din experiență și să producem alte propoziții corect formate prin adăugarea de noi cuvinte, în perechi care pot surprinde dependențele și acordurile necesare. Această adăugare a cuvintelor se face în concordanță cu niște selectori specificați, care pot asigura corectitudinea gramaticală. Iată deci că modelul gramaticilor contextuale, demonstrat ca fiind indicat pentru sintaxa limbajelor naturale, ne apare ca unul extrem de firesc.

Descoperite de S. Marcus în 1969, gramaticile contextuale suscită și astăzi un viu interes, atât prin aceea că ele reprezintă un foarte bun model pentru studiul limbajelor naturale, cât și prin eforturile care se fac, tocmai datorită acestui fapt, cu scopul proiectării și implementării unor analizori sintactici eficienți bazați pe aceste gramatici. Existența unui parser de tip Earley (a se vedea capitolul 4) pentru gramatici contextuale cu selectori lineari, regulați și independenți de context, care lucrează în timp polinomial [31], parser a cărui implementare se află într-un stadiu încă perfectibil, aduce din nou în prim plan gramaticile contextuale ca instrument adecvat în modelarea și procesarea limbajului natural.

CAPITOLUL 4

ANALIZA SINTACTICĂ BAZATĂ PE CONSTITUENȚI

O operație esențială în domeniul procesării limbajului natural este aceea de a atribui structuri sintactice acelor propoziții care sunt considerate admisibile sau corecte în conformitate cu regulile unei gramatici date. Este un fapt unanim acceptat acela că obiectele lingvistice sunt niște obiecte structurate, care însă nu își dezvăluie structura în mod evident. Înțelegerea sensului unei propoziții depinde în mod crucial de abilitatea avută în regăsirea *structurii* acesteia. În cazul vorbitorului nativ al unei limbi acest lucru se produce cel mai adesea în mod inconștient. Un dispozitiv computațional care deduce structura pornind de la șiruri de cuvinte corecte din punct de vedere gramatical se numește *analizor sintactic*.

Pentru *a formaliza studiul structurii sintactice a unei propoziții*, sunt necesare două elemente: **gramatica** - o construcție formală specifică unei structuri lingvistice - și **tehnicile de analiză sintactică**, care determină corectitudinea unei propoziții în conformitate cu regulile gramaticii.

Tehnicile de analiză sintactică sunt deosebit de variate și diferă în funcție de tipul de gramatică utilizat. (Ne-am referit deja la analiza sintactică de dependență sau de tip WG în §3.5). Majoritatea algoritmilor existenți utilizează însă gramaticile PS independente de context și, prin urmare, efectuează o **analiză sintactică bazată pe constituenți**.

Procesul de analiză sintactică este numit, în literatura de specialitate, **parsing**, iar un analizor sintactic este cunoscut sub denumirea de **parser**[1]. (Întrucât studiul gramaticilor PS, care sunt cel mai frecvent utilizate, s-a dezvoltat în special relativ la limba engleză, acești termeni englezești au fost preluați de cea mai mare parte a literaturii de specialitate scrise în diverse limbi). Termenul de **parsing** a desemnat inițial *analiza gramaticală în sens clasic* (analiza sintactică și cea morfologică tradiționale), după care sensul acestui concept s-a extins, ajungând să se refere la o serie de operații relativ variate, dar care sunt, într-un fel sau altul, analoage celei tradiționale). Această extindere a sensului este rezultatul apariției unor noi conceptualizări în diverse domenii, cum ar fi lingvistica teoretică, teoria limbajelor formale, inteligența artificială și psiholingvistica.

---

[1] De la verbul englezesc *to parse*, care înseamnă "a separa o propoziție în părțile sale componente".

În lingvistică s-a făcut trecerea de la descrierile neformale, lipsite de norme riguroase, oferite de gramatica tradițională, spre o nouă viziune, care tratează aceste descrieri ca pe niște obiecte formale, în particular ca pe niște structuri alcătuite din constituenți. *Aceste descrieri gramaticale formalizate (în particular structuri alcătuite din constituenți) reprezintă ieșirea (output) operației de parsing.* Aceasta din urmă poate fi acum privită ca fiind mai degrabă algoritmică decât euristică. În această viziune, operația de parsing constă în aplicarea unui șir de principii caracteristice fiecărei limbi, adică a unui șir de reguli PS, care se aplică (reprezentării) unei propoziții astfel încât să se obțină toate descrierile gramaticale adecvate ale respectivei propoziții. Nici o altă descriere a propoziției, cu excepția celor adecvate, nu poate rezulta în urma acestui proces. În acest mod se naște posibilitatea de a lua în considerare principii de tipul $NP + VP = S$ fie din punct de vedere analitic, ca instrucțiuni de atribuire a unor structuri unor propoziții date dintr-o limbă naturală, fie din punct de vedere sintetic, ca instrucțiuni de compunere a propozițiilor unei limbi. Mulțimea tuturor acestor principii constituie o **gramatică formală** pentru limba respectivă, gramatică care poate fi privită ca având o *funcție analitică sau funcție de parsing*, ori o *funcție sintetică sau funcție generativă*. Prin **analiză sintactică bazată pe constituenți** vom înțelege o analiză gramaticală de acest tip, adică utilizând o gramatică formală de tipul descris. Trebuie remarcat faptul că, atât analiza sintactică, cât și cea semantică, încep aproape întotdeauna cu descompunerea propoziției în grupuri mai mici de cuvinte, cum ar fi grupurile nominale, grupurile verbale, grupurile adjectivale etc. Din punct de vedere sintactic, acest lucru este motivat de presupunerea că, analizând în termeni de grupuri sintactice, se poate obține un grad mai mare de generalizare în descrierea structurilor aparținând limbajului natural, decât analizând în termeni de cuvinte individuale.

În cele ce urmează, ne vom referi la analiza sintactică bazată pe constituenți[2] și vom studia, în principal, două mari tipuri de parsing. Atunci când se

---

[2] Constituenții vor fi presupuși ca reprezentând structuri alcătuite din elemente adiacente, acesta fiind cadrul în care au fost concepute tehnicile clasice de parsing. Problema tehnicilor de analiză sintactică specifice *constituenților discontinui*, care nu constituie obiectul lucrării de față, a suscitat, la rândul ei, interesul cercetătorilor și a fost studiată în special de către McCawley (1982), Tomita (1991, 1995) și Bunt (1991, 1995). Harry Bunt (1995) propune un algoritm de analiză sintactică cu hartă pentru Gramatici PS discontinue bazat pe ideea centrală că un constituent nu trebuie privit ca o secvență de cuvinte, ci ca o *mulțime de cuvinte* înzestrată cu anumite relații de precedență, relații care există atât între cuvintele mulțimii, precum și între acestea și cuvinte din afara ei. În elaborarea acestei idei noțiunea centrală este aceea de *arbore discontinuu* ca structură de date de tip arbore care descrie constituentul discontinuu [12].

caută structura propozițiilor care pot fi construite de către regulile unei gramatici PS, există două tipuri principale de strategii care pot fi aplicate:

- strategia **top-down** (adică "de sus în jos"), care pleacă de la simbolul S și caută, pe diverse căi, să rescrie simbolurile, până când se ajunge la propoziția căutată (sau până când se epuizează toate variantele de construcție, în cazul în care se caută structura unei propoziții incorecte din punctul de vedere al gramaticii enunțate);
- strategia **bottom-up** (adică "de jos în sus"), care pleacă de la cuvintele propoziției și, prin procedee de deplasare - reducere (căutând membrii drepți ai regulilor și înlocuindu-i cu membrii stângi corespunzători), urmărește construirea arborelui de derivare de jos în sus, până când se ajunge la simbolul rădăcină S.

Vom studia, pe rând, algoritmii determinați de aceste strategii, variante optimizate ale acestora, precum și implementările lor în Prolog. Acești algoritmi de analiză sintactică vor încerca diverse modalități de combinare a unor reguli și deci a constituenților, cu scopul de a ajunge la o combinație pe baza căreia să se poată construi un arbore de derivare corespunzător structurii secvenței de intrare, adică structurii propoziției analizate. Pentru început, nu vom fi interesați de construcția efectivă a arborelui, ci doar de răspunsul afirmativ sau negativ la întrebarea dacă este posibilă construcția unui astfel de arbore (deci dacă secvența de intrare poate fi generată sau nu de către gramatica dată).

Atât analizorul sintactic de tip top-down, cât și cel de tip bottom-up trebuie să aibă capacitatea de a efectua un proces de backtracking, adică de a încerca aplicarea unei reguli și, în cazul în care aceasta nu reușește, de a reveni și a încerca o regulă diferită. Un analizor sintactic care are capacitatea de a efectua backtracking este numit un **parser nedeterminist**. După cum se știe, în Prolog procesul de backtracking se efectuează în mod automat. Aceasta face ca Prologul să reprezinte un limbaj de programare ideal pentru implementarea analizorilor sintactici de acest tip.

Specificăm faptul că intrarea pentru un parser va fi numită, în cele ce urmează, "șirul de intrare", chiar dacă în Prolog aceasta reprezintă, de obicei, o listă de atomi și nu un șir de caractere. În fine, pentru a implementa un parser, va fi necesară rescrierea gramaticii ca o mulțime de clauze Prolog (a se vedea §3.4).

4.1. Analiza sintactică top-down

O modalitate de a efectua analiză sintactică bazată pe constituenți de tip top-down este aceea de a conferi fiecărei reguli PS o interpretare procedurală.

Spre exemplu, regula

$$S \rightarrow NP\ VP$$

poate fi interpretată ca spunând: "pentru a analiza (izola) un S, analizează (izolează) mai întâi un NP şi apoi un VP". În acelaşi timp, o regulă de tipul

$$N \rightarrow mama$$

poate fi interpretată ca spunând: "pentru a analiza (izola) un N (substantiv), acceptă cuvântul *mama* din şirul de intrare". Procesul care ia naştere în virtutea acestor interpretări este adesea numit *coborâre recursivă*, deoarece fiecare regulă PS este transformată într-o procedură, iar aceste proceduri se pot apela una pe cealaltă, ori se pot autoapela în mod recursiv.

Procesul de analiză sintactică (parsing) devine, în acest context, un proces similar cu cel al satisfacerii interogărilor Prolog. Scopul iniţial este S; unele dintre reguli transformă acest scop în alte scopuri, iar alte reguli satisfac scopuri acceptând cuvinte. La fiecare pas al algoritmului analizorul sintactic fie transformă scopul curent într-un nou scop, fie satisface scopul curent prin acceptarea unui cuvânt. În anumite situaţii, analizorul sintactic revine asupra unei acţiuni anterioare, efectuând un proces de backtracking, aşa cum se va vedea în exemplul care urmează. În toate aceste privinţe, analiza sintactică top-down seamănă foarte mult cu modul în care Prologul rezolvă interogările. Se pare că aceasta nu reprezintă o coincidenţă, întrucât analiza sintactică bazată pe constituenţi a fost una dintre aplicaţiile pe care Alain Colmerauer le-a avut în vedere atunci când a conceput Prologul (Colmerauer 1978). Mai mult, formalismul DCG (în traducerea standard) implementează un dispozitiv de recunoaştere sau analizor sintactic de tip top-down şi care lucrează de la stânga la dreapta. Vom reveni asupra implementării Prolog după discutarea algoritmului propriu-zis.

Aşa cum s-a menţionat deja, strategia top-down pleacă de la simbolul S şi încearcă, pe diverse căi, să rescrie simbolurile, până când se ajunge la propoziţia căutată (dacă aceasta este acceptată de gramatica dată). Cu alte cuvinte, un parser top-down începe cu simbolul S şi încearcă să îl rescrie pe acesta sub forma unui şir de simboluri terminale care sunt în concordanţă cu acele cuvinte ce alcătuiesc şirul de intrare din punctul de vedere al categoriei fiecărui cuvânt. *Categoriile cuvintelor* pot fi indicate fie prin intermediul regulilor lexicale ale gramaticii, fie prin utilizarea unui lexicon (caz în care nu mai este necesar ca gramatica să conţină reguli lexicale). *Starea analizei* la un anumit moment este caracterizată şi poate fi reprezentată printr-o listă de simboluri constituind rezultatul operaţiilor efectuate până la acel moment, numită *lista simbolurilor*. Spre exemplu, analizorul începe din starea (S), iar după aplicarea regulii $S \rightarrow NP\ VP$ lista simbolurilor devine (NP VP). Dacă imediat după aceea se aplică regula $NP \rightarrow ART\ N$, atunci lista simbolurilor va fi (ART N VP)

ș.a.m.d. Analizorul ar putea continua în acest mod până când starea constă în întregime din simboluri terminale, după care ar putea verifica dacă șirul de intrare coincide cu ceea ce s-a obținut. Dar, un algoritm construit în acest mod ar fi extrem de neperformant, întrucât preia eventuale erori comise în utilizarea regulilor de rescriere, erori pe care nu le depistează decât mult mai târziu. Un analizor sintactic performant trebuie să facă verificarea șirului de intrare de îndată ce acest lucru este posibil.

Presupunând că este dată Gramatica nr.1 la care ne-am referit anterior, adică gramatica următoare (în notația PATR)

Regulă

$$S \rightarrow NP\ VP.$$

Regulă

$$VP \rightarrow V\ NP.$$

Regulă

$$VP \rightarrow V.$$

Cuvânt Dr. Popescu:

<cat> = NP.

Cuvânt surori:

<cat> = NP.

Cuvânt Colentina:

<cat> = NP.

Cuvânt pacienți:

<cat> = NP.

Cuvânt omoară:

<cat> = V.

Cuvânt angajează:

<cat> = V.

și că șirul de intrare este

Colentina angajează surori.

un *analizor sintactic top-down care procesează de la stânga la dreapta și efectuează o căutare de tip depth-first* va lucra, în mare, în felul următor:

Se caută un S.
Din ce poate să constea un S?
Un S poate să constea dintr-un NP urmat de un VP.
Se caută un NP.
Din ce poate să constea un NP?
Gramatica nu conține reguli pentru rescrierea lui NP.

Există o intrare lexicală care desemnează "surori" ca pe un membru al categoriei NP.

Este primul cuvânt al şirului de intrare "surori"?

Nu.

Există o intrare lexicală care desemnează "Colentina" ca pe un membru al categoriei NP.

Este primul cuvânt al şirului de intrare "Colentina"?

Da.

S-a găsit un NP constând din cuvântul "Colentina".

Se caută un VP.

*** Din ce poate să constea un VP?

Un VP poate să constea dintr-un V.

Se caută un V.

Din ce poate să constea un V?

Nu există reguli de rescriere referitoare la V.

Există o intrare lexicală care desemnează "omoară" ca pe un membru al categoriei V.

Este următorul cuvânt al şirului de intrare "omoară"?

Nu.

Există o intrare lexicală care desemnează "angajează" ca pe un membru al categoriei V.

Este următorul cuvânt al şirului de intrare "angajează"?

Da.

S-a găsit un V constând din cuvântul "angajează".

S-a găsit un VP constând dintr-un V care constă din cuvântul "angajează".

S-a găsit un S constând din:

un NP constând din cuvântul "Colentina" şi un VP constând dintr-un V constând din cuvântul "angajează".

S-a ajuns la sfârşitul şirului de intrare?

Nu.

Se revine la *** şi se ia o altă decizie (*backtracking*).

Se caută din nou un VP.

Din ce poate să constea un VP?

Un VP mai poate să constea dintr-un V urmat de un NP.

Se caută un V.

Din ce poate să constea un V?

ş.a.m.d.

Analizorul sintactic va procesa şirul de intrare acceptând cuvintele unul câte unul. Dacă **C** reprezintă tipul de constituent pe care analizorul sintactic îl

caută la un anumit moment (S, NP, V etc.), atunci algoritmul pentru analizarea unui constituent de tipul C este următorul:

Algoritmul 4.1

- Dacă C reprezintă un cuvânt individual, el este căutat în regulile lexicale ale gramaticii și acceptat din șirul de intrare.
- Altfel, se caută în regulile PS ale gramaticii pentru a-l rescrie pe C sub forma unei liste de constituenți și se analizează acei constituenți unul câte unul.

4.1.1. Algoritm de analiză sintactică top-down

O *stare a analizei* poate fi definită și printr-o pereche constând din lista simbolurilor (așa cum a fost ea descrisă anterior) și un număr ce indică poziția curentă în cadrul propoziției. Pozițiile se marchează între cuvinte, 1 fiind poziția din fața primului cuvânt. Spre exemplu, în cazul propoziției

$_1$ *Colentina* $_2$ *angajează* $_3$ *surori* $_4$

o posibilă stare este ((N VP)1), care indică faptul că analizorul sintactic trebuie să găsească un N urmat de un VP începând de la poziția 1. Noile stări sunt generate din cele vechi în funcție de cum primul simbol este sau nu simbol lexical. Dacă acesta reprezintă un simbol lexical (cum ar fi N din exemplul anterior) și dacă următorul cuvânt din șirul de intrare poate aparține respectivei categorii lexicale, atunci starea poate fi actualizată prin eliminarea primului simbol și actualizarea contorului de poziție.

Presupunând că sunt date gramatica PS independentă de context

$S \rightarrow NP\ VP$

$NP \rightarrow ART\ N$

$NP \rightarrow ART\ ADJ\ N$

$VP \rightarrow V$

$VP \rightarrow V\ NP$

și lexiconul (în notație PATR)

Cuvânt latră:
<cat> = V.
Cuvânt câine:
<cat> = N.
Cuvânt un:
<cat> = ART.

sau, mai simplu,

latră: V
câine: N
un: ART

și că șirul de intrare este reprezentat de propoziția

$_1$ *Un* $_2$ *câine* $_3$ *latră* $_4$

o stare tipică a analizei poate fi ((N VP)2). Această stare indică faptul că analizorul caută un N urmat de un VP pornind de pe poziția 2. Următoarea stare a analizorului va fi generată ținându-se cont de faptul că primul simbol este unul lexical (N). Deoarece cuvântul "câine" de la poziția 2 apare în lexicon ca fiind un N (substantiv), generarea stării următoare se va face prin eliminarea primului simbol și repoziționarea contorului. Aceasta va fi starea ((VP)3), care indică faptul că analizorul trebuie să găsească un VP pornind de la poziția 3. Dacă primul simbol este un neterminal (cum ar fi acest VP), atunci el este rescris utilizând o regulă a gramaticii. Spre exemplu, utilizând a patra regulă a gramaticii amintite noua stare ar fi

((V)3),

în timp ce prin utilizarea celei de a cincea reguli s-ar genera noua stare

((V NP)3).

Un algoritm de analiză sintactică complet trebuie să exploreze în mod sistematic toate noile stări posibile, tehnica care asigură cel mai bine acest lucru fiind aceea de backtracking. În această abordare, se vor genera toate noile stări posibile. Una dintre acestea va fi aleasă pentru a reprezenta starea următoare, iar celelalte vor fi salvate ca *stări de revenire* sau *stări backup*. Dacă se ajunge în situația în care starea curentă nu poate conduce la o soluție, se alege o nouă stare curentă din lista stărilor backup.

Un **algoritm de analiză sintactică top-down** lucrează, prin urmare, cu o listă de stări posibile, numită **lista posibilităților**. Primul element al acestei liste este **starea curentă**, care constă din o listă a simbolurilor și o poziție în cadrul propoziției, iar elementele rămase reprezintă **stările backup**. Spre exemplu, lista posibilităților

(((N)2) ((NUME)1) ((ADJ N)1))

indică pe post de stare curentă acea stare care constă din lista de simboluri (N) de la poziția 2 și stabilește faptul că există două posibile stări backup: una constând din lista de simboluri (NUME) la poziția 1 și cealaltă constând din lista de simboluri (ADJ N) la poziția 1. Algoritmul de analiză sintactică top-down este

Algoritmul 4.2

Se pleacă din starea inițială ((S)1), fără stări backup. Pașii algoritmului sunt:

1. *Selectează* starea curentă: se scoate prima stare din lista de posibilități; fie ea starea **C**. Dacă lista de posibilități este vidă, atunci algoritmul eșuează (i.e. nu se poate face cu succes analiza sintactică, în sensul că secvența de intrare este respinsă ca nefiind conformă cu gramatica limbajului) și **STOP**.

2. Dacă **C** are o listă de simboluri vidă, iar contorul indică poziția de la sfârșitul propoziției, atunci algoritmul are succes (i.e. propoziția este acceptată ca fiind corectă) și **STOP**.

3. Altfel, *generează* următoarele stări posibile. (Adăugarea acestor stări în lista posibilităților se va face în funcție de modul de organizare al listei, adică de modul în care se efectuează căutarea).

> **3.1.** Dacă primul simbol din lista de simboluri a lui **C** este unul lexical și următorul cuvânt al propoziției aparține acelei categorii lexicale, atunci se creează o nouă stare prin înlăturarea primului simbol din lista de simboluri și repoziționarea contorului de poziție. Starea nou creată se adaugă listei posibilităților.
>
> **3.2.** Altfel (i.e. primul simbol din lista de simboluri a lui **C** este un neterminal), se generează o nouă stare corespunzător fiecărei reguli a gramaticii care poate rescrie acest neterminal. Se adaugă listei posibilităților noile stări generate.

Acest tip de analiză sintactică bazată pe constituenți poate fi privită și ca un caz particular de **problemă de căutare**, așa cum este definită aceasta în inteligența artificială. În particular, analizorul sintactic top-down a fost descris de către noi în termenii următoarei *proceduri de căutare generalizate*: lista posibilităților este la început setată ca reprezentând starea inițială a analizei; se repetă pașii care urmează până când se înregistrează succes sau eșec. Pașii sunt:

1. Este selectată prima stare din lista posibilităților (și înlăturată din această listă).

2. Se generează noile stări prin încercarea fiecărei opțiuni posibile corespunzător stării selectate. (Se poate ca aceste noi stări să nu existe dacă ne aflăm pe un drum greșit).

3. Se adaugă stările generate la pasul **2** listei posibilităților.

În cazul organizării unei strategii de tip **depth-first**, lista posibilităților este o **stivă**. Cu alte cuvinte, pasul 1 preia întotdeauna primul element al listei, iar la pasul 3 noile stări generate sunt întotdeauna plasate la începutul listei, conducând la o strategie de tip LIFO. Prin contrast, în cazul organizării unei

strategii de tip **breadth-first**, lista posibilităților este manipulată ca o **coadă**. Pasul 3 adaugă noile stări generate la sfârșitul listei, conducând la o strategie de tip FIFO.

Diferența dintre cele două abordări este esențială. Astfel, în cadrul strategiei depth-first, o anumită interpretare este luată în considerare și extinsă până când eșuează; numai după aceea este studiată o a doua. În cazul strategiei breadth-first ambele interpretări sunt luate în considerare alternativ. Cu alte cuvinte, căutarea de tip depth-first studiază, pe rând, câte o singură ipoteză. Căutarea de tip breadth-first studiază ipotezele în paralel. O strategie de tip depth-first adesea avansează cu mare viteză către o soluție, dar în alte ocazii poate consuma suficient de mult timp urmând niște piste false. Strategia breadth-first explorează fiecare soluție posibilă până la o anumită adâncime înainte de a avansa. Mulți analizori sintactici folosiți în prezent utilizează strategia depth-first, care tinde să minimizeze numărul stărilor backup necesare. Prin aceasta, ea utilizează mai puțină memorie și necesită ținerea unei evidențe de mai mică amploare (spre deosebire de căutarea de tip breadth-first, care necesită scrierea unor programe ce pot manipula în mod explicit liste de stări alternative). Implementarea în Prolog a analizorilor sintactici ce utilizează strategia depth-first este, de asemenea, mult mai simplă, întrucât aceasta se bazează în mod esențial pe facilitatea Prologului de a efectua backtracking automat și deci căutare de tip depth-first.

Vom exemplifica, în continuare, analiza sintactică top-down efectuată cu Algoritmul 4.2 și implementând o strategie de căutare de tip depth-first, în cazul propoziției

$_1$ *Un* $_2$ *câine* $_3$ *latră* $_4$

Se utilizează gramatica și lexiconul definite anterior în legătură cu această propoziție. Pașii analizei sunt următorii:

Pas	Stare curentă	Stări backup	Comentarii
1.	((S)1)	-	Poziția inițială
2.	((NP VP)1)	-	Se rescrie S cu regula 1
3.	((ART N VP)1)	((ART ADJ N VP)1)	Se rescrie NP cu regulile 2 și 3
4.	((N VP)2)	((ART ADJ N VP)1)	Se reduce ART cu "Un"
5.	((VP)3)	((ART ADJ N VP)1)	Se reduce N cu "câine"
6.	((V)3)	((V NP)3) ((ART ADJ N VP)1)	Se rescrie VP cu regulile 4 și 5
7.	(()4)	((V NP)3) ((ART ADJ N VP)1)	Se reduce V cu "latră"
8.	**STOP**	-	Propoziție corectă

Exemplul a fost preluat din [4] şi adaptat limbii române. Pentru un al doilea exemplu în limba română vezi [6], iar pentru alte exemple în limba engleză vezi [4].

Analiza sintactică top-down, atunci când este efectuată de la stânga la dreapta, ca în exemplele discutate, va avea dificultăţi cu acele reguli care prezintă *recursivitate la stânga*. O astfel de regulă este, de pildă,

$$VP \rightarrow VP\ NP\ .$$

Dacă o gramatică admite reguli recursive la stânga, un parser top-down, care lucrează de la stânga la dreapta, va intra într-un ciclu infinit atunci când face alegeri greşite. Iar dacă se cer toate structurile posibile ale propoziţiei, astfel de decizii sunt inerente, întrucât analizorul trebuie să facă toate alegerile posibile.

În cele ce urmează, pentru a facilita înţelegerea implementării în limbajul Prolog a analizei sintactice top-down vom privi o stare ca fiind caracterizată exclusiv de o secvenţă de scopuri şi de o altă secvenţă de cuvinte rămase.

4.1.2. Implementare Prolog

Implementarea Prolog, la care ne-am referit şi la începutul acestei prezentări, este extrem de firească în ceea ce priveşte analizorii sintactici sau dispozitivele de recunoaştere top-down, care lucrează de la stânga la dreapta şi utilizează o strategie de căutare de tip depth-first. Un parser care foloseşte, din nou, lista diferenţă este următorul:

```
recunoaste(Categorie,S1,S2):-
        regula(Categorie,Fii),imperecheaza(Fii,S1,S2).
```

Prin urmare, un şir poate fi recunoscut ca o realizare a lui **Categorie** dacă există o regulă care permite lui **Categorie** să domine **Fii** şi dacă **Fii** se împerechează cu şirul respectiv. **Prin definiţie**, vom considera că o listă de categorii se împerechează cu un şir dacă:

- ambele sunt vide;
- ambele constau dintr-un singur cuvânt;
- o porţiune iniţială a şirului poate fi recunoscută ca reprezentând prima categorie din listă, iar restul listei de categorii se împerechează cu restul şirului.

Conform acestei descrieri, predicatul **imperecheaza** se defineşte în Prolog în felul următor:

```
imperecheaza([],S,S).
imperecheaza([Cuvant],[Cuvant|S],S).
imperecheaza([Categorie|Categorii],S1,S3):-
```

```
                    recunoaste(Categorie,S1,S2),
                    imperecheaza(Categorii,S2,S3).
```

În acest caz, regulile gramaticii trebuie date sub forma

```
regula(s,[np,vp]).
```

corespunzător regulilor PS și respectiv sub forma

```
regula(np,[surori]).
regula(v,[angajeaza]).
```

în cazul regulilor lexicale.

Interogarea se face în felul următor:

```
?- recunoaste(s,['Colentina',angajeaza,surori],[]).
```

Răspunsul sistemului corespunzător interogării de mai sus va fi

```
yes.
```

Dacă gramatica nu conține reguli lexicale, ci este atașat un **lexicon**, atunci regulile PS ale gramaticii vor fi reprezentate în același mod, iar intrările lexicale vor fi reprezentate sub forma

```
cuvant(np,surori).
cuvant(v,angajeaza).
```

În acest caz, utilizând Algoritmul 4.1, putem programa analizorul sintactic folosind predicatele **parse** și **parse_lista**. Predicatul **parse** analizează un constituent având categoria C, începând cu șirul de intrare S1 și sfârșind cu șirul de intrare S. În definirea acestui predicat vom ține seama de cele două situații prevăzute de Algoritmul 4.1. Predicatul **parse_lista** este identic cu predicatul **parse**, cu excepția faptului că se referă la o listă de constituenți și îi analizează pe fiecare în parte. Interogarea Prologului se realizează în același mod. Analizorul sintactic corespunzător Gramaticii nr.1, însoțite de **lexiconul** aferent (și în care corespondența dintre predicatele **parse** și **recunoaste** și respectiv **parse_lista** și **imperecheaza** este evidentă) va fi dat de următorul program scris în SICStus Prolog:

Programul 4.1

```
parse(C,[Cuvant|S],S):-cuvant(C,Cuvant).
parse(C,S1,S):-regula(C,Cs),parse_lista(Cs,S1,S).

parse_lista([C|Cs],S1,S):-parse(C,S1,S2),
                          parse_lista(Cs,S2,S).
parse_lista([],S,S).
```

```
regula(s,[np,vp]).
regula(vp,[v]).
regula(vp,[v,np]).

cuvant(np,'Colentina').
cuvant(np,'Dr.Popescu').
cuvant(np,pacienti).
cuvant(np,surori).
cuvant(v,omoara).
cuvant(v,angajeaza).
```

4.2. Analiza sintactică bottom-up

Deşi traducerea standard în programe Prolog a gramaticilor DC dă naştere unor analizori sintactici top-down, aceasta nu reprezintă, în nici un caz, unica traducere posibilă. Pe de altă parte, o importantă limitare a unui parser top-down este aceea că se ajunge la ciclare atunci când acesta tratează reguli recursive la stânga de tipul $A \rightarrow AB$, reguli care, de altfel, apar în limbajul natural. Un exemplu de astfel de regulă este

$$NP \rightarrow NP\ Conj\ NP$$

unde *Conj* este o conjuncţie de tipul şi/sau.

O modalitate de a trata regulile recursive la stânga o reprezintă utilizarea unui parser de tip *bottom-up*, care acceptă cuvinte şi încearcă să le combine în constituenţi, utilizând regulile unei gramatici date. Aşa cum s-a mai specificat, analiza sintactică care utilizează o *strategie bottom-up* (adică "de jos în sus") pleacă de la cuvintele propoziţiei şi, prin procedee de deplasare-reducere (căutând membrii drepţi ai regulilor şi înlocuindu-i cu membrii stângi corespunzători), urmăreşte construirea arborelui de derivare de jos în sus, până când se ajunge la simbolul rădăcină S.

În cazul aceleiaşi propoziţii oferite spre exemplificare

Un câine latră.

analizorul sintactic bottom-up lucrează în felul următor:

Se acceptă un cuvânt al propoziţiei - *un*.
Cuvântul *un* este un ART.
Se acceptă un alt cuvânt - *câine*.
Cuvântul *câine* este un N.

Un ART şi un N alcătuiesc un NP.
Se acceptă un nou cuvânt - *latră.*
Cuvântul *latră* este un V.
Un V singur constituie un VP.
Un NP şi un VP alcătuiesc un S.

O trăsătură importantă a acestui parser este aceea că, întrucât acţiunile sale sunt determinate numai de cuvinte efectiv găsite în cadrul propoziţiei, el nu prezintă limitarea de a cicla atunci când intervin reguli recursive la stânga. Acest analizor sintactic suferă, în schimb, de o altă limitare, aceea de a nu putea trata reguli de tipul

$$A \rightarrow \phi$$

deoarece nu dispune de nici o modalitate de a reacţiona în faţa unui constituent nul (care lipseşte).

4.2.1. Algoritm de deplasare-reducere

Analiza sintactică de tip bottom-up este adesea numită **parsing cu deplasare-reducere** [19]. Ea constă din două operaţii de bază: *deplasarea* cuvintelor în interiorul unei stive şi respectiv *reducerea* conţinutului stivei. Dăm, în continuare, analiza, privită în aceşti termeni, a aceluiaşi exemplu, adică a propoziţiei

Un câine latră.

Pas	Acţiune	Stivă	Şir de intrare
	START		*un câine latră*
1	Deplasare	*un*	*câine latră*
2	Reducere	*ART*	*câine latră*
3	Deplasare	*ART câine*	*latră*
4	Reducere	*ART N*	*latră*
5	Reducere	*NP*	*latră*
6	Deplasare	*NP latră*	-
7	Reducere	*NP V*	-
8	Reducere	*NP VP*	-
9	Reducere	*S*	-

Algoritmul de deplasare-reducere poate fi, prin urmare, formulat în felul următor:

Algoritmul 4.3

1. Se deplasează un cuvânt în stivă.

2. Se reduce stiva în mod repetat, utilizând intrările lexicale și regulile PS, până când această operație nu mai este posibilă.

3. Dacă mai există cuvinte în șirul de intrare se merge la Pasul 1; altfel **STOP**.

Este de remarcat faptul că, un parser bazat pe acest algoritm, spre deosebire de cel top-down, nu lucrează în manieră predictivă, în sensul că nu se așteaptă, la nici un pas al algoritmului, să găsească un constituent de un anumit tip.

4.2.2. Implementare Prolog

O implementare eficientă în Prolog a parsing-ului cu deplasare-reducere necesită, în esență, utilizarea a două tipuri de tehnici, după cum urmează:

1. *Construcția stivei se face înapoi*, prin aceasta înțelegându-se că se deplasează cuvintele de la *începutul* șirului de intrare la *începutul* stivei, ca în exemplul:

```
    START:      [ ]              [un,caine,latra]
Deplasare:      [un]             [caine,latra]
 Reducere:      [art]            [caine,latra]
Deplasare:      [caine,art]      [latra]
 Reducere:      [n,art]          [latra]
 Reducere:      [np]             [latra]
Deplasare:      [latra,np]       [ ]
 Reducere:      [v,np]           [ ]
 Reducere:      [vp,np]          [ ]
 Reducere:      [s]              [ ]
```

Această tehnică este eficientă deoarece face ca toate acțiunile să aibă loc la începutul fiecărei liste. Listele nu trebuie, prin urmare, parcurse, pentru a se ajunge la sfârșitul lor.

2. *Regulile se memorează înapoi.* Spre exemplu, regula

$$NP \rightarrow ART\ N$$

va fi transpusă în Prolog sub forma

```
iregula([n,art|X],[np|X]).
```

(Denumirea "iregula" sugerează o "regulă inversă" sau "regulă înapoi"). Ca urmare, pasul 2 al algoritmului anterior (pasul de reducere) se va executa mult mai rapid. Primul argument al regulii menționate, `[n,art|X]` coincide cu vârful stivei căreia i se aplică această regulă, în timp ce `[np|X]` reprezintă ceea ce devine stiva după reducere. Prin urmare, pasul de reducere al analizorului este foarte simplu: dacă există o regulă aplicabilă, aceasta este utilizată, după care se încearcă efectuarea unei noi reduceri; altfel, stiva este lăsată neschimbată. Vom reveni asupra unei variante mai performante a acestui algoritm, variantă care utilizează o structură de date numită "hartă".

Un **parser bottom-up cu deplasare-reducere**, care se bazează pe cele două tehnici amintite, de construcție a stivei și respectiv de memorare a regulilor, și care folosește Gramatica nr.1, este prezentat în continuare. Analizorul sintactic utilizează predicatul **parse (S, Rezultat)**, care analizează șirul de intrare S, pe care îl reduce la lista de categorii Rezultat. Programul corespunzător, scris în SICStus Prolog, este următorul:

Programul 4.2

```
parse(S,Rezultat):-depl_red(S,[],Rezultat).

% Predicatul depl_red(S,Stiva,Rezultat) analizeaza sirul de
% intrare S, unde Stiva este lista categoriilor analizate
% pana la momentul curent. Atunci cand sirul de intrare
% devine vid, inseamna ca toate cuvintele au fost deplasate
% in stiva si reduse, deci stiva contine chiar rezultatul.
% Predicatul depl_red se defineste recursiv astfel:

depl_red(S,Stiva,Rezultat):-deplasare(Stiva,S,StivaNoua,S1),
                            reducere(StivaNoua,StivaRedusa),
                            depl_red(S1,StivaRedusa,Rezultat).
depl_red([],Rezultat,Rezultat).

% OBSERVATIE: predicatul deplasare esueaza daca sirul de in-
% trare este vid. Sirul de intrare devine vid, atunci cand
% toate cuvintele sale au fost deplasate in stiva si reduse,
% deci in cazul tratat de clauza depl_red anterioara.
% Altfel, in cazul general, predicatul deplasare se defi-
% neste astfel:
```

```
deplasare(X,[H|Y],[H|X],Y).

% Predicatul reducere(Stiva,StivaRedusa) reduce in mod repe-
% tat varful stivei, pentru a forma constituenti mai putini,
% dar mai ampli. Definitia recursiva este:

reducere(Stiva,StivaRedusa):-iregula(Stiva,Stiva1),
                                reducere(Stiva1,StivaRedusa).
reducere(Stiva,Stiva).

% Definitia a tinut cont de cazul cand nu exista nici o
% iregula, deci nu se poate efectua nici o reducere. La
% clauza reducere(Stiva,Stiva). se ajunge atunci cand nu se
% poate aplica nici o iregula si stiva ramane neschimbata.

% Regulile PS ale gramaticii se memoreaza tot inapoi.

iregula([vp,np|X],[s|X]).
iregula([v|X],[vp|X]).
iregula([np,v|X],[vp|X]).

% Pentru ca lexiconul sa poata fi utilizat este nevoie, in
% plus, de specificarea urmatoarei "reguli inapoi":

iregula([Cuvant|X],[Cat|X]):-cuvant(Cat,Cuvant).

% Lexiconul este:

cuvant(np,'Dr.Popescu').
cuvant(np,'Colentina').
cuvant(np,surori).
cuvant(np,pacienti).
cuvant(v,omoara).
cuvant(v,angajeaza).
```

Interogarea Prologului se face astfel:

```
?- parse(['Colentina',angajeaza,surori],[s]).
```

Răspunsul sistemului va fi **yes**, ceea ce arată că **[s]** reprezintă, într-adevăr, lista categoriilor la care se reduce șirul de intrare. Cu alte cuvinte, propoziția

Colentina angajează surori.

a fost analizată sintactic cu succes, în sensul că ea este acceptată ca fiind corectă (în conformitate cu gramatica dată). O interogare a Prologului de tipul

```
?- parse(['Colentina',angajeaza,surori],[X]).
```

are ca răspuns

```
X=s?
```

Dacă, în continuare, se apasă tasta <Enter>, răspunsul sistemului este **yes**, ceea ce arată că şirul de intrare se poate reduce la categoria **s** şi deci reprezintă o propoziţie corectă în conformitate cu gramatica dată. Dacă, în loc de a se apăsa tasta <Enter>, se tastează "punct şi virgulă" [;], răspunsul sistemului este **no**, cu sensul că şirul de intrare se reduce la categoria **S** şi că aceasta este unica soluţie posibilă.

4.3. Analiza sintactică din colţul stâng

Aşa-numita "analiză sintactică din colţul stâng" este adesea descrisă ca fiind de tip bottom-up, dar ea reprezintă, în realitate, o combinaţie a strategiilor bottom-up şi top-down [19]. Această tehnică a fost prezentată de către Rosenkrantz şi Lewis (1970), precum şi de către Aho şi Ullman (1972), toţi aceşti autori atribuindu-i-o lui Irons (1961).

Ideea centrală care stă la baza acestui tip de analiză este aceea de a accepta un cuvânt, a determina tipul de constituent al cărui început îl marchează cuvântul respectiv şi a analiza apoi restul acelui constituent în manieră top-down. Arborele de derivare corespunzător propoziţiei este astfel "descoperit" începând din colţul din stânga jos.

Ca şi un parser top-down, un analizor sintactic "din colţul stâng" se aşteaptă întotdeauna să găsească un anumit tip de constituent şi, prin urmare, ştie că numai câteva dintre regulile gramaticii sunt relevante. Aceasta îl face mult mai eficient decât analizorul sintactic de tip bottom-up descris anterior. În plus, ca şi acesta din urmă, el poate trata regulile recursive la stânga fără a cicla, deoarece începe analiza fiecărui constituent prin acceptarea unui cuvânt din şirul de intrare.

Algoritmul de analiză sintactică corespunzător constă din două părţi distincte, şi anume: acceptarea şi identificarea unui cuvânt, urmate de formarea constituentului corespunzător. Algoritmul pentru analiza *unui constituent de tipul C* este următorul:

Algoritmul 4.4

1. Se acceptă un cuvânt din şirul de intrare şi i se determină categoria. Fie această categorie **W**.

2. Se completează **C** astfel: dacă **W=C**, atunci **STOP**;

altfel:

- cu ajutorul regulilor gramaticii se găseşte un constituent a cărui dezvoltare începe cu **W**; fie **P** acest constituent[3];
- se analizează în mod recursiv şi utilizând strategia **top-down** toate elementele rămase ale dezvoltării lui **P**;
- se înlocuieşte **W** cu **P** şi se merge înapoi la începutul pasului 2 (pentru a continua completarca lui **C**).

4.3.1. Implementare Prolog

Implementarea Prolog va utiliza predicatele **parse**, **parse_lista** şi **completeaza**.

Predicatul **parse(C, S1, S)** analizează un constituent de categorie **C**, începând cu şirul de intrare S1 şi încheind cu şirul de intrare S. Analiza se face conform Algoritmului 4.4, deci presupune acceptarea unui cuvânt de categorie **W** din şirul de intrare, urmată de completarea constituentului de tipul **C**:

```
parse(C,[Cuvant|S2],S):- cuvant(W,Cuvant),
                         completeaza(W,C,S2,S).
```

Predicatul **completeaza (W, C, S1, S)** verifică dacă **W** poate fi primul subconstituent al lui **C**, apoi analizează restul constituentului de tipul **C**. Dacă **C=W**, analiza se încheie

```
completeaza(C,C,S,S).
```

altfel continuă în mod recursiv:

```
completeaza(W,C,S1,S):- regula(P,[W|Rest]),
                        parse_lista(Rest,S1,S2),
                        completeaza(P,C,S2,S).
```

---

[3] Spre exemplu, dacă categoria **W** este *Det*, iar gramatica conţine regula PS *NP→Det N*, atunci se utilizează această regulă, care precizează pe Det N ca reprezentând o posibilă dezvoltare a lui NP şi se determină **P** ca fiind grup nominal (NP).

Lista de constituenți Rest este analizată urmând strategia clasică top-down și, prin urmare, se utilizează același predicat **parse_lista** care a fost definit cu prilejul implementării acestui tip de analiză sintactică (§4.1.2.). Regulile PS ale gramaticii și lexiconul aferent vor fi reprezentate tot ca în cazul analizorului sintactic top-down. În aceste condiții, interogarea Prologului se face, de asemenea, în același mod (§ 4.1.2.).

În cele ce urmează, prezentăm un parser (imperfect), care efectuează analiză sintactică din colțul stâng, corespunzător Gramaticii nr.2. Programul aferent, scris în SICStus Prolog, este următorul:

Programul 4.3

```
parse(C,[Cuvant|S2],S):-cuvant(W,Cuvant),
                         completeaza(W,C,S2,S).


parse_lista([C|Cs],S1,S):-parse(C,S1,S2),
                           parse_lista(Cs,S2,S).
parse_lista([],S,S).


completeaza(C,C,S,S).
completeaza(W,C,S1,S):-regula(P,[W|Rest]),
                       parse_lista(Rest,S1,S2),
                       completeaza(P,C,S2,S).


regula(s,[np,vp]).
regula(vp,[v,np]).
regula(np,[det,n]).


cuvant(det,un).
cuvant(det,niste).
cuvant(det,o).
cuvant(n,elev).
cuvant(n,elevi).
cuvant(n,carte).
cuvant(v,iubeste).
cuvant(v,iubesc).
```

4.3.1.1. Legături

Aşa cum a fost descris până în prezent, conform [19], parser-ul care efectuează analiză sintactică din colţul stâng poate numai parţial să trateze regulile de tipul $A \rightarrow \phi$, adică acele reguli care introduc constituenţi nuli. În esenţă, putem spune că nu va fi nici o problemă atunci când constituentul nul este întâlnit în timp ce se analizează sintactic top-down, întrucât parser-ul descris lucrează, în momentul respectiv, exact ca un analizor sintactic de acest tip. Aceasta va fi situaţia atunci când se aplică reguli de forma

$$A \rightarrow B \; C$$
$$C \rightarrow \phi$$

deoarece constituentul C va fi analizat top-down. Trebuie doar să i se specifice analizorului sintactic că, în situaţia în care se caută un C, se poate merge mai departe fără a analiza nimic. Această specificare poate fi realizată printr-o codificare a regulilor de tipul

```
regula(a,[b,c]).
regula(c,[]).
```

şi prin adăugarea următoarei clauze Prolog referitoare la predicatul **parse**:

```
parse(C,S2,S):-regula(W,[]),completeaza(W,C,S2,S).
```

(Această clauză spune că, dacă W este o categorie care poate fi reprezentată printr-un constituent nul, atunci este permisă oricând completarea lui W practic neîntreprinzând nici o acţiune).

Această completare a analizorului sintactic descris nu este însă suficientă pentru limbi ca engleza sau româna, limbi în care o regulă de tipul

$$NP \rightarrow Det \; N$$

poate fi însoţită de regula

$$Det \rightarrow \phi$$

Cu alte cuvinte constituentul nul poate fi un determinant nul, care apare la începutul unui grup nominal şi care, prin urmare, va fi analizat bottom-up. Dacă, spre exemplu, determinantul este un articol cu care începe un grup nominal ("o fată"), conform regulii

$$NP \rightarrow ART \; N$$

atunci, în limba română, putem avea tripletul de reguli

$S \rightarrow NP\ VP$ (*o fată citește; fata citește*)

$NP \rightarrow ART\ N$ (*o fată*)

$ART \rightarrow \phi$ (*fata*)

care spune analizorului sintactic să accepte determinantul nul ca reprezentând primul pas al analizei unui NP sau al analizei unui S. În cazul existenței unor astfel de reguli, analizorul sintactic descris va cicla.

Problema constă în aceea că analizorului de tip "din colțul stâng" i se permite, ca și celui bottom-up, să accepte constituenți nuli oricând dorește acest lucru. Soluția acestei probleme este aceea de a constrânge parser-ul prin adăugarea unui **tabel de legături** [19]. În afară de rezolvarea problemei constituenților nuli, astfel de tabele fac ca analizorul sintactic să devină mult mai eficient, așa cum se va vedea în cele ce urmează.

Tabelul de legături are rolul de a specifica *ce tipuri de constituenți pot să apară pe poziții de început.* Spre exemplu, tripletul de reguli

$S \rightarrow NP\ VP$

$NP \rightarrow Det\ N$

$VP \rightarrow V\ NP$

definește următoarele legături:

```
legatura(np,s).
legatura(det,np).
legatura(det,s).
legatura(v,vp).
legatura(X,X).
```

Ultima dintre acestea afirmă că orice constituent poate începe cu el însuși. Ea se aplică atunci când, spre exemplu, analizorul caută un constituent de tipul N și, în același timp, acceptă un substantiv. Introducerea tabelului de legături mărește eficiența parser-ului prin aceea că rezolvă problema existenței intrărilor lexicale multiple corespunzătoare aceluiași cuvânt. Acest tabel introduce o *relație* numită **legătura**, care spune că: o categorie **C1** este legată de o categorie **C2** dacă și numai dacă **C1** poate interveni în calitate de *constituent inițial* al lui **C2**. În mod evident, această relație este reflexivă și tranzitivă, iar luarea în considerare a ei rezolvă problema consumului excesiv de timp calculator în cazul acceptării de către parser a unui cuvânt ce aparține mai multor categorii lexicale. Predicatul **parse** trebuie modificat în consecință, adică în așa fel încât să existe și să fie exprimată o legătură între constituentul căutat la un anumit moment și cuvântul acceptat din șirul de intrare. Această legătură se exprimă în felul următor:

```
parse(C,[Cuvant|S2],S):-cuvant(W,Cuvant),
                        legatura(W,C),
                        completeaza(W,C,S2,S).
parse(C,S2,S):-regula(W,[]), % pentru constituenti nuli
               legatura(W,C),
               completeaza(W,C,S2,S).
```

Prin modificarea primei clauze Prolog amintite şi prin adăugarea celei de a doua, problema este rezolvată, în sensul că analizorul nu mai poate accepta constituenţi nuli în locaţii arbitrare. Prezentăm, în continuare, analizorul sintactic astfel modificat (şi utilizând tot Gramatica nr.2). Programul corespunzător, scris în SICStus Prolog, este următorul:

Programul 4.4

```
parse(C,[Cuvant|S2],S):-cuvant(W,Cuvant),
                        legatura(W,C),
                        completeaza(W,C,S2,S).
parse(C,S2,S):-regula(W,[]),
               legatura(W,C),
               completeaza(W,C,S2,S).

parse_lista([C|Cs],S1,S):-parse(C,S1,S2),
                          parse_lista(Cs,S2,S).
parse_lista([],S,S).

completeaza(C,C,S,S).
completeaza(W,C,S1,S):-regula(P,[W|Rest]),
                       parse_lista(Rest,S1,S2),
                       completeaza(P,C,S2,S).

regula(s,[np,vp]).
regula(vp,[]).
legatura(np,s).
regula(vp,[v,np]).
regula(np,[]).
```

```
legatura(v,vp).
regula(np,[det,n]).
regula(n,[]).
legatura(det,np).
legatura(det,s).
legatura(X,X).

cuvant(det,un).
cuvant(det,niste).
cuvant(det,o).
cuvant(n,elev).
cuvant(n,elevi).
cuvant(n,carte).
cuvant(v,iubeste).
cuvant(v,iubesc).
```

Acest parser poate încă să cicleze, şi anume atunci când întâlneşte un set de reguli de forma

$$A \rightarrow B\ A$$

$$B \rightarrow \phi$$

dar, în acest caz, ciclarea se datorează *regulilor în sine* şi nu analizorului sintactic. (Aceste reguli specifică un număr infinit de analize sintactice diferite pentru fiecare şir de intrare, după cum urmează: un A poate fi format dintr-un constituent B nul urmat de un alt A, ori din doi constituenţi nuli de tip B urmaţi de un alt A, ori din trei constituenţi nuli de tip B urmaţi de un alt A ş.a.m.d. Analizorul sintactic va executa un proces de backtracking infinit în încercarea de a găsi toate analizele posibile). Problema persistă în cazul tuturor gramaticilor care specifică, prin regulile lor, un număr infinit de analize sintactice.

4.3.1.2. BUP

Cea mai cunoscută implementare în Prolog a analizei sintactice din colţul stâng o reprezintă, după toate probabilităţile, aşa-numita BUP, concepută de către Matsumoto, Tanaka, Hirakawa, Miyoshi şi Yasukawa (1983).

Implementarea BUP (de la "bottom-up parser") transformă fiecare regulă PS într-o clauză Prolog al cărei cap nu este dat de nodul părinte ci de fiul cel mai din stânga. Astfel, regula

NP → Det N PP

se transformă în clauza Prolog

```
det(C,S1,S):-parse(n,S1,S2),parse(pp,S2,S3),np(C,S3,S).
```

Această clauză Prolog spune că "dacă tocmai s-a completat un constituent de tip Det, atunci se analizează un N şi apoi un PP; după care se apelează procedura care tratează un constituent completat de tip NP". Subliniem faptul că C reprezintă constituentul de *rang cel mai înalt* pe care analizorul sintactic încearcă să îl completeze. În acelaşi timp, S1, S2, S3 şi S reprezintă şirul de intrare în diverse momente ale analizei.

Pe lângă câte o clauză Prolog corespunzând fiecăreia dintre regulile PS, implementarea BUP necesită şi o aşa-numită "clauză de terminare" pentru fiecare tip de constituent. Astfel de clauze sunt:

```
np(np,S,S).
n(n,S,S).
det(det,S,S).
vp(vp,S,S).
v(v,S,S).
s(s,S,S).
```

Clauza

```
np(np,S,S).
```

afirmă, spre exemplu, că "dacă tocmai s-a acceptat un NP şi ceea ce se căuta era un grup nominal, atunci căutarea s-a încheiat". Prezentăm, în continuare, un parser BUP fără legături care efectuează analiză sintactică din colţul stâng şi care utilizează tot Gramatica nr.2. Predicatul **parse(C, S1, S)** analizează un constituent având categoria C, plecând de la şirul de intrare S1 şi ajungând la şirul de intrare S. Analizorul sintactic este cel din programul care urmează, program scris în SICStus Prolog:

Programul 4.5

```
parse(C,S1,S):-cuvant(W,S1,S2),
               P=..[W,C,S2,S],
               Call(P).


% Reguli PS si clauze de terminare


np(C,S1,S):-parse(vp,S1,S2),s(C,S2,S).
np(np,X,X).
```

```
det(C,S1,S):-parse(n,S1,S2),np(C,S2,S).
det(det,X,X).
v(C,S1,S):-parse(np,S1,S2),vp(C,S2,S).
v(v,X,X).

s(s,X,X).
vp(vp,X,X).
pp(pp,X,X).
n(n,X,X).

% Lexicon

cuvant(det,[un|X],X).
cuvant(det,[niste|X],X).
cuvant(det,[o|X],X).
cuvant(n,[elev|X],X).
cuvant(n,[elevi|X],X).
cuvant(n,[carte|X],X).
cuvant(v,[iubeste|X],X).
cuvant(v,[iubesc|X],X).
```

Interogarea Prologului se face în acelaşi mod:

```
?- parse(s,[niste,elevi,iubesc,o,carte],[]).
yes
```

sau

```
?- parse(s,[niste,elevi,iubesc],[]).
no
```

Se remarcă faptul că acest parser de tip BUP nu recunoaşte propoziţia

Nişte elevi iubesc.

ca fiind una corect generată în raport cu Gramatica nr.2, în timp ce parser-ul prezentat anterior (§4.3.1.1.) o va recunoaşte ca atare. (După acceptarea verbului "iubesc", acesta din urmă îşi propune să analizeze top-down un

constituent de tip VP prin admiterea unui constituent NP nul). Pentru a accepta (analiza cu succes) această propoziție, parser-ul BUP necesită introducerea în mod explicit în gramatică a regulii suplimentare $VP \rightarrow V$, sub forma:

```
v(C,S1,S):-vp(C,S1,S).
```

Implementarea BUP este extrem de eficientă datorită faptului că, așa cum rezultă și din programul Prolog prezentat, partea foarte delicată a procesului de căutare, aceea care urmează după completarea unui fiu aflat cel mai în stânga, este rezolvată de către cel mai rapid mecanism de căutare al Prologului, și anume mecanismul de găsire a unei clauze fiind dat predicatul corespunzător. Pentru o implementare BUP completă (care include legături), vezi [70].

4.4. Asupra eficienței strategiilor de tip bottom-up și top-down; ambiguitate

O primă *concluzie* care a fost pusă, deja, în evidență, este aceea că o strategie *pură* de tip bottom-up sau top-down nu va fi suficientă pentru a realiza o analiză sintactică eficientă. (Vom reveni asupra noțiunii de eficiență în acest context).

În același timp, *o primă problemă* care se ridică în urma prezentării tehnicilor de analiză sintactică de până acum este aceea că analizorii sintactici discutați până în prezent nu memorează **rezultatele intermediare**. În consecință, ei verifică din nou aceleași lucruri, care nu aveau cum să se schimbe până la momentul curent, consumând timp în mod inutil și afectând în acest fel execuția programului. Este motivul pentru care vom studia așa-numiții **analizori sintactici cu hartă**. Un analizor sintactic cu hartă folosește *o structură de date numită hartă* pentru a memora toate rezultatele intermediare obținute în cursul unei analize. Această creștere a eficienței va avea însă un cost, care nu este numai unul de mărire a complexității implementării: programul de analiză corespunzător va avea nevoie de mai mult spațiu.

O *a doua problemă* care se ridică, în mod natural, în acest context, este aceea a **eficienței** programelor de analiză sintactică. Eficiența nu înseamnă numai spațiul utilizat de program sau timpul necesar pentru recunoașterea unui șir, adică timpul cerut pentru a decide dacă acesta este corect din punct de vedere gramatical. Contează, în plus, timpul utilizat pentru a enumera *toate analizele posibile* ale unui șir de intrare, iar acesta este influențat de *gradul de ambiguitate* al propoziției (și s-a arătat că este, în cel mai rău caz, exponențial).

Pentru a vedea un exemplu de ambiguitate, vom adăuga o nouă intrare lexiconului Gramaticii nr.1, după cum urmează:

Cuvânt surori:

<cat> = V.

Aceasta permite Gramaticii nr.1 să admită propoziții ca

Dr. Popescu surori pacienți.

deoarece lexiconul său a fost afectat de o *ambiguitate lexicală* cauzată de apartenența cuvântului *surori* la două categorii lexicale diferite. Această ambiguitate lexicală va face ca cel puțin unul dintre analizorii sintactici descriși până acum să lucreze suplimentar: un parser bottom-up care găsește cuvântul "surori" îl va atribui pe acesta atât categoriei V, cât și categoriei NP și va explora fiecare dintre aceste posibilități.

Ambiguitatea lexicală de acest tip este omniprezentă în limba engleză. Se știe că majoritatea cuvintelor englezești sunt substantive și că multe substantive englezești pot fi folosite ca verbe. În general, majoritatea cuvintelor englezești pot aparține mai multor părți de vorbire, deci mai multor categorii lexicale, ceea ce determină existența unui mare număr de posibilități pentru generarea ambiguității. De asemenea, marea majoritate a cuvintelor englezești sunt polisemantice. Dacă un parser aparține unui sistem care are rolul de a realiza corespondența dintre propoziții englezești și înțelesuri ale lor, atunci el se va confrunta cu ambele aspecte ale ambiguității lexicale amintite. Prin contrast, un sistem experimental care nu face decât să atribuie structuri sintactice unor șiruri de intrare, poate ignora acele ambiguități lexicale de ordin semantic care nu sunt dublate de ambiguitate generată de apartenența la diverse categorii lexicale. Aceleași probleme referitor la ambiguitatea lexicală se ridică și în cazul limbii române. Și în această limbă multe cuvinte sunt polisemantice și pot aparține unor categorii lexicale diferite. Numeroase cuvinte, spre exemplu, pot fi atât substantive, cât și verbe (*cânt, blestem, botez*) ori atât substantive, cât și adjective, cât și adverbe (*bun*). De asemenea, în limba română, orice verb la participiul trecut poate funcționa și ca adjectiv (ca în *ușa stă deschisă* și respectiv *ușă deschisă*). Toate acestea, ca și multe alte fenomene întâlnite în limba română, sunt generatoare de ambiguitate lexicală.

În principiu, trebuie făcută o distincție clară între ambiguitatea lexicală și **ambiguitatea structurală** deși, în practică, prima o generează adesea pe cea din urmă, cele două fiind, prin urmare, legate între ele. Pentru a pune în mod clar în evidență diferența dintre cele două tipuri de ambiguitate, vom spune că propoziția

Merg spre bancă.

oferă un exemplu tipic de ambiguitate lexicală pură, generată de cele două sensuri diferite ale substantivului *bancă*, în timp ce propoziția

Observ băiatul cu un telescop.

conține o ambiguitate structurală pură, corespunzând celor două situații în care fie vorbitorul, fie băiatul, reprezintă posesorul telescopului. Un alt exemplu de ambiguitate structurală este cel din *frumoasa fată bogată*, fragment de propoziție pe care un parser îl poate analiza în două moduri diferite:

((frumoasa fată) bogată)

(frumoasa (fată bogată))

S-a arătat că în numeroase limbi (și cu precădere în limba engleză) există trei surse principale ale ambiguității structurale pure, și anume: atașarea în propoziție a grupurilor prepoziționale, coordonarea și alăturarea substantivelor. Pentru exemple referitoare la limba engleză și corespunzând tuturor acestor situații, vezi [29].

Discuția de până acum referitoare la ambiguitate s-a ocupat, aproape în exclusivitate, de așa-numita **ambiguitate globală**. Ambiguitatea globală se manifestă în acele cazuri în care unei expresii i se atribuie în mod corect două sau mai multe structuri, iar acele structuri se transmit, adică persistă în cadrul structurilor mai ample din care expresia face parte. În proiectarea și implementarea analizorilor sintactici suscită, în egală măsură, interes și fenomenul de **ambiguitate locală**. Aceasta se manifestă în acele cazuri în care unei expresii i se atribuie în mod corect două sau mai multe structuri, care s-ar putea să persiste sau nu în cadrul structurilor mai ample din care expresia face parte. Trebuie luat în considerare faptul că analiza sintactică a unei propoziții în totalitate neambigue poate presupune explorarea mai multor situații de ambiguitate locală în timpul procesului de căutare a unicei descrieri structurale a întregii propoziții. Activitățile dedicate eliminării sau minimizării ambiguității locale consumă mult timp calculator, din care cauză numeroase eforturi ale lingvisticii computaționale s-au concentrat în această direcție. La originea acestor eforturi se află lucrările lui M. P. Marcus (1980), care își propune să construiască analizori sintactici ce analizează propoziții neambigue în mod determinist. Aceasta înseamnă că un parser nu este indus în eroare de ambiguitățile locale și nu efectuează niciodată un proces de backtracking. În realitate, un astfel de parser are la bază o tehnică (*lookahead*) prin care privește înainte în cadrul șirului de intrare, până la o distanță determinată și limitată, înainte de a decide cum să interpreteze un cuvânt sau o secvență de cuvinte. Tehnica de *lookahead* poate fi privită ca o formă de backtracking cu multe constrângeri. Un parser de tip Marcus analizează numai limbaje independente de context, iar analizorii sintactici determiniști, în general, obțin o performanță adecvată utilizând o combinație de *lookahead* și euristică referitoare la frecvențele relative ale diferitelor construcții posibile. Întrucât ei nu formează obiectul acestei lucrări, vom reține doar faptul că ambiguitatea locală și cea globală constituie probleme cheie în analiza sintactică și că un parser determinist încearcă să elimine ambiguitatea locală.

4.5. Memorarea rezultatelor intermediare

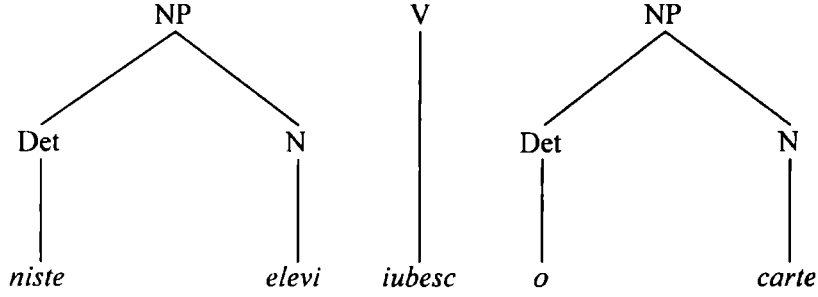
Așa cum am mai remarcat, analizorii sintactici discutați până în prezent nu memorează rezultatele intermediare obținute în cursul analizei, ceea ce face ca ei să efectueze aceleași verificări în repetate rânduri, consumând timp calculator și afectând în acest fel execuția programului. În cele ce urmează, vom studia o serie de tehnici ale analizei sintactice bazate tocmai pe memorarea rezultatelor intermediare (analiza sintactică cu hartă) și vom urma prezentarea acestor tehnici așa cum a fost ea realizată în [29].

4.5.1. Tabel de subșiruri bine formate

O primă structură de date care a fost introdusă [29] pentru ca un parser să memoreze în mod eficient rezultatele intermediare este aceea de **tabel de subșiruri bine formate (TSBF)**. Un TSBF reprezintă un mecanism prin care i se permite unui parser să păstreze o evidență a structurilor care au fost deja găsite, evitându-se astfel căutarea din nou a acestora. Încercăm să sugerăm, în cele ce urmează, cum s-a ajuns la soluția utilizării unui TSBF.

După cum se știe, există *redundanță* în spațiul de căutare al analizei (repetări ale acelorași situații). Un parser eficient trebuie, prin urmare, să fie înzestrat cu *memorie*, în sensul că el trebuie să poată înregistra constituenții găsiți deja, precum și structura acestora. Cu alte cuvinte, trebuie memorată informație despre analizele parțiale alternative ale șirului de intrare, fără a se lua decizii premature cu privire la corectitudinea fiecăreia. Există și alte situații și/sau modalități în care o asemenea abilitate a parser-ului de a memora rezultate intermediare poate fi utilă. Spre exemplu, în cazul unui șir de intrare pe care gramatica dată nu îl acceptă ca fiind corect, un sistem robust nu se va mulțumi să producă un mesaj de tipul "Nu există analiză", ci va dori din partea componentei sintactice să pună la dispoziția celorlalte componente (semantică și pragmatică) cât mai multe informații posibil despre structura sintactică. Cu alte cuvinte, un sistem robust de prelucrare a limbajului natural va fi interesat de valorificarea la maximum a rezultatelor intermediare găsite de componenta sintactică.

Există situații în care, fiind dată o gramatică, aceasta nu acceptă un anumit șir de intrare și, prin urmare, nu îi poate atribui un singur arbore de derivare. (Reprezentarea structurală a unui șir neambiguu admis de către o gramatică se realizează cel mai adesea prin intermediul arborilor). În schimb, sistemul poate fi interesat de secvența de arbori care analizează diferite părți ale șirului de intrare, adică de reprezentarea *structurilor parțiale*. Gramatica cu caracteristici nr. 5 nu admitea propoziția corectă *Niște elevi iubesc o carte*, dar majoritatea analizorilor sintactici vor găsi următoarele structuri parțiale:

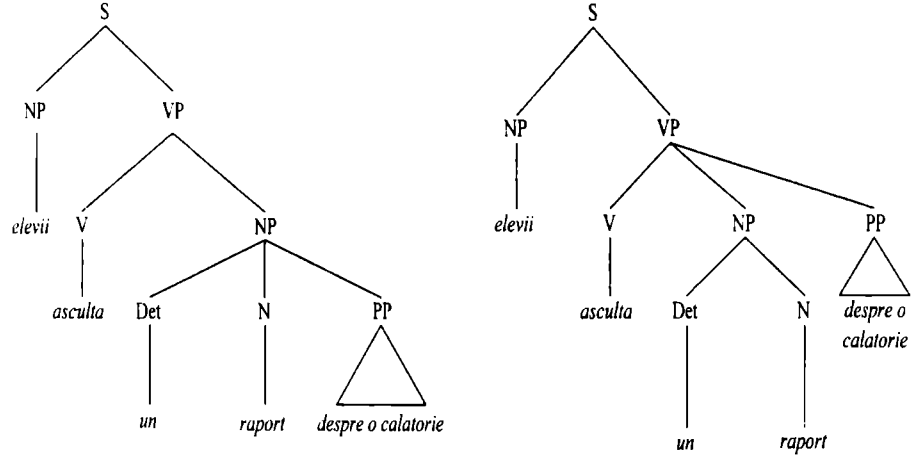


(În general, o problemă importantă care se pune este aceea a reprezentării structurilor parțiale, care s-ar putea să facă parte sau nu din reprezentarea finală, corespunzătoare analizei complete).

În exemplul anterior, în care am avut un șir de intrare incorect din punctul de vedere al gramaticii date, gramatica furnizează mai puține informații decât ar oferi un singur arbore de structură complet. Există însă șiruri de intrare despre care se poate da mai multă informație decât cea furnizată de un unic arbore (complet). *Reprezentarea structurilor alternative* corespunzătoare unui același șir de intrare este o altă problemă cu care se confruntă orice analizor sintactic. Propoziției

Elevii ascultă un raport despre o călătorie.

îi corespund două reprezentări structurale, după cum urmează:

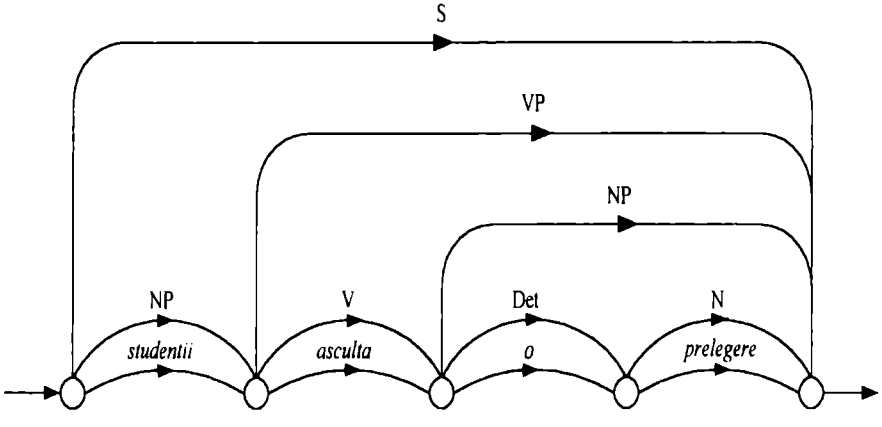


Acest fapt se datorează *ambiguității structurale* care se manifestă în cazul propoziției enunțate, ambiguitate determinată de posibilitățile diferite de atașare ale grupului prepozițional (PP) existent.

Conform acestei discuții, problema memorării rezultatelor parțiale ale analizei (ca, de altfel, și a celor finale), problemă pe care ne-am pus-o la începutul acestui paragraf, pare să ne conducă spre introducerea unei structuri de date constând dintr-o secvență de arbori ca cea din figura anterioară. Această structură de date poate însă deveni extrem de complexă, mai ales dacă se ia în considerare fenomenul de ambiguitate prezent în majoritatea limbilor naturale. În [29], a cărei linie de argumentație o urmăm aici, se propune ca primă soluție la aceste probleme de reprezentare structurală o tehnică prin care se reprezintă toată informația care ar fi necesară pentru a enumera secvențele posibile de arbori (i.e. arborii parțiali posibili, precum și acele porțiuni din șirul de intrare cărora le corespund), fără a face propriu-zis enumerarea. Reprezentarea care rezultă în acest mod constituie un așa-numit **tabel de subșiruri bine formate (TSBF)**.

Pentru a vedea în ce constă această reprezentare vom considera pozițiile de început și de sfârșit ale șirului de intrare ca fiind numerotate *0* și respectiv *n*, iar spațiile dintre cuvinte ca fiind numerotate de la stânga la dreapta, cu numere de la *1* la *n-1*. Un TSBF indică, pentru fiecare pereche (i,j) cu $0 \le i < j \le n$, categoriile care pot acoperi subșirul de cuvinte ce se găsește între *i* și *j*.

Modalitatea de a privi TSBF-urile pe care o adoptăm, în cele ce urmează, este aceea de a le considera niște grafuri orientate, aciclice, având un unic vârf inițial și un unic vârf final și ale căror vârfuri sunt etichetate de la *0* (cel inițial) la *n* (cel final), unde *n* reprezintă numărul de cuvinte al șirului de intrare. Arcele corespunzătoare sunt etichetate cu categorii (gramaticale și lexicale) și cu cuvinte, ca în figura următoare:



Un TSBF poate fi reprezentat ca o mulțime de arce, unde un arc este o structură cu următoarele atribute:

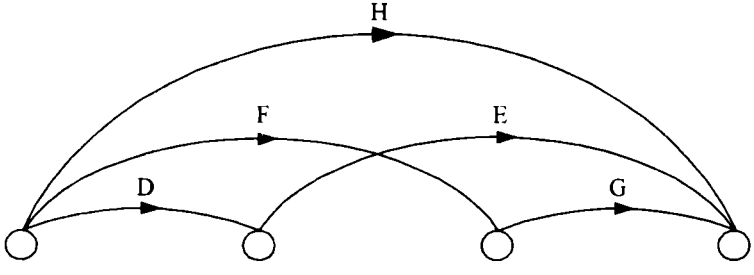
<START> = ... un întreg ...

<FINAL> = ... un întreg ...

<ETICHETĂ> = ... o categorie ...

În mod evident, un *arbore PS de structură* de tipul convențional este, de asemenea, un graf orientat aciclic. Acest graf are însă vârfurile etichetate cu categorii, în timp ce muchiile sunt neetichetate. În contextul utilizării unui TSBF arcele (muchiile) sunt purtătoare de etichete reprezentând categorii în timp ce vârfurile au, în esență, denumiri arbitrare. Această reprezentare a utilizat numerele întregi și a exploatat relația ' < ' asupra întregilor pentru a codifica în mod implicit capetele muchiilor orientate. Se observă că un TSBF, spre deosebire de un arbore PS, codifică în mod direct *ordinea grupurilor sintactice*: un constituent căruia îi corespunde în tabel un arc de la *i* la *j* precede un constituent căruia îi corespunde un arc de la *m* la *n* numai în cazul *j*<*m*. Arborii, în schimb, codifică relațiile de dominare imediată, indicând care grupuri sintactice sunt părți constitutive ale altor grupuri sintactice.

Un TSBF, așa cum a fost prezentat până acum, nu este suficient pentru a regăsi întreaga informație codificată într-o structură de date bazată pe arbori PS. Analizând un TSBF nu se poate determina, spre exemplu, regula gramaticii care legitimează un anume arc. În cazul TSBF-ului din figura următoare nu se poate decide dacă arcul etichetat cu H este legitimat de faptul că H domină D și E sau de faptul că H domină F și G:



Răspunsul la această întrebare nu poate fi dat fără a cunoaște regulile gramaticii. Această informație ar putea fi disponibilă dacă se transformă eticheta unui arc dintr-o simplă categorie într-un arbore de derivare - o categorie împreună cu secvența de arce corespunzând subgrupurilor sintactice imediate.

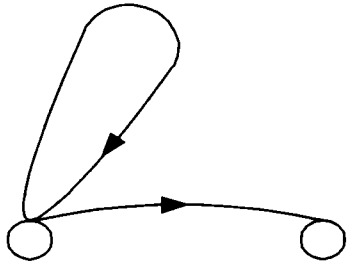
Utilizarea unui TSBF permite unui parser să se abțină de la a redescoperi fapte care au fost deja stabilite. Aceste tabele pot fi utilizate de către orice tip de parser. Ele reprezintă o structură de date complet neutră, ca și cea a arborilor PS. Un parser top-down, spre exemplu, poate folosi un TSBF în felul următor: se va impune cerința ca grupurile sintactice determinate să fie înregistrate ca intrări în diferite locații ale tabelului. Atunci când, la o anumită poziție din șirul

de intrare, se caută un grup sintactic de un anumit tip, tabelul va fi consultat. Dacă în tabel există o intrare de tipul dorit și care începe la poziția cerută, atunci se folosesc intrările din tabel ca reprezentând analizele posibile, fără ca această muncă să fie făcută, de fiecare dată, de către parser. Această strategie se bazează pe o anumită convenție legată de modul de inserare a analizelor parțiale în tabel: o intrare corespunzând unei categorii date și care începe la o poziție dată nu este creată decât atunci când toate intrările de acest tip pot fi create. Altfel, cea de-a doua parte a procesului de analiză, care se bazează pe prima, nu va lua în calcul toate analizele posibile. O astfel de convenție este necesară și unui dispozitiv de recunoaștere, deoarece *analize posibile diferite ale unei categorii, care încep de la o anumită poziție dată, pot avea poziții de sfârșit diferite.*

Prin memorarea rezultatelor intermediare și prin faptul că întotdeauna testează prezența unei categorii în tabel înainte de a încerca formarea acesteia, un parser care utilizează un TSBF poate evita executarea aceleiași operații de mai multe ori. Prețul care se plătește constă în spațiul consumat, dar, întrucât un TSBF nu conține redundanță, cerința de spațiu nu va fi niciodată mai mare decât un multiplu de n^2 (pentru o unică analiză). Evident, este vorba de găsirea *primei* analize a unui șir de intrare. În practică, putem fi interesați de găsirea tuturor analizelor posibile, complexitatea acestei operații fiind, atunci când limbajul conține ambiguitate arbitrară, de ordin exponențial. Cu toate acestea, începând din anii 70, foarte mulți analizori sintactici nedeterminiști ai limbajului natural au utilizat diverse variante de TSBF-uri.

4.5.2. Harta activă

Un TSBF de tipul celui prezentat este un instrument adecvat pentru a memora fapte referitoare la *structură*. Utilizarea unei asemenea structuri de date face, spre exemplu, ca un grup nominal determinat cu succes să nu trebuiască să fie găsit decât o singură dată. Analizorul sintactic nu va reuși însă să evite reinvestigarea ipotezelor care anterior au eșuat. Pentru a se evita duplicarea tentativelor anterioare de a analiza într-un anume mod, este necesară existența unei reprezentări explicite a *ipotezelor* și *țelurilor* diferite pe care parser-ul le ia în considerare și respectiv le are în orice moment al analizei. În cele ce urmează, vom fi preocupați de îmbogățirea structurii de date prezentate anterior (TSBF) în așa fel încât ea să poată ține evidența ipotezelor structurale și a scopurilor propuse. Pentru aceasta vom opera două mici *modificări asupra unui TSBF*. Astfel, în loc de a se cere ca graful orientat aflat la baza TSBF-ului să fie aciclic, se va permite existența unor arce care revin în vârful de unde au plecat, acesta fiind singurul tip de ciclu admis. Această relaxare a cerinței de aciclicitate, care introduce așa-numitele "arce vide", va fi justificată în cele ce urmează și va conduce la existența unor situații de tipul celei din figura următoare:



O a doua modificare efectuată constă în schimbarea etichetei arcelor de la o simplă categorie la o regulă a gramaticii, în felul următor: dacă $S \rightarrow NP\ VP$ este o regulă a gramaticii PS date, atunci se vor utiliza ca etichete ale arcelor obiecte de tipul

$$S \rightarrow \bullet\ NP\ VP$$

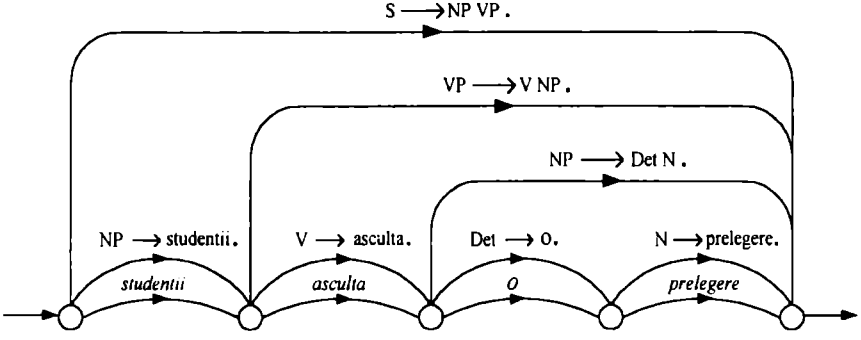
$$S \rightarrow NP\bullet\ VP$$

$$S \rightarrow NP\ VP\bullet$$

adică așa-numite "reguli punctate". Punctul care intervine în structura acestor etichete are rolul de a indica până unde a fost verificată de către parser ipoteza că regula respectivă se poate aplica. În particular, primul tip de etichetă va corespunde așa-numitelor arce vide. Ea denotă ipoteza că un constituent S poate fi găsit ca acoperind un subșir care începe de la nodul luat în considerare și că acest S acoperă subșirul respectiv în virtutea faptului că acesta din urmă poate fi acoperit, în egală măsură, de o secvență având structura NP VP. Eticheta indică faptul că o asemenea ipoteză a fost formulată, dar că ea nu a fost nici măcar parțial verificată. Cel de al doilea și al treilea tip de etichetă, care nu vor apărea niciodată în cazul arcelor vide, denotă aceeași ipoteză, dar indică și faptul că ea a fost parțial sau total confirmată. Cel de-al doilea tip de etichetă se va găsi numai în cazul arcelor care acopăr un NP. Cel de-al treilea tip de etichetă indică o ipoteză pe deplin confirmată și corespunde numai arcelor care acopăr un șir alcătuit dintr-un subșir de tip NP urmat de un subșir de tip VP. Acest tip de etichetă este cel mai apropiat echivalent al etichetelor constând din categorii care au fost utilizate în cazul TSBF-urilor, dar el furnizează o informație suplimentară. Este indicată, în plus, secvența care justifică prezența respectivei categorii.

Un TSBF care a fost modificat pentru a include ipoteze în maniera descrisă se numește **hartă activă**, iar analizorii sintactici care exploatează o asemenea structură de date se numesc **analizori sintactici cu hartă**. În cele ce urmează, harta activă va fi denumită, simplu, *hartă*, iar arcele vor fi denumite *muchii*. Mai precis, vom gândi harta ca fiind o structură de date constând dintr-o

mulțime de arce etichetate numite muchii. Arcele care reprezintă ipoteze neconfirmate vor fi numite **muchii active**, iar cele corespunzătoare ipotezelor confirmate vor fi numite **muchii inactive**. Cu alte cuvinte, o muchie inactivă reprezintă un rezultat, în timp ce o muchie activă reprezintă o ipoteză de structură. Hărțile pot reprezenta întreaga informație inclusă în TSBF-uri. Figura următoare prezintă același exemplu pe care l-am utilizat în cazul TSBF-urilor, de data aceasta într-o notație cu hărți:



Ca și TSBF-urile, hărțile sunt complet neutre în raport cu strategia de parsing folosită.

Conform descrierii anterioare, o hartă poate fi reprezentată ca o mulțime de structuri, fiecare dintre acestea având atributele

< START > = ... un întreg ...

< FINAL > = ... un întreg ...

< ETICHETA > = ... o categorie ...

< GĂSIT > = ... o secvență de categorii ...

< DEGĂSIT > = ... o secvență de categorii ...

unde <ETICHETA> reprezintă membrul stâng al regulii punctate corespunzătoare, < GĂSIT > este secvența de categorii din membrul drept aflate la stânga punctului, iar <DEGĂSIT> este secvența de categorii din membrul drept situate la dreapta punctului. În această reprezentare, o muchie având valoarea corespunzătoare a lui <DEGĂSIT> secvența vidă va fi o muchie inactivă, în timp ce toate celelalte muchii vor fi active. Pentru a reprezenta astfel de înregistrări vom folosi adesea notații de tipul < i, j, R >, unde i și j sunt numere întregi, iar R desemnează o regulă punctată. Astfel, structura

< 0, 2, S → NP• VP >

reprezintă următoarea muchie activă:

< START > = 0

< FINAL > = 2

< ETICHETA > = S

< GĂSIT > = < NP >

< DEGĂSIT > = < VP >

4.5.2.1. Analiza sintactică cu hartă

În cele ce urmează, ne propunem să utilizăm structura de date descrisă - harta - pentru a face analiză sintactică. Vom avea în vedere faptul că harta reprezintă, în ultimă instanță, o mulțime de muchii și, ca și în prezentările anterioare, nu vom fi preocupați de problema construirii efective a arborilor PS de structură.

Pentru a vedea procesul care reprezintă esența analizei sintactice cu hartă vom considera un exemplu. Să presupunem că, într-un anumit moment al analizei, harta conține, printre altele, următoarele muchii:

{ < 0, 2, S → NP• VP >,

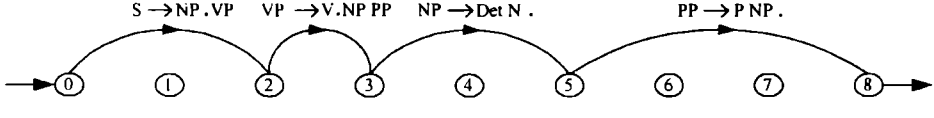
< 2, 3, VP → V• NP PP >,

< 3, 5, NP → Det N• >,

< 5, 8, PP → P NP• >,

... }

Aceste muchii pot fi reprezentate ca în figura următoare, care arată două muchii active și două muchii inactive:



(Pentru claritate am omis celelalte muchii care au fost obținute înainte de a se ajunge la situația din figură).

Prima muchie activă reprezintă o ipoteză despre o propoziție care a găsit un grup nominal (chiar dacă nu este cel reprezentat pe figură), iar acum caută un grup verbal. Pentru a satisface această ipoteză trebuie găsită pe hartă o muchie corespunzătoare *inactivă* care începe la poziția 2. Harta dată nu oferă decât o ipoteză privitoare la *eventuala* prezență a unei astfel de muchii (pentru moment

activă). Până când această a doua ipoteză nu va fi confirmată, nu se poate efectua nici o acțiune privitoare la prima muchie activă. Din această cauză, parser-ul își va concentra atenția asupra celei de a doua muchii active din figură.

Cea de-a doua muchie activă reprezintă o ipoteză privitoare la un grup verbal care a găsit deja un verb, iar acum caută un grup nominal urmat de un grup prepozițional. Ipoteza aceasta caută un grup nominal care începe la poziția 3. O astfel de muchie (inactivă) există. Prin urmare, ipoteza referitoare la grupul verbal a fost confirmată în continuare, chiar dacă nu în întregime. Această confirmare parțială se reprezintă prin adăugarea pe hartă a noii muchii active $< 2, 5, VP \rightarrow V\ NP\bullet\ PP >$. Aceasta reprezintă o nouă ipoteză, care caută un grup prepozițional începând de la poziția 5. O muchie inactivă corespunzătoare există. Prin urmare, noua ipoteză referitoare la grupul verbal este pe deplin confirmată. În mod corespunzător, se poate adăuga pe hartă muchia inactivă $< 2, 8, VP \rightarrow V\ NP\ PP\bullet >$. S-a ajuns la următoarea structură a muchiilor:

$\{ < 0, 2, S \rightarrow NP\bullet\ VP >,$

$< 2, 3, VP \rightarrow V\bullet\ NP\ PP >,$

$< 2, 5, VP \rightarrow V\ NP\bullet\ PP >,$

$< 2, 8, VP \rightarrow V\ NP\ PP\bullet >,$

$< 3, 5, NP \rightarrow Det\ N\bullet >,$

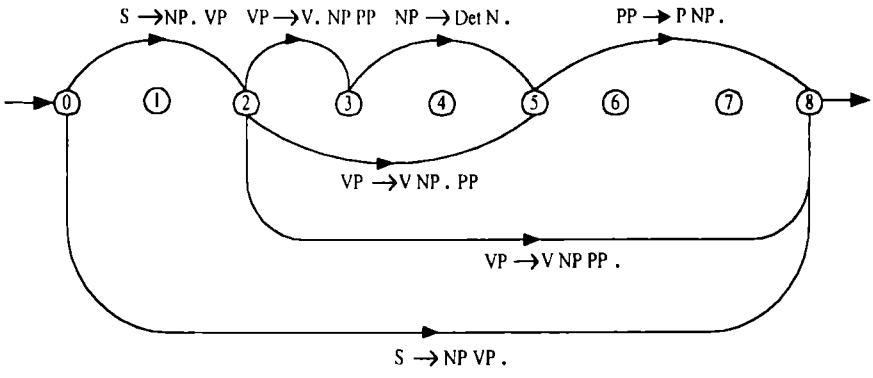
$< 5, 8, PP \rightarrow P\ NP\bullet >,$

$\ldots \}$

Întorcându-ne la prima muchie activă vom observa că ipoteza indusă de aceasta este acum pe deplin confirmată, întrucât a fost găsit un grup verbal care începe la poziția 2. Prin urmare, se poate adăuga muchia inactivă

$< 0, 8, S \rightarrow NP\ VP\bullet >$

și harta arată acum ca în figura următoare:



Faptul că există o *muchie de tip S* care acoperă întreaga lățime a grafului înseamnă că s-a înregistrat succes în a analiza șirul de intrare ca pe o propoziție. (Alte analize mai pot exista, dar una dintre ele a fost deja găsită).

Menționăm faptul că, cea mai bună abordare, ca și în cazul TSBF-urilor, este aceea de a folosi harta numai dacă ea este *completă* relativ la un anumit tip de constituent intervenind la o anumită poziție. Spre exemplu, dacă se caută un constituent de tip NP la poziția 3 din șirul de intrare, se utilizează harta numai atunci când se cunoaște faptul că ea conține toate grupurile nominale posibile intervenind la poziția 3. Altfel, se poate ignora harta și efectua analiza sintactică în modul convențional, fără utilizarea acestei structuri de date. Dacă, în timpul acestui tip de analiză, nu se găsește ceea ce se căuta, atunci se afirmă faptul că harta este completă relativ la tipul de constituent căutat la poziția respectivă (și se include acest fapt în baza de date Prolog).

4.5.2.1.1. Regula fundamentală

Procesul descris anterior reprezintă esența analizei sintactice cu hartă. Ceea ce s-a făcut, de fapt, a fost aplicarea aceleiași reguli de trei ori: dacă o muchie activă întâlnește o muchie inactivă de categoria dorită, atunci se adaugă hărții o nouă muchie, acoperitoare pentru ambele. M. Kay, ale cărui lucrări le urmăm în această prezentare, numește aceasta "regula fundamentală a analizei sintactice cu hartă". **Regula fundamentală** poate fi formulată [29] după cum urmează:

Regulă fundamentală

Dacă harta conține muchiile $< i, j, A \rightarrow W1.B\ W2 >$ *și* $< j, k, B \rightarrow W3. >$, *unde A și B sunt categorii, iar W1, W2 și W3 reprezintă secvențe de categorii sau cuvinte (posibil vide), atunci se adaugă hărții muchia* $< i, k, A \rightarrow W1\ B.\ W2 >$.

Prin urmare, operațiile de bază ale unui analizor bazat pe hărți folosesc o *combinare între arce active și constituenți compleți*. Rezultatul este sau un constituent complet nou sau un arc activ nou, extensie a arcului activ originar.

Se remarcă faptul că regula fundamentală nu precizează dacă noua muchie este activă sau inactivă, dar acest fapt nu este necesar, el fiind în întregime determinat de W2, după cum acesta este sau nu vid. Se observă, de asemenea, că *regula efectuează numai adăugări pe hartă*. Ea nu înlătură de pe hartă muchiile active care au înregistrat succes. Acest lucru este necesar deoarece o astfel de muchie poate avea succes grație unei alte muchii inactive, care apare ulterior pe hartă. Îndepărtarea ei poate face, prin urmare, ca parser-ul să nu găsească toate analizele posibile ale unui șir de intrare.

Pentru a descrie în întregime un analizor sintactic cu hartă este necesar să tratăm, în plus, următoarele trei probleme: inițializarea, strategia de invocare a regulilor și strategia de căutare.

4.5.2.1.2. Inițializare

Inițializarea este necesară deoarece nu putem aplica regula fundamentală unei hărți care nu conține nici o muchie. (Este necesară existența cel puțin a unei muchii active și a unei muchii inactive pentru a se putea declanșa un algoritm de analiză sintactică).

A inițializa harta înseamnă a asigura existența unor muchii *inactive*. Acest lucru poate fi ușor realizat consultând lexiconul, care înregistrează cărei categorii lexicale îi aparține fiecare cuvânt. Arce multiple vor corespunde acelor cazuri de ambiguitate în care un același cuvânt aparține mai multor categorii lexicale.

Presupunând că, în cazul exemplului de până acum, este dat lexiconul

Cuvânt studenții:

< cat > = NP.

Cuvânt ascultă:

< cat > = V.

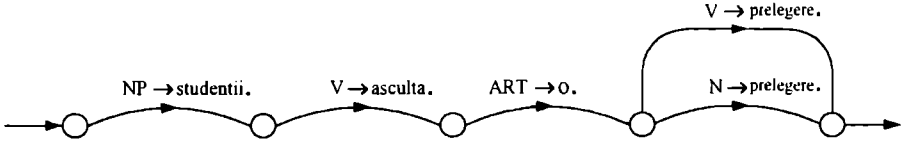
Cuvânt o:

< cat > = ART.

Cuvânt prelegere:

< cat > = N, V.

harta inițializată este următoarea:



4.5.2.1.3. Invocarea regulilor

Muchiile furnizate de pasul de inițializare nu sunt suficiente pentru ca analiza sintactică să poată începe. Ele reprezintă muchiile inactive cu care va debuta un algoritm de parsing cu hartă. Pentru aplicarea regulii fundamentale este însă necesară crearea, în egală măsură, a unor muchii active. Un principiu simplu care asigură crearea unor astfel de muchii este următorul: de fiecare dată când se adaugă hărții o muchie inactivă de categorie C, se adaugă și o nouă muchie activă vidă, începând din același vârf și corespunzând fiecărei reguli a gramaticii care necesită un constituent de categorie C în rolul fiului celui mai din stânga. (Prin muchie activă vidă înțelegem, așa cum era de așteptat, o muchie activă care urmează să își stabilească componentele și care, prin urmare, începe din și revine în același vârf).

Această strategie de invocare a regulilor va da naștere unei analize sintactice de tip bottom-up. Vom lua în considerare și alte strategii de invocare a regulilor în viitor. Pentru moment, ne vom limita la strategia amintită, care ar putea fi rezumată prin intermediul următoarei reguli:

Regulă bottom-up

Dacă se adaugă hărții muchia $< i, j, C \rightarrow W1. >$, *atunci, pentru fiecare regulă a gramaticii de forma* $B \rightarrow C\ W2$, *se adaugă hărții muchia* $< i, i, B \rightarrow .C\ W2 >$.

Cu ajutorul acestei reguli se pot adăuga hărții muchii active după pasul de inițializare. Este evident faptul că regula fundamentală și regula bottom-up sunt suficiente pentru a asigura găsirea tuturor analizelor posibile ale șirului de intrare. *Modul de interacțiune* al regulii bottom-up cu regula fundamentală depinde de ordinea în care diferitele muchii sunt adăugate hărții. Această chestiune depinde, la rândul ei, de *strategia de căutare*, problemă asupra căreia vom reveni.

Strategia de analiză sintactică cu hartă pe care tocmai am prezentat-o presupune că o regulă este regăsită de îndată ce este localizat elementul cel mai din stânga al membrului ei drept. Această indexare după fiul cel mai din stânga sugerează o strategie de tip "din colțul stâng". O altă caracteristică interesantă a acestei tehnici de analiză sintactică o constituie utilizarea muchiilor active. Folosirea acestora în modul prezentat face să fie păstrate ipoteze despre grupuri sintactice care *ar putea fi prezente*, urmând să fie depistate în viitor. Muchiile active furnizează o modalitate eficientă de a memora reguli care s-ar putea să fie aplicate (bottom-up) într-o fază ulterioară. Atunci când vom lua în considerare un alt tip de regulă ("regula top-down"), vom vedea modalități mai eficiente decât aceasta de a utiliza muchiile active.

Pentru moment, vom reține faptul că există două mari strategii de invocare a regulilor: cea bottom-up (la care ne-am referit deja) și cea top-down.

Indiferent de strategia aplicată, vom avea în vedere faptul că fiecare muchie activă reprezintă o ipoteză care trebuie explorată. Dacă această ipoteză înregistrează succes, fie și numai parțial, atunci ea poate genera noi ipoteze (muchii active și inactive), grație acțiunii regulii fundamentale, precum și a strategiei de invocare a regulilor care se adoptă.

Primii analizori sintactici pe care îi vom prezenta au în comun două caracteristici importante, și anume:

- utilizează baza de date Prolog pentru memorarea hărții;
- nu lucrează în mod explicit cu o structură de date de tip "agendă", ci se bazează, ca strategie de control, pe procesul de backtracking pe care Prologul îl efectuează.

4.5.2.1.4. Elemente de organizare

Implementarea unui analizor sintactic cu hartă va trebui să țină cont de câteva elemente suplimentare, care au fost numite "elemente de organizare" [29].

Mai întâi, atunci când una dintre regulile discutate propune adăugarea unei noi muchii, aceasta va trebui inclusă în hartă numai dacă ea nu se află deja acolo. Prin urmare, adăugarea oricărei muchii trebuie să fie precedată de o verificare a existenței acesteia. (Fără această verificare harta nu ar putea înlătura duplicatele din spațiul de căutare).

Organizarea construirii arborelui PS corespunzător șirului analizat este o altă problemă care trebuie luată în considerare. Ca și în cazul TSBF-urilor, ideea de bază este aceea de a avea ca elemente ale structurii de date pe care o formează muchiile *arbori* (reprezentați sub formă de liste), nu simple categorii. Această cerință va afecta secvența categoriilor găsite corespunzător unei muchii. Astfel, atributul GĂSIT al unei muchii va fi dat de secvența arborilor de derivare corespunzători grupurilor sintactice deja găsite, iar nu de simple categorii.

Regula fundamentală trebuie modificată pentru a lua în considerare construcția arborilor în sensul amintit.

4.5.3. Implementarea unui parser bottom-up cu hartă

O primă implementare în Prolog a unui **parser bottom-up cu hartă** pe care o vom lua în considerare va utiliza însăși baza de date Prolog ca depozitar al hărții. Memorarea muchiilor se va face prin declararea lor în următorul format:

```
muchie(START, FINAL, ETICHETA, DEGĂSIT, GĂSIT).
```

Aşa cum se arată în [29], aceasta este cea mai simplă implementare posibilă, chiar dacă nu şi cea mai flexibilă.

Primul pas al algoritmului de analiză sintactică constă în *iniţializarea hărţii* prin introducerea muchiilor inactive corespunzător fiecărui cuvânt din şirul de intrare. Predicatul **init_harta**, care realizează iniţializarea, are ca al treilea argument al său un şir de cuvinte, iar ca prim argument un număr reprezentând poziţia de pe hartă de la care se porneşte. Rolul său este acela de a introduce pe hartă cuvintele adecvate şi de a instanţia cel de-al doilea argument la poziţia finală de pe hartă. Dacă şirul de cuvinte este vid, predicatul nu face nimic şi întoarce poziţia din şir. Altfel, el ia primul cuvânt din şir, adaugă pe hartă câte o muchie *inactivă* corespunzător fiecărei apariţii a cuvântului în lexicon şi apoi continuă în acelaşi mod cu celelalte cuvinte ale şirului. Predicatul **init_harta** se defineşte, prin urmare, astfel:

```
init_harta(V0, V0, []).

init_harta(V0, Vn, [Cuvant|Cuvinte]):-

        V1 is V0 + 1,

        pentru_fiecare(cuvant(Categorie, Cuvant),

              adauga_muchie(V0, V1, Categorie, [],

              [Cuvant, Categorie])),

        init_harta(V1, Vn, Cuvinte).
```

Predicatul **pentru_fiecare** are rolul de a executa un proces de backtracking în căutarea tuturor modalităţilor de satisfacere a primului său argument ca scop Prolog. Corespunzător fiecăreia dintre acestea este invocat cel de-al doilea argument ca scop Prolog. În cazul de faţă, se apelează **adauga_muchie** pentru fiecare combinaţie de tipul cuvânt - categorie găsită de predicatul **cuvant**. Definiţia exactă a predicatului **pentru_fiecare** poate fi văzută în cadrul programului Prolog prezentat la sfârşitul acestui paragraf.

Se observă faptul că în argumentul GĂSIT al fiecărei muchii care trebuie adăugată se include însăşi categoria, întrucât scopul nostru este construirea arborelui de derivare (reprezentat ca o listă de categorii). Prin organizarea argumentului GĂSIT ca o listă incluzând categoria şi arborii de derivare corespunzători grupurilor sintactice deja găsite se construiesc în mod automat arbori de derivare.

Predicatul **adauga_muchie** face însă mai mult decât să afirme existenţa muchiei, pe care o introduce în baza de date. El determină, în plus, ce alte muchii trebuie incluse în hartă ca o consecinţă a acestei adăugări. Mai întâi,

acest predicat verifică dacă muchia deja există, caz în care nu se întreprinde nimic:

```
adauga_muchie(V0, V1, Categorie, Categorii, Analiza):-

        muchie(V0, V1, Categorie, Categorii, Analiza),!.
```

Altfel, este introdusă muchia în baza de date, după care sunt adăugate eventuale alte muchii *active* a căror prezență pe hartă este determinată de includerea muchiei amintite.

În general, acest predicat se poate afla în una dintre următoarele două situații:

- adăugarea unei muchii inactive;
- adăugarea unei muchii active.

Dacă se adaugă hărții, de la **V1** la **V2**, o muchie **inactivă** (argumentul DEGĂSIT este lista vidă) și având eticheta **Categorie1**, atunci, corespunzător fiecărei reguli a gramaticii care are în membrul drept fiul cel mai din stânga de **Categorie1**, predicatul **adauga_muchie** va adăuga o *muchie activă vidă* la **V1**, de tipul dictat de către regula care se aplică. Aceasta este situația în care se utilizează regula bottom-up. Partea a doua a aceleiași clauze Prolog va trata cazul în care se aplică regula fundamentală, și anume: pentru fiecare muchie *activă* care ajunge în **V1** și care caută pe **Categorie1** se adaugă o nouă muchie, care ajunge în **V2** și care poate fi activă sau inactivă. Prin urmare, clauza care se ocupă de *tot ceea ce trebuie făcut în cazul adăugării pe hartă a unei* ***muchii inactive*** este următoarea:

```
adauga_muchie(V1,V2,Categorie1,[],Analiza):-

        asserta(muchie(V1,V2,Categorie1,[],Analiza)),

        pentru_fiecare(regula(Categorie2,[Categorie1|

                        Categorii]),

        adauga_muchie(V1,V1,Categorie2,[Categorie1|Categorii],

                        [Categorie2])),

        pentru_fiecare(muchie(V0,V1,Categorie2,

                        [Categorie1|Categorii], Analize),

        adauga_muchie(V0, V2, Categorie2, Categorii,

                        [Analiza|Analize])).
```

În *cazul adăugării pe hartă a unei* ***muchii active***, se caută fiecare muchie *inactivă* care ar putea permite extinderea muchiei active respective. În această situație, se aplică numai regula fundamentală, după cum urmează:

```
adauga_muchie(V0,V1,Categorie1,[Categorie2|Categorii],
              Analize):-
        asserta(muchie(V0,V1,Categorie1,
                [Categorie2|Categorii],Analize)),
        pentru_fiecare(muchie(V1,V2,Categorie2,[],Analiza),
        adauga_muchie(V0,V2,Categorie1,Categorii,
                      [Analiza|Analize])).
```

Analizorul sintactic este complet, dar se va adăuga un ultim predicat, **parse**, utilizat în interogarea Prologului. Predicatul **parse(Cat, Sir)** îndeplinește următoarele atribuții:

- specifică vărful inițial;
- afișează rezultatele analizei;
- înlătură toate muchiile create în timpul analizei.

Atunci când în rolul argumentului **Cat** se utilizează **S** (simbolul de start) se va face analiza sintactică a unui șir de intrare reprezentând o întreagă propoziție. În general, argumentul **Cat** specifică categoria sintactică care trebuie analizată ori recunoscută (după cum se consideră că am implementat un analizor sintactic sau un dispozitiv de recunoaștere). Afișarea rezultatelor analizei se va face cu ajutorul predicatului **mwrite**. Predicatul **mwrite(Analiza)** este definit direct în program. Acest predicat are rolul de a afișa imaginea în oglindă (inversă) a unui arbore codificat sub formă de listă.

Predicatul **parse(Cat, Sir)** specifică, prin urmare, vârful inițial, după care apelează predicatul **init_harta(V0, Vn, Sir)** care realizează efectiv analiza sintactică. Predicatul **parse(Cat, Sir)** trebuie să țină cont de cele două situații posibile:

- nu există nici o muchie de tipul
  muchie(V0, Vn, Cat, [ ], _),
  deci șirul de intrare nu este corect din punctul de vedere al categoriei specificate și al gramaticii date;

- există măcar o muchie de tipul
  muchie(V0, Vn, Cat, [ ], _),
  adică măcar o analiză a şirului de intrare în conformitate cu gramatica dată şi cu categoria specificată.

În primul dintre cele două cazuri predicatul **parse** eşuează şi nu trebuie decât să fie retrase din baza de date muchiile create pe parcursul analizei. În cel de-al doilea caz se caută toate analizele posibile ale şirului de intrare, se afişează acestea şi se retrag din baza de date toate muchiile create:

```
parse(Cat,Sir):-
            V0 is 1,
            init_harta(V0,Vn,Sir),
            ((\+ muchie(V0,Vn,Cat,[],_),
               retractall(muchie(_,_,_,_,_)),!,fail);
              (pentru_fiecare(muchie(V0,Vn,Cat,[],Analiza),
                              mwrite(Analiza)),
               retractall(muchie(_,_,_,_,_)))).
```

Programul complet, scris în SICStus Prolog şi reprezentând implementarea parser-ului bottom-up cu hartă descris, care utilizează Gramatica nr. 2, este următorul:

Programul 4.6

```
:-dynamic muchie / 5 .

init_harta(V0,V0,[]).
init_harta(V0,Vn,[Cuvant|Cuvinte]):-
     V1 is V0+1,
     pentru_fiecare(cuvant(Categorie,Cuvant),
                    adauga_muchie(V0,V1,Categorie,[],
                                  [Cuvant,Categorie])),
     init_harta(V1,Vn,Cuvinte).


pentru_fiecare(X,Y):-X,executa(Y),fail.
pentru_fiecare(_,_):-true.


executa (Y):-Y,!.
```

```
adauga_muchie(V0,V1,Categorie,Categorii,Analiza):-
              muchie(V0,V1,Categorie,Categorii,Analiza),!.
adauga_muchie(V1,V2,Categorie1,[],Analiza):-
              asserta(muchie(V1,V2,Categorie1,[],Analiza)),
              pentru_fiecare(regula(Categorie2,
                                    [Categorie1|Categorii]),
                             adauga_muchie(V1,V1,Categorie2,
                             [Categorie1|Categorii],[Categorie2])),
              pentru_fiecare(muchie(V0,V1,Categorie2,
                     [Categorie1|Categorii],Analize),
                     adauga_muchie(V0,V2,Categorie2,Categorii,
                                   [Analiza|Analize])).
adauga_muchie(V0,V1,Categorie1,[Categorie2|Categorii],
               Analize):-
               asserta(muchie(V0,V1,Categorie1,
                              [Categorie2|Categorii],Analize)),
           pentru_fiecare(muchie(V1,V2,Categorie2,[],Analiza),
                          adauga_muchie(V0,V2,Categorie1,
                                Categorii,[Analiza|Analize])).

parse(Cat,Sir):-
               V0 is 1,
               init_harta(V0,Vn,Sir),
               ((\+ muchie(V0,Vn,Cat,[],_),
               retractall(muchie(_,_,_,_,_)),!,fail);
               (pentru_fiecare(muchie(V0,Vn,Cat,[],Analiza),
                               mwrite(Analiza)),
               retractall(muchie(_,_,_,_,_)))).

mwrite(Arbore):-
               oglinda(Arbore,Imagine),
               write(Imagine),
               nl.
```

```
oglinda([],[]):-!.
oglinda(Atom,Atom):-atomic(Atom).
oglinda([X1|X2],Imagine):-
                          oglinda(X1,Y2),
                          oglinda(X2,Y1),
                          append(Y1,[Y2],Imagine).

append([], L, L).
append([H|T], L, [H|T1]) :- append(T, L, T1).

regula(s,[np,vp]).
regula(vp,[v,np]).
regula(np,[det,n]).

cuvant(det,un).
cuvant(det,niste).
cuvant(det,o).
cuvant(n,elev).
cuvant(n,elevi).
cuvant(n,carte).
cuvant(v,iubeste).
cuvant(v,iubesc).
```

Interogarea Prologului se face după cum urmează:

```
?- parse(s,[un,elev,iubeste,o,carte]).
```

sau

```
?- parse(vp,[iubeste,o,carte]).
```

În cazul celui de-al doilea exemplu de interogare, de pildă, răspunsul sistemului va fi:

```
[vp,[v,iubeste],[np,[det,o],[n,carte]]]
yes
```

Șirul de intrare a fost acceptat ca reprezentând un grup verbal, iar analiza sintactică a acestuia este afișată pe ecran.

4.5.4. Strategii alternative de invocare a regulilor

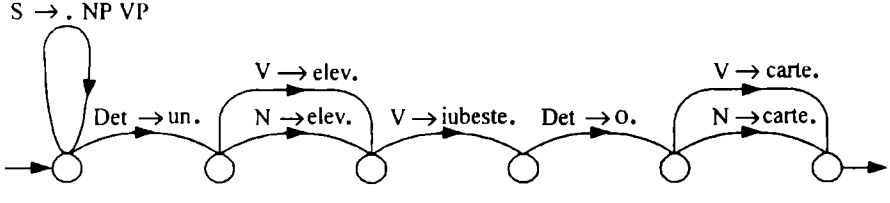
Analizorul sintactic implementat în §4.5.3 a fost unul de tip bottom-up deoarece invocă o regulă de acest gen. Aşa cum s-a văzut, structura de date numită *hartă* este neutră cu privire la strategia de căutare adoptată (depth-first, breadth-first etc.). Aceeaşi neutralitate se extinde şi asupra strategiei de invocare a regulilor. În cele ce urmează, vom prezenta şi implementa, conform [29], un parser care invocă o regulă top-down, adică o regulă care constă din următorii paşi:

Strategia top-down

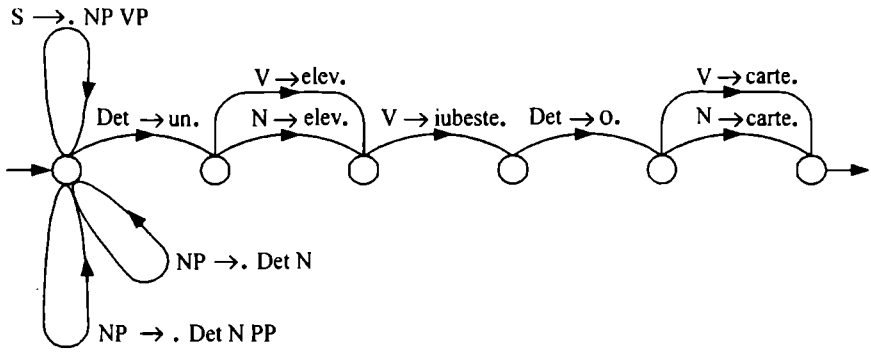
(1) În momentul iniţializării, pentru fiecare regulă de forma $A \rightarrow W$, *unde A reprezintă o categorie care poate "acoperi" o hartă (în mod tipic A desemnează categoria S), se adaugă hărţii muchia* $< 0, 0, A \rightarrow .W >$.

(2) Dacă se adaugă hărţii muchia $< i, j, C \rightarrow W1.BW2 >$, *atunci, pentru fiecare regulă de forma* $B \rightarrow W$, *se inserează pe hartă şi muchia* $< j, j, B \rightarrow .W >$.

Aici, prima clauză asigură faptul că procesul de analiză sintactică începe prin adăugarea unei mulţimi de muchii active vide etichetate cu S şi adăugate vârfului iniţial, ca în exemplul următor (care utilizează Gramatica nr. 2, căreia i s-au adăugat intrările lexicale `cuvant(v,elev)`, `cuvant(v,carte)` şi regula *NP → Det N PP*):



Cea de a doua clauză a strategiei top-down asigură faptul că fiecare muchie activă produce noi muchii active vide, care urmăresc să găsească prima categorie din lista DEGĂSIT corespunzătoare muchiei active adăugate. În cazul exemplului nostru, după aplicarea acestei clauze, harta iniţializată devine:



În mod evident, este posibilă şi conceperea altor strategii de invocare a regulilor, eventual a unor strategii care să combine regulile bottom-up şi top-down. În orice astfel de situaţie trebuie avută în vedere problema *completitudinii* analizorului sintactic, adică a capacităţii acestuia de a găsi toate analizele posibile ale unui şir de intrare.

4.5.5. Implementarea unui parser top-down cu hartă

Programul de implementare a unui **parser top-down cu hartă** este similar cu cel prezentat în §4.5.3, singurele modificări datorându-se faptului că strategia de invocare a regulilor nu mai este una de tip bottom-up, ci este una top-down.

În cazul de faţă, predicatul **parse** realizează iniţializarea hărţii cu muchii lexicale inactive, stabileşte categoria simbolului considerat ca fiind iniţial sau de start (de obicei S) şi adaugă la începutul hărţii muchii active vide etichetate cu această categorie:

```
parse(V0,Vn,Sir):-
                    init_harta(V0,Vn,Sir),
                    initial(Simbol),
                    init_activ(V0,Simbol).
```

Precizăm că, şi de această dată, fiecare adăugare a unei muchii pe hartă poate determina adăugarea unor alte muchii, proces care va fi urmărit în program.

Definiţia predicatului **init_harta** este aceeaşi din programul corespunzător parser-ului bottom-up. (Diferită va fi doar definiţia predicatului **adauga_muchie**, asupra căreia vom reveni).

În ceea ce priveşte definiţia predicatului **init_activ**, aceasta stipulează faptul că, pentru fiecare regulă care extinde **Categorie**, predicatul **init_activ(Varf,Categorie)** creează o muchie activă vidă, în **Varf**, bazată pe acea regulă:

```
init_activ(V0,Categorie):-
      pentru_fiecare(regula(Categorie,Categorii),
                     adauga_muchie(V0,V0,Categorie,Categorii,
                                   [Categorie])).
```

Atât **init_harta** cât şi **init_activ** apelează predicatul **adauga_muchie**, a cărui definiţie va distinge între adăugarea pe hartă a unei muchii active sau a uneia inactive. Înainte de adăugarea oricărei muchii însă acest predicat verifică dacă muchia deja există, caz în care nu face nimic:

```
adauga_muchie(V1,V2,Categorie,Categorii,Analiza):-
                muchie(V1,V2,Categorie,Categorii,Analiza),!.
```

În *cazul adăugării pe hartă a unei* ***muchii inactive*** se are în vedere numai regula fundamentală:

```
adauga_muchie(V1,V2,Categorie1,[],Analiza):-
       asserta(muchie(V1,V2,Categorie1,[],Analiza)),
       pentru_fiecare(muchie(V0,V1,Categorie2,
                             [Categorie1|Categorii],Analize),
                      adauga_muchie(V0,V2,Categorie2,Categorii,
                             [Analiza|Analize])).
```

În *cazul adăugării pe hartă a unei* ***muchii active*** se au în vedere atât regula fundamentală, cât şi regula top_down:

```
adauga_muchie(V1,V2,Categorie1,[Categorie2|Categorii],
                Analize):-
       asserta(muchie(V1,V2,Categorie1,[Categorie2|Categorii],
                      Analize)),
       pentru_fiecare(muchie(V2,V3,Categorie2,[],
                             Analiza),
                      adauga_muchie(V1,V3,Categorie1,Categorii,
                                    [Analiza|Analize])),
       init_activ(V2,Categorie2).
```

Predicatul **test(Sir)** are rolul de a

- specifica vârful inițial
- afișa rezultatele analizei
- înlătura toate muchiile create

și este definit după aceleași principii care au condus la definirea predicatului **parse** din § 4.5.3:

```
test(Sir):-
        V0 is 1,
        initial(Simbol),
        parse(V0,Vn,Sir),
        ((\+ muchie(V0,Vn,Simbol,[],_),
        retractall(muchie(_,_,_,_,_)),!,fail);
        (pentru_fiecare(muchie(V0,Vn,Simbol,[],Analiza),
                        mwrite(Analiza)),
        retractall(muchie(_,_,_,_,_)))).
```

În mod evident, programul va trebui să specifice care este simbolul inițial (sau simbolul de start al gramaticii). Execuția programului corespunzătoare unui simbol de start diferit va fi posibilă, dar va avea ca urmare schimbarea bazei de fapte Prolog. Programul complet, scris în SICStus Prolog și reprezentând implementarea parser-ului top-down cu hartă descris, care utilizează Gramatica nr. 2 în forma ei inițială, cu simbolul de start S, este următorul:

Programul 4.7

```
:-dynamic muchie / 5 .

parse(V0,Vn,Sir):-
                init_harta(V0,Vn,Sir),
                initial(Simbol),
                init_activ(V0,Simbol).


init_harta(V0,V0,[]).
init_harta(V0,Vn,[Cuvant|Cuvinte]):-
     V1 is V0+1,
     pentru_fiecare(cuvant(Categorie,Cuvant),
```

```
                    adauga_muchie(V0,V1,Categorie,[],
                                  [Cuvant,Categorie])),
      init_harta(V1,Vn,Cuvinte).

pentru_fiecare(X,Y):- X, executa(Y), fail.
pentru_fiecare(_,_):- true.

executa(Y):- Y,!.

init_activ(V0,Categorie):-
          pentru_fiecare(regula(Categorie,Categorii),
                         adauga_muchie(V0,V0,Categorie,
                                       Categorii,[Categorie])).

adauga_muchie(V1,V2,Categorie,Categorii,Analiza):-
          muchie(V1,V2,Categorie,Categorii,Analiza),!.
adauga_muchie(V1,V2,Categorie1,[],Analiza):-
          asserta(muchie(V1,V2,Categorie1,[],Analiza)),
          pentru_fiecare(muchie(V0,V1,Categorie2,
                                [Categorie1|Categorii],Analize),
          adauga_muchie(V0,V2,Categorie2,Categorii,
                        [Analiza|Analize])).
adauga_muchie(V1,V2,Categorie1,[Categorie2|Categorii],
              Analize):-
   asserta(muchie(V1,V2,Categorie1,[Categorie2|Categorii],
                  Analize)),
          pentru_fiecare(muchie(V2,V3,Categorie2,[],Analiza),
                         adauga_muchie(V1,V3,Categorie1,
                                       Categorii,
                                       [Analiza|Analize])),
          init_activ(V2,Categorie2).

test(Sir):- V0 is  1,
```

```
    initial(Simbol),
    parse(V0,Vn,Sir),
    ((\+ muchie(V0,Vn,Simbol,[],_),
    retractall(muchie(_,_,_,_,_)),!,fail);
    (pentru_fiecare(muchie(V0,Vn,Simbol,[],Analiza),
                    mwrite(Analiza)),
    retractall(muchie(_,_,_,_,_)))).

mwrite(Arbore):-
                oglinda(Arbore,Imagine),
                write(Imagine),
                nl.

oglinda([],[]):- !.
oglinda(Atom,Atom):-atomic(Atom).
oglinda([X1|X2],Imagine):-
                        oglinda(X1,Y2),
                        oglinda(X2,Y1),
                        append(Y1,[Y2],Imagine).
append([], L, L).
append([H|T], L, [H|T1]):- append(T, L, T1).

regula(s,[np,vp]).
regula(vp,[v,np]).
regula(np,[det,n]).

cuvant(det,un).
cuvant(det,niste).
cuvant(det,o).
cuvant(n,elev).
cuvant(n,elevi).
cuvant(n,carte).
cuvant(v,iubeste).
```

```
cuvant(v,iubesc).

initial(s).
```

Interogarea Prologului se face astfel:

```
?- test([un,elev,iubeste,o,carte]).
```

sau

```
?- test([iubeste,o carte]).
```

dacă se declară în program

```
initial(vp).
```

În acest din urmă caz răspunsul sistemului va fi:

```
[vp,[v,iubeste],[np,[det,o],[n,carte]]]

yes.
```

4.5.6. Strategia de căutare: utilizarea agendei

Așa cum s-a mai arătat, fiecare muchie *activă* reprezintă o *ipoteză* care trebuie explorată. Dacă o astfel de ipoteză înregistrează succes, fie el și parțial, atunci este foarte probabil ca ea să genereze noi ipoteze (muchii active și inactive), datorită acțiunii regulii fundamentale, precum și a strategiei de invocare a regulilor care a fost adoptată. În mod evident, pe parcursul derulării analizei trebuie luate decizii cu privire la *ordinea* în care aceste ipoteze vor fi analizate.Modul în care sunt luate aceste decizii constituie *strategia de căutare*.

În efectuarea *analizei sintactice cu hartă* s-a constatat că este foarte utilă folosirea unei structuri de date suplimentare, sub forma unei **agende**, care să memoreze ipotezele de analizat sau muchiile care trebuie adăugate hărții. În acest mod se evită investigarea unor ipoteze identice și se iau decizii cu privire la ordinea și momentul în care este analizată fiecare ipoteză.

O posibilitate existentă este aceea de a organiza agenda ca pe o **stivă**. Astfel, pe măsură ce sunt generate noi muchii, acestea se plasează în stivă. Atunci când trebuie selectată o muchie cu care să se lucreze este aleasă cea din vârful stivei. Această tehnică produce un comportament similar cu cel al căutării de tip depth-first. Fiecare posibilitate existentă este urmărită cât de mult posibil înainte de a se încerca o alternativă a ei. O altă strategie posibilă este aceea de a privi agenda ca pe o **coadă**. În acest caz, pe măsură ce se produc noi muchii, ele sunt plasate la sfârșitul cozii. Atunci când trebuie aleasă o muchie cu care să se lucreze este selectată cea din fața cozii, ceea ce produce un comportament similar cu cel al căutării de tip *breadth_first*. Dacă muchiile diferite generate la fiecare pas reprezintă alternative, atunci putem spune că sistemul petrece numai

o mică cantitate de timp analizând fiecare posibilitate înainte de a comuta la următoarea.

Multe alte strategii sunt posibile prin manevrarea agendei în moduri diferite. Spre exemplu, muchiile *active* pot fi poziționate în agendă în funcție de cât de aproape se află de stadiul reprezentării unor constituenți completi. Într-o strategie de acest tip muchiile vide ar fi plasate la sfârșitul listei ce reprezintă agenda, în timp ce acelea care mai caută o unică categorie ar fi plasate la începutul ei, ori pe dos. Pot fi favorizate muchiile care rezultă din aplicarea anumitor reguli ale gramaticii față de cele care rezultă în urma aplicării altor reguli, exploatându-se, în acest fel, rezultatele și informațiile statistice existente referitoare la utilizarea limbajului natural. De asemenea, anumite muchii pot fi favorizate în conformitate cu informațiile obținute din alte componente ale sistemului, cum ar fi cea semantică. Dacă se dorește ca analizorul sintactic implementat să producă *toate* analizele posibile, atunci harta va produce, până la urmă, exact același rezultat, indiferent de strategia de căutare impusă. Prin urmare, dacă *ordinea* în care analizele posibile ale șirului de intrare sunt generate nu prezintă interes, atunci alegerea strategiei de căutare rămâne, în esență, arbitrară. Dacă, în schimb, se caută o unică analiză, ori numai acele analize care respectă anumite condiții (ce s-ar putea să nu derive exclusiv din regulile gramaticii), ori dacă se dorește execuția în paralel a unui alt proces (cum ar fi interpretarea semantică a constituenților), atunci strategia de căutare aleasă poate avea implicații serioase asupra eficienței, fenomen asupra căruia vom reveni.

4.5.7. Implementarea controlului flexibil

Analizorii sintactici cu hartă descriși până în prezent au în comun două caracteristici majore:

- utilizează baza de date Prolog pentru memorarea hărții;
- în ceea ce privește *strategia de control*, nu lucrează în mod explicit cu o agendă, ci *se bazează pe procesul de backtracking* efectuat în mod automat de către Prolog.

Această abordare ușurează implementarea și scurtează codul sursă al programului, dar sacrifică flexibilitatea. În cele ce urmează, vom menține harta utilizată până în prezent și vom lucra în mod explicit cu o structură de date suplimentară de tipul agendei descrise anterior. Acestea vor fi manevrate ca liste de muchii care sunt transmise ca argumente de la predicat la predicat. În acest mod devine posibilă comutarea între procesarea de tip depth-first și cea de tip breadth-first, prin simpla interschimbare a ordinii argumentelor într-o unică apelare de predicat.

Structura de date care va sta la baza implementărilor analizorilor sintactici pe care îi vom prezenta în continuare este tot aceea de *muchie*, având următoarea reprezentare propusă în [29]:

muchie(START, FINAL, ETICHETA, DEGĂSIT).

Se observă eliminarea componentei finale GĂSIT, utilizată în reprezentarea de până acum a muchiilor. Aceasta înseamnă că, pentru a simplifica codul sursă, programele prezentate în cele ce urmează vor implementa mai degrabă un *dispozitiv de recunoaştere* şi nu un analizor sintactic propriu-zis. Astfel, în momentul interogării Prologului, programul va răspunde **yes** sau **no**, după cum şirul de intrare este recunoscut sau nu ca fiind corect din punctul de vedere al gramaticii date şi eventual al categoriei specificate, fără a afişa pe ecran analiza sintactică propriu-zisă a acestuia, ca în cazurile de până acum (§4.5.3 şi §4.5.5). *Hărţile şi agendele* vor fi privite ca nişte *liste* ale căror elemente sunt muchii având reprezentarea amintită.

4.5.7.1. Implementarea unui parser bottom-up cu hartă şi agendă

Pasul de iniţializare al algoritmului de analiză sintactică bottom-up va fi asigurat de către predicatul (de iniţializare) **init_agenda(Sir, Varf, Agenda)**, care are rolul de a crea o *agendă iniţială* ce conţine *muchii lexicale*, fiind dat şirul de intrare **Sir**, care începe de la nodul **Varf**. Dacă şirul de intrare este vid, atunci se creează o agendă iniţială vidă:

```
init_agenda([],_,[]).
```

Altfel, se izolează primul cuvânt, se consultă lexiconul şi se plasează într-o agendă toate muchiile inactive care "acopăr" cuvântul. Această agendă se adaugă (prin intermediul predicatului **append**, definit, în modul uzual, direct în program) în faţa agendei obţinute ca urmare a consultării lexiconului relativ la celelalte cuvinte:

```
init_agenda([Cuvant|Cuvinte],V0,Agenda):-
                V1 is V0+1,
                findall(muchie(V0,V1,Categorie,[]),
                        cuvant(Categorie,Cuvant),
                        Agenda1),
                init_agenda(Cuvinte,V1,Agenda2),
                append(Agenda1,Agenda2,Agenda).
```

Menționăm că predicatul **findall**, predefinit în SICStus Prolog, are rolul de a forma, în cel de-al treilea argument al său, o listă care conține, pentru fiecare soluție a celui de-al doilea argument, instanțierea adecvată a primului argument. Atunci când cel de-al doilea argument al său nu are soluții, predicatul **findall** produce, în cel de-al treilea argument, lista vidă. În cazul de față, **findall** realizează o iterație a tuturor modurilor posibile de a găsi **Cuvant** ca pe o instanțiere a lui **Categorie**. Pentru fiecare soluție existentă, termenul **muchie** adecvat este inclus în lista rezultat (**Agenda 1**).

Predicatul **extinde_muchii(Agenda, InHarta, OutHarta)** este folosit pentru adăugarea muchiilor din **Agenda** unei hărți inițiale **InHarta**, adăugare ce trebuie să ia în considerare atât regula fundamentală, cât și regula bottom-up, pentru a genera noi muchii. Aceste acțiuni au ca rezultat crearea unei noi hărți numite **OutHarta**. Atunci când agenda este vidă, acest predicat reprezintă funcția identitate:

```
extinde_muchii([],Harta,Harta).
```

Altfel, predicatul ia în considerare prima muchie din agendă și vede dacă aceasta este deja plasată pe hartă. În caz afirmativ, el continuă parcurgerea agendei:

```
extinde_muchii([Muchie|Agenda1],Harta1,Harta2):-
                    membru(Muchie,Harta1),!,
                    extinde_muchii(Agenda1,Harta1,Harta2).
```

În caz contrar, **extinde_muchii** adaugă hărții prima muchie, caută muchiile noi care sunt generate ca urmare a acestei adăugări, include aceste noi muchii în agendă și apoi continuă procesul de analiză sintactică utilizând noua agendă și harta mărită:

```
extinde_muchii([Muchie|Agenda1],Harta1,Harta3):-
                    Harta2 = [Muchie|Harta1],
                    muchii_noi(Muchie,Harta2,Muchii),
                    adauga_muchii(Muchii,Agenda1,Agenda2),
                    extinde_muchii(Agenda2,Harta2,Harta3).
```

O muchie se adaugă unei liste de muchii existente fie constatând că ea face deja parte din listă, fie printr-o includere directă în lista respectivă (pe post de cap al listei):

```
adauga_muchie(Muchie,Muchii,Muchii):-
                    membru(Muchie,Muchii),!.
adauga_muchie(Muchie,Muchii,[Muchie|Muchii]).
```

Definiția aproape evidentă a predicatului **membru** poate fi văzută direct în program.

Două liste de muchii sunt combinate prin apelarea predicatului **adauga_muchie** într-o manieră recursivă, de către predicatul **adauga_muchii**:

```
adauga_muchii([],Muchii,Muchii).
adauga_muchii([Muchie|Muchii],Muchii1,Muchii3):-
                    adauga_muchie(Muchie,Muchii1,Muchii2),
                    adauga_muchii(Muchii,Muchii2,Muchii3).
```

Revenind la predicatul **extinde_muchii**, care adaugă hărții prima muchie din agendă în cazul în care agenda este nevidă iar muchia nu se regăsește deja pa hartă, trebuie analizată posibila apariție a unor noi muchii ca urmare a acestei adăugări. Cu alte cuvinte, trebuie luate în considerație acțiunile regulii fundamentale și respectiv ale regulii bottom-up. Această analiză va fi realizată de către predicatul **muchii_noi(Muchie, Harta, Muchii)**, care face corespondența dintre **Muchie** și **Harta** și o mulțime de noi muchii (**Muchii**), apărută în urma adăugării lui **Muchie** la **Harta**. Același predicat are rolul de a adăuga hărții muchiile active vide care declanșează procesul de analiză sintactică.

În definirea predicatului **muchii_noi** trebuie luate în considerare cele două cazuri posibile în inserarea pe hartă a unei noi muchii:

- adăugarea unei muchii inactive;
- adăugarea unei muchii active.

Dacă se adaugă, de la **V1** la **V2**, o muchie inactivă etichetată **Categorie1**, atunci, pentru fiecare regulă care are în membrul drept fiul cel mai din stânga de **Categorie1**, se creează în **V1** o muchie activă vidă de tipul impus de către regula care se utilizează. Aceasta este consecința aplicării regulii bottom-up. În continuare, conform regulii fundamentale, pentru fiecare muchie activă existentă care se termină în **V1** și caută **Categorie1**, se adaugă o muchie care ajunge în **V2** și care s-ar putea să fie activă ori inactivă. În final, cele două mulțimi de muchii care rezultă se combină. Acestea sunt acțiunile care derivă din *adăugarea pe hartă a unei* ***muchii inactive***. Ele sunt exprimate de următoarea clauză Prolog:

```
muchii_noi(muchie(V1,V2,Categorie1,[]),Harta,Muchii):-
   findall(muchie(V1,V1,Categorie2,[Categorie1|Categorii1]),
          regula(Categorie2,[Categorie1|Categorii1]),Muchii1),
   findall(muchie(V0,V2,Categorie3,Categorii2),
      membru(muchie(V0,V1,Categorie3,[Categorie1|Categorii2]),
                    Harta),Muchii2),
   adauga_muchii(Muchii1,Muchii2,Muchii).
```

În *situația adăugării pe hartă a unei* ***muchii active*** va acționa numai regula fundamentală. În acest caz, se caută fiecare muchie inactivă care ar putea permite extinderea muchiei active și se adaugă muchiile găsite operând extinderile de rigoare:

```
muchii_noi(muchie(V1,V2,Categorie1,[Categorie2|Categorii]),
                    Harta,Muchii):-
          findall(muchie(V1,V3,Categorie1,Categorii),
            membru(muchie(V2,V3,Categorie2,[],Harta),Muchii).
```

Întrucât toate cazurile posibile au fost luate în considerare, vom introduce un ultim predicat, **parse**, utilizat în interogarea Prologului și care reunește predicatele **init_agenda** și **extinde_muchii**. Predicatul **parse(Cat, Sir)** îndeplinește atribuțiile de a

- specifica vârful inițial;
- verifica dacă harta conține într-adevăr o muchie care acoperă *întreg* șirul de intrare relativ la categoria specificată:

```
parse(Cat,Sir):-
                init_agenda(Sir,0,Agenda),
                extinde_muchii(Agenda,[],Harta),
                membru(muchie(0,M,Cat,[]),Harta),
                N is M+1,
                \+ (membru(muchie(_,N,_,_),Harta)).
```

Programul complet, scris în SICStus Prolog și reprezentând implementarea parser-ului bottom-up descris, care utilizează Gramatica nr. 2 (în forma ei inițială), este următorul:

Programul 4.8

```
:- dynamic muchie / 4.

init_agenda([],_,[]).
init_agenda([Cuvant|Cuvinte],V0,Agenda):-
              V1 is V0+1,
              findall(muchie(V0,V1,Categorie,[]),
                      cuvant(Categorie,Cuvant),Agenda1),
              init_agenda(Cuvinte,V1,Agenda2),
              append(Agenda1,Agenda2,Agenda).

append([],L,L).
append([H|T],L,[H|T1]):-append(T,L,T1).

extinde_muchii([],Harta,Harta).
extinde_muchii([Muchie|Agenda1],Harta1,Harta2):-
                 membru(Muchie,Harta1),!,
                 extinde_muchii(Agenda1,Harta1,Harta2).
extinde_muchii([Muchie|Agenda1],Harta1,Harta3):-
                 Harta2 = [Muchie|Harta1],
                 muchii_noi(Muchie,Harta2,Muchii),
                 adauga_muchii(Muchii,Agenda1,Agenda2),
                 extinde_muchii(Agenda2,Harta2,Harta3).

membru(X,[X|_]).
membru(X,[_|Y]):- membru(X,Y).

adauga_muchie(Muchie,Muchii,Muchii):-
               membru(Muchie,Muchii),!.
adauga_muchie(Muchie,Muchii,[Muchie|Muchii]).

adauga_muchii([],Muchii,Muchii).
```

```
adauga_muchii([Muchie|Muchii],Muchii1,Muchii3):-
                adauga_muchie(Muchie,Muchii1,Muchii2),
                adauga_muchii(Muchii,Muchii2,Muchii3).

muchii_noi(muchie(V1,V2,Categorie1,[]),Harta,Muchii):-
           findall(muchie(V1,V1,Categorie2,
                   [Categorie1|Categorii1]),
               regula(Categorie2,[Categorie1|Categorii1]),
                      Muchii1),
           findall(muchie(V0,V2,Categorie3,Categorii2),
                   membru(muchie(V0,V1,Categorie3,
                          [Categorie1|Categorii2]),Harta),
                          Muchii2),
           adauga_muchii(Muchii1,Muchii2,Muchii).
muchii_noi(muchie(V1,V2,Categorie1,[Categorie2|Categorii]),
             Harta,Muchii):-
           findall(muchie(V1,V3,Categorie1,Categorii),
               membru(muchie(V2,V3,Categorie2,[]),Harta),
               Muchii).

parse(Cat,Sir):-
                init_agenda(Sir,0,Agenda),
                extinde_muchii(Agenda,[],Harta),
                membru(muchie(0,M,Cat,[]),Harta),
                N is M + 1,
                \+ (membru(muchie(_,N,_,_),Harta)).

regula(s,[np,vp]).
regula(vp,[v,np]).
regula(np,[det,n]).

cuvant(det,un).
cuvant(det,niste).
```

```
cuvant(det,o).
cuvant(n,elev).
cuvant(n,elevi).
cuvant(n,carte).
cuvant(v,iubeste).
cuvant(v,iubesc).
```

Interogarea Prologului se face astfel:

```
?- parse(s,[un,elev,iubeste,o,carte]).
yes
```

sau

```
?- parse(vp,[iubeste,o,carte]).
yes
```

Se observă că analizorul sintactic considerat operează în manieră **depth-first** (datorită ordinii în care muchiile sunt adăugate în agendă). Astfel, în prezent, agenda este organizată ca o *stivă*, noile muchii care rezultă în urma adăugării unei muchii fiind inserate în fața agendei, adică în vârful stivei.

Agenda poate fi organizată, în egală măsură, ca o *coadă*, caz în care analizorul sintactic va explora spațiul de căutare în manieră **breadth-first**.

Diferența în organizarea agendei poate fi foarte ușor pusă în evidență în cadrul definiției predicatului **extinde_muchii**. Ordinea primelor două argumente (reprezentând agenda existentă și noile muchii) ale predicatului **adauga_muchii**, apelat de către **extinde_muchii** pentru a construi noua agendă, este cea care determină organizarea acesteia:

```
extinde_muchii([Muchie|Agenda1],Harta1,Harta3):-
        Harta2 = [Muchie|Harta1],
        muchii_noi(Muchie,Harta2,Muchii),
        %adauga_muchii(Muchii,Agenda1,Agenda2),
        %procesare depth-first
        adauga_muchii(Agenda1,Muchii,Agenda2),
        %procesare breadth-first
        extinde_muchii(Agenda2,Harta2,Harta3).
```

4.5.7.2. Implementarea unui parser top-down cu hartă și agendă

Programul care implementează un parser top-down în acest context va utiliza o serie dintre predicatele definite anterior (§4.5.7.1). Este vorba despre predicatele **init_agenda**, **adauga_muchie**, **adauga_muchii** și **extinde_muchii**, care toate rămân neschimbate.

Predicatul **init_agenda** creează o agendă inițializată cu muchii lexicale. Întrucât, în acest caz, nu acționează regula bottom-up, această agendă poate fi considerată ca reprezentând harta inițială.

Este necesară, de asemenea, introducerea pe harta inițială a unor muchii active, lucru care va fi realizat, ca și în trecut, de predicatul **init_activ**. Definiția acestuia este una directă: pentru fiecare regulă care extinde **Categorie**, predicatul **init_activ(Categorie, Varf, Muchii)** creează, la **Varf**, o muchie activă vidă bazată pe regula respectivă. Muchia este introdusă în lista **Muchii**, după cum urmează:

```
init_activ(Categorie,Varf,Muchii):-
      findall(muchie(Varf,Varf,Categorie,Categorii),
              regula(Categorie,Categorii),Muchii).
```

Regula fundamentală a analizei sintactice cu hartă, precum și regula top-down vor fi încorporate în predicatul **muchii_noi(Muchie, Harta, Muchii)**, care face corespondența dintre **Muchie** și **Harta** și o mulțime de muchii noi, numită **Muchii**, care ia naștere prin adăugarea lui **Muchie** la **Harta**. În definirea predicatului **muchii-noi** trebuie din nou luate în considerare cele două cazuri posibile în inserarea pe hartă a unei noi muchii:

- adăugarea unei muchii inactive;
- adăugarea unei muchii active.

Dacă se adaugă, de la **V1** la **V2**, o muchie inactivă etichetată **Categorie1**, atunci, pentru fiecare muchie activă existentă, care se termină în **V1** și care caută **Categorie1**, se adaugă o nouă muchie, care se termină în **V2** și care poate fi atât activă, cât și inactivă:

```
muchii_noi(muchie(V1,V2,Categorie1,[]),Harta,Muchii):-
   findall(muchie(V0,V2,Categorie2,Categorii),
       membru(muchie(V0,V1,Categorie2,[Categorie1|Categorii]),
              Harta),
       Muchii).
```

Această clauză tratează *cazul adăugării unei **muchii inactive***.

În situația în care *se adaugă hărții o **muchie activă***, trebuie avute în vedere atât acțiunea regulii fundamentale, cât și cea a regulii top-down. Prin urmare, se vor executa următoarele trei operații:

- se creează muchiile active vide relevante care caută următoarea categorie (strategie top-down);
- se caută fiecare muchie inactivă care permite extinderea muchiei active adăugate și se creează, în mod efectiv, noi muchii, prin extinderea acesteia;
- se combină cele două mulțimi de muchii rezultate.

Clauza Prolog corespunzătoare, care încorporează aceste trei acțiuni, este următoarea:

```
muchii_noi(muchie(V1,V2,Categorie1,[Categorie2|Categorii]),
           Harta,Muchii):-
   init_activ(Categorie2,V2,Muchii1),
   findall(muchie(V1,V3,Categorie1,Categorii),
           membru(muchie(V2,V3,Categorie2,[]),Harta),Muchii2),
   append(Muchii1,Muchii2,Muchii).
```

Interogarea Prologului se va realiza prin intermediul predicatului **parse(Cat, Sir)**, care

- inițializează harta;
- creează, la începutul hărții, muchia (muchiile) activă vidă corespunzătoare categoriei analizate (recunoscute);
- apelează predicatul **extinde_muchii**, care realizează efectiv analiza sintactică a șirului de intrare **Sir** relativ la categoria specificată **Cat**;
- elaborează harta până la final efectuând o verificare a sfârșitului șirului de intrare (i.e. vede dacă harta conține într-adevăr o muchie care acoperă *întreg* șirul de intrare relativ la categoria specificată;
- verifică dacă s-a înregistrat succes în analiza (recunoașterea) șirului de intrare:

```
parse(Cat,Sir):-
                init_agenda(Sir,0,Harta1),
```

```
        init_activ(Cat,0,Agenda),
        extinde_muchii(Agenda,Harta1,Harta2),
        membru(muchie(0,M,Cat,[]),Harta2),
        N is M+1,
        \+ (membru(muchie(_,N,_,_),Harta2)).
```

Programul complet, scris în SICStus Prolog şi reprezentând implementarea parser-ului top-down descris, care utilizează Gramatica nr. 2 (în forma ei iniţială), este următorul:

Programul 4.9

```
:- dynamic muchie / 4.

init_agenda([],_,[]).
init_agenda([Cuvant|Cuvinte],V0,Agenda):-
            V1 is V0+1,
            findall(muchie(V0,V1,Categorie,[]),
                    cuvant(Categorie,Cuvant),
                    Agenda1),
            init_agenda(Cuvinte,V1,Agenda2),
            append(Agenda1,Agenda2,Agenda).


append([],L,L).
append([H|T],L,[H|T1]):-append(T,L,T1).


extinde_muchii([],Harta,Harta).
extinde_muchii([Muchie|Agenda1],Harta1,Harta2):-
                membru(Muchie,Harta1),!,
                extinde_muchii(Agenda1,Harta1,Harta2).
extinde_muchii([Muchie|Agenda1],Harta1,Harta3):-
                Harta2 = [Muchie|Harta1],
                muchii_noi(Muchie,Harta2,Muchii),
                adauga_muchii(Muchii,Agenda1,Agenda2),
                extinde_muchii(Agenda2,Harta2,Harta3).
```

```
membru(X,[X|_]).
membru(X,[_|Y]):- membru(X,Y).

adauga_muchie(Muchie,Muchii,Muchii):-
                membru(Muchie,Muchii),! .
adauga_muchie(Muchie,Muchii,[Muchie|Muchii]).

adauga_muchii([],Muchii,Muchii).
adauga_muchii([Muchie|Muchii],Muchii1,Muchii3):-
              adauga_muchie(Muchie,Muchii1,Muchii2),
              adauga_muchii(Muchii,Muchii2,Muchii3).

init_activ(Categorie,Varf,Muchii):-
      findall(muchie(Varf,Varf,Categorie,Categorii),
              regula(Categorie,Categorii),Muchii).

muchii_noi(muchie(V1,V2,Categorie1,[]),Harta,Muchii):-
           findall(muchie(V0,V2,Categorie2,Categorii),
                   membru(muchie(V0,V1,Categorie2,
                                 [Categorie1|Categorii]),
                          Harta),
                   Muchii).
muchii_noi(muchie(V1,V2,Categorie1,[Categorie2|Categorii]),
           Harta,Muchii):-
   init_activ(Categorie2,V2,Muchii1),
   findall(muchie(V1,V3,Categorie1,Categorii),
           membru(muchie(V2,V3,Categorie2,[]),Harta),Muchii2),
   append(Muchii1,Muchii2,Muchii).

parse(Cat,Sir):-
               init_agenda(Sir,0,Harta1),
               init_activ(Cat,0,Agenda),
               extinde_muchii(Agenda,Harta1,Harta2),
```

```
                membru(muchie(0,M,Cat,[]),Harta2),
                N is M+1,
                \+ (membru(muchie(_,N,_,_),Harta2)).

regula(s,[np,vp]).
regula(vp,[v,np]).
regula(np,[det,n]).

cuvant(det,un).
cuvant(det,niste).
cuvant(det,o).
cuvant(n,elev).
cuvant(n,elevi).
cuvant(n,carte).
cuvant(v,iubeste).
cuvant((v,iubesc).
```

Interogarea Prologului se face astfel:

```
? - parse(s,[un,elev,iubeste,o,carte]).
yes
```

sau

```
?- parse(vp,[iubeste,o,carte]).
yes
```

4.5.7.3. Variante ale analizorilor sintactici bottom-up şi top-down cu hartă şi agendă

Orice variantă a analizorilor sintactici cu hartă prezentaţi deja va lua în considerare faptul că operaţia de bază efectuată de un asemenea parser constă în combinarea unei muchii active cu o muchie inactivă, aceasta din urmă reprezentând un constituent gata completat, adică un aşa-numit **constituent complet**. Rezultatul acestei operaţii de bază este fie apariţia unui nou constituent complet, fie a unei noi muchii active, care reprezintă o extindere a muchiei active iniţiale.

Variantele prezentate aici şi preluate din [4] ale analizorilor sintactici implementaţi până în acest moment organizează în mod diferit lista

reprezentând *agenda*, în sensul că aceasta *conține numai constituenți compleți*. Astfel, noii constituenți compleți, rezultați în urma aplicării regulii fundamentale, sunt incluși în agendă, până în momentul în care vor fi adăugați hărții. Acest proces este descris în mod exact de către următorul *Algoritm de extindere a muchiilor*:

Algoritmul 4.5

Pentru a adăuga un constituent C de la poziția p_1 la poziția p_2, execută:

1. Inserează C în hartă de la p_1 la p_2.
2. Pentru orice muchie activă de forma $X \rightarrow X_1 \ldots \bullet C \ldots X_n$ de la p_0 la p_1, adaugă o muchie activă nouă $X \rightarrow X_1 \ldots C\bullet \ldots X_n$ de la p_0 la p_2.
3. Pentru orice muchie activă de forma $X \rightarrow X_1 \ldots X_n \bullet C$ de la p_0 la p_1, adaugă în agendă un nou constituent de tipul X, de la p_0 la p_2.

Fiind dat Algoritmul 4.5 de extindere a muchiilor, **algoritmul de analiză sintactică bottom-up cu hartă și agendă** este următorul:

Algoritmul 4.6

Execută următorii pași până când se parcurge întreg șirul de intrare:

1. Dacă agenda este vidă, se caută și se introduc în agendă toate interpretările pentru următorul cuvânt din șirul de intrare.
2. Se alege un constituent din agendă. Fie acesta constituentul C de la poziția p_1 la poziția p_2.
3. Pentru fiecare regulă a gramaticii de forma $X \rightarrow CX_1 \ldots X_n$, se adaugă o muchie activă de forma $X \rightarrow C\bullet X_1 \ldots X_n$ de la poziția p_1 la poziția p_2.
4. Se adaugă hărții constituentul C utilizând Algoritmul 4.5 de extindere a muchiilor.

Ca și până acum, se poate folosi o strategie de căutare de tip depth-first sau una de tip breadth-first, după cum agenda este implementată sub formă de stivă ori de coadă. Remarcăm faptul că, pentru aplicarea unei strategii veritabile de tip breadth-first, este necesară citirea întregului șir de intrare cu adăugarea în agenda inițială a tuturor interpretărilor cuvintelor acestuia înainte de declanșarea algoritmului propriu-zis de analiză sintactică.

În cele ce urmează, prezentăm un **algoritm de analiză sintactică top-down cu hartă** care se bazează, de asemenea, pe o *agendă alcătuită numai din constituenți compleți* și pe Algoritmul 4.5 de extindere a muchiilor. Diferența fundamentală față de algoritmul top-down cu hartă prezentat anterior (§4.5.7.2) constă tot în faptul că agenda este formată numai din constituenți compleți. De asemenea, în algoritmul care urmează, inițializarea agendei nu se va face prin introducerea prealabilă a tuturor muchiilor lexicale posibile, cuvintele șirului de intrare fiind pe rând interpretate cu ajutorul lexiconului, atunci când se constată că agenda este vidă. În abordarea top-down care urmează și a cărei implementare este lăsată în seama cititorului, sunt generate muchii active noi ori de câte ori o nouă muchie activă este adăugată hărții, conform **Algoritmului de introducere top-down a muchiilor** [care reproduce pasul general (2) al strategiei top-down enunțate în §4.5.4]:

Algoritmul 4.7

Pentru a adăuga o muchie $S \rightarrow C_1 \ldots \bullet C_i \ldots C_n$ care se termină la poziția j, execută:

Pentru fiecare regulă a gramaticii de forma $C_i \rightarrow X_1 \ldots X_k$ *adaugă în mod recursiv noua muchie* $C_i \rightarrow \bullet X_1 \ldots X_k$ *de la poziția j la poziția j.*

Algoritmul de analiză sintactică top-down cu hartă este următorul:

Algoritmul 4.8

Inițializare: Pentru fiecare regulă a gramaticii de forma $S \rightarrow X_1 \ldots X_k$, se adaugă o muchie etichetată $S \rightarrow \bullet X_1 \ldots X_k$ utilizând *Algoritmul de introducere top-down a muchiilor.*

Analiză: Execută până când se parcurge întreg șirul de intrare:

1. Dacă agenda este vidă, caută interpretările posibile ale cuvântului următor al șirului de intrare și adaugă-le agendei.
2. Selectează un constituent din agendă. Fie acesta constituentul C.
3. Combină C cu fiecare muchie activă de pe hartă folosind Algoritmul de extindere a muchiilor. Adaugă agendei eventualii constituenți noi care rezultă.
4. Adaugă hărții orice muchie activă creată la pasul 3 utilizând Algoritmul de introducere top-down a muchiilor.

Algoritmii 4.6 și 4.8 nu reprezintă un salt calitativ din punctul de vedere al eficienței, dar ușurează implementarea analizorilor sintactici top-down și bottom-up în alte limbaje decât Prologul, și anume în *limbajele care admit accesul direct*. Caracteristica acestor *variante* ale algoritmilor top-down și bottom-up cu hartă și agendă prezentați anterior este aceea că în agendă se depun numai constituenți completi. Prin urmare, agenda este menținută ca având o structură cât mai simplă, restul fiind plasat pe hartă, care se prelucrează în *acces direct*. Tocmai de aceea se recomandă implementarea acestor algoritmi în alte limbaje decât Prologul.

Majoritatea algoritmilor de analiză sintactică au fost dezvoltați la mijlocul anilor '60, de către informaticieni, cu scopul principal de a analiza limbajele de programare și nu limbajul natural. O trimitere clasică pentru studiul acestui domeniu este aceea la Aho, Sethi și Ullman (1986) dar, mai ales, la Aho și Ullman (1972).

Conceptul de hartă este descris în special de către Kay (1973; 1980) și a fost preluat și adaptat de către numeroase sisteme ulterioare bazate pe analizori sintactici. Algoritmul 4.6, prezentat aici, este similar Algoritmului de analiză sintactică din colțul stâng al lui Aho și Ullman (1972), în timp ce Algoritmul 4.8 este similar celui descris de către Earley (1970) și, prin urmare, este numit Algoritmul lui Earley.

Analiza sintactică de tip Earley este definită pentru gramatici independente de context, algoritmul corespunzător lucrând în manieră top-down și de la stânga la dreapta. Algoritmul lui Earley (1970) este printre primii algoritmi bazați pe gramatici PS independente de context care explorează în mod sistematic îmbunătățirile legate de eficiență realizate prin memorarea rezultatelor intermediare. Printre caracteristicile algoritmului lui Earley se numără următoarele proprietăți:

- analizează propoziții alcătuite din n cuvinte într-un interval de timp proporțional cu cel mult n^3, ceea ce se apropie de cea mai bună performanță posibilă;
- manevrează constituenții nuli în mod corect;
- nu ciclează atunci când întâlnește reguli recursive la stânga ($A \rightarrow AB$)[4].

Ideea centrală a acestui algoritm este aceea că harta poate memora atât constituenți completi, cât și constituenți incompleți, la sfârșitul procesului de analiză sintactică toate analizele alternative găsindu-se pe hartă. Pentru o

[4] S-a arătat că algoritmul lui Earley ciclează, totuși, în cazul utilizării unei reguli a gramaticii în care unul dintre nodurile fii are un argument care conține un argument al nodului părinte, adică în cazul unei reguli de tipul: a(X) --> a(f(X)).

descriere şi o implementare în Prolog a algoritmului lui Earley, a se vedea şi [19].

4.5.8. Eficienţă

Aşa cum se arată în [29], eficienţa unui analizor sintactic cu hartă dat, relativ la un şir dat de intrare, se află într-o relaţie invers proporţională cu cantitatea de "structură în exces" care va fi prezentă pe hartă la sfârşitul procesului de analiză sintactică.

Dacă, spre exemplu, şirul de intrare nu este corect relativ la gramatica specificată, dar harta conţine un număr mare de muchii la sfârşitul analizei, aceasta înseamnă că analizorul sintactic a efectuat un număr semnificativ de operaţii inutile. Într-un astfel de caz ar fi de dorit ca parser-ul să construiască foarte puţine muchii înainte de a da răspunsul conform căruia şirul de intrare nu poate fi analizat din punctul de vedere al gramaticii date.

În situaţia în care şirul de intrare este corect iar harta conţine un număr mare de muchii, atunci parser-ul este unul eficient dacă fiecare muchie inactivă prezentă este folosită în una sau mai multe dintre structurile atribuite şirului şi dacă fiecare dintre muchiile active prezente reprezintă o ipoteză care, până la urmă, va înregistra succes. Muchiile inactive neutilizate, precum şi muchiile active care nu conduc la nimic reflectă, în mod direct, lipsa de eficienţă a parser-ului. Experimentarea cu strategiile alternative de invocare a regulilor are ca scop tocmai reducerea la maximum a acestei ineficienţe.

Eficienţa unui analizor sintactic cu hartă este, de asemenea, mult influenţată de modul în care sunt armonizate structurile de date utilizate pentru a susţine operaţiile de bază care se repetă în decursul procesului de analiză sintactică. Principalele operaţii de bază, în decursul cărora parser-ul regăseşte informaţie utilizând atât harta, cât şi gramatica, sunt următoarele:

- atunci când *se adaugă hărţii o muchie inactivă*, parser-ul caută (pe hartă) muchiile active necesare aplicării regulii fundamentale;
- atunci când *se adaugă hărţii o muchie activă*, parser-ul caută (pe hartă) muchiile inactive necesare aplicării regulii fundamentale;
- atunci când *se adaugă hărţii o muchie inactivă*, parser-ul caută toate regulile gramaticii în care prima categorie a membrului drept coincide cu categoria muchiei, cu scopul de a aplica regula bottom-up;
- atunci când *se adaugă hărţii o muchie activă*, parser-ul caută toate regulile gramaticii al căror membru stâng reprezintă prima categorie cerută de muchie, cu scopul de a aplica regula top-down.

Este evident faptul că performanța ar scădea în mod dramatic odată cu mărirea hărții dacă parser-ul ar fi nevoit să parcurgă toate muchiile acesteia ori de câte ori caută o muchie de un anumit tip. Acesta este tipul de performanță scăzută care se obține atunci când harta este reprezentată sub forma unei simple liste de structuri de date de tip *muchie*, ca în cazul analizorilor sintactici prezentați anterior (§4.5.3, §4.5.5, §4.5.7.1, §4.5.7.2). O implementare mai eficientă ar putea grupa, spre exemplu, muchiile hărții într-o structură de date indexată după poziția de început și de sfârșit a fiecărei muchii, ori în conformitate cu anumite categorii relevante ale muchiei (cum ar fi eticheta unei muchii inactive sau prima categorie de tip <DEGĂSIT> a unei muchii active). În acest mod căutarea ar putea progresa rapid spre o mulțime restrânsă de potențiale muchii relevante, fără a se lua în considerare întreaga hartă. Aceleași observații sunt valabile relativ la organizarea agendei.

Analizorii sintactici prezentați în acest capitol nu au fost proiectați în mod special după criteriul vitezei. (Numai DCG și BUP compilează regulile gramaticii în clauze Prolog executabile. În rest, regulile sunt memorate ca fapte Prolog și sunt manevrate în momentul execuției). O analiză comparativă a vitezei algoritmilor amintiți conduce la concluzia [19] că, pe măsură ce algoritmul devine mai "bun", procesul de analiză sintactică se realizează din ce în ce mai încet, unul dintre algoritmii cei mai neperformanți, *din acest punct de vedere*, fiind cel al lui Earley. Rezultă în mod clar că analiza sintactică bazată pe constituenți este un proces care nu se realizează ușor și că, atât analizorii sintactici bazați pe backtracking, cât și cei bazați pe utilizarea unei structuri de date de tip hartă, care reduce backtracking-ul, prezintă imperfecțiuni și neajunsuri. Tocmai de aceea proiectarea de analizori sintactici performanți rămâne o problemă de actualitate și reprezintă una dintre sferele de preocupări de cea mai mare importanță în domeniul procesării limbajului natural.

Algoritmul lui Earley este extrem de important din punct de vedere "istoric", deoarece el a demonstrat că analiza sintactică bazată pe reguli PS se poate realiza în *timp polinomial*, i.e. într-un interval de timp proporțional cu n^k, unde n este lungimea șirului de intrare, iar k reprezintă o constantă. În continuare, Earley a demonstrat că întotdeauna $k \leq 3$. Acesta este un rezultat extrem de important, deoarece mulți alți analizori sintactici (cum ar fi cel cu deplasare - reducere) necesită, în cazul cel mai nefavorabil, *timp exponențial* i.e. proporțional cu k^n (care, pentru o valoare ridicată a lui n, este mult mai mare decât n^3). Analizorii sintactici de tipul celui cu deplasare - reducere necesită un interval de timp exponențial, întrucât ei încearcă determinarea separat a fiecărui arbore de derivare. Prin contrast, Algoritmul lui Earley a demonstrat că analiza sintactică poate fi organizată în jurul unor principii și nu prin încercarea consecutivă a tuturor combinațiilor posibile.

Așa cum se observă însă în [19], aceste rezultate celebre au, din diverse motive, o importanță mai mică decât ar putea părea la prima vedere. Astfel, realitatea lingvistică arată că un parser pentru limbajul natural nu se va

confrunta niciodată cu valori mari ale lui n. Chiar și analizorii sintactici care lucrează în timp exponențial pot fi rapizi atunci când n este mic, iar în limbile naturale este puțin probabil să întâlnim propoziții alcătuite din mai mult de 30 de cuvinte. Atunci când intervin, cu adevărat, propoziții sau fraze lungi, acestea pot fi despărțite în propoziții de lungime mai mică, legate între ele prin conjuncții de tipul lui *și*, care pot fi analizate separat. Mai mult, rezultatele referitoare la n^3 și k^n sunt cele obținute pentru cazul cel mai nefavorabil, adică în situația în care parser-ul ia, pe cât posibil, decizii greșite, utilizând regula indicată numai după ce le-a încercat pe toate cele care nu se aplică. Această situație extremă se regăsește relativ rar în practică. Un al treilea motiv invocat [19] este acela că rezultatul privitor la n^3 este valabil numai în cazul gramaticilor în care nodurile nu au argumente. Barton, Berwick și Ristad (1987) au demonstrat că procesul de parsing, în cazul general, este NP - COMPLET - adică aparține unei clase de probleme care necesită timp exponențial - dacă li se permite gramaticilor să aibă *caracteristici* (i.e. să admită argumente ale nodurilor, cum ar fi *singular* sau *plural*, împreună cu regulile de concordanță aferente) și dacă cuvintele pot fi ambigue. Gramaticile care descriu limbajul natural necesită, în mod evident, aceste cerințe.

Progresul în dezvoltarea unor analizori sintactici mai eficienți pare, așadar, să depindă, cu precădere, de următorii doi factori: studiul atent al complexității în cazurile tipice și/sau de frecvență medie (și nu în cazul cel mai nefavorabil), precum și continua descoperire a constrângerilor insuficient explorate cărora trebuie să li se supună gramaticile referitoare la limbajul natural.

Un număr imens de tehnici de parsing diferite a fost dezvoltat de-a lungul timpului, unele dintre acestea referindu-se la limbajul natural, altele la limbajele de programare, iar altele la ambele tipuri de limbaje. Printre lucrările de valoare referitoare la tehnici de parsing dedicate limbajului natural se numără Winograd (1983), Kay (1980), King (1983), Dowty, Karttunen și Zwicky (1985) și Tomita (1991). Pentru implementarea în Prolog a diverși analizori sintactici, vezi și Dahl și Saint-Dizier (1985, 1988), Abramson și Dahl (1989), Gal, Lapalme, Saint-Dizier și Somers (1991).

Acest capitol s-a concentrat, cu precădere, asupra analizorilor sintactici concepuți pentru a lucra cu mulțimi arbitrare de reguli PS. Dar este un fapt bine cunoscut acela că limbile naturale au o structură proprie. Regulile PS care intervin în limbajul natural sunt supuse diverselor constrângeri și, în același timp, limbajul natural presupune o serie întreagă de fenomene care nu pot fi exprimate în exclusivitate de către regulile PS. În multe limbi, spre exemplu, ordinea cuvintelor este parțial sau complet variabilă. Regulile PS nu funcționează corect în astfel de cazuri și pentru astfel de limbi. În consecință, există o întreagă varietate de abordări diferite, care se aplică acestor situații, ca și altora, care apar suficient de frecvent, constituind problemele specifice cu care se confruntă domeniul procesării limbajului natural. Nu întâmplător au fost prezentate cele două abordări total diferite pe care le reprezintă utilizarea *gramaticilor de dependență* și a *gramaticilor contextuale* în modelarea

fenomenelor tipice limbajului natural. Reținem faptul că procesul de analiză sintactică se desfășoară într-o accepție total diferită în cazul gramaticilor de dependență și că, printre cele mai recente preocupări legate de sintaxă, se numără și găsirea unui algoritm de analiză sintactică bazat pe gramatici contextuale. Aceste tipuri de gramatici sunt doar două dintre cele propuse în mod special pentru studiul limbajului natural. Alte formalisme, care sunt mai puternice decât acela al gramaticilor independente de context și care pot manevra dependențele apărute între cuvinte aflate la mare distanță în cadrul propoziției au fost, de asemenea, definite. Un astfel de formalism este cel al **gramaticilor TAG** (de la "tree-adjoining grammars"), dezvoltat de Joshi în 1985.

M. P. Marcus (1978, 1980) inaugurează o importantă tendință în cercetările legate de analizorii sintactici concepuți în mod special pentru limbajul natural. Parser-ul lui M. P. Marcus, la care ne-am referit anterior (§4.4), nu efectuează backtracking și se caracterizează printr-un "lookahead" limitat. Acordă un regim special nodurilor de tip NP și are o performanță, relativ la propozițiile ambigue din punct de vedere structural, surprinzător de apropiată de cea umană (vezi și Kac 1982).

În ceea ce privește gramaticile independente de context, ele rămân suportul principal al algoritmilor de analiză sintactică cunoscuți până în prezent. Pentru a putea surprinde mai bine fenomenele tipice limbajului natural ele sunt adesea extinse prin utilizarea caracteristicilor, mai precis definindu-se constituenții printr-o mulțime de caracteristici. O astfel de extensie permite ca aspecte ale limbajului natural cum ar fi acordul și subcategorizarea (a se vedea §3.4.2) să fie tratate într-un mod intuitiv și, în același timp, cât se poate de concis. Gramaticile independente de context augmentate, la care ne vom referi în capitolul 5 al lucrării de față, constituie un formalism puternic, care poate surprinde multe dintre fenomenele generale ale limbajului natural.

La rândul ei, *eficiența* unui parser bazat pe gramatici independente de context poate fi substanțial îmbunătățită prin aplicarea unei strategii de căutare de tip **best-first**, de către un analizor sintactic care utilizează gramaticile independente de context porbabiliste.

4.5.9. Analizori sintactici de tip best-first

Algoritmii de analiză sintactică de tip best - first urmăresc explorarea cu prioritate a constituenților având cele mai mari probabilități și folosesc, în consecință, gramaticile independente de context probabiliste. Scopul acestor algoritmi este acela de a găsi cât mai rapid analiza cea mai bună, adică cea mai probabilă, a șirului de intrare, fără explorarea integrală a spațiului de căutare. Prin aplicarea unui algoritm de acest tip se urmărește ca posibilitățile având asociate cele mai mici probabilități să nu fie niciodată explorate.

Probabilitatea unui anumit arbore de derivare poate fi determinată folosind un algoritm de analiză sintactică cu hartă standard, în cadrul căruia

probabilitatea fiecărui constituent este calculată utilizând probabilitățile subconstituenților săi și probabilitatea regulii aplicate. Spre exemplu, atunci când analizorul sintactic primește șirul de intrare E de categorie C (de pildă C≡S) și utilizează o regulă *i* cu *n* subconstituenți corespunzând intrărilor E_1, ... , E_n, probabilitatea lui E se va calcula după cum urmează:

$$\mathbf{P(E) = P(Regula\ } i \mid \mathbf{C) * P(E_1) * ... * P(E_n)} \qquad (4.1)$$

Cea mai simplă abordare pentru a calcula probabilitatea aplicării unei reguli, necesară în formula (4.1), este aceea de a număra de câte ori este folosită regula în cadrul unui corpus de propoziții deja analizate. Să considerăm, spre exemplu, o categorie C, în contextul unei gramatici care conține *m* reguli R_1, ... , R_m având membrul stâng identic cu C. Estimarea probabilității de a folosi regula R_j în derivarea lui C se face după cum urmează:

$$\mathbf{P(R_j \mid C) \cong Numără(\#\ utilizări\ R_j) / \sum_{i=1}^{m} (\#\ utilizări\ R_i)} \qquad (4.2)$$

În ceea ce privește categoriile lexicale, ale căror probabilități intervin, de asemenea, în formula (4.1), se recomandă utilizarea "probabilităților înainte" și nu a probabilităților de generare lexicală (a se vedea §3.3.1.2). Aceasta va produce estimații mai bune, întrucât este o metodă prin care se ia, parțial, în considerație contextul șirului de intrare. Este posibilă folosirea unui algoritm de analiză sintactică cu hartă standard căruia i se adaugă un pas la care se calculează probabilitatea fiecărei intrări în momentul adăugării acesteia pe hartă.

Scopul unui parser best - first este, prin urmare, acela de a găsi cât mai repede cea mai probabilă analiză a șirului de intrare reducând, la maximum, explorarea spațiului de căutare. Se observă că toți algoritmii de analiză sintactică cu hartă prezentați până acum pot fi modificați relativ ușor pentru a lua în considerație mai întâi constituenții cei mai probabili. Raportându-ne, spre exemplu, la algoritmii prezentați în §4.5.7.3, modificarea acestora va consta în transformarea agendei într-o **coadă de priorități** - o structură în care elementele cel mai bine cotate sunt întotdeauna plasate primele în coadă. Analizorul sintactic va prelua întotdeauna constituentul de rang cel mai înalt din agendă și îl va adăuga hărții.

Această unică modificare a strategiei de căutare s-a dovedit însă [4] a fi insuficientă. Complicația care intervine se referă la aceea că algoritmii de analiză sintactică cu hartă la care ne-am referit (§4.5.7.3) depind de faptul că parser-ul lucrează în mod sistematic de la stânga la dreapta, procesând complet acei constituenți care intervin pe primele poziții în șirul de intrare înainte de a-i lua în considerație pe următorii. Algoritmul modificat trebuie însă să admită ca, dacă ultimul cuvânt al șirului de intrare are "scorul" cel mai mare, atunci el să fie primul adăugat hărții. Problema care derivă de aici este că nu se pot, pur și

simplu, adăuga hărții muchii active urmând ca extinderea lor să se realizeze la pașii ulteriori ai algoritmului. Este posibil ca acel constituent necesar extinderii unei anumite muchii *active* să se găsească deja pe hartă. De aceea, ori de câte ori se adaugă hărții o muchie activă, trebuie imediat verificată posibilitatea extinderii ei, dată fiind harta curentă. Pentru aceasta, este necesară modificarea Algoritmului 4.5 de extindere a muchiilor în așa fel încât, la pasul 2, să se facă *o verificare a constituenților care există deja pe hartă.* Noul *algoritm* complet *de extindere a muchiilor* este următorul:

Algoritmul 4.9

Pentru a adăuga un *constituent* C de la poziția p_1 la poziția p_2, execută:

1. Inserează C pe hartă de la poziția p_1 la poziția p_2.
2. Pentru orice muchie activă de forma $X \rightarrow X_1 \ldots \bullet C \ldots X_n$ de la poziția p_0 la poziția p_1, adaugă o nouă muchie activă $X \rightarrow X_1 \ldots C\bullet \ldots X_n$ de la poziția p_0 la poziția p_2.

Pentru a adăuga hărții o *muchie activă* $X \rightarrow X_1 \ldots C\bullet C' \ldots X_n$ de la poziția p_0 la poziția p_2, execută:

1. Dacă C este ultimul constituent (i.e. muchia este completă), adaugă un nou constituent de tipul X în agendă.
2. Altfel, dacă există pe hartă, de la poziția p_2 la poziția p_3, un constituent Y de categorie C', atunci adaugă în mod recursiv o muchie activă $X \rightarrow X_1 \ldots CC'\bullet \ldots X_n$ de la poziția p_0 la poziția p_3. (Adăugarea acestei muchii poate conduce la adăugarea altor muchii sau la crearea unor noi constituenți).

Chiar dacă nu ia în considerație fiecare constituent posibil, un parser best - first garantează determinarea interpretării celei mai probabile a șirului de intrare (interpretarea care are probabilitatea cea mai mare). Acest lucru este ușor de demonstrat. Astfel, să presupunem că analizorul sintactic a găsit interpretarea S_1 cu probabilitate p_1. Cea mai importantă proprietate a "scorului probabilist" calculat de formula (4.1) este aceea că probabilitatea unui constituent este întotdeauna mai mică (sau cel mult egală) decât probabilitatea oricăruia dintre subconstituenții săi. Astfel, dacă ar exista o altă interpretare S_2 având scorul p_2, mai mare decât p_1, atunci aceasta ar trebui să fie construită pe baza unor subconstituenți de probabilitate p_2 sau mai mare decât p_2. Ceea ce înseamnă că toți acești subconstituenți ar fi adăugați hărții înaintea lui S_1. Dar aceasta ar determina completarea muchiei care construiește S_2 și, prin urmare, S_2 s-ar afla în agendă. Întrucât S_2 are un scor mai mare decât S_1, ea ar fi prima luată în considerație.

Deşi ideea care stă la baza implementării unui parser best-first este una cât se poate de simplă şi directă, o serie de probleme se ridică atunci când se doreşte aplicarea ei în practică. O primă problemă constă în aceea că, dacă se foloseşte o metodă multiplicativă pentru a combina scorurile constituenţilor, atunci aceste scoruri scad vertiginos, pe măsură ce se acopăr porţiuni tot mai mari din şirul de intrare. În practică, unde se folosesc gramatici de mari dimensiuni, această scădere este atât de drastică încât căutarea care se face seamănă foarte mult cu una de tip breadth-first: mai întâi se construiesc toţi constituenţii de lungime 1, apoi toţi cei de lungime 2 ş.a.m.d. În acest fel, parser-ul se depărtează de ţelul găsirii rapide a soluţiei care are probabilitatea cea mai mare. Această problemă este tratată în mod diferit de diverşi autori, majoritatea sistemelor folosind o altă funcţie pentru calcularea scorului constituenţilor. Spre exemplu, se poate utiliza [4] cel mai mic dintre scorurile subconstituenţilor şi regulii aplicate:

$$\mathbf{Scor(C) = MIN(Scor(C \rightarrow C_1,...,C_n), Scor(C_1), ... , Scor(C_n)).} \qquad (4.3)$$

Se poate, de asemenea, lua în consideraţie media scorurilor tuturor subconstituenţilor.

Varinate ale parser-ului best-first discutat există, majoritatea acestora bazându-se pe modalităţi diferite de calcul al probabilităţilor care intervin. O variantă destul de apreciată a fost aceea a unui **parser best-first dependent de context**, care utilizează o metodă alternativă de calcul a *probabilităţilor regulilor*, metodă ce foloseşte şi înglobează mai multă informaţie lexicală dependentă de context. Ideea care stă la baza acestui tip de parser exploatează observaţia conform căreia primul cuvânt al unui constituent reprezintă, cel mai adesea, centrul sau *capul* în jurul căruia se organizează acesta şi, prin urmare, are un efect extrem de important asupra probabilităţilor regulilor corespunzătoare. Această observaţie sugerează un nou tip de probabilitate a regulilor, relativă la primul cuvânt şi notată P(R | C,W). Această probabilitate se estimează astfel:

$$\mathbf{P(R \mid C, w) = \frac{Numără(\#\ ori\ regula\ R\ folosită\ pentru\ cat.C\ începând\ cu\ w)}{Numără\ (\#\ ori\ cat.C\ începe\ cu\ w)}} \qquad (4.4)$$

Efectul acestei modificări constă în aceea că probabilităţile devin senzitive la cuvintele particulare care intervin în şirul de intrare. Un aspect foarte important al acestei proprietăţi este, spre exemplu, acela că regulile senzitive la context codifică preferinţele verbelor pentru diferite tipuri de subcategorizare (a se vedea §5.2.1).

Întrucât folosirea metodelor statistice în procesarea limbajului natural nu constituie obiectul lucrării de faţă, nu vom intra în alte detalii legate de analizorii sintactici de tip best-first. Reţinem numai ideea că *eficienţa unui parser* poate fi substanţial îmbunătăţită prin utilizarea unei **strategii de căutare de tip best-first**, în care constituenţii de rang cel mai înalt sunt primii adăugaţi pe hartă.

CAPITOLUL 5

CARACTERISTICI ŞI GRAMATICI AUGMENTATE. GRAMATICI DE UNIFICARE

Majoritatea analizorilor sintactici se bazează pe gramatici independente de context. Aşa cum s-a mai remarcat, acestea sunt însă prea restrictive pentru a putea reflecta întreaga bogăţie şi complexitate a limbajului natural. De aceea, vom realiza, în cele ce urmează, o extindere a acestor gramatici prin asocierea unei mulţimi de **caracteristici**.

Sistemele de caracteristici sunt, în prezent, frecvent utilizate pentru a reprezenta informaţia *morfologică, sintactică* şi *semantică*. Vom fi, în mod special, interesaţi de combinarea structurilor de caracteristici (unificare), de relaţiile de incluziune existente între acestea, precum şi de consecinţele folosirii lor în procesul de analiză sintactică.

Limbajul natural se caracterizează prin existenţa a numeroase *restricţii de acord*, precum şi a unor variate *tipuri de acord*. Cel mai frecvent întâlnit este acordul în număr, care poate fi tratat prin intermediul caracteristicilor. Mânuirea unor caracteristici suplimentare (cum ar fi cele care se referă la acordul în persoană sau caz, acordul între subiect şi predicat etc.) ar mări continuu dimensiunile gramaticii. Pentru tratarea unor astfel de fenomene, extrem de importante şi de frecvente, formalismul gramatical a fost extins cu scopul de a permite constituenţilor să admită caracteristici (a se vedea şi capitolul 3, §3.4.2). În acest mod, dimensiunea **gramaticii augmentate** rămâne aceeaşi cu cea a gramaticii originale, în timp ce prima ia în consideraţie şi restricţiile de felul celor amintite.

Pentru a realiza acest tip de extindere, un *constituent* va fi definit ca reprezentând o **structură de caracteristici**, adică o corespondenţă între caracteristici şi valori ale lor, corespondenţă ce defineşte proprietăţile relevante ale constituentului. O **structură de caracteristici** reprezintă, prin urmare, o mulţime de *caracteristici (atribute)* şi de *valori*. Ea conţine *cel mult* o valoare corespunzător fiecărei caracteristici. Spre exemplu, în *notaţia convenţională*, matricea de caracteristici

$$\begin{bmatrix} a : b \\ c : d \end{bmatrix}$$

este o structură de caracteristici care conţine valoarea b a caracteristicii a, valoarea d a caracteristicii c şi nici o valoare a caracteristicii e. În acelaşi timp, structura

$$\begin{bmatrix} a : b \\ a : c \end{bmatrix}$$

nu reprezintă o structură de caracteristici, deoarece nu atribuie o unică valoare lui *a*. Asemenea obiecte nu atribuie mai mult de o valoare unui atribut și nu au obligația să atribuie o valoare fiecărui atribut. Din punct de vedere matematic, matricile de caracteristici reprezintă, prin urmare, *funcții parțiale* de la mulțimea caracteristicilor (atributelor) la mulțimea valorilor.

O caracteristică este, în ultimă instanță, doar o denumire, în timp ce o valoare poate fi ori un simbol atomic (cum ar fi un atom Prolog), ori o altă structură de caracteristici. Astfel, structurile de caracteristici pot fi utilizate pentru a reprezenta constituenți oricât de complecși. Pentru aceasta, s-a admis ca structurile de caracteristici să intervină ele însele ca valori:

$$\begin{bmatrix} a : b \\ c : \begin{bmatrix} d : e \\ f : g \\ h : \begin{bmatrix} i : j \\ k : l \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Astfel de structuri de caracteristici imbricate permit gruparea caracteristicilor în diverse moduri care se vor dovedi extrem de utile.

***Regulile** unei gramatici augmentate* nu mai sunt formulate în termeni de simple categorii ci în termeni de structuri de caracteristici.

Variabilele sunt admise ca *valori ale caracteristicilor*, astfel încât o regulă să se poată aplica unei game largi de situații. Spre exemplu, o regulă de structură a grupului verbal ar putea fi următoarea:

$$\begin{array}{ccccc} VP & \longrightarrow & V & & NP \\ [num: X] & & [num: X] & & [caz: acc] \end{array}$$

Această regulă afirmă că un constituent de tip VP poate fi alcătuit din doi subconstituenți, primul fiind un verb (V) și celălalt un grup nominal (NP), organizat în jurul unui substantiv în cazul acuzativ. Caracteristica de număr a grupului verbal trebuie să fie identică cu cea a verbului.

O altă regulă de structură, de această dată a grupului nominal, ar putea fi:

$$\begin{array}{ccccc} NP & \longrightarrow & ART & & N \\ [num: X] & & [num: X] & & [num: X] \end{array}$$

Această regulă afirmă că un constituent de tip NP poate consta dintr-un subconstituent de tip ART, urmat de unul de tip N, caracteristica de număr a celor trei constituenți fiind identică. Regula nu impune vreo restricție asupra nici uneia dintre celelalte caracteristici pe care NP, ART și N le-ar putea avea.

În ambele reguli, fiecare nod are o **structură de caracteristici** proprie, adică o mulțime de caracteristici și de valori ale lor.

5.1. Analiza sintactică folosind caracteristici

Algoritmii de analiză sintactică descriși anterior pentru gramatici independente de context pot fi, în mod evident, adaptați utilizării gramaticilor independente de context augmentate, adică gramaticilor cu caracteristici. Această adaptare presupune generalizarea algoritmului astfel încât el să realizeze împerecherea dintre reguli și constituenți. Spre exemplu, algoritmii de analiză sintactică cu hartă prezentați în capitolul 4 foloseau toți o operație de extindere a muchiilor active cu un nou constituent. Astfel, un constituent X putea extinde o muchie de forma

$$C \rightarrow C_1 \ldots C_i \bullet X \ldots C_n$$

producând o nouă muchie de forma

$$C \rightarrow C_1 \ldots C_i X \bullet \ldots C_n$$

O operație similară poate fi efectuată în cazul gramaticilor cu caracteristici, dar analizorul sintactic va fi nevoit să instanțieze variabilele muchiei inițiale înainte ca aceasta să poată fi extinsă cu constituentul X.

Algoritmul poate fi specificat în felul următor:

Algoritmul 5.1

Fiind date o muchie A în care constituentul aflat după simbolul • se numește NEXT și un nou constituent X utilizat pentru extinderea muchiei,

1. Se caută o instanțiere a variabilelor astfel încât toate caracteristicile specificate în NEXT să fie găsite în X;
2. Se creează o nouă muchie A' care reprezintă o copie a lui A, cu excepția instanțierilor variabilelor determinate la pasul 1;
3. Se actualizează A' în modul uzual (caracteristic unui parser cu hartă).

Atunci când sunt folosite mulțimi de valori pentru variabile, cum ar fi {3s 3p} pentru caracteristica de număr *num* a verbului și respectiv a grupului verbal, procesul de identificare (împerechere) decurge similar, cu observația că

valoarea dată variabilei va fi aleasă dintre cele listate. Dacă o variabilă este folosită în interiorul unui constituent, atunci una dintre valorile ei posibile trebuie să corespundă cerinței regulii. Dacă atât regula, cât și constituentul conțin variabile, rezultatul este o variabilă având ca domeniu de valori intersecția mulțimilor de valori asociate regulii, respectiv constituentului.

5.2. Sisteme de caracteristici generalizate și gramatici de unificare

Structurile de caracteristici sunt deosebit de utile pentru generalizarea noțiunii de gramatică independentă de context. În fapt, aceste structuri pot fi ele însele generalizate într-un asemenea grad încât existența gramaticii independente de context să nu mai fie necesară. Rolul lor poate fi extins astfel încât să se elimine complet folosirea gramaticilor. În acest fel, întreaga gramatică poate fi specificată ca o mulțime de restricții între diverse structuri de caracteristici. Astfel de sisteme sunt cunoscute sub denumirea de **gramatici de unificare**.

Definiția 5.1

O gramatică de unificare este orice gramatică care:

(i) codifică informația în caracteristici și valori ale acestora;

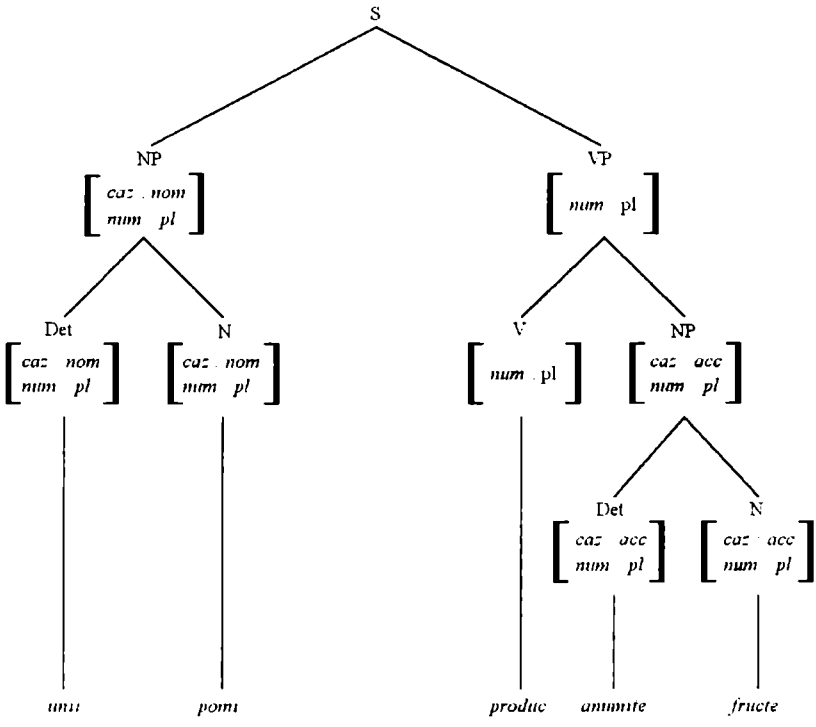
(ii) acordă valori caracteristicilor numai prin unificare (și nu prin vreun alt proces de calcul).

În cele ce urmează, vom reveni asupra procesului de unificare. Pentru moment, ne propunem să privim caracteristicile dintr-un punct de vedere mai teoretic.

5.2.1. Caracteristici

Chiar dacă formalismul gramaticilor de unificare este unul relativ nou, folosirea caracteristicilor reprezintă un procedeu cunoscut și valorificat de mult timp, des întâlnit chiar și în gramatica tradițională. Caracteristicile sunt frecvent utilizate încă de Chomsky (1965), precum și de către alți lingviști ai aceleiași perioade, în timp ce unificarea ca proces își face apariția în lingvistică abia în anii '80.

Deși în introducerea acestui capitol ne-am referit în exclusivitate la restricțiile de acord, trebuie să remarcăm faptul că există și alte modalități prin care se poate ajunge la adnotarea cu caracteristici a unui arbore sintactic ca cel din figura următoare:



Ori de câte ori se aplică o regulă PS cu caracteristici, structurile de caracteristici ale regulii trebuie unificate cu structurile de caracteristici corespunzătoare din arbore. Chiar dacă toate procesele care duc la adnotarea arborelui de derivare cu caracteristici vor fi tratate prin unificare, putem distinge câteva tipuri de asemenea procese diferite. Cele mai importante ar putea fi socotite următoarele:

- *regulile de acord* cer ca o caracteristică a unui nod să se împerecheze cu o caracteristică a altuia; spre exemplu, un substantiv la plural necesită existența unui determinant care are, de asemenea, numărul plural;
- *procesul de atribuire*, prin care gramatica pretinde ca unele caracteristici să aibă anumite valori particulare; spre exemplu, obiectul direct al unui verb să fie în cazul acuzativ;
- *procesul de pătrundere sau de infiltrare* datorită căruia anumite caracteristici ale unui întreg grup sintactic reprezintă copii ale caracteristicilor cuvântului ce constituie centrul (sau capul) grupului; spre exemplu, caracteristica de număr "se infiltrează" de la verb în întreg grupul verbal căruia acesta îi aparține.

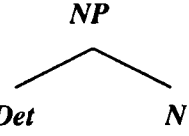
Așa cum se arată în [19], înainte de anii '80 gramaticile generative tratau aceste procese și altele de același tip ca reprezentând procese efectuate de

regulile transformaționale. Cu alte cuvinte, regulile PS generau arborele, după care transformările copiau caracteristici în diverse locuri (noduri din arbore). Viziunea predominantă actualmente este aceea că toate caracteristicile pot fi justificate în urma efectuării unei singure operații, aceea de ***unificare***, care se aplică odată cu fiecare regulă PS.

Astfel, după cum se știe, rolul fiecărei reguli PS este acela de a legitima o anume porțiune din arbore. Spre exemplu, regula

$$NP \longrightarrow Det\ N$$

legitimează structura:



O gramatică generează un arbore dacă și numai dacă fiecare porțiune a arborelui este legitimată de o regulă. Caracteristicile fiecărui nod din arbore trebuie să fie unificate cu cele ale nodurilor corespunzătoare din regulă. Spre exemplu, regula

$$\begin{array}{lll} NP & \longrightarrow\ Det & N \\ [num: X] & [num: X] & [num: X] \end{array}$$

cere ca structura *[num:X]* să fie unificată cu structurile de caracteristici ale lui NP, Det și N. În acest mod se asigură faptul că NP, Det și N au aceeași caracteristică de număr, ținându-se cont de regulile de acord și efectuându-se, totodată, un proces de "infiltrare" relativ la grupul nominal. *Valoarea efectivă* a caracteristicii *num* (singular sau plural) va fi dată de intrările lexicale propriu-zise.

Menționăm faptul că, într-o gramatică de unificare, regulile PS sunt de tipul celei de mai sus, iar intrările lexicale sunt de tipul:

$$\begin{bmatrix} Pronume \\ caz : nom \\ num : sg \end{bmatrix} \longrightarrow \textbf{el}$$

sau

$$\begin{bmatrix} N \\ num : sg \end{bmatrix} \longrightarrow \textbf{câine}$$

Printre cele mai importante sunt intrările lexicale corespunzătoare verbelor, care trebuie să țină cont de *subcategorizarea*[1] acestora, prin intermediul unei

[1] În gramatica generativă, *regula de subcategorizare* este un tip de regulă care, alături de regulile de structură, constituie componentul de bază al gramaticii, aplicându-se categoriilor lexicale numai după ce nici una dintre regulile de structură nu se mai poate aplica. Realizează rescrierea categoriei lexicale V în funcție de trăsături contextuale, descrise atât în termeni sintactici categoriali, cât și în termenii trăsăturilor semantice ale

caracteristici pe care o vom numi *subcat*. Această caracteristică va distinge, în cazul nostru, între verbele care nu admit obiect direct (valoare 1) și cele care admit obiect direct (valoare 2):

$$V \longrightarrow \textbf{latră}$$
$$\begin{bmatrix} num : sg \\ subcat : 1 \end{bmatrix}$$

$$V \longrightarrow \textbf{sperie}$$
$$\begin{bmatrix} num : sg \\ subcat : 2 \end{bmatrix}$$

În aceste condiții, iată un alt exemplu de regulă PS (pentru verbe care admit obiecte directe):

$$VP \longrightarrow V \quad NP$$
$$[num: X] \quad \begin{bmatrix} subcat : 2 \\ num : X \end{bmatrix} \quad [caz: acc]$$

S-a arătat că regulile de structură bazate pe unificare lucrează la fel de bine atunci când sunt aplicate bottom-up, top-down sau în orice altă ordine.

5.2.2. Unificare

Caracteristicile dintr-o structură de caracteristici sunt identificate prin intermediul numelui, nu al poziției. Prin urmare,

$$\begin{bmatrix} pers : 2 \\ num : plural \end{bmatrix} \text{ și } \begin{bmatrix} num : plural \\ pers : 2 \end{bmatrix}$$

reprezintă aceeași structură de caracteristici.

Două structuri pot fi **unificate** dacă ele se pot combina fără a genera contradicții. Spre exemplu,

$$\begin{bmatrix} x : y \\ z : t \end{bmatrix} \text{ și } \begin{bmatrix} x : y \\ u : v \end{bmatrix}$$

se unifică producând structura

$$\begin{bmatrix} x : y \\ z : t \\ u : v \end{bmatrix}.$$

---

categoriilor vecine. În contextul de mai sus, *subcategorizarea verbelor* înseamnă clasificarea acestora după cum acceptă sau nu un complement direct, după cum acceptă un complement prepozițional sau acceptă două complemente etc. (vezi și [9], p. 481, s.v. *subcategorizare*).

Acest proces seamănă foarte bine cu procesul de unificare din Prolog. Diferența principală este că acele caracteristici care nu sunt instanțiate sunt pur și simplu ignorate. Prin urmare, nu este necesară folosirea unei "variabile anonime". Variabilele care au nume lucrează în același fel ca în Prolog. Spre exemplu,

$$\begin{bmatrix} w : y \\ z : t \end{bmatrix} \quad \text{și} \quad \begin{bmatrix} w : X \\ u : X \end{bmatrix}$$

prin unificare produc

$$\begin{bmatrix} w : y \\ z : t \\ u : y \end{bmatrix}.$$

Cea de-a doua structură de caracteristici nu acordă valori lui *a* și *e*, dar impune cerința ca valorile lui *a* și *e* să coincidă. Aceasta seamănă cu ceea ce se întâmplă atunci când, în Prolog, unificăm f(b,d,_) cu f(X,_,X).

Unificarea structurilor de caracteristici poate, de asemenea, eșua. Într-un asemenea caz, regula gramaticii care pretinde unificarea eșuează în egală măsură, în sensul că nu se poate aplica. Spre exemplu, structurile de caracteristici

$$\begin{bmatrix} x : y \\ z : t \end{bmatrix} \quad \text{și} \quad \begin{bmatrix} x : t \\ u : v \end{bmatrix}$$

nu se pot unifica. (Unificarea eșuează deoarece x nu poate avea simultan valorile y și t într-o aceeași structură de caracteristici).

O descriere completă a **unificării structurilor de caracteristici**, preluată de noi din [19], ar putea fi următoarea:

- Pentru a unifica două structuri de caracteristici se unifică valorile tuturor caracteristicilor.
- Dacă o caracteristică intervine în una dintre structuri, dar nu și în cealaltă, ea va fi, pur și simplu, inclusă în structura rezultată în urma unificării.
- Dacă o caracteristică intervine în ambele structuri, se unifică valorile ei astfel:
  - pentru a unifica valori care reprezintă simboluri atomice, se verifică dacă acestea sunt egale; în cazul în care nu sunt egale, unificarea eșuează;
  - pentru a unifica o variabilă cu orice altceva, se acordă variabilei o valoare egală cu cea corespunzător căreia se face unificarea;
  - pentru a unifica valori care sunt structuri de caracteristici, se aplică acest proces în mod recursiv.

Iată câteva exemple:

$$\begin{bmatrix} x : y \\ z : X \end{bmatrix} \text{ și } \begin{bmatrix} z : \begin{bmatrix} t : u \\ v : p \end{bmatrix} \\ q : r \end{bmatrix} \text{ se unifică în } \begin{bmatrix} x : y \\ z : \begin{bmatrix} t : u \\ v : p \end{bmatrix} \\ q : r \end{bmatrix}$$

$$\begin{bmatrix} x : g \\ z : X \end{bmatrix} \text{ și } \begin{bmatrix} x : Y \\ y : h \\ z : \begin{bmatrix} t : Y \\ u : v \end{bmatrix} \end{bmatrix} \text{ se unifică în } \begin{bmatrix} x : g \\ y : h \\ z : \begin{bmatrix} t : g \\ u : v \end{bmatrix} \end{bmatrix}$$

Așa cum se remarcă în [19], putem admite ca valorile să fie termeni Prolog de orice tip, fără a schimba puterea computațională a formalismului. Aceasta deoarece orice termen Prolog poate fi tradus într-o structură de caracteristici. Spre exemplu, `f(a,b)` poate deveni

$$\begin{bmatrix} \textit{functor} : f \\ \textit{arg1} : a \\ \textit{arg2} : b \end{bmatrix}$$

iar lista `[a,b,c]` poate fi redată ca:

$$\begin{bmatrix} \textit{cap} : a \\ \textit{coada} : \begin{bmatrix} \textit{cap} : b \\ \textit{coada} : \begin{bmatrix} \textit{cap} : c \\ \textit{coada} : \textit{nil} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

De aceea se utilizează, ori de câte ori este convenabil, termenii Prolog ca un înlocuitor al structurilor de caracteristici. Unificarea structurilor de caracteristici se reduce, în acest caz, la unificarea uzuală a termenilor în Prolog.

Unul dintre cele mai mari avantaje ale unificării, care a fost deja exploatat în Prolog, este faptul că acest proces este *independent de ordine*. Dacă se unifică o mulțime de structuri de caracteristici, rezultatul obținut va fi același, indiferent de ordinea în care se face unificarea. Aceasta înseamnă că este adesea posibil să se folosească o singură gramatică de unificare pentru diverși algoritmi de analiză sintactică diferiți. Așa cum se arată în [19], nu are importanță care dintre unificări se realizează prima, atâta timp cât se efectuează toate operațiile de unificare cerute. Aceasta conferă o mare libertate programatorului care concepe un analizor sintactic.

Independența față de ordine a unificării elimină, de asemenea, una dintre întrebările mai mult sau mai puțin lipsite de conținut ale gramaticii transformaționale [19]: în cazul acordului dintre subiect și predicat, spre exemplu, caracteristica de număr este copiată de la subiect la verb sau invers?

Evident, acest lucru nu trebuie să prezinte importanță, mai ales în cazul efectuării unor operații cum ar fi conceperea unui analizor sintactic. Totuși, o gramatică transformațională trebuie să facă în așa fel încât procesul de copiere să se desfășoare într-o direcție anume. O gramatică de unificare nu se supune unei asemenea cerințe. Ea nu face decât să stipuleze faptul că sunt egale caracteristicile de număr ale subiectului și ale verbului.

Operația de unificare poate fi definită și în termenii **relații de extindere** dintre două structuri de caracteristici, relație care reprezintă conceptul de bază al unei gramatici de unificare [4]:

Definiția 5.2

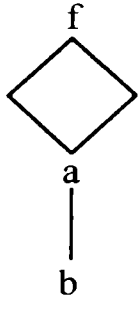
O structură de caracteristici F1 **extinde** (sau "este mai precisă decât") o structură de caracteristici F2 dacă orice valoare de caracteristică din F1 este specificată în F2.

Definiția 5.3

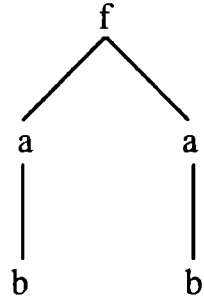
Două structuri de caracteristici **unifică** dacă există o structură de caracteristici care reprezintă o extindere a ambelor.

Structura de caracteristici minimală cu această proprietate se numește **cel mai general unificator**.

Anumite **proprietăți ale caracteristicilor** pot fi puse în evidență cu referire directă la procesul de unificare din Prolog. Spre exemplu, dacă două caracteristici au aceeași valoare, valorile lor desemnează *același obiect*, nu două obiecte diferite, însă asemănătoare. Un exemplu din Prolog care ilustrează această proprietate este acela în care se unifică `f(X,X)` cu `f(a(b),_)`. Rezultatul unificării este `f(a(b),a(b))`. Important este faptul că rezultatul nu conține doi `a(b)` diferiți; el conține doi pointeri către *același* `a(b)`. Structura arborescentă corespunzătoare este, prin urmare, de tipul



și nu de tipul



Sructurile de caracteristici lucrează în același mod. Dacă se unifică

$$\begin{bmatrix} p : X \\ q : X \end{bmatrix} \text{ și } [p : [a : b]]$$

se obține structura

$$\begin{bmatrix} p : [a : b] \\ q : [a : b] \end{bmatrix} \tag{5.1}$$

în care p și q au *aceeași* valoare, nu două valori care arată identic. Pentru a exprima mai clar faptul că cele două apariții ale lui $[a: b]$ reprezintă o unică structură, se poate atribui acesteia un număr de identificare, cu ajutorul căruia structura (5.1) se exprimă astfel:

$$\begin{bmatrix} p : (1)[a : b] \\ q : (1) \end{bmatrix} \tag{5.2}$$

Aici, primul număr (1) reprezintă o etichetă a structurii $[a: b]$. Atunci când se utilizează (1) a doua oară se indică faptul că aceeași structură reprezintă, în același timp, valoarea unei alte caracteristici. Iată un *exemplu*, preluat din [19], în care această notație se dovedește foarte utilă: vom presupune că se grupează anumite caracteristici care țin de acord (așa-numite "agreement features") în cadrul unei structuri numite ***agr***. In acest cadru dorim să construim o regulă $S \rightarrow NP\ VP$ astfel încât:

- categoriile NP și VP să aibă aceleași caracteristici de tip ***agr*** (persoană, număr etc.);
- NP să aibă caracteristica de tip ***agr*** *caz* : *nom* (i.e. cazul nominativ).

Cu alte cuvinte, se dorește combinarea următoarelor două reguli

$$\begin{matrix} S & \longrightarrow & NP & VP \\ & & [agr : X] & [agr : X] \end{matrix}$$

$$S \longrightarrow \underset{[agr:[caz:nom]]}{NP} \quad VP$$

în una singură. Problema care se ridică este aceea că, dacă i se acordă lui NP o caracteristică notată *agr* : *X*, ca în cadrul primei reguli, nu există nici o modalitate de a ne referi la o caracteristică anume din cadrul structurii ***agr***, așa cum este necesar în cea de-a doua regulă. Problema poate fi rezolvată utilizând notația anterioară în felul următor:

$$S \longrightarrow \underset{[agr:(1)\ [caz:nom]]}{NP} \quad \underset{[agr:(1)]}{VP} \tag{5.3}$$

Regula (5.3) poate fi interpretată astfel: în cadrul regulii de structură *S* → *NP VP* caracteristicile ***agr*** ale categoriilor NP și VP trebuie unificate, ele devenind o unică structură notată (1). În plus, (1) trebuie să unifice cu [*caz:nom*]. Această unificare produce efectul secundar de a atribui *caz:nom* și lui VP, ceea ce reprezintă o operație inofensivă, întrucât verbele nu pot avea caracteristica de caz. (În același timp, sugerează un motiv pentru a nu include caracteristica de caz în grupul ***agr***. *Gruparea caracteristicilor în diverse structuri* reprezintă, prin urmare, o operație delicată, care necesită întreaga atenție a programatorului.)

Vom mai remarca, în final, faptul că gramaticile de unificare, așa cum ne-am referit până în prezent la ele, pot fi ușor convertite în gramatici independente de context. (Aceasta va conduce la adaptarea algoritmilor de analiză sintactică clasici, concepuți referitor la gramaticile independente de context, în cazul utilizării gramaticilor de unificare). Într-adevăr, dacă se înlătură toate caracteristicile regulilor gramaticilor de unificare la care ne-am referit până acum, se obțin reguli PS independente de context. Chiar și etichetele nodurilor pot fi tratate ca reprezentând caracteristici. Astfel, în loc de

$$\underset{\begin{bmatrix} caz:nom \\ num:pl \end{bmatrix}}{NP}$$

putem scrie

$$\begin{bmatrix} cat:np \\ caz:nom \\ num:pl \end{bmatrix}$$

Se poate afirma că fiecare gramatică de unificare are o "coloană vertebrală", independentă de context. Acest lucru este adevărat [19] dacă fiecare nod al fiecărei reguli are o caracteristică de tip ***cat*** a cărei valoare *nu conține variabile*. (În general, valoarea lui ***cat*** este un simbol atomic, cum ar fi *np*, dar Jackendoff (1977) și Gazdar (1985) studiază ce se poate realiza atunci când această valoare reprezintă o structură de caracteristici).

Desigur, atunci când categoriile constituenților nu sunt specificate, adaptarea algoritmilor de analiză sintactică clasici în cazul gramaticilor de unificare nu mai este atât de evidentă. Se poate însă arăta că [4], atâta timp cât există o submulțime finită de caracteristici, astfel încât cel puțin una dintre caracteristici este specificată în cadrul fiecărui constituent, *gramatica de unificare poate fi convertită într-o gramatică independentă de context.* Această mulțime finită de caracteristici este adesea numită **coloana vertebrală independentă de context a gramaticii**.

Așa cum am mai menționat, spre deosebire de caracteristici ca atare, gramaticile de unificare reprezintă o extindere (a gramaticilor independente de context) relativ nouă. Mai mult, gramaticile de unificare în sine nu reprezintă o teorie gramaticală. Ele sunt, mai degrabă, un *cadru*, o structură, pe care se bazează unele teorii gramaticale, în special acelea referitoare la gramaticile funcționale. Formalismul gramaticilor de unificare dezvoltat în cadrul acestui capitol este adaptat pentru a face posibilă implementarea în Prolog.

5.2.3. Notația PATR

Pentru a desemna structurile de caracteristici s-a folosit, până în prezent, notația convențională, adică aceea care utilizează matricile de caracteristici. În cele ce urmează, ne vom familiariza cu o altă notație consacrată.

Definiția 5.4

Un **drum** reprezintă o descriere a locului în care se găsește o anumită informație în cadrul unei structuri de caracteristici imbricate.

Spre exemplu, în cadrul structurii

$$\begin{bmatrix} p : \begin{bmatrix} a : b \\ c : \begin{bmatrix} d : e \\ f : g \end{bmatrix} \\ h : i \end{bmatrix} \\ q : r \end{bmatrix}$$

drumul *p:c:d* conduce la valoarea *e*. Un drum poate fi privit ca o funcție parțială care, fiind dată o structură de caracteristici, întoarce o valoare sau nu.

Aceasta sugerează o modalitate diferită de a scrie regulile gramaticilor de unificare. În loc de a folosi variabile în cadrul structurilor de caracteristici, putem specifica faptul că valorile anumitor drumuri trebuie să fie egale. M.A. Covington numește aceasta "stil ecuațional" [19]. Spre exemplu, în loc de

$$S \longrightarrow \quad NP \qquad VP$$
$$\begin{bmatrix} pers : P \\ num : X \\ caz : nom \end{bmatrix} \begin{bmatrix} pers : P \\ num : X \end{bmatrix}$$

putem scrie (într-o notație pe care am mai folosit-o):

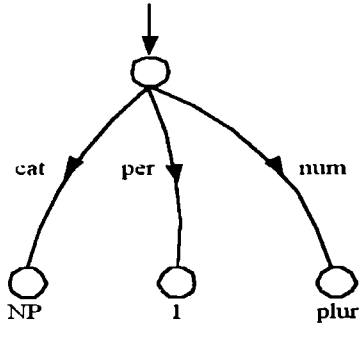
$S \longrightarrow NP\ VP$
<NP pers> = <VP pers>
<NP num> = <VP num>
<NP caz> = nom

Aceasta este notația utilizată de programul PATR-II (Shieber 1985). Interpretarea ecuațiilor cu caracteristici este esențială în construirea constituenților.

Covington dezvoltă o extensie a limbajului Prolog numită GULP, în care pot fi utilizate atât stilul convențional, cât și cel ecuațional. Atât GULP, cât și PATR-II, permit folosirea drumurilor. Notația GULP a fost introdusă în cadrul unui program cunoscut sub denumirea de "Graph Unification Logic Programming" (Covington 1989). O prezentare a extensiei GULP a Prologului pentru gramatici de unificare poate fi găsită în [19]. Pentru o implementare a PATR în Prolog, a se vedea §5.2.7.

5.2.4. Reprezentarea structurilor de caracteristici sub forma unor grafuri orientate aciclice

Din punct de vedere matematic, matricile de caracteristici pot fi privite ca reprezentând niște funcții parțiale de la mulțimea atributelor la mulțimea valorilor. O modalitate alternativă de a privi caracteristicile, din punct de vedere matematic, utilizează **grafurile orientate aciclice** (așa-numitele **DAG**-uri, de la "directed acyclic graph"), grafuri în care muchiile sunt etichetate cu atribute, iar nodurile terminale sunt etichetate cu valori. Un exemplu de DAG în care nodurile terminale sunt etichetate exclusiv cu valori atomice este următorul:

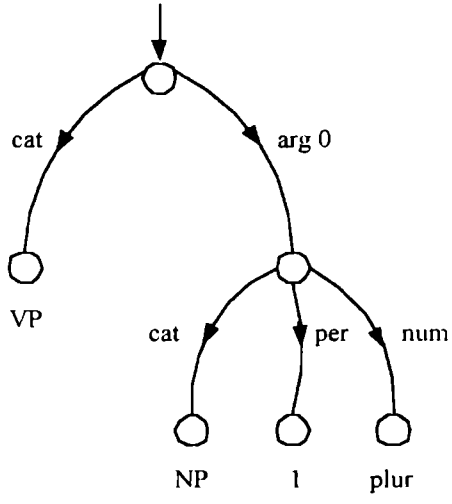


Aşa cum s-a mai arătat, este permis ca structurile de caracteristici să intervină ele însele ca valori.

Definiţia 5.5

O caracteristică are ca valoare o *categorie* dacă şi numai dacă ea nu are o valoare atomică.

În exemplul următor, caracteristica *arg0* are ca valoare o categorie, în timp ce *cat, per* şi *num* au toate valori atomice. O notaţie de tipul *<arg0 num> = plur* specifică atât un drum în DAG-ul din figură (*<arg0 num>*), cât şi eticheta vârfului de la capătul acestui drum (*plur*):



Existenţa caracteristicilor ale căror valori sunt categorii, în sensul Definiţiei 5.4, permite ca foarte multe generalizări gramaticale semnificative să fie exprimate într-un mod destul de direct. În mod standard sunt invocate, spre exemplu, principii de împerechere prin egalare a caracteristicilor pentru a asigura identitatea anumitor caracteristici corespunzătoare unor vârfuri adiacente. Astfel de principii reprezintă ecuaţii care afirmă că anumite caracteristici corespunzătoare unui vârf trebuie să fie identice cu cele care apar în altul. Multe teorii gramaticale actuale se bazează pe principiul ecuaţional, conform căruia membrii ecuaţiei sunt furnizaţi de o anumită clasă de specificaţii de caracteristici, aşa cum apar ele la "categoria mamă" şi la "categoria fiică", aceea care manifestă trăsăturile morfologice asociate cu respectivele caracteristici. Această "categorie fiică" este cunoscută ca reprezentând **capul** grupului sintactic. Majoritatea grupurilor sintactice au un singur cap. Astfel, spre exemplu, un grup verbal moşteneşte timpul verbului său, întrucât acesta din urmă reprezintă capul grupului sintactic VP. Efectele

acestui principiu pot fi stipulate în cadrul fiecărei reguli dacă se presupune că acele caracteristici care sunt relevante se găsesc toate pe o unică ramură a DAG-ului. Vom numi eticheta acestei ramuri ***cap***. Atunci, în notația PATR, se poate scrie o regulă tipică referitoare la grupul verbal astfel:

Regula

$$VP \longrightarrow V\ NP:$$
$$< \text{V cap} > = < \text{VP cap} > .$$

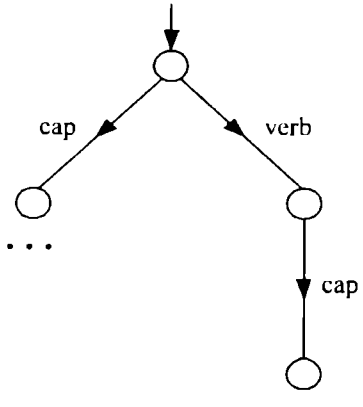
Această regulă impune ca valoarea caracteristicii *cap* a verbului (V) și aceea a grupului sintactic părinte (VP) să fie identice. Dacă, spre exemplu, caracteristica *cap* a verbului conține o pereche (*atribut, valoare*) care nu se regăsește întocmai în structura caracteristicii *cap* a grupului verbal, atunci regula nu este aplicabilă. Valorile lui *cap* la care ne referim nu vor fi, de regulă, atomice, ci vor reprezenta, la rândul lor, DAG-uri.

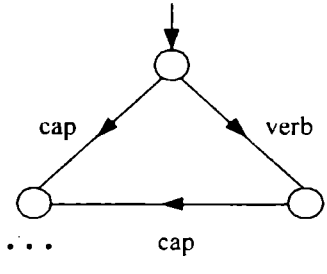
Până în acest moment, grafurile cu caracteristici la care ne-am referit au fost întotdeauna arbori. Totuși, atunci când se operează cu caracteristici ale căror valori sunt categorii, este convenabil să se pună în valoare flexibilitatea oferită de DAG-urile generale. Să considerăm, spre exemplu, următoarea regulă extinsă referitoare la grupul verbal:

Regula

$$VP \longrightarrow V\ NP:$$
$$< \text{V cap} > \quad = < \text{VP cap} >$$
$$< \text{VP verb} > = < \text{V} > .$$

Există două moduri în care categoria gramaticală VP descrisă de o astfel de regulă poate fi reprezentată sub forma unui graf:



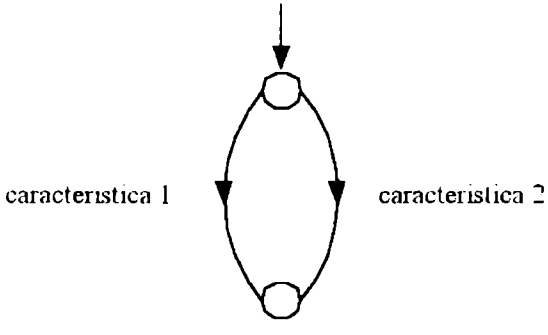


Prima structură presupune că există *două copii* ale substructurii care apare de două ori, în timp ce a doua presupune folosirea în comun a acesteia, deci se reduce la a împărți substructura comună. Alegerea uneia sau a alteia dintre aceste reprezentări se face în funcție de semnificația pe care dorim să o atribuim simbolului " = ". Interpretarea tipică este aceea în care apare o substructură comună ("sharing interpretation"). În acest caz, simbolul "=" înseamnă că cele două entități reprezintă o *aceeași* structură, nu două structuri diferite, care întâmplător au aceleași valori ale tuturor caracteristicilor. Devine, prin urmare, imposibil să se opereze adăugări numai în una dintre structuri. Abordarea de tip "sharing" (partajare) este mult mai atractivă din punct de vedere computațional, întrucât structurile de date utilizate vor fi mai mici atunci când nu trebuie menținută decât o singură copie. În concluzie, simbolul " = " va fi interpretat ca specificând faptul că două categorii sau părți ale unor categorii trebuie să partajeze, iar nu să aibă aceeași valoare. Din punct de vedere computațional, punctul esențial este acela că putem reprezenta două substructuri care partajează chiar înainte de a ști totul despre acestea. În astfel de condiții devine trivială problema actualizării uneia dintre ele, atunci când se operează o adăugare în cea de-a doua.

Pentru a opera cu DAG-uri care prezintă un fenomen de tip "sharing" (partajare), trebuie să se permită ca vârfurile terminale să rămână neetichetate. Spre exemplu, DAG-ul minimal care satisface

< caracteristica 1 > = < caracteristica 2 >

nu are nici un vârf etichetat, fiind cel din figura următoare:



Din păcate, odată cu permiterea existenței unor vârfuri terminale neetichetate, devine posibilă existența mai multor grafuri diferite care transmit exact aceeași informație.

Printre proprietățile mai importante ale DAG-urilor, rezultate din modul lor de construcție, se numără:

- deși un DAG poate exprima faptul că o categorie nu conține nici o informație despre o caracteristică dată, un astfel de graf nu poate exprima faptul că o categorie nu poate avea o valoare pentru o caracteristică dată; prin urmare, un DAG care specifică prezența unei

caracteristici fără a oferi nici o informație suplimentară referitoare la valoarea acesteia nu poate intra în conflict cu nici un alt DAG privitor la acea caracteristică și, în consecință, nu afirmă nimic relativ la acea caracteristică;

- a afirma că un DAG are o valoare pentru o caracteristică dată fără a furniza nici o informație despre acea valoare, ori fără a spune că valoarea respectivă partajează cu o altă valoare, nu transmite nici o informație reală.

5.2.5. Structuri de caracteristici în Prolog

Un DAG pentru structuri de caracteristici va fi reprezentat în Prolog, conform [29], ca o listă de perechi de tipul (*caracteristică, valoare*), listă a cărei coadă este, de obicei, lăsată deschisă ca variabilă. Această variabilă va reprezenta așa-numitul **rest** al DAG-ului. Atunci când va fi necesară extinderea unui DAG cu scopul de a include valori pentru mai multe caracteristici, aceasta se va realiza prin instanțierea în continuare a acestei variabile, pe care o vom numi *variabilă rest*. Astfel:

```
[persoana:3,numar:sing|_]
```

reprezintă o structură cu caracteristicile **persoana** și **numar**. Valorile acestor caracteristici sunt **3** și respectiv **sing**. Să mai notăm faptul că " : " este un operator ale cărui proprietăți sintactice pot fi definite după cum urmează:

```
?- op(500,xfy,:).
```

Acest operator este folosit numai ca modalitate de a cupla o caracteristică și o valoare într-o unică structură, care poate interveni ca element al unei liste. Atunci când o valoare reprezintă ea însăși o colecție complexă de caracteristici, se poate folosi o listă imbricată de același tip. Spre exemplu, în lista

```
[cat:v,arg0:[cat:np,caz:nom,numar:sing|_]|_]
```

valoarea caracteristicii **arg0** este un obiect complex. El include valori cel puțin pentru caracteristicile **cat**, **caz** și **numar**. Ambele exemple conduc la niște DAG-uri sub formă de arbore. Pentru a putea reprezenta DAG-uri generale, este necesară găsirea unei modalități de a reprezenta fenomenul de partajare dintre diferitele părți ale unei categorii. În acest sens, pentru a indica faptul că două caracteristici trebuie să primească aceeași valoare, se va folosi o variabilă Prolog, după cum urmează: un DAG în care *capul* și *capul verbului* utilizează în comun o valoare este

```
[cat:vp,cap:X,verb:[cap:X|_]|_]
```

în timp ce

```
[cat:vp,cap:X,verb:[cap:Y|_]|_]
```

reprezintă un DAG în care acestea nu împart între ele aceeaşi valoare. Din această cauză, a doua structură menţionată transmite, în esenţă, aceeaşi informaţie ca cea furnizată de structura

```
[cat:vp|_]
```

Sunt relativ frecvente situaţiile în care este necesară desemnarea unui DAG în care două "subDAG-uri" (pe care le vom numi *DAG-uri secundare*) partajează o valoare, fiind deja cunoscute anumite informaţii despre aceasta. În astfel de cazuri trebuie reprezentată în ambele locuri întreaga informaţie cunoscută despre valoarea respectivă şi, în plus, trebuie indicat faptul că *restul* descrierii valorii este, de asemenea, partajat. Spre exemplu,

```
[cat:vp,cap:[numar:sing|X],
      verb:[cap:[numar:sing|X]|_]|_]
```

denotă o structură în care *capul* împarte valoarea cu *capul verbului*, dar în care se cunoaşte, în plus, informaţia că *numărul capului* este **sing**.

Ca descriere, un DAG este întotdeauna *deschis într-o parte*, ceea ce înseamnă că el poate fi extins în mod consistent prin adăugarea de informaţii referitoare la noi caracteristici, care nu fuseseră menţionate în structura lui iniţială. În Prolog acest lucru poate fi realizat prin instanţierea variabilei care reprezintă *restul* DAG-ului la o nouă structură conţinând noua informaţie referitoare la caracteristici şi care, la rândul ei, se încheie cu o *variabilă rest*. Spre exemplu:

```
?- D1=[cat:vp|R1],
   R1=[cap:[numar:sing|X]|R2],
   R2=[numar:sing|R3].

   D1=[cat:vp,cap:[numar:sing|X],numar:sing|R3]
```

(Răspunsul complet al Prologului, la interogarea de mai sus, este

```
D1=[cat:vp,cap:[numar:sing|X],numar:sing|R3],
R1=[cap:[numar:sing|X],numar:sing|R3],
R2=[numar:sing|R3]
```

aceasta reprezentând o soluţie unică).

Atunci când se doreşte codificarea faptului că două DAG-uri complexe partajează, trebuie să se verifice că, dacă primul DAG este extins, atunci şi cel de-al doilea va fi extins în acelaşi mod. Acest lucru poate fi realizat făcând ca atât *variabilele rest*, cât şi toate valorile caracteristicilor deja existente să împartă între ele.

Vom mai nota faptul că este posibilă construcția unor liste Prolog similare, care nu corespund însă unui DAG în accepția prezentă. Iată un exemplu de o asemenea structură greșită:

```
[cat:vp,cap:[numar:sing|X],
     verb:[cap:[numar:plur|X]|_]|_]
```

Această structură presupune existența a două DAG-uri secundare care împart *restul*, dar nu împart caracteristicile de **numar**, lucru care nu este posibil în structura unui DAG.

Pe de altă parte, un DAG "legal" îl constituie, spre exemplu, următoarea structură

```
[cat:vp,cap:[persoana:3,numar:sing|X],
     verb:[cap:[numar:sing,persoana:3|X]|_]|_]
```

care ilustrează faptul că ordinea perechilor (*caracteristică, valoare*) nu are nici o importanță în reprezentarea unui DAG. Aceasta, împreună cu deschiderea într-o parte a DAG-urilor, constituie cele două diferențe fundamentale dintre structurile de caracteristici discutate aici și cele utilizate de către gramaticile DC cu caracteristici din capitolul 3 al lucrării de față.

Pentru a construi structurile specificate de către gramatică în timpul procesului de analiză sintactică este vital să putem calcula unificarea a două categorii (în sensul Definiției 5.5), categorii reprezentate sub formă de DAG-uri.

5.2.6. Unificarea în Prolog

Așa cum se remarcă în [29], foarte mulți algoritmi de unificare folosiți în practică conduc la o aceeași problemă: construirea unor structuri cu cicluri. Pentru a preveni acest lucru, algoritmul de unificare ar trebui să conțină un așa-numit *test de ocurență*, adică un test care să se asigure că o structură nu este unificată cu o altă structură care conține structura inițială. Un astfel de test induce un cost extrem de mare în ceea ce privește timpul de execuție, motiv pentru care este adesea omis de către diversele implementări ale unificării.

Se pot imagina algoritmi de unificare care construiesc graful rezultat în urma unificării de la început, chiar dacă acesta diferă foarte puțin de unul sau celălalt dintre grafurile inițiale. În practică însă se utilizează, de regulă, un așa-numit *algoritm distructiv*, adică un algoritm care presupune efectuarea de modificări asupra structurii de date reprezentând unul dintre DAG-urile inițiale, astfel încât graful cu modificările operate să reprezinte rezultatul unificării. *Atunci când nu se impune păstrarea structurii de date inițiale*, această abordare

este cea mai eficientă. Implementarea ei în Prolog [29] este facilitată de reprezentarea sub formă de listă a DAG-urilor prezentată în §5.2.5. În cadrul *unificării distructive*, informația care se găsește în unul dintre grafuri, dar nu se regăsește în celălalt, este copiată direct în cel de-al doilea graf (și, uneori, viceversa). Problema utilizării în comun ("sharing") este tratată în același mod ca în §5.2.5. Vom mai nota faptul că, deși Prologul se bazează pe unificare, este vorba de unificarea termenilor, nu a DAG-urilor, distincție care trebuie făcută și datorită căreia unificarea DAG-urilor în Prolog este departe de a reprezenta o operație elementară, chiar dacă ea se efectuează într-un mod mult mai simplu decât în Lisp, spre exemplu.

Întrucât folosim variabile Prolog pentru a reprezenta partea necunoscută a unui DAG, este firesc ca, în cele ce urmează, să implementăm unificarea în manieră *distructivă*. Unificarea DAG-urilor va fi distructivă în același mod în care este distructivă și unificarea Prolog. Prin urmare, o unificare realizată cu succes va avea ca rezultat instanțierea celor două DAG-uri implicate în unificare, astfel încât întreaga informație existentă să fie partajată de către acestea. Esența algoritmului de unificare implementat constă în a parcurge perechile de tipul (*caracteristică, valoare*) existente în primul DAG, corespunzător fiecărei astfel de perechi realizându-se o extindere a celui de-al doilea DAG, astfel încât acesta să conțină aceeași pereche. În final, se instanțiază *variabila rest* de la sfârșitul primului DAG la informația existentă în cel de-al doilea DAG, informație care nu a fost încă întâlnită. Spre exemplu, unificarea realizată de predicatul **unifica** în urma interogării

```
?- unifica([cat:vp|X],[numar:sing|Y]).
```

începe prin extinderea celui de-al doilea DAG pentru ca acesta să conțină informația codificată în mod explicit în primul:

`(Y=[cat:vp|Z])`	Primul DAG	Al doilea DAG
	`[cat:vp|X]`	`[numar:sing,cat:vp|Z]`

Procesul de unificare continuă cu instanțierea lui **X** la acele părți ale celui de-al doilea DAG care nu au fost încă parcurse și copiate:

`(X=[numar:sing|Z])`	Primul DAG	Al doilea DAG
	`[cat:vp,numar:sing|Z]`	`[numar:sing,cat:vp|Z]`

Ca urmare, se obțin două DAG-uri care partajează informația în toate privințele, grafuri obținute prin simpla instanțiere a DAG-urilor inițiale.

Plecând de la acest exemplu, vom defini unificarea, conform [29], în mod *recursiv*. Operația de bază pe care o efectuăm constă în parcurgerea (recursivă) a listei furnizate de primul DAG, cu extinderea celui de-al doilea DAG, astfel încât acesta să includă informația relevantă. Această operație este urmată de instanțierea variabilei rest a primului DAG la o parte a celui de-al doilea DAG. Vom face în așa fel încât, atunci când se va ajunge la variabila de la sfârșitul

primului DAG, cel de-al doilea argument al predicatului **unifica** să fie exact partea din cel de-al doilea DAG inițial de care avem nevoie. Prima clauză Prolog din definiția unificării se ocupă de acest *caz de bază*. *Variabila rest* corespunzătoare primului argument este făcută să aibă o valoare identică cu DAG-ul furnizat de cel de-al doilea argument. Această clauză va acoperi și alte situații utile, cum ar fi aceea în care se dorește unificarea a două DAG-uri atomice:

```
unifica(Dag,Dag):-!.                                    (5.4)
```

Restul definiției predicatului **unifica** constă în următoarea clauză Prolog:

```
unifica([Drum:Valoare|Dag1],Dag):-
        drumval(Dag,Drum,Valoare,Dag2),                  (5.5)
        unifica(Dag1,Dag2).
```

Această parte a definiției împarte primul DAG în perechea inițială **Drum: Valoare** și un rest (**Dag1**), după care folosește predicatul **drumval** pentru a se asigura că **Drum** are valoarea **Valoare** în cel de-al doilea DAG (**Dag**). În plus, predicatul **drumval** întoarce, în cel de-al patrulea argument al său, un nou DAG, **Dag2**, care conține tot ceea ce exista în **Dag** cu excepția perechii (*caracteristică, valoare*) corespunzătoare lui **Drum**. În final, predicatul **unifica** își propune unificarea celor două DAG-uri rămase, **Dag1** și **Dag2**.

Faptul că, în apelarea recursivă, se utilizează rezultatul furnizat de predicatul **drumval** și nu DAG-ul inițial **Dag**, înseamnă că, atunci când procesul recursiv ajunge la *cazul de bază*, cel de-al doilea argument al predicatului **unifica** va conține numai informație referitoare la caracteristici care inițial existau în cel de-al doilea DAG, dar nu și în primul. Al doilea argument al lui **unifica** va furniza, prin urmare, valoarea adecvată pentru instanțierea primei *variabile rest*.

Definirea predicatului **drumval** se realizează, de asemenea, în două etape. Dacă caracteristica dorită, **Caract**, se găsește în primul element al DAG-ului, atunci valoarea ei (**Valoare1**) este extrasă și verificată pentru unificare cu valoarea argument (**Valoare2**):

```
drumval([Caract:Valoare1|Dag],Caract,Valoare2,Dag):-     (5.6)
        !,unifica(Valoare1,Valoare2).
```

Altfel, se parcurge în mod recursiv lista, căutându-se o intrare corespunzătoare acelei caracteristici:

```
drumval([Dag|Dag1],Caract,Valoare,[Dag|Dag2]):-          (5.7)
        drumval(Dag1,Caract,Valoare,Dag2).
```

Vom mai observa faptul că folosirea lui *cut* în prima clauză din definiția predicatului **drumval** semnifică aceea că, odată găsită în DAG o intrare

corespunzător caracteristicii dorite **Caract**, chiar dacă valoarea înregistrată (**Valoare1**) nu unifică cu valoarea dorită (**Valoare2**), nu se va lua în considerație posibilitatea căutării mai departe în listă a altor intrări corespunzătoare acestei caracteristici.

Presupunând că avem programul Prolog de unificare alcătuit din clauzele (5.4) - (5.7), putem realiza interogarea Prologului în diverse moduri. Astfel, predicatul **drumval** poate fi apelat relativ la un DAG care deja conține o intrare pentru caracteristica dorită:

```
?- drumval([cat:vp,numar:sing|X],numar,Valoare,Rest).

Rest=[cat:vp|X],

Valoare=sing ? ; <Enter>

no
```

Ultimul mesaj furnizat (**no**) arată unicitatea soluției găsite. Apelarea se poate face, în egală măsură, pentru un DAG care nu menționează în mod explicit caracteristica respectivă. În acest caz, se va face instanțierea *variabilei rest* astfel încât DAG-ul să aibă o valoare pentru acea caracteristică după aceea:

```
?- drumval([cat:vp,numar:sing|X],timp,prezent,Rest).

X=[timp:prezent|_A],

Rest=[cat:vp,numar:sing|_A] ? ; <Enter>

no
```

În aceeași categorie se înscrie o apelare de forma:

```
?- drumval(Dag,numar,plur,Rest).

Dag=[numar:plur|Rest] ? ; <Enter>

no
```

În fine, dacă se apelează **drumval** cu o valoare a unei caracteristici care intră în conflict cu valoarea deja înregistrată, scopul (și, prin urmare, întreaga unificare) *eșuează*:

```
?- drumval([cat:vp,numar:sing|X],numar,plur,Rest).

no
```

Această versatilitate a predicatului este extrem de utilă pentru a acoperi *toate cazurile* care pot să apară în decursul unificării.

5.2.7. Implementarea PATR în Polog

Implementarea PATR în Prolog pe care o prezentăm aici este preluată din [29] și constă, practic, în înglobarea în cadrul Prologului a unui limbaj formal extrem de asemănător cu ceea ce am numit "notația PATR".

O **regulă** PATR, într-o notație similară celor furnizate de Prolog, ar putea fi următoarea:

Regula

s → *np vp* :
<np per> = <vp per>
<np num> = <vp num>.

Această regulă afirmă că o categorie (DAG) a membrului stâng poate fi realizată printr-o succesiune de două categorii (DAG-uri) din membrul drept, cu condiția ca aceste categorii să satisfacă anumite condiții. Legătura dintre categoria membrului stâng și cele ale membrului drept poate fi exprimată prin intermediul unui predicat **regula**, care are două argumente. Mai întâi, DAG-urile care intervin trebuie să aibă valorile corecte pentru caracteristica **cat**. Ultimele două trebuie, în plus, să împartă (partajeze) valorile caracteristicilor **per** și **num**. Aceste constrângeri pot fi formulate, cu ajutorul predicatului **drumval**, după cum urmează:

```
regula(S,[NP,VP]):-
        drumval(S,cat,s,_),
        drumval(NP,cat,np,_),
        drumval(VP,cat,vp,_),
        drumval(NP,per,X,_),
        drumval(VP,per,X,_),
        drumval(NP,num,Y,_),
        drumval(VP,num,Y,_).
```

Dacă predicatului **regula** îi sunt furnizate DAG-uri complet instanțiate corespunzător lui **S**, **NP** și **VP**, acesta va înregistra succes exact în cazul în care DAG-urile satisfac condițiile regulii PATR. Dacă aceluiași predicat îi sunt furnizate DAG-uri care nu specifică valori pentru toate caracteristicile relevante, acesta va încerca să instanțieze DAG-urile astfel încât condițiile să fie satisfăcute. Aceasta înseamnă că predicatul **regula** va calcula *extinderile minime* ale categoriilor care îi sunt date, astfel încât acestea să satisfacă regula.

Aceasta este strategia de bază folosită pentru a codifica conținutul PATR în clauze Prolog. În cele ce urmează, vom fi mai preocupați de eleganța notației și vom lua măsurile necesare pentru a obține o notație mai clară, mai transparentă și mai apropiată de notația PATR inițială, fără a schimba

conținutul prezentat în vreun mod semnificativ. Din punct de vedere sintactic, nu va fi necesară decât definirea a patru operatori infixați, care se vor adăuga operatorului :, utilizat anterior:

```
?- op(500,xfx,:).

?- op(500,xfx,--->).

?- op(600,xfx, = = = ).

?- op(400,xfx,egula).

?- op(600,xfx,uv).
```

Mai întâi, vom redenumi **regula** prin folosirea operatorului infixat ---> și vom introduce un predicat === care este foarte asemănător cu **drumval**. Acest predicat va folosi pe **drumval**, atunci când îi sunt furnizate ca prim argument un DAG și un nume de caracteristică (legate între ele prin operatorul :), pentru a găsi valoarea de caracteristică relevantă din DAG și a o unifica cu cel de-al doilea argument. În aceste condiții, regula anterioară poate fi reformulată după cum urmează:

```
S ---> [NP,VP]:-

   S:cat = = = s,

   NP:cat = = = np,

   VP:cat = = = vp,

   NP:per = = = X,

   VP:per = = = X,

   NP:num = = = Y,

   VP:num = = = Y.
```

Această notație poate fi ușor modificată pentru a deveni și mai apropiată de notația PATR inițială. Astfel, definiția lui === poate fi generalizată pentru a permite unei perechi de tipul *DAG:caracteristică* să intervină atât ca prim argument, cât și ca cel de-al doilea argument. Mai mult, generalizarea poate fi făcută astfel încât să fie permise atât o unică denumire de caracteristică, cât și o întreagă secvență de nume de caracteristici (despărțite între ele prin două puncte). În virtutea acestei generalizări, vor fi permise construcții de tipul:

```
NP:num = = = VP:num

NP:agr:per = = = 1
```

În fine, operatorul **egula** poate face ca reprezentarea să fie și mai apropiată de notația PATR inițială. Utilizând acest operator, regula la care ne-am referit devine:

```
R egula S ---> [NP,VP]:-
   S:cat = = = s,
   NP:cat = = = np,
   VP:cat = = = vp,
   NP:per = = = VP:per,
   NP:num = = = VP:num.
```

Se observă că funcția variabilei Prolog R este pur decorativă. Într-o implementare alternativă, am fi putut defini '**Regula**' ca reprezentând un operator unar (fiind necesară utilizarea permanentă a ghilimelelor). În actuala implementare, ori de câte ori vom folosi predicatul --->, primul argument va fi o structură de forma

Variabilă **egula** MS

unde prin MS am desemnat adevăratul membru stâng al regulii. Aceasta este una dintre reprezentările cele mai apropiate de sintaxa PATR inițială.

Intrările lexicale vor fi tratate în mod similar. Fiecare dintre acestea exprimă o legătură între cuvântul definit și DAG-ul corespunzător acelui cuvânt. Un exemplu de intrare lexicală este

Cuvânt eu:
<cat>=np
<per>=1
<num>=sing.

Dacă notăm prin C DAG-ul asociat unui cuvânt, atunci intrarea lexicală amintită poate fi transcrisă în Prolog după cum urmează:

```
cuvant(C,eu):-
   drumval(C,cat,np,_),
   drumval(C,per,1,_),
   drumval(C,num,sing,_).
```

În cele ce urmează, vom redenumi **cuvant** utilizând operatorul infixat **uv** și vom folosi, ca și înainte, predicatul ===, pentru a da regulii următoarea formă:

```
C uv eu:-
   C:cat = = = np,
   C:per = = = 1,
   C:num = = = sing.
```

Se observă că aici variabila C nu mai are un rol pur decorativ, ea desemnând DAG-ul asociat cuvântului. Condițiile asupra acestei variabile sunt exprimate în corpul clauzei.

Analiza efectuată poate fi puțin schimbată pentru a simplifica regula de rescriere referitoare la S astfel:

Regula

$s \rightarrow np\ vp$:

<np agr> = <vp agr>.

Aceasta se traduce în felul următor:

```
R egula S ---> [NP,VP]:-
   S:cat = = = s,
   NP:cat = = = np,
   VP:cat = = = vp,
   NP:agr = = = VP:agr.
```

Această modificare operată asupra analizei se reflectă și în cazul intrărilor lexicale. Prin urmare,

Cuvânt eu:

<cat> = np

<agr per> = 1

<agr num> = sing.

se traduce astfel:

```
C uv eu:-
   C:cat = = = np,
   C:agr:per = = = 1,
   C:agr:num = = = sing.
```

Cu excepția detaliilor minore de punctuație, cele două mari diferențe între *notația PATR* și *extensia de tip PATR a Prologului* definită aici sunt:

- necesitatea utilizării unor variabile explicite în cazul celei din urmă;
- aspectul sintactic al drumurilor.

Definițiile date operatorilor au efectul de a face ca sintaxa de tip PATR a regulilor să fie acceptată de către interpretorul Prolog, dar nu furnizează nici o informație legată de semnificație și nu permit efectuarea vreunei operații importante cu aceste reguli. În acest scop sunt necesare noi definiții. Să considerăm, spre exemplu, operatorul ===. Dorim ca semantica acestuia să fie

precizată astfel încât **X===Y** să fie adevărat dacă și numai dacă atât X, cât și Y desemnează același obiect. Prin urmare:

```
X = = = Y:-
   denota(X,Z),
   denota(Y,Z).
```

Vom permite ca variabilele și atomii să se desemneze pe ele însele:

```
denota(Var,Var):-var(Var),!.
denota(Atom,Atom):-atomic(Atom),!.
```

Perechile de tipul **Dag:Drum** desemnează valoarea lui **Drum** în **Dag**:

```
denota(Dag:Drum,Valoare):-
   drumval(Dag,Drum,Valoare,_).
```

Se observă că cel de-al doilea argument al lui **drumval** s-ar putea să nu fie atomic. Definiția existentă a predicatului **drumval** presupune însă ca acest argument să fie de natură atomică. Din această cauză, devine necesar să *prefațăm* definiția lui **drumval** cu o a treia clauză. Această clauză va stipula faptul că, dacă cel de-al doilea argument este el însuși o pereche, se aplică **drumval** primului element (**Caract**) al acestei perechi, pentru a regăsi valoarea corespunzătoare (**Dag2**) în **Dag1**, după care se aplică **drumval** celui de-al doilea element (**Drum**) cu referire la **Dag2**:

```
drumval(Dag1,Caract:Drum,Valoare,Dag):-
        !,drumval(Dag1,Caract,Dag2,Dag),
        drumval(Dag2,Drum,Valoare,_).
```

Întrucât operatorii **:** și **===** sunt singurii care intervin în corpul regulilor, furnizarea unei interpretări pentru aceștia este suficientă pentru a da o interpretare și operatorilor **uv** și **--->**, deoarece aceștia apar exclusiv în capul unei reguli.

5.2.8. Implementarea în Prolog a unui dispozitiv de recunoaștere bazat pe gramatici de unificare

Pentru a exemplifica modul în care pot fi utilizate gramaticile având o transcriere de tipul celei prezentate anterior (§5.2.7), vom programa, în cele ce urmează, un *dispozitiv de recunoaștere* relativ simplu, care efectuează o *analiză din colțul stâng* și folosește o strategie de invocare a regulilor de tip *bottom-up*. Dispozitivul de recunoaștere implementat nu va utiliza o structură de date de tip hartă și nu va presupune decât existența prealabilă a unor reguli scrise în notația

de tip PATR (§5.2.7). Şirurile de caractere vor fi reprezentate sub forma listei diferenţă.

Un prim predicat pe care îl vom utiliza asociază unui cuvânt al şirului de intrare categoria corespunzătoare (prin categorie înţelegându-se aici un DAG ce reprezintă o structură de caracteristici). Acelaşi predicat trebuie să poată manevra producţiile vide:

```
frunza(C,[Cuvant|X],X):-C uv Cuvant.

frunza(C,X,X):- R egula C--->[ ].
```

Dar utilizarea listelor diferenţă sugerează un mod mai compact şi mai elegant de a defini acest predicat, folosind *notaţia DCG*:

```
frunza(Dag)-->[Cuvant],{Dag uv Cuvant}.

frunza(Dag)-->{_egula Dag --->[ ]}.
```

Notaţia DCG va fi utilizată şi în continuarea acestei implementări. Pentru a recunoaşte un şir ca reprezentând o ocurenţă a lui **Dag1** trebuie să luăm în consideraţie o frunză iniţială **Dag0** şi să demonstrăm că **Dag0** constituie un colţ stâng al lui **Dag1**, adică o categorie care ar apărea în partea din stânga jos a arborelui de derivare corespunzător:

```
recunoaste(Dag1)-->frunza(Dag0),colt_stang(Dag0,Dag1).
```

Dacă *sfârşitul şirului a fost atins*, atunci **Dag1** constituie un colţ stâng al lui **Dag2** numai dacă cele două Dag-uri unifică:

```
colt_stang(Dag1,Dag2)-->[ ],{unifica(Dag1,Dag2)}.
```

Altfel, **Dag1** constituie un colţ stâng al lui **Dag2** dacă

- există o regulă care are categorie mamă pe **Dag0**, categoria cea mai din stânga **Dag1** şi categoriile rămase desemnate prin **Dags**;
- restul şirului poate fi recunoscut ca fiind desemnat prin **Dags**;
- **Dag0** este un colţ stâng al lui **Dag2**:

```
colt_stang(Dag1,Dag2)-->
        {_ egula Dag0--->[Dag1|Dags]},
        recunoaste_rest(Dags),
        colt_stang(Dag0,Dag2).
```

Predicatul **recunoaste_rest** are următoarea definiţie recursivă:

```
recunoaste_rest([ ])-->[ ].
recunoaste_rest([Dag|Dags])-->
        recunoaste(Dag),
        recunoaste_rest(Dags).
```

Aceasta completează implementarea dispozitivului de recunoaştere din colţul stâng, cu excepţia existenţei unui predicat prin intermediul căruia să se realizeze interogarea Prologului. Acesta este predicatul **test**, a cărui definire se va face direct în program. (Este de remarcat faptul că predicatul **test** se defineşte cu ajutorul predicatului **recunoaste**, care are trei argumente. Cele două argumente suplimentare - a se vedea definiţia predicatului **recunoaste** - reprezintă consecinţa traducerii în Prolog a notaţiei DCG utilizate în această definiţie.)

Programul complet, scris în SICStus Prolog şi reprezentând implementarea dispozitivului de recunoaştere descris, care utilizează Gramatica nr. 6, este următorul:

Programul 5.1

```
X = = = Y:-denota(X,Z),denota(Y,Z).


denota(Var,Var):-var(Var),!.
denota(Atom,Atom):-atomic(Atom),!.
denota(Dag:Drum,Valoare):-
        drumval(Dag,Drum,Valoare,_).


drumval(Dag1,Caract:Drum,Valoare,Dag):-
         !,drumval(Dag1,Caract,Dag2,Dag),
         drumval(Dag2,Drum,Valoare,_).
drumval([Caract:Valoare1|Dag],Caract,Valoare2,Dag):-
         !,unifica(Valoare1,Valoare2).
drumval([Dag|Dag1],Caract,Valoare,[Dag|Dag2]):-
         drumval(Dag1,Caract,Valoare,Dag2).


unifica(Dag,Dag):-!.
unifica([Drum:Valoare|Dag1],Dag):-
         drumval(Dag,Drum,Valoare,Dag2),
         unifica(Dag1,Dag2).


frunza(Dag)-->[Cuvant],{Dag uv Cuvant}.
frunza(Dag)-->{_ egula Dag--->[]}.
```

```
recunoaste(Dag1)-->frunza(Dag0),
                   colt_stang(Dag0,Dag1).

colt_stang(Dag1,Dag2)-->[],{unifica(Dag1,Dag2)}.
colt_stang(Dag1,Dag2)-->{_ egula Dag0--->[Dag1|Dags]},
                         recunoaste_rest(Dags),
                         colt_stang(Dag0,Dag2).

recunoaste_rest([])-->[].
recunoaste_rest([Dag|Dags])-->recunoaste(Dag),
                              recunoaste_rest(Dags).

test(Sir):-recunoaste(Categ,Sir,[]).

%Gramatica nr.6%

_ egula S--->[NP,VP]:-
 S :cat = = = s,
 NP:cat = = = np,
 VP:cat = = = vp,
 S :num = = = NP:num,
 S :num = = = VP:num,
 NP:num = = = VP:num.
_ egula VP--->[V,NP]:-
 VP:cat = = = vp,
 V :cat = = = v,
 NP:cat = = = np,
 VP:num = = = V:num.
_ egula NP--->[Det,N]:-
 NP:cat = = = np,
 Det:cat = = = det,
 N :cat = = = n,
 NP:num = = = Det:num,
```

```
NP:num = = = N:num,
Det:num = = = N:num,
Det:gen = = = N:gen.

C uv un:-
  C:cat = = = det,
  C:num = = = sing,
  C:gen = = = masc.
C uv niste:-
  C:cat = = = det,
  C:num = = = plural,
  C:gen = = = masc.
C uv o:-
  C:cat = = = det,
  C:num = = = sing,
  C:gen = = = fem.
C uv elev:-
  C:cat = = = n,
  C:num = = = sing,
  C:gen = = = masc.
C uv elevi:-
  C:cat = = = n,
  C:num = = = plural,
  C:gen = = = masc.
C uv carte:-
  C:cat = = = n,
  C:num = = = sing,
  C:gen = = = fem.
C uv iubeste:-
  C:cat = = = v,
  C:num = = = sing.
C uv iubesc:-
  C:cat = = = v,
```

```
C:num = = = plural.
```

Interogarea Prologului se realizează în felul următor:

```
?- test([un,elev,iubeste,o,carte]).
yes
```

sau

```
?- test([iubeste,o,carte]).
yes
```

sau

```
?- test([un,elev,iubeste]).
no
```

Remarcăm, din nou, faptul că dispozitivul de recunoaștere / parser-ul descris este unul care nu folosește o hartă. De altfel, toți algoritmii de analiză sintactică cu hartă prezentați anterior utilizau gramatici PS având exclusiv categorii monoatomice. Gramaticile de unificare permit existența unor *categorii complexe*, ceea ce face ca implementarea unor analizori sintactici cu hartă să îmbrace forme specifice, care însă nu constituie obiectul lucrării de față. Reținem doar faptul că există trei opțiuni de bază referitoare la modul în care se poate face analiza sintactică atunci când o gramatică permite existența categoriilor complexe, și anume:

- expandarea gramaticii la o gramatică PS independentă de context monoatomică;
- efectuarea analizei sintactice folosind "coloana vertebrală" a unei gramatici PS independente de context monoatomice;
- încorporarea în analizorul sintactic a unor mecanisme speciale.

Referitor la prima dintre aceste opțiuni vom remarca faptul că cea mai simplă modalitate de folosire a caracteristicilor poate fi privită ca o modalitate de specificare a unor mulțimi de reguli într-o formă abreviată. Spre exemplu, regula

Regula

$$S \longrightarrow NP\ VP:$$
$$\langle NP\ num \rangle = \langle VP\ num \rangle.$$

poate fi privită ca o stenogramă a următoarelor două reguli:

Regula

$$S \longrightarrow NP\_sing \quad VP\_sing.$$

Regula

$$S \longrightarrow NP\_plur \quad VP\_plur.$$

În acest caz, strategia care trebuie aplicată pentru a realiza analiza sintactică constă în expandarea mulțimii regulilor urmată de efectuarea analizei folosind tehnicile standard. Această strategie este însă aplicabilă numai în cazul gramaticilor de dimensiuni foarte mici, deci a gramaticilor experimentale și nu a celor care reprezintă realitatea lingvistică a diverselor limbi naturale.

Cea de-a doua opțiune amintită se bazează pe faptul că, atunci când există o caracteristică (sau o mulțime de caracteristici) astfel încât fiecare DAG din fiecare regulă a gramaticii specifică o anumită valoare pentru acea caracteristică, un proces inițial de analiză sintactică poate avea loc pe baza gramaticii PS independente de context monoatomice rezultate prin luarea în considerație exclusiv a acestei caracteristici și ignorarea celorlalte. Desigur, atunci când o serie de caracteristici sunt neglijate în timpul procesului de parsing, analizorul sintactic va pierde timp explorând ipoteze și formulând potențiale analize care ar fi excluse de către întreaga gramatică.

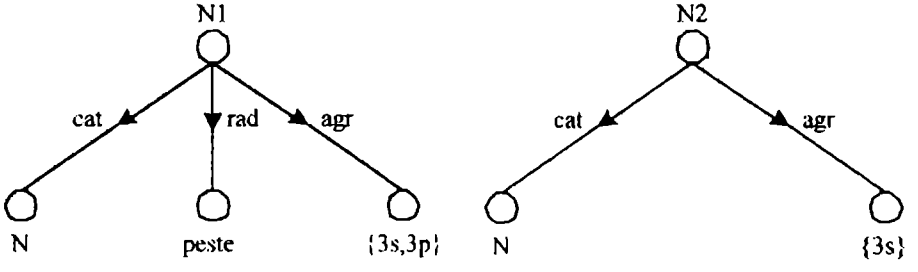
În fine, atunci când în analizorul sintactic se încorporează mecanisme speciale, problema centrală care se pune este aceeași, și anume dacă categoriile sunt atomi sau structuri de caracteristici arbitrare. Cerința esențială este ca operațiile de bază ale analizorului sintactic (testarea egalității, căutarea regulilor etc.) să fie redefinite astfel încât să accepte o structură de date diferită pentru categorii. Această cerință relativ simplă maschează însă un număr de probleme speciale care se ridică și atunci când se încearcă analiza sintactică direct pe baza categoriilor complexe. Această abordare rămâne, probabil, cea mai eficientă, în pofida tuturor problemelor care pot să apară legat de ea. (O analiză a problemelor specifice care se ridică în acest caz, precum și a unora dintre ideile standard pentru soluționarea lor poate fi văzută în [29].

Întrucât algoritmii standard de analiză sintactică cu hartă prezentați anterior utilizează gramatici PS având exclusiv categorii monoatomice, iar adaptarea lor în cazul existenței categoriilor complexe necesită efectuarea unor operații specifice ce ridică o serie de probleme speciale, ne vom plasa, în continuare, în contextul analizorilor sintactici care nu folosesc o structură de date de tip hartă.

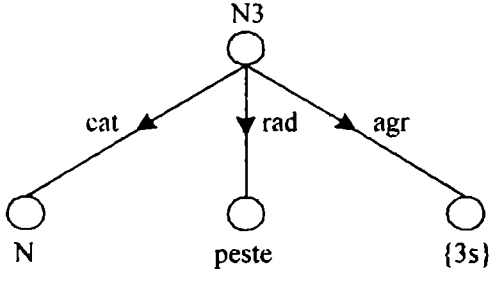
5.3. Analiza sintactică bazată pe gramatici de unificare

În cele ce urmează, vom formula un *algoritm de construcție a unui nou constituent*, care, la rândul său, se bazează pe un *algoritm general de unificare a grafurilor*, lăsând în sarcina cititorului programarea acestora. Având un algoritm de formare a unui nou constituent, devine posibilă construcția unui *parser* prin utilizarea oricăruia dintre algoritmii standard de analiză sintactică. Tocmai de aceea, în prezent, suntem mai puțin interesați de limbajul de programare folosit și ne concentrăm atenția asupra unei formulări mai generale, pe pași, a algoritmilor care formează suportul analizei sintactice bazate pe gramatici de unificare.

Formalismul bazat pe unificare a fost definit prin reprezentarea structurilor de caracteristici sub formă de DAG-uri. În cele ce urmează, vom folosi aceeași reprezentare, în care fiecare constituent și valoare sunt redate ca un nod (vârf al grafului), în timp ce caracteristicile sunt reprezentate sub formă de arce (muchii) etichetate. Iată, spre exemplu, două DAG-uri reprezentând două grupuri nominale:



Unificarea a două structuri de caracteristici este, în continuare, definită sub forma unui algoritm de unificare a două grafuri. Vom formula un *algoritm de unificare a grafurilor*, care, în urma aplicării sale nodurilor N_1 și N_2, va genera noul constituent



Folosind această terminologie, *algoritmul de unificare a grafurilor* este

Algoritmul 5.2

Pentru a unifica un DAG având ca rădăcină nodul N_i cu un DAG având ca rădăcină nodul N_j, execută:

1. Dacă N_i coincide cu N_j, atunci întoarce N_i; Stop.
2. Dacă atât N_i, cât și N_j sunt noduri terminale, se compară etichetele lor: dacă intersecția acestora este nevidă, se construiește un nou nod care are ca etichetă această intersecție. Altfel, DAG-urile nu se pot unifica.
3. Dacă N_i și N_j nu sunt noduri terminale, atunci se creează un nou nod N. Pentru fiecare arc etichetat cu F de la nodul N_i la nodul NF_i,
   - **3a.** Dacă există un arc etichetat cu F de la nodul N_j la nodul NF_j, atunci se unifică în mod recursiv nodurile NF_i cu NF_j. Se construiește un arc etichetat cu F de la N la rezultatul acestei construcții recursive.
   - **3b.** Dacă nu există nici un arc etichetat cu F care pleacă din nodul N_j, atunci se construiește un arc etichetat cu F de la N la NF_i.
   - **3c.** Pentru fiecare arc etichetat cu F de la nodul N_j la nodul NF_j, corespunzător căruia nu există un arc etichetat cu F care pleacă din nodul N_i, se creează un nou arc etichetat cu F de la nodul N la nodul NF_j.

Se observă că unificarea descrisă de acest algoritm nu este una distructivă, întrucât algoritmii de analiză sintactică sunt prezentați aici în forma lor cea mai generală, așa cum au fost ei preluați din [4]. Fiind în posesia acestui algoritm, putem enunța *algoritmul de construcție a unui nou constituent*:

Algoritmul 5.3

Intrare:

- Producția $X_0 \rightarrow X_1 \ldots X_n$;
- SC1, SC2, ... , SCn - subconstituenții corespunzători lui $X_1, \ldots, X_n$;
- o mulțime de ecuații cu caracteristici de forma $F_i = V$.

Algoritm:

1. Se creează un nod CC0 ca rădăcină a noii structuri cu caracteristici.
2. Pentru fiecare i, $1 \leq i \leq n$:
   - se face o copie a DAG-ului de rădăcină SCi și se notează cu CC_i rădăcina acestei copii;
   - se trasează un arc etichetat cu i de la nodul CC0 la nodul CC_i.
3. Pentru fiecare ecuație cu caracteristici de forma $F_i = V$, unde V reprezintă o valoare, se urmează legătura F de la nodul CC_i la un nod N_i și se unifică N_i cu V.
4. Pentru fiecare ecuație cu caracteristici (de forma $F_i = G_j$),

4a. Dacă există o legătură F din CC_i și o legătură G din CC_j, atunci
  i. se urmează legătura F până la un nod N_i și legătura G până la un nod N_j;
  ii. se unifică N_i cu N_j, folosind Algoritmul 5.2 de unificare a grafurilor, creându-se astfel un nou nod X;
  iii. toate arcele care intrau fie în N_i, fie în N_j sunt aduse în X.

4b. Dacă nu există nici o legătură F din CC_i, dar există o legătură G de la nodul CC_j la un nod N_j, se creează o legătură F de la nodul CC_i la nodul N_j.

4c. Dacă nu există nici o legătură G din CC_j, dar există o legătură F de la nodul CC_i la un nod N_i, se creează o legătură G de la nodul CC_j la nodul N_i.

Ieșire: Un DAG care satisface toate ecuațiile cu caracteristici.

Pe baza acestui algoritm de construcție a unui nou constituent folosind ecuațiile cu caracteristici se poate proiecta un parser prin adaptarea oricăruia dintre algoritmii standard de analiză sintactică prezentați anterior. Subliniem faptul că algoritmii enunțați în acest paragraf presupun că este dată categoria fiecărui constituent în parte. Desigur, atunci când categoriile constituenților nu sunt specificate, adaptarea algoritmilor de analiză sintactică standard, bazați pe gramatici independente de context, în vederea utilizării gramaticilor de unificare, nu mai este atât de evidentă. Așa cum s-a mai remarcat, se poate însă arăta că, atâta timp cât există o submulțime finită de caracteristici, astfel încât cel puțin una dintre acestea să fie specificată în cadrul fiecărui constituent, gramatica de unificare poate fi convertită într-o gramatică independentă de context.

O altă particularitate extrem de importantă a gramaticilor de unificare este aceea că ele permit codificarea a mult mai multă informație în cadrul *lexiconului*. Într-adevăr, aproape întreaga gramatică poate fi codificată în interiorul lexiconului, ea ajungând să includă foarte puține reguli propriu-zise, aspect asupra căruia vom reveni.

Gramaticile independente de context augmentate au fost utilizate în diverse modele computaționale, începând cu momentul introducerii *gramaticilor cu atribute* de către Knuth (1968), autor care le-a folosit în procesul de parsing referitor la limbajele de programare. De atunci, numeroase sisteme au utilizat reguli adnotate de diverse forme.

Studiul gramaticilor de unificare se bazează în special pe lucrările lui Kay (1982) și pe sistemul PATR-II (Shieber, 1984; 1986). O considerabil de importantă activitate de cercetare în domeniu se desfășoară și în prezent. O analiză pertinentă a domeniului poate fi găsită în Shieber (1986). Formalisme diferite pentru structurile de caracteristici au fost dezvoltate de-a lungul timpului, dintre care amintim: Rounds (1988), Johnson (1991) și Shieber (1992).

5.4. Reprezentarea cunoştinţelor lexicale. Lexiconul

O altă trăsătură importantă a gramaticilor de unificare este aceea că ele permit codificarea unui număr mult mai mare de informaţii în interiorul lexiconului. Aproape întreaga gramatică poate fi încorporată în lexicon, ceea ce micşorează în mod simţitor numărul de reguli ale gramaticii.

Lexiconul unui sistem de procesare a limbajului natural trebuie să furnizeze o codificare sistematică şi accesibilă a unei informaţii extrem de variate asupra cuvintelor. Lexiconul este, alături de sistemul de reguli, o parte componentă a gramaticii. Prin conceptul de lexicon[2] se înţelege o listă de unităţi lexicale aranjate într-o ordine anumită şi conţinând o informaţie specifică în funcţie de scopul urmărit.

Informaţia *minimală* care trebuie inclusă în lexicon este de natură sintactică. Astfel, informaţiile tipice furnizate de un lexicon se referă la: categoria (gramaticală sau lexicală) a cuvântului, posibilităţile de subcategorizare, caz, număr, persoană, gen, mod etc. Informaţia prevăzută diferă în mod esenţial în funcţie de limbă şi de tipul cuvântului care se tratează. Dacă ea se rezumă la cea minimală, adică la cea de tipul menţionat anterior, vor exista două consecinţe relativ neplăcute. Mai întâi, un parser nu va putea face altceva decât analiză sintactică, întrucât un lexicon conceput în acest fel nu furnizează nici o informaţie cu ajutorul căreia să se distingă, spre exemplu, între sensul propoziţiilor *bărbatul iubeşte femeia* şi *femeia urăşte bărbatul*. Din punctul de vedere al lexiconului discutat, aceste două şiruri de cuvinte vor fi identice. În al doilea rând, un astfel de lexicon va trebui să afişeze în mod exhaustiv fiecare formă flexionară a fiecărui cuvânt, indiferent dacă acea formă este regulată sau nu, ceea ce măreşte în mod considerabil şi inutil atât dimensiunea lexiconului, cât şi efortul corespunzător de implementare. Următoarele concluzii privitoare la conceperea lexiconului se desprind imediat:

- Pentru ca un lexicon să fie util şi în efectuarea altor operaţii în afara celei de analiză sintactică, el trebuie să includă şi *informaţie semantică* privitor la cuvintele pe care le defineşte.

---

[2] Termenul *lexicon* este preferat pentru gramaticile formale de tip generativ. În gramaticile formale, se face distincţia între lexiconul mental, intern fiecărui vorbitor nativ, conţinând întreaga informaţie sintactică şi semantico-sintactică asupra categoriilor sintactice şi a formativelor lexicale, informaţie care, în concepţia chomskyană, este parţial înnăscută şi parţial dobândită, şi lexiconul gramaticii, propus de cercetător în intenţia de "descoperire" a celui mental. Lexiconul, ca parte componentă a gramaticii, este alcătuit, la rândul lui, dintr-un vocabular auxiliar, setul de simboluri reprezentând categoriile sintactice, şi un vocabular terminal, setul de lexeme şi de morfeme, din limba supusă modelării. În ansamblul gramaticii, poziţia şi rolul lexiconului, precum şi structura articolelor au diferit de la un model de gramatică la altul (vezi [9], p. 274, s. v. *lexicon*).

- Pentru ca un lexicon să evite redundanța, el trebuie să constea din rădăcini de cuvinte, împreună cu informația morfologică și sintactică necesară pentru a putea deduce formele regulate ale cuvintelor. În același timp, lexiconul trebuie să conțină intrări separate corespunzătoare formelor neregulate.

Sistemele moderne de procesare a limbajului natural acordă o importanță tot mai mare proiectării lexiconului, ceea ce ne face ca, în cele ce urmează, să privim cu o mai mare atenție atât informația care trebuie reprezentată, cât și modul ei de reprezentare. Ne vom referi, mai întâi, la *informația de natură **sintactică*** care trebuie codificată în cadrul unui lexicon. Specialiștii în lingvistică computațională consideră că există trei tipuri principale de informație sintactică asociată unui cuvânt:

- *categoria lexicală* (partea de vorbire) a cuvântului (verb, adjectiv etc.);
- posibilitățile de *combinare* ale cuvântului (complementele și eventual subiectul său);
- anumite *proprietăți* relevante din punct de vedere sintactic, *care se moștenesc* (spre exemplu, în cazul substantivelor, genul).

Într-o sintaxă bazată pe caracteristici, toate trei tipurile de informație amintite sunt, în general, codificate în cadrul categoriei sintactice a cuvântului. Exemplu:

Lexem[3] mănâncă:

<cat> = V
<arg0 cat> = NP
<arg0 caz> = nom
<arg1 cat> = NP
<arg1 caz> = acc
<arg2 cat> = PP
<arg2 pform> = la.

Această intrare lexicală afirmă că verbul românesc "mănâncă" este un verb care subcategorizează pentru un obiect direct NP (caz acuzativ), adică este un verb tranzitiv și un complement indirect reprezentat printr-un grup prepozițional introdus de prepoziția "la". Același verb se combină cu un subiect NP în cazul nominativ. Este evident că folosim caracteristica **arg0** pentru a codifica informația referitoare la subiectul verbului, caracteristica **arg1** pentru codificarea informației referitoare la obiectul direct și caracteristica **arg2**

[3] Cuvânt sau parte de cuvânt care servește ca suport minimal al semnificației (vezi [22], p. 569, s.v. *lexem*)

pentru a codifica informația referitoare la obiectul indirect reprezentat prin grupul prepozițional. Un exemplu de propoziție care corespunde acestei descrieri a verbului *mănâncă* este

Fata mănâncă desertul la restaurant.

Evident, în cazul verbelor care permit existența mai multor complemente se pot folosi mai multe caracteristici de tipul **argi**. S-a arătat că, în majoritatea limbilor, ar putea fi suficiente patru asemenea caracteristici de tip argument pentru descrierea gramaticii. Aceste caracteristici ale verbelor vor fi utilizate de reguli ca

Regula
VP ⟶ *V X1 X2 :*
<V arg1> = X1
<V arg2> = X2
<V arg3> = 0.

sau

Regula
VP ⟶ *V X1 X2 X3 :*
<V arg1> = X1
<V arg2> = X2
<V arg3> = X3.

Toate limbile naturale conțin mii de intrări lexicale, corespunzătoare verbelor sau altor părți de vorbire, identice între ele (în sensul că prezintă aceleași caracteristici). Pentru a evita mărirea inutilă a dimensiunilor lexiconului, se pot folosi abrevieri exprimate sub forma unor *macro*-uri:

```
Macro sin_Vi : {verb intranzitiv - "merg"}
    <cat> = V
    <arg0 cat> = PP
    <arg0 pform> = la.  {"merg la plajă"}

                sau

Macro sin_Vt : {verb tranzitiv - "mănâncă"}
    sin_Vi
    <arg1 cat> = NP
    <arg1 caz>  = acc.  {"mănâncă desertul la restaurant"}
```

Ideea unui *macro* este de a avea un unic simbol, spre exemplu **sin_Vi**, care abreviază o întreagă mulțime de specificații de caracteristici. Ori de câte ori se include numele unui *macro* în cadrul unei intrări lexicale este ca și cum s-ar fi inclus întreaga mulțime a specificațiilor pe care acel nume le abreviază. Există definiții de *macro*-uri care invocă alte *macro*-uri, ca în exemplul anterior (**sin_Vt**). Prin utilizarea *macro*-urilor intrările lexicale devin mult mai scurte, ele putând fi de tipul:

Lexem mănâncă:
sin_Vt.

sau

Lexem mănâncă:
sin_Vt
<arg2 cat> = NP
<arg2 caz> = nom.

Se observă că numeroase cuvinte vor avea în continuare mai multe intrări lexicale corespunzător categoriilor lexicale diferite pe care le pot avea.

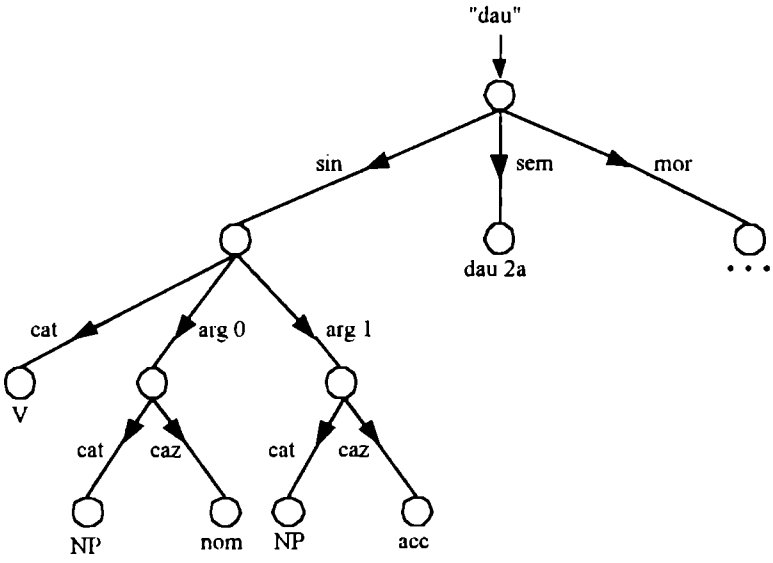
Macro-urile reprezintă un instrument util în scrierea unui lexicon. Cea mai importantă rămâne însă *interpretarea lor din punct de vedere computațional.* Astfel, există mai multe momente posibile în care numele unui *macro* poate fi expandat la forma lui întreagă (extinsă). Acest lucru poate fi făcut în momentul creării lexiconului, fiecărui cuvânt fiindu-i asociat un DAG în întregime specificat, caz în care lexiconul ar avea dimensiuni mult prea mari. Expandarea definiției bazate pe *macro*-uri a unei intrări lexicale se mai poate face atunci când utilizarea acelei intrări devine efectiv necesară. În practică aceasta pare a fi soluția cea mai adecvată. O a treia alternativă ar putea amâna și mai mult expandarea *macro*-urilor. Astfel, regăsirea unei intrări lexicale ar putea întoarce un DAG care să conțină nume de *macro*-uri neexpandate. Acestea ar urma să fie expandate numai atunci când acel DAG (sau o porțiune relevantă a lui) ar trebui să fie unificat cu un alt DAG.

Un alt aspect care trebuie luat în considerație relativ la lexicon este acela al intrărilor lexicale multiple corespunzând unui același verb, substantiv etc. Intrările multiple nu sunt folosite numai pentru a indica variante sintactice ci corespund, în egală măsură, diferențelor semantice. Aceasta înseamnă că *intrările lexicale trebuie să conțină și* ***informație semantică***. Într-o primă *abordare semantică simplistă*, vom privi sensul lui *mănâncă* din "ea mănâncă" ca fiind o funcție cu un argument - adică ceva care trebuie să se combine cu un singur obiect pentru a forma o propoziție. Pe de altă parte, sensul aceluiași *mănâncă* din "ea mănâncă desertul" este o funcție cu două argumente (semnificând o combinare cu două obiecte). Vom distinge între aceste sensuri utilizând notația **mananca1a** pentru cazul cu un singur argument, **mananca2a** pentru cazul cu două argumente ș.a.m.d.. Caracterul alfabetic final din denumirile acestor sensuri este cel prin intermediul căruia se face distincția între *variantele semantice* cu același număr de argumente ale cuvintelor:

citim3a pentru "noi citim o carte la școală"
citim3b pentru "noi citim la școală o carte"

Pentru moment, vom presupune că **mananca1a**, **mananca2a**, **citim3a**, **citim3b** etc. sunt, pur și simplu, *constante logice* cu care lucrează *limbajul de reprezentare semantică* folosit. În această viziune semantică simplistă vom ignora legăturile logice care ar putea exista între aceste constante, caz în care lexiconul nu trebuie decât să indice care constante corespund căror cuvinte.

În cele ce urmează, vom presupune că DAG-urile corespunzătoare intrărilor lexicale se ramifică inițial în trei ramuri: **sin** (sintaxă), **sem** (semantică) și **mor** (morfologie). Iată, spre exemplu, DAG-ul asociat intrării lexicale "dau":



Anterior, intrările lexicale au luat în considerație numai ramura corespunzătoare sintaxei, un *macro* pentru o asemenea intrare lexicală putând fi acum reprezentat după cum urmează:

Macro sin_Vi :
<sin cat> = V
<sin arg0 cat> = NP
<sin arg0 caz> = nom.

În prezent, dorim ca intrările lexicale să furnizeze și un minimum de informație semantică, ceea ce va duce la scrierea lor în felul următor:

Lexem mănâncă:
sin_Vi
<sem> = mananca1a.
Lexem mănâncă:
sin_Vt
<sem> = mananca2a.

Noile intrări lexicale conțin informație sintactică și semantică, dar sunt lipsite de **informație morfologică**, pe care o vom adăuga în continuare. Vom continua să ne referim la verbe, dar, întrucât morfologia verbului este extrem de complexă în limba română (ceea ce face ca generarea cu calculatorul a formelor verbale, spre exemplu, să fie o operație extrem de delicată), vom exemplifica cu ajutorul verbului englezesc, urmând argumentația din [29] ca,

de altfel, în cazul întregii discuții referitoare la lexicon. În limba engleză un verb poate avea maximum opt forme distincte, dintre care una este întotdeauna *rădăcina*, iar o alta este întotdeauna previzibilă (participiul prezent sau forma de gerunziu, care se termină în **-ing**) dacă rădăcina este cunoscută. Iată cele opt forme ale verbului cel mai neregulat din limba engleză, verbul *to be*:

radacina - be
forma 1 - am
forma 2 - are
forma 3 - is
forma 4 - was
forma 5 - were
forma 6 - been
forma 7 - being

Vom folosi caracteristica **radacina** pentru codificarea rădăcinii verbului și caracteristicile **forma1** până la **forma7** pentru codificarea celorlalte potențiale șapte forme verbale. Un verb englezesc *regulat* cunoaște, în schimb, numai patru forme distincte, toate putând fi formate cu ajutorul rădăcinii. Spre exemplu, verbul regulat *to stamp* (a ștampila) cunoaște următoarele opt forme, dintre care numai patru sunt distincte:

radacina - stamp
forma 1 - stamp
forma 2 - stamp
forma 3 - stamps
forma 4 - stamped
forma 5 - stamped
forma 6 - stamped
forma 7 - stamping

În cele ce urmează, vom adopta un tip de analiză în care fiecare dintre forme se compune dintr-o *tulpină*[4] și un *sufix*. Acestea reprezintă *forma de bază* a cuvântului, de la care se formează toate celelalte (și care se confundă cu rădăcina în cazul verbelor regulate) și respectiv *desinența*, care se adaugă în mod specific fiecărei forme. În acest cadru și urmând tipul de analiză din [29], putem concepe o abreviere, sub forma unui *macro*, pentru *verbele englezești regulate*, după cum urmează:

Macro mor_regV:

[4] Tulpina se confundă uneori cu rădăcina, iar alteori nu. Spre exemplu, în limba română, perfectul simplu "cântai" are ca rădăcină pe *cânt*, iar ca temă a perfectului sau tulpină pe *cânta*. La rândul său, mai mult ca perfectul "cântasem" are tot rădăcina *cânt*, iar ca tulpină pe *cântase*, la care se adaugă desinențe specifice.

<mor forma1 tulpina> = <mor radacina>
<mor forma1 sufix> = ε
<mor forma2 tulpina> = <mor radacina>
<mor forma2 sufix> = ε
<mor forma3 tulpina> = <mor radacina>
<mor forma3 sufix> = ε
<mor forma4 tulpina> = < mor radacina>
<mor forma4 sufix> = ed
<mor forma5 tulpina> = <mor radacina>
<mor forma5 sufix> = ed
<mor forma6 tulpina> = <mor radacina>
<mor forma6 sufix> = ed
<mor forma7 tulpina> = <mor radacina>
<mor forma7 sufix> = ing.

unde ε indică sufixul vid. Interpretarea dorită este aceea conform căreia tulpina şi sufixul se vor combina într-un cuvânt în concordanţă cu regulile ortografice ale limbii engleze, astfel încât, spre exemplu, "love" şi "ing" dau naştere cuvântului "loving" şi nu cuvântului "loveing". Fiind dat *macro*-ul de mai sus, în această interpretare, intrarea lexicală corespunzătoare lui "love" va fi:

Lexem love:
<mor radacina> = love
mor_regV
sin_Vt
<sem> = love2a.

Prima linie este, în mod evident, redundantă, întrucât denumirea intrării lexicale este identică cu forma reprezentând rădăcina. *Prin convenţie*, vom denumi întotdeauna o intrare lexicală în acest mod, ceea ce înseamnă că o intrare lexicală de tipul

Lexem XXX:
<mor radacina> = XXX
YYY
...
ZZZ.

va fi abreviată prin

Lexem XXX:
YYY
...
ZZZ.

Conform acestor convenţii, intrarea lexicală anterioară pentru "love" devine:

Lexem love:
mor_regV
sin_Vt
<sem> = love2a.

Evident, pot fi concepute *macro*-uri şi pentru verbe care nu sunt regulate. Important este modul în care un sistem de procesare a limbajului natural poate folosi cunoştinţele lexicale codificate în lexicon. Un sistem căruia i se prezintă un şir de cuvinte (neanalizate) ca input va trebui cel mai adesea să determine caracteristicile relevante ale fiecărui cuvânt, adică caracteristicile sintactice necesare în procesul de parsing şi caracteristicile semantice necesare în determinarea sensului. Dar, întrucât toate cunoştinţele lexicale oferite de lexicon se referă la rădăcinile cuvintelor, pentru a determina caracteristicile relevante este necesar să determinăm modul în care un cuvânt dat poate fi analizat ca reprezentând o formă a unei rădăcini existente în lexicon. Pentru aceasta, este necesar să putem analiza un cuvânt în termeni de posibile tulpini şi sufixe din care este format.

Odată ce au fost determinate posibilele tulpini şi sufixe pentru un cuvânt dat, un analizor poate, în principiu, să extragă informaţia necesară în procesul de parsing din intrarea lexicală corespunzătoare acelui cuvânt. Ceea ce deranjează cel mai mult în acest moment al analizei este faptul că nu există încă nici o legătură între sintaxă şi morfologie. Un lexicon de tipul celui descris aici ne spune, spre exemplu, că "give" are pe "given" ca valoare a caracteristicii **forma6**, dar nu oferă nici un indiciu asupra contextelor sintactice specifice în care forma **forma6** a verbului ar putea să intervină (spre exemplu, după "have", ca în "have given"). Este necesară, prin urmare, *o modalitate de a obţine caracteristicile sintactice particulare ale anumitor* ***forme flexionare*** *ale cuvintelor plecând de la intrările lexicale*. Acest lucru poate fi realizat prin augmentarea lexiconului cu o definiţie a modurilor posibile în care *o formă flexionară poate fi pusă în legătură cu un lexem*, aşa cum apare acesta în lexicon.

O modalitate de a trata problema relaţiilor dintre un cuvânt şi un lexem este aceea a utilizării unei *clauze a formei cuvântului* (CFC), care enunţă condiţiile în care relaţiile sunt satisfăcute. Iată, spre exemplu, o clauză a formei cuvântului referitoare la clasa verbelor englezeşti aflate la timpul prezent şi persoana a treia, singular, a cărei interpretare este evidentă:

CFC trei_sing:
<cuvant mor forma> = <lexem mor forma3>
<cuvant sin> = <lexem sin>
<cuvant sin cat> = V
<cuvant sin arg0 per> = 3
<cuvant sin arg0 num> = sing
<cuvant sin timp> = prezent
<cuvant sem> = <lexem sem>.

CFC-urile exprimă, prin urmare, relaţiile dintre formele flexionare şi intrările lexicale. Atunci când sunt date un cuvânt şi un posibil lexem de care acesta ar putea fi legat, se caută o CFC care ar putea fi aplicată. Ca şi în aplicarea unei

reguli a gramaticii se încearcă construirea unei extensii a DAG-ului reprezentând cuvântul (care nu va conține decât informație referitoare la tulpină și la sufix) și a DAG-ului reprezentând lexemul (care conține numai informație generală ce se aplică tuturor formelor acestuia) în așa fel încât să fie satisfăcute condițiile stipulate de către CFC. În urma calculării acestor extensii, cuvântul va moșteni proprietăți sintactice și semantice suplimentare din partea lexemului, precum și proprietăți specifice enunțate în cadrul CFC.

În cazul existenței unui ipotetic *analizor* al limbajului, *accesul lexical* ar avea mai multe etape. Cuvintele reprezentând input-ul ar fi, mai întâi, procesate de către un analizor morfologic pentru a obține posibile valori ale tulpinilor și ale sufixelor. Corespunzător fiecărui astfel de rezultat sunt regăsite în lexicon posibile lexeme. Pentru fiecare posibilă descriere parțială a cuvântului și pentru fiecare lexem de care aceasta ar putea fi legată, sunt invocate posibile CFC-uri. Fiecare invocare realizată cu succes are ca rezultat o descriere completă a cuvântului (DAG), care poate fi folosită în etape ulterioare, cum ar fi aceea de analiză sintactică (parsing). Este evident faptul că, în cadrul unui sistem în care există un număr mare de intrări lexicale și de CFC-uri, indexarea structurilor de date lexicale va fi vitală pentru realizarea unei procesări eficiente. (Spre exemplu, lexemele relevante pot fi regăsite în mod eficient pentru a se combina cu descrieri parțiale ale cuvintelor dacă intrările lexicale sunt indexate după posibilele valori ale tulpinilor în diferitele forme. Astfel, intrarea "give" ar putea fi accesată rapid atât pe baza tulpinii "give", cât și a tulpinii "gave". În plus, o CFC ar putea fi indexată după posibilele sufixe care indică relații ș.a.m.d.)

5.4.1. Implementarea lexiconului în Prolog

Implementarea *macro*-urilor și a definițiilor lexemelor se face [29] în contextul sistemului Prolog PATR existent. Exemplificăm, în continuare, cu două definiții de *macro* referitoare la informația sintactică. Predicatul **macro** pune în legătură numele unui *macro* cu DAG-ul pe care acesta îl reprezintă:

```
macro(sin_Vi,L):-
    L:sin:cat = = = v,
    L:sin:arg0:cat = = = np,
    L:sin:arg0:caz = = = nom.

macro(sin_Vt,L):-
    macro(sin_Vi,L),
    L:sin:arg1:cat = = = np,
    L:sin:arg1:caz = = = acc.
```

Dacă se va invoca scopul

```
?- macro(sin_Vi,L).
```

unde L este un DAG existent, atunci scopurile introduse de regula dată de **macro** vor servi la extinderea (instanțierea) DAG-ului existent cu orice perechi de tipul (*caracteristică, valoare*) necesare în satisfacerea definiției.

O modalitate de a defini un *macro* pentru verbe regulate este aceea de a atribui fiecărei forme a lexemului caracteristicile **tulpina** și **sufix** (având semnificația evidentă dată de denumirile lor):

```
L:mor:radacina = = = R,
L:mor:forma3:tulpina = = = R,
L:mor:forma3:sufix = = = s.
```

Într-o limbă ca engleza, unde adăugarea sufixelor sa face conform unor reguli bine stabilite, putem simplifica scrierea prin utilizarea operatorului '+' în împerecherea tulpinilor cu sufixele corespunzătoare. Această utilizare a lui '+' necesită adăugarea unei clauze suplimentare în definirea predicatului **denota**:

```
denota(R+S, R+S):-!.
```

Un *macro* corespunzător morfologiei verbelor englezești regulate este atunci următorul:

```
macro(Vmor_reg,L):-
    L:mor:radacina = = = R,
    L:mor:forma1 = = = R + '',
    L:mor:forma2 = = = R + '',
    L:mor:forma3 = = = R + s,
    L:mor:forma4 = = = R + ed,
    L:mor:forma5 = = = R + ed,
    L:mor:forma6 = = = R + ed,
    L:mor:forma7 = = = R + ing.
```

Aceste *macro*-uri pot fi invocate în *definiții ale lexemelor*. În acest scop vom utiliza operatorul **exem**, definit prin analogie cu operatorii existenți **egula** și **uv**:

```
L exem love:-
    L:mor:radacina = = = love,
    macro(sin_Vt,L),
    macro(Vmor_reg,L),
    L:sem = = = love2a.
```

Se observă că rolul variabilei L nu este unul decorativ, această variabilă desemnând DAG-ul asociat lexemului dat.

La rândul ei, o CFC poate fi privită ca reprezentând o modalitate de a deriva, într-un mod regulat, cuvinte suplimentare pornindu-se de la lexeme. Această interpretare firească sugerează o implementare a acestor clauze sub forma unor clauze suplimentare pentru predicatul **uv**:

```
C uv Forma:-
    L:mor:forma3 = = = Forma,
    L exem _,
    C:sem = = = L:sem,
    C:mor = = = Forma,
    C:sin = = = L:sin,
    C:sin:arg0:per = = = 3,
    C:sin:arg0:num = = = sing,
    C:sin:timp = = = prezent.
```

CAPITOLUL 6

ELEMENTE DE SEMANTICĂ COMPUTAȚIONALĂ

Semantica este o ramură a lingvisticii, dar și a altor științe (filozofie, logică, psihologie), al cărei obiect de studiu este **sensul**. Semantica reprezintă *nivelul la care limba realizează contactul cu lumea reală*. Modul în care poate fi descris înțelesul enunțurilor aparținând limbajului natural a rămas pentru o lungă perioadă de timp relativ neclar. Studiul semanticii începe să se dezvolte după 1970, atunci când o serie de "instrumente" adecvate sunt furnizate în special de către logica matematică și teoria mulțimilor.

În funcție de diversele aspecte ale sensului luate în considerație, putem spune [9] că se delimitează:

- semantica lingvistică;
- semantica aparținând altor științe.

Din cea de-a doua categorie face parte și *semantica logică*, care are o îndelungată tradiție. Ea se bazează [9] pe pozitivismul logic și își propune să aprecieze condițiile de adevăr ale enunțurilor. Bazându-se pe formalizarea logică, ea se mai numește și *semantică formală*. Studiul sensului se face cu ajutorul logicii matematice, care furnizează concepte și o notație simbolică, utile analizei limbajului.

În ceea ce privește analiza limbajului, reamintim faptul că există diferite niveluri la care aceasta se realizează, lucrarea de față concentrându-se asupra nivelului sintactic și a celui semantic. Dacă la nivel sintactic eram interesați de rolul structural al fiecărui cuvânt și al fiecărui grup sintactic, *analiza semantică* se ocupă de sensurile cuvintelor și de modul în care aceste sensuri se combină în cadrul propozițiilor pentru a forma înțelesul unei întregi propoziții. Acesta este studiul *sensului independent de context*, adică a sensului pe care o propoziție îl are indiferent de contextul în care ea este folosită.

În semantică este fundamentală distincția dintre **sens** (înțeles) și **referire**. Astfel, sensul unui cuvânt determină la ce se poate referi acesta. În propoziția

Autoarea a scris o introducere în procesarea limbajului natural.

cuvântul *autoarea* **înseamnă** "persoană care creează o operă literară, artistică, științifică sau publicistică", dar **se referă la** Florentina Hristea. În mod evident, *autoarea* s-ar putea referi și la altcineva - la oricine a scris o carte de același tip și este de sex feminin. Sensul unui cuvânt rămâne însă neschimbat, determinând la ce se poate referi cuvântul respectiv.

Există multe tipuri diferite de referire. Într-o propoziție de tipul

Leul este regele animalelor.

cuvântul *leul* se referă la o întreagă specie, nu la un anume leu. Conceptul de *referire* a fost îndelung studiat, inclusiv de către logicienii medievali (care îl numeau *suppositio*) și ale căror clasificări sunt încă folosite de unii autori.

Multe cuvinte sunt **ambigue**, adică au mai multe înțelesuri. Adesea, aceste sensuri multiple sunt complet distincte, iar într-un context particular vorbitorul trebuie să aleagă sensul adecvat al unui cuvânt, proces numit **dezambiguizare**. În timp ce oamenii rezolvă, cel mai adesea, problemele legate de dezambiguizare cu relativă ușurință, tehnicile computaționale aferente nu sunt încă suficient de perfecționate.

Propoziții întregi pot fi, de asemenea, ambigue, fie deoarece cuvintele care intră în alcătuirea lor sunt ambigue, fie deoarece propoziția are mai multe structuri sintactice posibile. Un exemplu celebru de acest tip îl constituie propoziția

L-am văzut pe băiatul acela cu telescopul.

în care *cu telescopul* se referă fie la *băiatul acela*, fie la *văzut*, comunicând ori care băiat a fost văzut, ori cum am reușit să îl vedem. (Varianta englezească ambiguă a propoziției, citată adesea în literatura de specialitate, este *I saw the boy with the telescope*).

Compunerea semantică este procesul de combinare a sensurilor cuvintelor pentru a forma sensurile diverselor grupuri sintactice sau ale propozițiilor. Astfel, înțelesul propoziției

Vulpile iubesc găinile.

este, în mod clar, o combinație a sensurilor cuvintelor *vulpile*, *iubesc* și *găinile*. Este însă la fel de clar faptul că aceste sensuri nu sunt alăturate într-un mod care să nu țină cont de nici o *structură*, întrucât, dacă ar fi așa, propoziția *Vulpile iubesc găinile* ar însemna același lucru cu *Găinile iubesc vulpile*, interpretare evident incorectă.

O modalitate de a explica compunerea semantică este aceea a reprezentării sensurilor sub forma unor *formule logice*. Iată câteva exemple în acest sens:

iubesc (vulpile, găinile) pentru *Vulpile iubesc găinile.*
$(\forall x)vulpe(x) \rightarrow roșu(x)$ pentru *Toate vulpile sunt roșii.*

Lambda notația, asupra căreia vom reveni, furnizează o modalitate de a codifica formule din care anumite informații lipsesc, ca în exemplele următoare:

(λx) *iubesc (x,găinile)* corespunzător lui *iubesc găinile*
$(\lambda y)(\lambda x)$ *iubesc (x,y)* corespunzător lui *iubesc*

Aceasta creează reprezentări pentru sensuri ale cuvintelor, ale grupurilor sintactice, precum și ale unor întregi propoziții.

Există și expresii ale căror sensuri nu pot fi deduse analizând sensurile părților componente. Astfel de expresii (cum ar fi *a spăla putina* cu sensul de *a fugi*) nu se înscriu în sfera semanticii compoziționale și ele trebuie să fie incluse, în mod explicit, în lexicon.

Pentru *a descrie sensul* diverselor formulări și enunțuri ale limbajului natural este necesară o modalitate precisă de a descrie informația pe care acestea o conțin. Așa cum am mai menționat, instrumentele folosite sunt furnizate în special de logică și de teoria mulțimilor, acestea fiind utile la "traducerea" unui limbaj natural în Prolog, în special atunci când scopul urmărit este de a răspunde interogărilor unor baze de date. O teorie semantică bazată pe astfel de instrumente și de principii se numește *semantică teoretică a modelelor*. Ea se bazează pe **modele**, care sunt niște *baze de cunoștințe* definite cu multă precizie. Vom reveni pe larg asupra acestor noțiuni (§6.4).

6.1. Semantică și forma logică

Există, așa cum s-a menționat deja, diverse teorii ale sensului, care iau în considerație aspecte diferite ale acestuia. Un prim nivel la care se poate analiza noțiunea de sens este acela al *sensului independent de context*. La acest nivel este produsă așa-numita **formă logică**, care poate fi obținută direct din structura sintactică a propoziției. Structura unei propoziții nu reflectă sensul acesteia. *Reprezentarea sensului independent de context al unei propoziții se numește **forma ei logică***. Forma logică codifică posibilele sensuri ale cuvintelor și identifică legăturile semantice dintre cuvinte și grupurile sintactice existente. Deoarece trebuie să fie independentă de context, forma logică nu conține rezultatele nici unei analize care să presupună interpretarea propoziției în context. Apare, prin urmare, necesitatea dezvoltării unui *limbaj formal* în care să fie posibilă specificarea sensului fără a face referiri la însuși limbajul natural. Chiar și în condițiile existenței unui asemenea limbaj, definirea noțiunii de sens al unei întregi propoziții rămâne însă o sarcină dificilă.

Problema care se pune este aceea a definirii noțiunii de sens independent de context[1] al propoziției. Cu alte cuvinte, întrebarea căreia i se caută răspuns

---

[1] Utilizarea limbii *în context* face obiectul de studiu al **pragmaticii**. Granița dintre semantică și pragmatică este neclară și diverși autori folosesc acești termeni în mod diferit. În general, putem spune că pragmatica include acele aspecte ale comunicării care pătrund dincolo de condițiile exacte de adevăr ale fiecărei propoziții. Astfel, în timp ce sintaxa și semantica studiază propozițiile, pragmatica (care nu face obiectul acestei lucrări) studiază *actele vorbirii* și situațiile în care este folosită limba. Spre exemplu, atunci când un vorbitor îi adresează unui posibil interlocutor întrebarea *Poți*

este dacă există un nivel la care o propoziție are un unic înțeles, ea putând însă să fie utilizată în scopuri diferite. Aceasta este o problemă extrem de complexă, dar există numeroase avantaje ale unei asemenea abordări, care o fac demnă de luat în studiu. Un prim argument în acest sens îl constituie faptul că, dacă o atare separare poate fi făcută, aceasta înseamnă că sensul propozițiilor poate fi studiat în detaliu, fără a fi luate în seamă complicațiile generate de utilizările acestora. În particular, dacă propozițiile nu ar avea nici un sens independent de context, atunci studiul limbii nu ar putea fi separat de studiul gândirii umane în general și de context. S-a constatat însă că există multe exemple de constrângeri bazate pe sensuri ale cuvintelor care apar ca fiind independente de context.

Definiția 6.1

Reprezentarea sensului independent de context este numită **formă logică**. Operația care asociază unei propoziții forma sa logică este numită **interpretare semantică**. Procesul de asociere a formei logice limbajului final de reprezentare a cunoștințelor (KR - de la "knowledge representation") se numește **interpretare contextuală**.

Pentru nevoile prezente vom presupune că limbajul KR îl reprezintă calculul predicatelor de ordinul întâi (FOPC). Această presupunere fiind făcută,

---

închide ușa? și acesta nu face decât să răspundă *Da*, înseamnă că cel din urmă nu a înțeles sensul formulării, care în mod implicit reprezenta o rugăminte. Această nuanță îi scapă unui vorbitor care nu cunoaște pragmatica limbii române.

Un concept important în pragmatică este *implicatura* (tip de deducție pragmatică). Prin implicatura realizată de o propoziție înțelegem acea informație care nu face parte din sensul propoziției, dar care este oricum dedusă din aceasta. Astfel, propoziția *Poți închide ușa?* la care ne-am referit anterior este și o rugăminte politicoasă, nu numai o întrebare. Spre deosebire de sensul unei propoziții, implicarea realizată de aceasta poate fi anulată.

O altă problemă de care se ocupă pragmatica este aceea a *presupunerii* ori *supoziției*. Presupunerea realizată de o formulare constă în acele fapte care trebuie să fie adevărate pentru ca respectiva formulare să fie ori adevărată, ori falsă. Spre exemplu, atât formularea *Președintele țării este reales din patru în patru ani*, cât și formularea *Președintele țării nu este reales din patru în patru ani*, presupune că țara în cauză este o republică. Dacă ea nu reprezintă o republică, ci o monarhie constituțională, atunci propozițiile menționate nu sunt nici adevărate, nici false. Presupunerea realizată de o formulare poate fi și ea anulată (*Președintele țării nu este reales din patru în patru ani întrucât țara este o monarhie constituțională*), spre deosebire de sensul de bază al unei propoziții, care nu poate fi niciodată anulat.

Un obiect de studiu important al pragmaticii îl constituie *discursul* (modul în care propozițiile sunt legate între ele pentru a transmite informație). Interesează în mod special structura discursului, care este de o importanță crucială mai ales atunci când se dorește ca un calculator să genereze text aparținând limbajului natural. O problemă esențială și insuficient explorată o constituie organizarea discursului de către un calculator în același mod în care o fac ființele umane.

se pune întrebarea care este statutul formei logice. În unele abordări forma logică este definită ca reprezentând înțelesul ad litteram al propoziției, iar limbajul formei logice coincide cu limbajul KR. Pe termen lung aceasta ar presupune însă ca reprezentarea cunoștințelor să fie mult mai complexă decât este ea la ora actuală în diverse sisteme ale inteligenței artificiale. Multe aspecte ale limbii, cum ar fi interpretarea timpurilor verbale sau determinarea domeniilor cuantificatorilor, depind și de context și, prin urmare, nu pot fi unic determinate la nivelul formei logice. Desigur, toate aceste aspecte ar putea fi tratate, la nivelul formei logice, ca fiind ambigue, ceea ce este însă nepractic deoarece ar conduce la existența unui mare număr de forme logice posibile corespunzător fiecărei propoziții. Recomandăm, prin urmare, acele abordări în care limbajul formei logice nu face parte din limbajul KR.

O abordare diferită și promițătoare în ceea ce privește statutul limbajului formei logice este cea dezvoltată în lingvistica ultimelor decenii, abordare care folosește noțiunea de **situație** ca mulțime particulară de circumstanțe ale lumii înconjurătoare și care a generat o întreagă teorie lingvistică, numită *semantica situațiilor*. Din punct de vedere formal, o situație poate fi privită ca reprezentând o mulțime de obiecte și de relații între aceste obiecte. Spre exemplu, o situație extrem de simplă poate fi aceea constând din două obiecte, o minge notată B0005 și o persoană notată P86 și incluzând o unică relație care spune că persoana este posesoarea mingii. Această situație poate fi codificată sub forma mulțimii

{(MINGE B0005), (PERSOANA P86), (POSEDA P86 B0005)}

Limba creează diverse tipuri speciale de situații, în funcție de informația care este transmisă. Se presupune că, în orice conversație sau text, există o situație a discursului care memorează informația transmisă până la momentul curent. O nouă propoziție este întotdeauna interpretată cu privire la această situație și produce o situație nouă, care include și informația transmisă de noua propoziție. În această viziune, forma logică este o *funcție* care realizează corespondența dintre situația discursului în cadrul căreia un enunț a fost formulat și o nouă situație a discursului, rezultată în urma apariției enunțului respectiv. Să presupunem, spre exemplu, că situația codificată anterior a fost creată de niște propoziții (anterioare) care descriau mingea și pe proprietarul acesteia. Enunțul *Mingea este roșie* poate produce o nouă situație, care constă din situația veche, căreia i se adaugă faptul (nou) că mingea este roșie, adică acela că B0005 are proprietatea ROSU:

{(MINGE B0005), (PERSOANA P86), (POSEDA P86 B0005), (ROSU B0005)}.

Chiar dacă cea mai mare parte a limbajului este dependentă de context, există totuși suficientă structură semantică a acestuia care se comportă independent de context și care poate fi deci folosită în procesul de interpretare semantică pentru a produce forma logică. Majoritatea acestor cunoștințe semantice constau din tipul de informație care poate fi oferită de un dicționar -

proprietățile semantice de bază ale cuvintelor (i.e. dacă acestea se referă la relații, obiecte etc.), ce sensuri diferite sunt posibile pentru fiecare cuvânt, ce sensuri pot fi combinate pentru a forma o structură semantică mai largă etc.

6.1.1. Limbajul formei logice

Limbajul formei logice cel mai des utilizat seamănă cu FOPC, deși există multe alte forme echivalente de reprezentare, cum ar fi cele bazate pe rețele, care, în esență, au la bază aceleași principii.

În accepția prezentă *sensurile* cuvintelor vor servi ca **atomi** sau **constante** ale reprezentării. Aceste constante pot fi clasificate în funcție de natura lucrurilor pe care le desemnează. Spre exemplu, acele constante care descriu obiecte ale lumii înconjurătoare, inclusiv obiecte abstracte cum ar fi evenimentele și situațiile, se numesc **termeni**. Constantele care descriu relații și proprietăți se numesc **predicate**. O propoziție a limbajului este formată dintr-un predicat urmat de un număr adecvat de termeni care reprezintă argumentele acestuia. Spre exemplu, propoziția din limbajul formei logice corespunzătoare propoziției *Andrei este student* din limbajul natural se construiește din termenul ANDREI1 și din constanta predicat STUDENT1 și este următoarea:

(ANDREI1 STUDENT1)

Predicatele care admit un unic argument sunt **predicate unare** sau **proprietăți**; acelea care admit două argumente, cum ar fi IUBESTE1, se numesc **predicate binare** etc. Propoziția limbajului formei logice corespunzătoare enunțului *Ion iubește pe Maria* al limbajului natural utilizează un predicat binar și este următoarea:

(IUBESTE1 ION1 MARIA1)

Se observă că diferitele categorii de cuvinte corespund unor tipuri diferite de constante ale formei logice. Numele proprii, spre exemplu, cum ar fi *Maria*, sunt reprezentate prin termeni. Substantivelor comune, cum ar fi *student*, le corespund predicate unare, în timp ce verbelor le corespund predicate *n*-are, unde *n* se stabilește în funcție de posibilitățile de subcategorizare ale fiecărui verb în parte.

Propoziții mult mai complexe sunt construite utilizând o nouă clasă de constante numite **operatori logici**. Spre exemplu, operatorul NON permite construirea unei propoziții care afirmă că o anumită propoziție nu este adevărată. Propoziția corespunzătoare lui *Ion nu o iubește pe Maria* este, în limbajul formei logice:

(NON(IUBESTE1 ION1 MARIA1))

Majoritatea limbilor naturale conțin operatori care combină două sau mai multe propoziții pentru a forma o propoziție complexă. FOPC conține operatori

cum ar fi disjuncția (V) sau conjuncția (&), în timp ce limba engleză, spre exemplu, conține operatori similari ca *or, and, if, only if* etc. Operatorii corespunzători din limba română sunt *sau, și, dacă, numai dacă*. Cuvintele cu rol de conectori ale limbajului natural adesea presupun relații mult mai complexe între propoziții. Spre exemplu, conjuncția *și* corespunde operatorului logic &, dar implică, adesea, o succedare temporală, ca în propoziția *Am ajuns la facultate și am dat examenul*, în care, anterior acțiunii de a susține examenul, vorbitorul a trebuit să ajungă într-un anumit loc (la facultate). Forma generală a unei asemenea propoziții este

(conector propoziție propoziție).

Spre exemplu, forma logică a propoziției *Ion iubește pe Maria sau Ion iubește pe Ana* este următoarea:

(SAU1(IUBESTE1 ION1 MARIA1)(IUBESTE1 ION1 ANA1))

Limbajul formei logice descris aici va permite existența atât a operatorilor corespunzând unor sensuri ale cuvintelor limbajului natural, precum și a operatorilor cum ar fi &, adică a operatorilor preluați direct din FOPC. Acești din urmă operatori logici vor fi utilizați pentru a uni propoziții care nu sunt legate în mod explicit.

Cu ajutorul propozițiilor limbajului formei logice definite până în prezent nu putem defini decât propoziții ale limbajului natural constând din forme verbale simple și substantive proprii. Pentru a exprima propoziții mult mai complexe este necesară definirea unor structuri semantice suplimentare. O asemenea structură extrem de importantă este **cuantificatorul**. In FOPC există numai doi cuantificatori: $\forall$ și $\exists$. Majoritatea limbilor naturale, cum ar fi engleza, conțin o gamă mult mai largă de cuantificatori: *all, some, most, many, a few* și alții. (Corespondenții acestor cuantificatori în limba română sunt *toți, unii, majoritatea, mulți, câțiva.*) Pentru a permite existența acestor cuantificatori, sunt introduse, ca și în logica de ordinul întâi, variabile, dar cu o diferențiere importantă. După cum se știe, în logica de ordinul întâi o variabilă își menține semnificația numai în cadrul domeniului cuantificatorului. Astfel, două instanțieri ale aceleiași variabile x care intervin în două formule diferite - de pildă în formulele $\exists x.P(x)$ și $\exists x.Q(x)$ - sunt tratate ca reprezentând variabile complet diferite, neexistând nici o legătură între ele. Limbajul natural se comportă în mod fundamental diferit. Pentru a ilustra acest fapt, vom lua în considerație cele două propoziții *Un student intră în sala de examen. El se îndreaptă spre catedră.* Prima propoziție introduce în discuție un nou obiect, și anume un oarecare student. La studentul introdus din punct de vedere existențial în această primă propoziție se face însă referire și în propoziția a doua, prin intermediul pronumelui *El*. Prin urmare, variabilele își continuă existența după ce au fost introduse. Pentru a permite acest lucru, ori de câte ori este introdusă o variabilă a discursului, acesteia i se atribuie un nume unic (care

nu a mai fost folosit până atunci). O propoziție ulterioară poate apoi să se refere din nou la acest termen.

Cuantificatorii limbajului natural au ranguri restrânse și, prin urmare, sunt mai complecși decât cei ai FOPC. Astfel, în FOPC o formulă de forma $\forall x.P(x)$ este adevărată dacă și numai dacă $P(x)$ este adevărat pentru toate obiectele din domeniu (i.e. x poate fi oricare termen al limbajului). Astfel de enunțuri sunt rare în limbajul natural. Enunțurile frecvente în limbajul natural sunt de tipul *majoritatea studenților învață* și reclamă existența unor așa-numiți **cuantificatori generalizați**. Acești cuantificatori sunt utilizați în formulări având structura generală

(variabilă cuantificator: propoziție-restricție corp-propoziție)

Spre exemplu, propoziției *Majoritatea studenților învață* îi corespunde forma logică

(MAJORITATEA1 **d1**:(STUDENT1 **d1**)(ÎNVAȚĂ1 **d1**))

Aceasta înseamnă că majoritatea obiectelor **d1** care satisfac (STUDENT1 **d1**) satisfac, de asemenea, (ÎNVAȚĂ1 **d1**). Acesta este un sens fundamental diferit de cel al formei logice

(MAJORITATEA **d2**: (ÎNVAȚĂ1 **d2**)(STUDENT1 **d2**))

care transcrie propoziția *Majoritatea celor care învață sunt studenți.*

O clasă foarte importantă de cuantificatori generalizați corespunde *articolelor* hotărât și nehotărât. Astfel, afirmația *Studentul învață* are forma logică

(ART_L **x**:(STUDENT1 **x**)(INVAȚĂ1 **x**))

în timp ce *Un student învață* se reprezintă sub forma următoare:

(UN **x**:(STUDENT1 **x**)(INVAȚĂ1 **x**))

În limba engleză există cuantificatori generalizați corespunzător articolelor *the* și *a*. Propoziției *The dog barks* îi corespunde următoarea formă logică:

(THE **x**:(DOG1 **x**)(BARKS1 **x**))

Remarcăm faptul că fiecare dintre aceste propoziții presupune utilizarea contextului pentru înțelegerea exactă a sensului ei. Referindu-ne la primele două propoziții românești vom nota faptul că formele lor logice sunt corecte numai dacă există un student unic determinat de context și dacă acel student învață. Evident, în realitatea înconjurătoare există mulți studenți, astfel încât folosirea contextului pentru a-l identifica pe cel corect este crucială în înțelegerea propoziției. Astfel de aspecte, care țin mai degrabă de domeniul pragmaticii, nu fac obiectul discuției în momentul de față. În prezent, vom reține doar existența acestei importante clase de cuantificatori generalizați.

Grupuri nominale complexe vor da naştere unor restricţii, la rândul lor, mult mai complexe. Spre exemplu, propoziţia *Un student conştiincios învaţă* necesită introducerea unei restricţii de tip conjuncţie şi are următoarea formă logică:

(UN **x**:(&(STUDENT1 **x**)(CONŞTIINCIOS **x**))(ÎNVAŢĂ1 **x**))

Această formă logică va exprima adevărul numai dacă există un **x** unic din punctul de vedere al contextului astfel încât (&(STUDENT1 **x**) (CONŞTIINCIOS **x**)) este adevărat şi, în plus, acest **x** învaţă.

O altă construcţie trebuie introdusă pentru a trata formele de plural care apar în propoziţii precum *Studenţii învaţă*. Pentru aceasta se defineşte un nou tip de constantă, sub forma unui operator unar, constantă numită **operator predicat**. Aceasta va avea ca argument un predicat şi va produce un nou predicat. Pentru construcţia formelor de plural se introduce operatorul predicat PLUR. Astfel, dacă STUDENT1 este un predicat care este adevărat pentru orice student, atunci (PLUR STUDENT1) este un predicat adevărat pentru orice mulţime de studenţi. Operatorul predicat PLUR este util în special în cazul verbelor care au la unele persoane aceeaşi formă la singular şi la plural, ca în exemplul menţionat. Reprezentarea înţelesului propoziţiei *Studenţii învaţă* este următoarea:

(ART_I **x**:((PLUR STUDENT1)**x**)(ÎNVAŢĂ1 **x**))

În cazul folosirii articolului nehotărât, forma logică a propoziţiei *Nişte studenţi învaţă* este

(NIŞTE **x**:((PLUR STUDENT1)**x**)(ÎNVAŢĂ1 **x**))

Remarcăm aici atât folosirea *cuantificatorului generalizat* (corespunzând articolului nehotărât) NIŞTE, cât şi marcarea formei de plural prin utilizarea *operatorului predicat* PLUR. Un exemplu pentru limba engleză este cel al reprezentării sensului propoziţiei *The dogs bark* (câinii latră) după cum urmează:

(THE **x**:((PLUR DOG1)**x**)(BARKS1 **x**))

Grupurile nominale al căror substantiv este la plural introduc o nouă formă de ambiguitate. Să remarcăm, în acest sens, faptul că înţelesul firesc al propoziţiei *studenţii învaţă* este acela că există o mulţime (un grup) specificată de studenţi şi că fiecare dintre membrii acesteia învaţă. Această interpretare este urmarea unei aşa-numite ***citiri distributive***, în care predicatul ÎNVAŢĂ1 se distribuie fiecărui element al mulţimii. Prin contrast, vom lua în consideraţie propoziţia *Studenţii s-au întâlnit la facultate*. În acest caz nu are sens interpretarea conform căreia fiecare student, în mod individual, s-a întâlnit. Afirmaţia este mai degrabă adevărată pentru întreaga mulţime de studenţi. Aceasta se numeşte ***citire colectivă***. Există însă propoziţii care permit ambele

interpretări şi, prin urmare, sunt ambigue. O astfel de propoziţie este *Doi studenţi au cumpărat un video*, care poate să însemne fie că cei doi studenţi au cumpărat fiecare câte un video (într-o citire distributivă), fie că cei doi au cumpărat un video împreună (într-o citire colectivă).

O ultimă construcţie pe care o vom introduce este cea a aşa-numiţilor **operatori modali**, care par similari operatorilor logici, înregistrând însă unele deosebiri importante. În mod concret, termenii din domeniul unui operator modal pot avea o interpretare diferită de cea obişnuită. Aceasta afectează concluziile care se pot trage dintr-o propoziţie. Spre exemplu, să presupunem că Andrei are două nume şi că există persoane care îl cunosc sub numele de Tudor. Există atunci două sensuri care sunt egale, şi anume: ANDREI1=TUDOR22. În cadrul unei propoziţii relativ simple nu va avea importanţă care dintre aceste două constante este folosită: dacă (STUDIOS TUDOR22) este adevărat, atunci şi (STUDIOS ANDREI1) este adevărat şi reciproc. Situaţia este aceeaşi în cadrul unor propoziţii mai complexe, dar formate prin utilizarea operatorilor logici. Astfel, dacă (SAU (OCUPAT ANDREI1)(LIBER ANDREI1)) este adevărat, atunci (SAU (OCUPAT TUDOR22)(LIBER TUDOR22)) este în egală măsură adevărat şi reciproc. Vom remarca însă faptul că aceleaşi propoziţii nu sunt interschimbabile în cadrul domeniului unui operator modal cum ar fi CREDE1 (corespunzător verbului *a crede*). Spre exemplu, din faptul că Ioana crede că Andrei este conştiincios, ceea ce se transcrie în forma logică

(CREDE1 IOANA1 (CONŞTIINCIOS ANDREI1))

nu rezultă neapărat că Ioana crede şi că Tudor este conştiincios, adică

(CREDE1 IOANA1 (CONŞTIINCIOS TUDOR22))

întrucât s-ar putea ca Ioana să nu ştie că Andrei şi Tudor sunt una şi aceeaşi persoană. Avem aici o situaţie în care doi terneni sunt egali, dar nu sunt interschimbabili. Ca urmare, nici propoziţiile corespunzătoare nu sunt interschimbabile. Aceste propoziţii se referă la ceea ce crede Ioana, nu la faptul că Tudor şi Andrei sunt una şi aceeaşi persoană.

Prin urmare, termenii egali nu pot fi substituiţi în mod liber atunci când intervin în interiorul domeniului unui operator modal. Această restricţie este cel mai adesea numită ***eşecul substituţiei în contexte modale***. Reamintim faptul că operatorii modali sunt necesari pentru a reprezenta sensul unor verbe cum ar fi *a crede, a vrea, a dori, a suspecta* etc.

O categorie importantă de operatori modali care intervin în studiul limbajului natural sunt aşa-numiţii *operatori de timp*, pe care îi vom desemna prin cuvintele-cheie TRECUT, PREZENT şi VIITOR. Exemplele de până acum au ignorat în totalitate efectul pe care îl poate produce timpul verbal. Utilizând aceşti noi operatori se poate însă reprezenta diferenţa de sens dinte *Ion o vede pe Ana, Ion a văzut-o pe Ana* şi *Ion o va vedea pe Ana*, propoziţii cărora le corespund următoarele forme logice:

(PREZENT (VEDE1 ION1 ANA1))
(TRECUT (VEDE1 ION1 ANA1))
(VIITOR (VEDE1 ION1 ANA1))

Este evident faptul că aceştia sunt operatori modali, întrucât ei prezintă proprietatea de eşec al substituţiei. Pentru a verifica acest fapt, vom lua în consideraţie următorul exemplu: fie operatorul TRECUT şi două constante ANDREI1 şi STUDENT1, care sunt acum egale, indicând faptul că Andrei este în prezent student. Dar în trecut Andrei nu devenise încă student (el tocmai a intrat la facultate), prin urmare ANDREI1 nu era egal cu STUDENT1. Luând acest fapt în consideraţie, precum şi pe acela că, în trecut, Andrei a citit cursul de procesarea limbajului natural i.e.

(TRECUT (CITEŞTE1 ANDREI1 CURS1))

nu se poate trage concluzia că studentul a citit cursul în trecut i.e.

(TRECUT (CITEŞTE1 STUDENT1 CURS1))

deoarece în trecut Andrei nu devenise încă student. Prin urmare, el a citit cursul, dar nu în calitate de student, iar operatorul TRECUT verifică eşecul substituţiei.

Prin aceasta specificarea limbajului de bază al formei logice a devenit completă. Diverse extensii posibile ale limbajului de bază prezentat îl fac pe acesta să poată exprima diferite fenomene semantice, cum ar fi ambiguitatea.

Cea mai bună prezentare a limbajului formei logice este cea realizată în [4], pe care o urmăm şi aici. Pentru *limbajul de bază* al formei logice vezi şi Anexa 1.

Intenţia noastră a fost aceea de a prezenta *o reprezentare semantică independentă de context* numită ***formă logică***. O astfel de reprezentare este necesară pentru

- a simplifica procesul computaţional prin care se determină structuri semantice pornindu-se de la structura sintactică;
- a modulariza procesarea propoziţiilor prin separarea şi înlăturarea efectelor contextuale.

În concluzie, putem spune că *limbajul formei logice* foloseşte multe concepte provenind din FOPC, inclusiv termeni, predicate, propoziţii şi operatori logici. Extensiile cele mai semnificative ale FOPC de bază care au fost operate aici sunt cele constând din introducerea următoarelor tipuri de cuantificatori şi de operatori:

- *cuantificatori generalizați*[2], care indică o legătură între două mulțimi și corespund unor cuvinte cum ar fi *fiecare, unii, câțiva, majoritatea* etc.;
- *operatori modali*, care identifică diferite modalități de a lua în considerație propozițiile și corespund unor cuvinte cum ar fi *crede* sau *speră*; o clasă importantă de operatori modali o constituie operatorii de timp;
- *operatori predicat*, care realizează corespondența dintre un predicat existent și un nou predicat, asemenea operatorului pentru formele de plural, operator care ia în considerație un predicat ce descrie indivizi având o proprietate P și produce un predicat ce descrie mulțimi de indivizi, fiecare având proprietatea P.

Limbajul formei logice, așa cum a fost el definit până în prezent, comportă câteva aspecte extrem de importante, printre care amintim:

- poate codifica într-o manieră eficientă multe forme de *ambiguitate* (prin permiterea inserării sensurilor alternative oriunde este admis un singur sens);
- permite reprezentarea unor *complemente circumstanțiale* suplimentare fără a fi necesară introducerea unor predicate distincte pentru fiecare combinație posibilă a argumentelor față de verb;
- permite reprezentări bazate pe *roluri tematice*; chiar dacă rolurile pot fi extrem de diferite de la un sistem la altul, acest mod de reprezentare este unul uzual în sistemele concepute pentru prelucrarea limbajului natural.

Toate aceste aspecte vor fi investigate, succint, în cele ce urmează, analiza lor determinând unele nuanțări în ceea ce privește specificarea limbajului formei logice.

6.1.2. Codificarea ambiguității în forma logică

Limbajul formei logice poate codifica într-o manieră eficientă multe forme de ambiguitate, așa cum se va vedea și în cele ce urmează.

Prin **ambiguitate** se înțelege posibilitatea de a da două sau mai multe interpretări unei construcții sau unui component al ei, ca o consecință semantică a fenomenelor de omonimie[3] și polisemie[4]. În funcție de tipul de omonimie

---

[2] Pentru cuantificatorii existențial și universal standard există formule în FOPC-ul standard echivalente cu formele cuantificatorilor generalizați. Celorlalți cuantificatori generalizați nu le corespund însă forme echivalente în FOPC-ul standard.

[3] *Omonimia* este o relație dintre două sau mai multe cuvinte, morfeme și construcții care au aceeași formă și sensuri diferite.

[4] *Polisemia* este capacitatea majorității cuvintelor din limbile naturale de a avea mai multe sensuri.

prezent în construcție, ambiguitatea poate fi lexicală, morfologică, sintactică, pragmatică. Dacă apare ca o consecință a polisemiei, ambiguitatea este lexicală.

Ambiguitatea constituie o problemă serioasă în timpul procesului de *interpretare semantică*. Un *cuvânt* se definește ca fiind *ambiguu* din punct de vedere semantic dacă i se pot atribui mai multe sensuri, caz în care, așa cum s-a menționat deja, el generează *ambiguitate lexicală*.

Un aspect important care trebuie luat în considerație atunci când se discută sensul independent de context este cel al ocurenței unor restricții care iau naștere între sensurile cuvintelor. Adesea sensul corect al unui cuvânt poate fi identificat pe baza înțelesului și a structurii restului propoziției. Anumite sensuri ale aceluiași verb, spre exemplu, pot interveni numai atunci când acesta este folosit ca verb tranzitiv, în timp ce altele intervin numai atunci când este folosit ca verb intranzitiv. Una dintre cele mai importante sarcini ale interpretării semantice este să folosească astfel de restricții pentru a reduce numărul de sensuri posibile ale fiecărui cuvânt și deci ambiguitatea lexicală.

Pe lângă acest tip de ambiguitate, la nivel semantic există considerabil de multă *ambiguitate structurală*. Propoziția *Elevi și studenți conștiincioși învață limbajul Prolog* este, spre exemplu, ambiguă în ceea ce privește însușirea elevilor și studenților de a fi conștiincioși. (Mai exact, nu se știe dacă limbajul Prolog este învățat atât de elevi conștiincioși cât și de studenți conștiincioși sau numai de studenți conștiincioși și de elevi de orice fel.) Deși această formă de ambiguitate are și consecințe semantice, rădăcinile sale se află în structura sintactică, care nu poate decide dacă conjuncția folosită se referă la două grupuri nominale (Elevi) și (studenți conștiincioși) sau la unicul grup nominal ((Elevi și studenți) conștiincioși).

Este un fapt bine stabilit acela că reprezentărilor făcute în scop computațional li se impune să poată trata problemele legate de ambiguitate. O propoziție tipică va avea mai multe structuri sintactice posibile, fiecare dintre acestea putând, de obicei, să genereze mai multe forme logice. În plus, cuvintele propoziției ar putea avea sensuri multiple. Toate acestea reprezintă surse de ambiguitate. Tocmai de aceea nu suntem interesați de simpla enumerare a formelor logice posibile, ci de indicarea unei forme logice care să codifice ambiguitatea. Mulți cercetători privesc această codificare a ambiguității ca pe un nivel separat al reprezentării, diferit de forma logică și o numesc **formă cvasilogică**.

Pentru a împiedica o veritabilă explozie de forme logice, putem folosi o tehnică similară cu cea utilizată în sintaxă pentru manevrarea valorilor multiple ale caracteristicilor. Mai exact, așa cum am menționat deja, *oriunde este admis un sens atomic se va folosi o mulțime de sensuri atomice posibile*. Aceasta este abordarea care își propune să trateze principala sursă de ambiguitate în forma logică, și anume existența, în cazul multor cuvinte, a **sensurilor multiple**. Spre exemplu, cuvântul *banca* are cel puțin două sensuri: BANCA1, obiectul pe care ne putem așeza și BANCA2, instituția unde ne putem depune economiile. Din

această cauză, propoziția *Ana privea o bancă* este ambiguă atunci când este scoasă din context. O unică formă logică poate însă reprezenta ambele posibilități, după cum urmează:

(O **b1**: ({BANCA1 BANCA2}**b1**)(TRECUT(PRIVEȘTE1 ANA1 **b1**)))

Aceasta reprezintă abrevierea următoarelor două forme logice posibile:

(O **b1**: (BANCA1 **b1**) (TRECUT(PRIVEȘTE1 ANA1 **b1**)))

și

(O **b1**: (BANCA2 **b1**) (TRECUT(PRIVEȘTE1 ANA1 **b1**)))

Unul dintre cele mai complexe tipuri de ambiguitate în forma logică provine din **domeniul relativ de acțiune al cuantificatorilor și operatorilor**. Aceasta este o formă de *ambiguitate structurală pur semantică*, care are la bază o unică structură sintactică. Astfel, o propoziție precum *Fiecare student stimează un profesor* este ambiguă datorită existenței a două tipuri de citiri diferite (adică a două interpretări diferite), datorate modului de a defini domeniul cuantificatorilor. În cazul concret al acestei propoziții nu este clar dacă există un singur profesor pe care îl stimează toți studenții ori dacă fiecare student stimează un profesor diferit. Structura sintactică a propoziției este aceeași în ambele cazuri, diferența constând în modul în care li se atribuie domenii cuantificatorilor. Cele două interpretări ale propoziției menționate anterior corespund în mare următoarelor două formulări din FOPC:

$\exists p.\ Profesor(p)\ \&\ \forall s.\ Student(s) \supset Stimează(s,p)$
$\forall s.\ Student(s) \supset \exists p.\ Profesor(p)\ \&\ Stimează(s,p)$

Prin urmare, propoziția dată are o unică structură sintactică, dar structura ei semantică este ambiguă. Nu există nici o metodă independentă de context pentru a rezolva această ambiguitate, astfel încât ea trebuie reprezentată în forma logică finală a propoziției. În loc de a enumera toate domeniile posibile - ceea ce ar duce la o creștere exponențială a interpretărilor bazată pe numărul construcțiilor de domenii - vom introduce o abreviere în limbajul formei logice, abreviere care are rolul de a strânge laolaltă mai multe interpretări posibile. Mai exact, forma logică abreviată nu va conține nici un fel de informație relativ la domeniu. Construcții specifice (cum ar fi cuantificatorii generalizați) vor fi tratate din punct de vedere sintactic ca termeni și vor apărea în poziția indicată de structura sintactică a propoziției. Astfel de construcții sunt marcate prin paranteze unghiulare pentru a indica abrevierea referitoare la domenii. Spre exemplu, formele logice posibile ale propoziției *Fiecare student stimează un profesor* sunt prinse în unica formă logică ambiguă

(STIMEAZA1<FIECARE **s1**(STUDENT1 **s1**)> <UN **p1**(PROFESOR1 **p1**)>)

Aceasta abreviază ambiguitatea dintre următoarele două forme logice:

(FIECARE **s1**:(STUDENT1 **s1**)(UN **p1**:(PROFESOR1 **p1**)(STIMEAZĂ1 **s1** **p1**)))

și

(UN **p1**:(PROFESOR1 **p1**)(FIECARE **s1**:(STUDENT1 **s1**)(STIMEAZĂ1 **s1** **p1**)))

Chiar dacă economia realizată în acest exemplu nu este una semnificativă, trebuie avut în vedere faptul că o propoziție în care sunt posibile, de pildă, cinci construcții ale domeniului ar avea 120 (5 factorial) forme logice. Convenția de abreviere adoptată permite comprimarea tuturor acestora într-o unică reprezentare.

În plus, dacă restricția într-un cuantificator generalizat este o propoziție care utilizează un singur predicat unar, se poate folosi o abreviere suplimentară, care are rolul de a elimina variabila. Spre exemplu, forma

<FIECARE **s1** (STUDENT **s1**)>

va fi adesea abreviată astfel:

<FIECARE **s1** STUDENT>

Multe construcții din limbajul natural sunt senzitive la domenii, inclusiv toți cuantificatorii generalizați. Spre exemplu, în cazul propoziției *În fiecare facultate decanul este amabil*, în aproape orice context *decanul* intră în domeniul lui *fiecare facultate* (sensul fiind acela că, în cadrul fiecărei facultăți, există un alt decan).

Ca o ultimă observație referitoare la reducerea numărului de forme logice posibile, ne vom referi la numele proprii și la pronume. În ceea ce privește numele proprii, în accepția de până acum, s-a presupus că fiecare dintre acestea identifică un sens care denotă un obiect al domeniului. Aceasta a fost o abordare utilă în introducerea ideilor de bază referitoare la forma logică. Numele proprii trebuie însă interpretate în context, iar un nume ca *Ana* se va referi la persoane diferite în diverse situații. Noua abordare la care ne referim aici va utiliza o variabilă a discursului care are proprietatea de a avea numele specificat. Această construcție este introdusă ca o funcție specială de forma

(NUME <variabilă> <nume>)

care are ca efect producerea în contextul curent a unui obiect cu numele respectiv. Astfel, forma logică a propoziției *Andrei alerga* devine

(<TRECUT ALERGA1>(NUME **a1** "ANDREI"))

Argumente similare pot fi invocate în cazul pronumelor, care vor fi tratate prin utilizarea funcției speciale

(PRO <variabilă> <propoziție>)

Spre exemplu, forma cvasilogică a propoziției *Fiecare student îl stima* este:

(<TRECUT STIM1> <FIECARE **s1** STUDENT1>(PRO **s2** (ÎL1 **s2**)))

Aici ÎL1 reprezintă sensul pentru *îl* și *pe el*, iar din punct de vedere formal este un predicat adevărat pentru acele obiecte care satisfac restricțiile (în acest caz de a fi în viață și de sex masculin) impuse oricărui antecedent. Ca și în cazul cuantificatorilor generalizați, atunci când restricția este un predicat unar, formele corespunzătoare pronumelor sunt adesea abreviate. Spre exemplu, forma logică anterioară corespunzătoare pronumelui va fi, de obicei, scrisă astfel: (PRO **s2** ÎL1).

Construcțiile descrise aici au rolul de a reduce în mod dramatic numărul formelor logice care trebuie determinate pentru o propoziție dată, fără a avea însă pretenția de a putea surprinde toate formele de ambiguitate posibile. În viitor vor exista și propoziții care vor necesita mai multe forme logice, chiar și atunci când structura sintactică atribuită lor este unică.

6.1.3. Verbe și stări în formă logică; roluri tematice

În analiza făcută până în prezent corespondența dintre verbe și sensurile adecvate s-a realizat cu verbul acționând ca predicat în forma logică. Această abordare este de natură să nu poată reflecta anumite generalizări extrem de utile. Pentru a studia această situație, vom lua în considerație următoarele trei propoziții, care folosesc toate verbul *a sparge*:

1. *Ion a spart geamul cu ciocanul.*
2. *Ciocanul a spart geamul.*
3. *Geamul s-a spart.*

Intuitiv, toate aceste propoziții descriu același tip de eveniment, variind doar detaliile referitoare la acesta. Din această cauză ar fi de dorit ca verbul *a sparge* să corespundă aceluiași sens în fiecare caz. Problema care apare constă în faptul că, în cele trei afirmații, modul de folosire a verbului pare să indice sensuri verbale de arități diferite. Primul sens pare a fi o *relație ternară* între Ion, fereastră și ciocan, cel de-al doilea o *relație binară* între ciocan și geam, iar al treilea o *relație unară* în care intervine numai geamul. Se pare că sunt necesare trei sensuri distincte ale verbului *a sparge*: SPARGE1, SPARGE2 și SPARGE3, care diferă prin aritate și produc forme logice de tipul:

1. (<TRECUT SPARGE1> (NUME **i1** "Ion") <ART_L **w1** GEAM1> <ART_L **h1** CIOCAN1>)
2. (<TRECUT SPARGE2> <ART_L **h1** CIOCAN1> <ART_L **w1** GEAM1>)
3. (<TRECUT SPARGE3> <ART_L **w1** GEAM1>)

Pentru a garanta faptul că fiecare predicat este interpretat în mod corespunzător, reprezentarea ar trebui completată cu axiome care să asigure că ori de câte ori 1 este adevărat și 2 este adevărat și că ori de câte ori 2 este adevărat și 3 este

adevărat. Aceste axiome sunt de obicei numite *postulate ale sensului.* Specificarea unor asemenea restricții pentru fiecare verb în parte pare însă extrem de laborioasă și deci neconvenabilă.

Pentru a rezolva această problemă, Davidson (1967) propune o formă alternativă de reprezentare, care se caracterizează prin introducerea noțiunii de eveniment. În reprezentarea lui Davidson și folosind notațiile noastre actuale, o propoziție de tipul *Ion îl sparge* se traduce prin

(∃ **e1** : (SPARGE **e1** (NUME **i1** "Ion")(PRO **j1** ÎL1)))

ceea ce semnifică faptul că **e1** este un eveniment în care Ion sparge geamul menționat.

Reprezentarea sensului propoziției *Ion îl sparge cu ciocanul* este, în această accepție, următoarea:

(∃ **e1** : (&(SPARGE **e1** (NUME **i1** "Ion")(PRO **j1** ÎL1))
(INSTR **e1** <ART_L **h1** CIOCAN>)))

Avantajul acestei reprezentări constă în aceea că diverse tipuri de complemente circumstanțiale suplimentare, cum ar fi *cu ciocanul, în hol, din greșeală* etc. pot fi adăugate reprezentării de bază prin intermediul predicatelor care se referă la evenimentul respectiv. În acest fel este suficientă definirea numai a unuia dintre sensurile verbului *a sparge*, definire cu ajutorul căreia vor putea fi tratate toate cazurile posibile.

Această abordare este în concordanță cu o teorie apărută la începutul anilor '70, care își păstrează valabilitatea și astăzi influențând studiile actuale și care afirmă că există o mulțime limitată de relații semantice abstracte care se pot stabili între un verb și argumentele sale. Aceste relații sunt numite **roluri tematice** sau **roluri de caz**[5]. Ceea ce caracterizează această abordare este faptul

---

[5] Studiile asupra rolurilor tematice își au originea în lucrările lui Fillmore (1968) referitoare la așa-numita **gramatică a cazului**. Aceasta reprezintă un tip de gramatică și de teorie generală a limbii propus de Ch. J. Fillmore în lucrările din 1968-1972 care, păstrând o concepție de tip generativ și de tip transformațional, precum și înțelegerea organizării limbii pe două niveluri, de adâncime și de suprafață, propune o viziune cu totul diferită asupra organizării structurii de adâncime și dă o nouă semnificație componentului transformațional. În această concepție, *structura de adâncime este organizată exclusiv semantic, pe categorii logico-semantice de tipul rolurilor, numite cazuri*, iar structura sintactică apare numai în suprafață, revenind în exclusivitate transformărilor.

Menționăm faptul că, în gramatica generativ-transformațională, se operează cu *structură de adâncime* și *structură de suprafață*. Structura de suprafață reprezintă nivelul mai puțin abstract, mai apropiat de structurile sintactice observabile, incluzând anomalii sintactico-semantice de tipul elipsei. Prin contrast, structura de adâncime este un nivel de reprezentare foarte abstract, niciodată direct observabil. Există mari diferențe de concepere a structurii de adâncime și a raportului ei cu structura de suprafață între diversele tipuri de gramatică generativă. În modelul generativ standard

că deși diverși cercetători au utilizat de-a lungul timpului mulțimi de roluri diferite, numărul de roluri necesar a rămas întotdeauna relativ mic. Astfel, revenind la exemplele anterioare, vom remarca faptul că, din punct de vedere intuitiv, *Ion, ciocanul* și *fereastra* au același rol semantic în fiecare dintre propozițiile menționate. *Ion* este factorul activ, actorul (rolul **agent**), *fereastra* este obiectul (rolul **temă**), iar *ciocanul* este instrumentul (rolul **instrument**), toate intervenind în actul de spargere. Aceasta sugerează o reprezentare a înțelesului propoziției similară celei folosite în rețelele semantice, unde totul este exprimat în termeni de relații unare și binare. În mod concret, folosind cele trei roluri tematice amintite, putem reprezenta înțelesul propoziției *Ion sparge geamul* în felul următor:

(∃ **e** (&(SPARGE **e**)(AGENT **e** (NUME **i1** "Ion"))
(TEMA **e** <ART_L **w1** GEAM>)))

Deoarece astfel de construcții sunt extrem de frecvente, vom introduce o nouă notație pentru ele. Astfel, forma abreviată a unei aserțiuni de tipul

(∃ **e** : (&(Eveniment-p **e**)(Relație$_1$ **e** obj$_1$)...(Relație$_n$ **e** obj$_n$)))

va fi

(Eveniment-p **e** [Relație$_1$ obj$_1$]...[Relație$_n$ obj$_n$])

În particular, forma cvasilogică a propoziției *Ion a spart geamul*, atunci când se utilizează această abreviere, este următoarea:

(<TRECUT SPARGE1> **e1** [AGENT(NUME **i1** "Ion")]
[TEMA<ART_L **w1** GEAM1>])

Au fost găsite argumente pentru a trata în mod similar și verbele care nu exprimă evenimente. Astfel, dacă propoziția *Maria era nefericită* poate fi reprezentată folosind un predicat unar în felul următor

(<TRECUT NEFERICIT>(NUME **j1** "Maria"))

(Chomsky, 1965), structura de adâncime este concepută sintactic, reprezentând structurile de constituenți în termeni de categorii sintactice. Rolul structurii de adâncime este de a genera structurile "bine-formate" dintr-o limbă prin aplicarea regulilor de structură a frazei și de subcategorizare. În modelul Fillmore, structura de adâncime nu mai este concepută sintactic, ci logico-semantic, în termenii categoriilor de caz, fiind introduse șase cazuri. Pentru detalii, vezi [9], s.v. *gramatică, suprafață, adâncime*.

Ideile lui Fillmore au fost adoptate și adaptate atât în lingvistică, cât și în filozofie de către mulți alți cercetători (a se vedea, spre exemplu, Jackendoff (1972), Dowty (1989), Parsons (1990)). Tehnici similare pot fi întâlnite în multe sisteme computaționale. În astfel de sisteme, rolurile tematice sunt cel mai adesea legate de reprezentarea cunoștințelor care se află la baza sistemului (spre exemplu, Charniak (1981)).

se pune problema modului în care trebuie tratate complementele care pot să apară în formulări de tipul *Maria a fost nefericită la întâlnire*. Pentru rezolvarea unor astfel de situații, literatura de specialitate generalizează noțiunea de eveniment astfel încât evenimentele să includă și **stări**, după care se folosește aceeași tehnică. Spre exemplu, NEFERICIT poate fi considerat un predicat care afirmă că argumentul său este o *stare de nefericire*. Utilizând, în plus, rolul temă pentru a-l include pe Ion, obținem reprezentarea:

(∃ **s** <TRECUT NEFERICIT> **s**)(TEMA **s** (NUME **j1** "Maria"))

Folosind aceleași convenții de abreviere definite în cazul evenimentelor, putem reprezenta propoziția *Maria a fost nefericită la întâlnire* sub forma următoare:

(<TRECUT NEFERICIT> **s** [TEMA(NUME **j1** "Maria")]
[LA-LOC **m1** ÎNTÂLNIRE])

În particular, remarcăm, în cadrul acestei reprezentări, apariția unui nou rol tematic - LOC - care descrie *locul* în care s-a petrecut evenimentul generalizat, în cazul nostru locul în care s-a instalat starea de nefericire. Întrucât complementul circumstanțial respectiv se formează cu prepoziția *la*, am numit acest rol tematic LA-LOC. Roluri tematice similare se întâlnesc în limba engleză: rolul FROM-LOC, care descrie de unde vine ceva sau cineva, ca în *from here*; rolul TO-LOC, care descrie destinația, ca în *to the ground*; rolul PATH-LOC, care descrie traiectoria sau drumul, ca în *along the gorge* etc. Există și alte roluri tematice și o întreagă teorie lingvistică referitoare la acestea, care nu constituie însă obiectul acestei lucrări.

Revenind la forma logică a propozițiilor, vom remarca faptul că unii autori o consideră pe aceasta incapabilă de a surprinde întreaga bogăție și expresivitate a limbajului natural, argumentând faptul că nici chiar introducerea variabilelor eveniment ca argumente ale predicatelor atomice nu este suficientă în acest sens. Totuși, utilizarea *evenimentelor* și a *stărilor* în forma logică s-a dovedit eficientă pentru dezvoltarea ideilor de bază ale semanticii computaționale.

Alte forme ale reprezentărilor discutate aici și preluate din [4] există în literatura de specialitate. În general, se vor folosi diverse reprezentări și variante ale formei logice, considerate adecvate scopului propus.

6.1.4. Interpretare semantică și interpretare contextuală

Așa cum se arată în [4], pentru a dezvolta o teorie a interpretării semantice este nevoie, mai întâi, să dezvoltăm un *model structural*, ca și în cazul sintaxei. Pentru sintaxă a fost introdusă noțiunea de categorie sintactică, după care au fost studiate modalități de a combina categoriile de bază pentru a forma structuri mai ample. Aceeași strategie va fi aplicată în cazul semanticii. S-ar putea crede că unitatea semantică de bază o reprezintă cuvântul sau morfemul, dar această abordare eșuează datorită prezenței fenomenului de ambiguitate. De

pildă, în [4], se ia în considerație exemplul verbului *to go* (a merge) din limba engleză. Într-un dicționar tipic al limbii engleze verbului *to go* îi corespund peste 40 de intrări. Fiecare dintre aceste definiții reflectă un **sens** diferit al cuvântului. Pentru anumite sensuri sunt date sinonime. Pentru *to go* ar putea fi date sinonime precum *to move* (a se mișca), *do depart* (a pleca), *to pass* (a trece), *to vanish* (a dispărea), *to set out* (a se porni) ș.a.. Multe dintre acestea evidențiază un nou sens al verbului *to go*. Desigur, dacă acestea sunt sinonime perfecte ale unora dintre sensurile lui *to go*, atunci verbele însele vor avea unele sensuri identice. De pildă, unul dintre sensurile lui *to go* (a merge) va fi identic cu unul dintre sensurile lui *to depart* (a pleca). Prin urmare, există cuvinte care au *sensuri sinonime*. Chiar și în acest caz, având în vedere faptul că fiecare cuvânt al unei limbi are unul sau mai multe sensuri, rezultă că ne confruntăm cu un număr foarte mare de sensuri ale cuvintelor.

Din fericire, s-a observat că diferitele sensuri pot fi organizate într-o mulțime de *clase de obiecte* prin care putem clasifica universul. *Mulțimea diferitelor clase de obiecte aparținând unei reprezentări se numește* ***ontologia*** *acelei reprezentări.*

Pentru a trata un limbaj natural este necesară o ontologie mult mai largă decât găsim, de regulă, în studiile despre logica formală. Asemenea clasificări ale obiectelor suscită de mult timp interes și ele se regăsesc, de pildă, în scrierile lui Aristotel. Clasele majore sugerate de acesta au fost substanța (obiectele fizice), cantitatea (cum ar fi numerele), calitatea (cum ar fi "roșu aprins"), relația, locul, timpul, poziția, starea, acțiunea și sentimentul. Acestei liste îi pot fi adăugate, conform [4], alte clase, cum ar fi: evenimentele, ideile, conceptele și planurile. Două dintre cele mai importante clase sunt **acțiunile** și **evenimentele**.

Evenimentele sunt lucruri care se petrec în univers (în lumea înconjurătoare) și care sunt importante în multe teorii semantice deoarece ele furnizează o structură pentru a organiza interpretarea propozițiilor. *Acțiunile* sunt lucruri pe care le fac *agenții*, provocând, în acest fel, producerea unui eveniment. Ca în cazul tuturor obiectelor aparținând unei ontologii, ne putem referi la acțiuni și la evenimente prin intermediul pronumelor, ca în următorul fragment de discurs: *Am ridicat greutățile. Acesta este un lucru neplăcut.* Aici, pronumele *acesta* se referă la acțiunea de a ridica greutățile.

O altă categorie care poate influența substanțial analiza semantică este **situația**. Așa cum s-a menționat anterior, o situație se referă la o mulțime particulară de circumstanțe și poate fi privită ca subsumând noțiunea de eveniment. În multe cazuri, o situație poate acționa ca o abstracțiune a lumii înconjurătoare raportat la o anumită localizare și un anumit moment de timp. Spre exemplu, propoziția *Am ascultat și apoi am aplaudat la susținerea tezei de doctorat* se referă la o mulțime de activități desfășurate într-un anumit loc și la un anumit moment de timp, descrise ca reprezentând situația *susținerea tezei de doctorat*.

Există, așa cum s-a menționat anterior, diverse *teorii ale sensului*, care iau în considerație aspecte diferite ale acestuia. *Un prim nivel la care se poate analiza noțiunea de sens este acela al sensului independent de context.* La acest nivel este produsă așa-numita **formă logică**, care poate fi obținută direct din structura sintactică a propoziției. Așa cum am mai arătat, chiar dacă cea mai mare parte a limbajului este dependentă de context, există totuși suficientă structură semantică a acestuia care se comportă independent de context și care poate fi deci folosită în procesul de interpretare semantică pentru a produce forma logică. Deoarece forma logică trebuie să fie independentă de context, apare necesitatea dezvoltării unui limbaj formal în care să fie posibilă specificarea sensului fără a face referiri la însuși limbajul natural. Acesta este ceea ce am numit *limbajul formei logice.*

Limbajul formei logice dezvoltat aici este preluat din [4]. El își are rădăcinile în multe și variate surse. Cea mai importantă sursă timpurie o constituie limbajul de reprezentare folosit în sistemul LUNAR (Woods, 1978), care conține ideile inițiale corespunzătoare multora dintre dataliile reprezentărilor ulterioare. Influențe puternice mult mai recente le constituie limbajele formei logice dezvoltate de către Schubert și Pelletier (1982), Hwang și Schubert (1993), Moore (1981).

Ca multe alte limbaje ale formei logice, cel preluat din [4] și prezentat aici se bazează în mod esențial pe tehnici de introducere a *evenimentelor* în ontologie, cu scopul de a putea fi tratate argumentele opționale și diversele complemente. Această tehnică, astăzi consacrată, își are originile într-o lucrare a lui Davidson (1967), care prezintă numeroase argumente în favoarea acestei abordări. Alți autori care folosesc *sisteme bazate pe eveniment* aducând contribuții importante la fundamentarea limbajului formei logice sunt Moore (1981), Alshawi (1992) și, mai ales, Hobbs (1987).

Forma logică constituie, prin urmare, reprezentarea sensului independent de context și ea apare ca o reprezentare intermediară. **Interpretarea semantică** este procesul prin care se asociază unei propoziții forma sa logică. La rândul său, procesul de asociere a formei logice limbajului final de reprezentare a cunoștințelor (limbajul KR) constituie **interpretarea contextuală** (conform Definiției 6.1).

O versiune simplificată a ceea ce am putea numi **stadiile interpretării** este înfățișată în Fig. 6.1. Exemplul considerat este propoziția *O minge este roșie*, la care ne-am referit anterior.

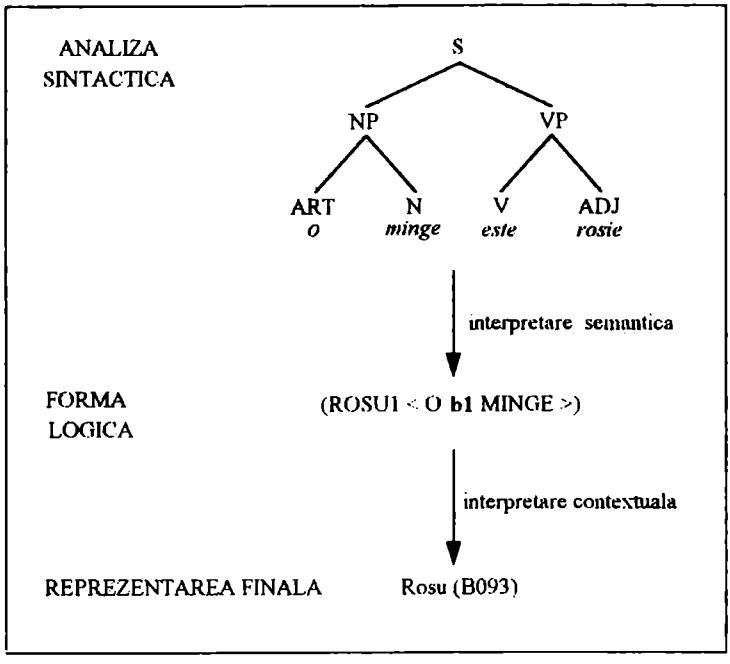


Fig. 6.1 Stadiile interpretării; forma logică ca reprezentare intermediară

În acest exemplu forma logică este folosită ca *reprezentare intermediară*. (Reamintim faptul că forma logică poate fi tratată și ca o funcție între situații). Legătura dintre forma logică și structurile sintactice discutate în primele capitole ale acestei lucrări se realizează atunci când un parser permite determinarea formelor logice în timpul procesului de analiză sintactică.

6.2. Legătura dintre sintaxă și semantică

6.2.1. Interpretarea semantică și compunerea semantică

Realizarea unei legături între formele logice și structurile sintactice este esențială deoarece aceasta ar permite determinarea formelor logice în timpul procesului de parsing, adică realizarea *interpretării semantice*.

Una dintre principalele premise de la care se pleacă adesea în efectuarea interpretării semantice este aceea că ea reprezintă un "proces compozițional" sau de compunere. *Compunerea semantică* este procesul de combinare a sensurilor cuvintelor pentru a forma sensurile diverselor grupuri sintactice sau ale propozițiilor. Prin urmare, în această abordare, sensul unui constituent

derivă exclusiv din sensurile subconstituenților săi. Modelul sintactic bazat pe gramatici independente de context reprezintă, spre exemplu, o teorie de aceeași natură pentru sintaxă. În cadrul acestui model regulile se aplică în funcție de categoria subconstituenților, fără a fi în vreun fel luată în considerație structura internă a acestora. O regulă de tipul $S \rightarrow NP\ VP$, de pildă, se aplică indiferent ce tip de grup nominal intervine. Ori de câte ori în gramatică se adaugă o nouă regulă referitoare la structura grupului nominal, o întreagă clasă de noi propoziții va putea fi acceptată de către gramatică. Aceasta este o proprietate extrem de utilă și deci de atractivă, a cărei aplicare s-a încercat și în interpretarea semantică. Abordările semantice computaționale se bazează pe construirea sensului constituent cu constituent, sensul unui constituent fiind produs de către o funcție computațională bine definită (i.e. un program), care folosește sensurile subconstituenților ca input.

Compunerea semantică este foarte atrăgătoare din punct de vedere computațional. Dar construirea unei asemenea teorii referitoare la interpretarea semantică ridică o serie de probleme greu de rezolvat.

O primă problemă care se ridică este aceea a existenței unei inconsistențe structurale între structura sintactică și structura formelor logice. În cazul propoziției *Ana iubește fiecare copil*, spre exemplu, analiza sintactică dezvăluie structura

((Ana)(iubește(fiecare copil)))

Forma logică (neambiguă) a propoziției într-o formă de tipul predicat - argument este:

(FIECARE **d**:(COPIL1 **d**)(IUBEȘTE1 **l1** (NUME **j1** "Ana")**d**))

Aceasta stabilește că pentru fiecare copil **d** există un eveniment **l1** care constă în faptul că Ana îl iubește pe **d**. Se observă că nu există o corespondență bijectivă între părțile formei logice și constituenții puși în evidență în urma analizei sintactice. Grupul sintactic *fiecare copil*, spre exemplu, este un subconstituent al grupului verbal *iubește fiecare copil*. Interpretarea lui semantică, pe de altă parte - structura care utilizează un cuantificator generalizat (FIECARE **d**:(COPIL1 **d**)...) - pare a avea sensul grupului verbal ca parte integrantă a ei. Un fenomen mult mai grav este acela că interpretarea lui *fiecare copil* pare a fi spartă în două, producând atât structura de cuantificator din afara predicatului, cât și un argument pentru predicat. Prin urmare, este greu de imaginat modul în care sensul lui *fiecare copil* ar putea fi reprezentat în mod separat și apoi folosit pentru a construi sensul întregii propoziții. Aceasta este una dintre cele mai importante și mai greu de rezolvat probleme care se ridică în compunerea semantică[6].

---

[6] O modalitate de a rezolva această problemă este introducerea formelor logice necuantificate. Dacă ne propunem, ca rezultat al interpretării semantice, producerea

O a doua problemă pe care nu o poate rezolva compunerea semantică este cea a expresiilor de tipul *a spăla putina* (cu sensul de *a fugi*), expresii ale căror sensuri nu pot fi deduse analizând sensurile cuvintelor componente. O posibilă abordare a acestei probleme o constituie atribuirea de înțelesuri semantice unor întregi grupuri de cuvinte, astfel încât unitatea de bază în analiza sensului să nu o mai constituie exclusiv cuvintele (sau morfemele), ci și anumite grupuri de cuvinte (sau propoziții), considerate a avea un sens primar.

În ciuda acestor neajunsuri, compunerea semantică pare a fi una dintre metodele cele mai indicate în procesul de interpretare semantică. Dacă ne plasăm într-un cadru în care procesul de interpretare semantică este unul compozițional, atunci trebuie să putem atribui o structură semantică oricărui constituent sintactic. Preluând exemplele din [4], ne propunem să atribuim un sens relativ uniform fiecărui grup verbal care poate interveni în orice regulă care presupune existența unui VP în calitate de subconstituent. Cazul cel mai simplu care poate fi luat în discuție este cel al unui grup verbal construit în jurul unui verb intranzitiv, ca în propoziția *Andrei plângea*. O posibilă interpretare este aceea că sensul grupului verbal *plângea* este de predicat unar care este adevărat referitor la orice obiect care în trecut plângea. Pentru a putea generaliza această abordare ar trebui ca fiecare VP să aibă sensul dat de un predicat unar. Un caz mai complex este cel al verbului tranzitiv care intervine în propoziția *Ion o sărută pe Maria*, propoziție având următoarea formă logică:

(SĂRUTĂ1 **k1** (NUME **j1** "Ion")(NUME **s1** "Maria"))

Conform abordării anterioare, sensul grupului verbal *sărută pe Maria* ar putea fi un *predicat unar* care este adevărat relativ la orice obiect care o sărută pe Maria. Este, prin urmare, util să putem exprima predicate unare de o atare complexitate.

Formalismul necesar în exprimarea unor predicate unare atât de complexe este furnizat de **lambda calcul**. Acesta reprezintă probabil instrumentul cel mai util, din punct de vedere tehnic, care se folosește în semantică. Lambda calculul a fost introdus de către logicianul Alonzo Church (1941) ca o modalitate de a transforma formulele în proprietăți.

În logica formală, *muritor(Socrate)* înseamnă *Socrate este muritor*, iar *muritor(Platon)* înseamnă *Platon este muritor*. Se pune atunci problema exprimării lui *muritor*. Conform lui Church, proprietatea de a fi muritor este exprimată de formula

$(\lambda x)muritor(x)$

---

unei forme logice necuantificate, atunci versiunea necuantificată a propoziției analizate ar fi

(IUBEȘTE1 **l1** (NUME **j1** "Ana") <FIECARE **d** COPIL1>)

care, din punctul de vedere al structurii, este mult mai apropiată de structura sintactică.

unde λx înseamnă că x din formulă trebuie furnizat ca argument. Un alt exemplu extrem de familiar este acela al definirii unei funcții $f: \mathbf{R} \rightarrow \mathbf{R}$ printr-o formulă de tipul

$$f(x)=7x^2 + 1.$$

În realitate, putem spune că aceasta nu reprezintă o definiție a lui f, ci a lui $f(x)$. Se presupune că definițiile lui $f(y)$, $f(z)$, $f(a+b+c)$ sunt analoage, fiind obținute prin înlocuirea lui x cu y, z și respectiv $a+b+c$. Dacă, în schimb, se definește funcția f ca fiind

$$f'=(\lambda x)7x^2 + 2$$

este exprimat în mod explicit faptul că x nu face parte din definiție, ci reprezintă doar un indicator pentru o valoare care urmează să fie furnizată ulterior.

În Prolog, o *lambda expresie* va fi un termen cu două argumente, o variabilă și respectiv un termen în care acea variabilă poate interveni, ca în exemplul:

```
lambda(X,titlu(X,programator))
```

Menționăm că *lambda* nu are nici o semnificație în Prolog și că reprezintă un functor arbitrar. Pentru mai multă conciziune, în viitor vom folosi caracterul ^ ca pe un operator infixat în lambda expresii. În această notație vom scrie

```
X^titlu(X,programator)
```

și vom spune că am definit *programator*. În lambda notația obișnuită aceasta s-ar scrie

$$(\lambda x)titlu(x,programator).$$

Vom mai observa faptul că nu este necesară o declarare suplimentară, întrucât ^ este deja un operator infixat. În expresii aritmetice el denotă exponențierea; în cadrul argumentelor lui `setof` și `bagof` el indică faptul că anumite variabile trebuie să primească toate valorile posibile. Nici una dintre aceste utilizări nu intră în conflict cu folosirea lui prezentă (referitoare la lambda expresii). Să mai notăm în final și faptul că ^ este asociativ la dreapta, astfel încât `X^Y^f(X,Y)` este corect și înseamnă `X^(Y^f(X,Y))`. Lambda notația tratează exact în acest fel argumentele multiple. Cu alte cuvinte, este cerut un prim argument și este întoarsă o lambda expresie care nu mai cere decât un singur alt argument.

Abandonând, pentru moment, implementarea în Prolog și întorcându-ne la exemplul luat în discuție, propoziția *Ion o sărută pe Maria*, remarcăm faptul că, în particular, expresia

(λx(SĂRUTĂ1 **k1** x(NUME **s1** "Maria")))

reprezintă un predicat care are un singur argument. X poate fi privit ca un parametru, iar acest predicat este adevărat referitor la orice obiect O cu proprietatea că, dacă se înlocuiește x cu O în expresia dată, se obține o propoziție adevărată. Se poate construi o propoziție pornind de la o lambda expresie și un argument. Spre exemplu, în limbajul formei logice, următoarea formulare este o propoziție:

((λx(SĂRUTĂ1 **k1** x(NUME **s1** "Maria")))(NUME **j1** "Ion"))

Această propoziție este adevărată dacă și numai dacă (NUME **j1** "Ion") satisface predicatul (λx(SĂRUTĂ1 **k1** x(NUME **s1** "Maria"))), care, prin definiție, este adevărat dacă și numai dacă

(SĂRUTĂ1 **k1** (NUME **j1** "Ion")(NUME **s1** "Maria"))

este adevărat. Se spune că această ultimă expresie a fost obținută **aplicând** lambda expresia (λx(SĂRUTĂ1 x(NUME **s1** "Maria"))) argumentului (NUME **j1** "Ion"). Această operație se numește **lambda reducerc**[7].

Având în vedere faptul că, pentru a realiza o împerechere perfectă între sintaxă și semantică, este necesară introducerea unor concepte precum lambda expresia, s-ar putea crede că această metodă nu este cea mai adecvată pentru a realiza interpretarea semantică. S-a constatat însă că, pe măsură ce o gramatică devine din ce în ce mai mare, tratând fenomene cât mai complexe, teoria compunerii semantice își mărește eficiența. Spre exemplu, folosindu-se această metodă, pot fi cu ușurință unite grupurile verbale, chiar și atunci când acestea au structuri total diferite. Astfel, în propoziția

Ana adoarme și închide o carte.

există două grupuri verbale: *adoarme*, care din punct de vedere semantic este interpretat ca un *predicat unar*, adevărat relativ la cineva care adoarme, de pildă (λa(ADOARME1 **e1** a)); *închide o carte*, tot predicat unar, adevărat relativ la cineva care închide o carte, cu alte cuvinte

---

[7] Lambda calculul reprezintă un limbaj extrem de puternic, în care una dintre axiomele cruciale este

((λx Px) a) = P{x|a}

unde Px este o formulă arbitrară în care intervine x, iar P{x|a} reprezintă formula în care fiecare apariție a lui x a fost înlocuită cu a. Pornind de la această axiomă se pot defini două operații principale: lambda reducerea (parcurgerea axiomei de la stânga la dreapta) și lambda abstracția (parcurgerea axiomei de la dreapta la stânga). În interpretarea semantică ne interesează cel mai mult lambda reducerea deoarece această operație are tendința de a simplifica formulele. Întrucât lambda reducerea nu face decât să înlocuiască o formulă cu una mai simplă care este egală cu ea, această operație ar putea fi eliminată, din punct de vedere formal, din procesul de interpretare semantică. Neutilizarea lambda reducerii ar face însă ca răspunsurile, deși corecte, să fie aproape de nedescifrat.

(λa(ÎNCHIDE1 **l1** a <O **c1** CARTE1>))

Aceste două predicate unare se pot combina pentru a forma un predicat unar complex, adevărat referitor la cineva care îndeplinește ambele acțiuni, aceea de a adormi și de a închide cartea, adică

(λa(&(ADOARME1 **e1** a)(ÎNCHIDE1 **o1** a <O **c1** CARTE1>)))

Aceasta din urmă reprezintă forma unui VP, care se poate combina cu alți constituenți, inclusiv cu un alt grup verbal. Ea poate fi aplicată, spre exemplu, unui grup nominal care are forma logică (NUME **s1** "Ana") pentru a reconstitui sensul propoziției inițiale:

(&(ADOARME1 **l1** (NUME **s1** "Ana"))
(ÎNCHIDE1 **o1** (NUME **s1** "Ana") <O **c1** CARTE1>))

Acesta a fost doar un exemplu de interpretare semantică utilizând operația de compunere semantică. În general, *fiecare grup sintactic major corespunde unei anumite construcții semantice*. S-a constatat că grupurilor verbale (VP) și grupurilor prepoziționale (PP) le corespund predicate unare (eventual sub forma unor expresii complexe provenind din lambda expresii), unui S îi corespunde o întreagă propoziție, în timp ce grupurilor nominale (NP) le corespund termenii. Categoriilor relativ minore le corespund expresii care definesc rolul lor în construirea așa-ziselor categorii majore. Întrucât fiecărui constituent dintr-o categorie sintactică de un anumit tip îi corespunde același tip de construcție semantică, aceștia pot fi tratați în mod uniform. Nu este necesar, spre exemplu, să fie cunoscută exact structura unui anumit grup verbal. Atâta timp cât se cunoaște faptul că, din punct de vedere semantic, acestuia îi corespunde un predicat unar, el va putea fi folosit în construcția sensului oricărui alt constituent mai amplu, care îl conține.

6.2.2. O gramatică și un lexicon cu interpretare semantică

În cele ce urmează, continuăm să fim interesați de *interpretarea semantică*, adică de *determinarea formelor logice în timpul procesului de analiză sintactică*. Problema care se pune este aceea de a "calcula" forma logică utilizând caracteristicile în timpul procesului de parsing.

Pentru ca exemplele să fie cât mai simple, forma logică pe care ne propunem să o determinăm va fi o structură de tipul predicat - argument care nu utilizează o reprezentare cu roluri tematice. O generalizare de acest fel, care să identifice roluri tematice, va fi însă posibilă. Alegerea noastră se oprește la structurile mai simple menționate, întrucât acestea permit ca toate verbele care au aceeași structură de subcategorizare să fie tratate în același mod.

Principala extensie necesară pentru a determina forma logică în timpul procesului de analiză sintactică utilizând caracteristici este aceea de a adăuga o

caracteristică, pe care o vom numi SEM, fiecărei *intrări lexicale* și fiecărei *reguli a gramaticii*. Spre exemplu, o regulă a gramaticii de forma

$$S \rightarrow NP\ VP$$

s-ar putea transforma în

(S **SEM** (?semvp ?semnp)) → (NP **SEM** ?semnp)(VP **SEM** ?semvp)

Dacă acestei reguli îi sunt furnizați un subconstituent de tip NP având caracteristica SEM

(NUME **m1** "Maria")

și un subconstituent de tip VP cu SEM

(λa(VEDE1 **e8** a (NUME **i1** "Ion")))

atunci caracteristica SEM a noului constituent S este expresia

((λa(VEDE1 **e8** a (NUME **i1** "Ion"))) (NUME **m1** "Maria"))

Această expresie poate fi simplificată folosind lambda reducerea. În acest mod se obține formula

(VEDE1 **e8** (NUME **m1** "Maria") (NUME **i1** "Ion"))

care este forma logică corectă a întregii propoziții. Fig. 6.2 ne arată *arborele de derivare* corespunzător acestei propoziții, indicând caracteristica de tip SEM a fiecărui constituent.

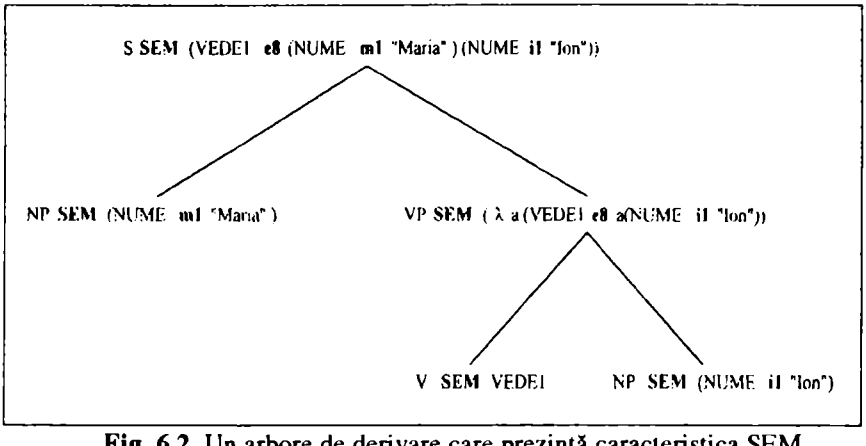


Fig. 6.2 Un arbore de derivare care prezintă caracteristica SEM

În cadrul lexiconului caracteristica SEM este folosită pentru a indica toate sensurile posibile ale fiecărui cuvânt. În general, orice cuvânt va avea un sens diferit pentru fiecare posibilitate de subcategorizare pe care o are, întrucât fiecăreia dintre acestea îi corespunde un predicat de o altă aritate. Spre

exemplu, relativ la verbul *a gândi* în lexicon vor trebui să existe două intrări, una pentru cazul în care subcategorizarea este de tipul "nimic", ca în propoziția

El gândește.

iar alta pentru cazul în care subcategorizarea este de tipul "grup prepozițional", ca în propoziția

El gândește asupra unei probleme.

Cele două intrări din lexicon, reflectând aceste două situații, ar putea fi:

gândește (v **SEM** GÂNDEȘTE1 **VFORMA** baza **SUBCAT**_nimic)
gândește (v **SEM** GÂNDEȘTE-ASUPRA1 **VFORMA** baza **SUBCAT**_PP:asupra)

Atunci când un cuvânt are forme diferite în funcție de caracteristicile sale de natură sintactică, sunt necesare, de asemenea, intrări multiple în lexicon. Iată intrările lexicale corespunzătoare substantivului *bărbat* cu pluralul *bărbați* (caracteristica **AGR** se referă la acord):

bărbat (n **SEM** BĂRBAT1 **AGR** 3s)
bărbați (n **SEM** (PLUR BĂRBAT1) **AGR** 3p)

Un lexicon trebuie să conțină informații despre toate cuvintele diferite care pot interveni, aceste informații incluzând toate restricțiile relevante legate de valorile caracteristicilor. Atunci când un cuvânt este ambiguu, el va fi descris prin intrări multiple în lexicon, câte una pentru fiecare utilizare diferită a sa. Toate acestea fac din scrierea unui lexicon adecvat o sarcină extrem de complexă, acesta fiind și motivul pentru care revenim asupra subiectului în noul context al interpretării semantice.

Există numeroase modalități de tratare a problemei conceperii lexiconului, în funcție de modul în care acesta va fi folosit ulterior. Așa cum a fost descrisă până acum crearea lui, lexiconul ar avea probabil niște dimensiuni mult prea mari. Datorită faptului că aproape toate cuvintele au tendința de a se înscrie în anumite clișee morfologice mai mult sau mai puțin regulate, putem evita includerea explicită în lexicon a multor forme ale cuvintelor. Majoritatea verbelor englezești, de pildă, folosesc o aceeași mulțime de sufixe pentru a indica diferite forme: *-s* este adăugat pentru a marca persoana a treia singular a prezentului, *-ed* pentru perfectul compus, *-ing* pentru participiul prezent ș.a.m.d. Dacă nu se face nici un fel de analiză morfologică, lexiconul trebuie să conțină fiecare dintre aceste forme, ceea ce nu este însă necesar. Pentru a nu mări inutil dimensiunile lexiconului suntem interesați ca, ori de câte ori este posibil, să creăm o unică intrare în lexicon corespunzător unui anumit cuvânt. Ideea este aceea de a memora în lexicon *forma de bază* a cuvântului, urmând ca celelalte forme ale sale să fie derivate prin utilizarea unor reguli independente de context. În cazul exemplului considerat, acela al verbelor englezești, forma de bază a verbului este memorată în lexicon, după care se utilizează reguli independente de context pentru a combina această formă de bază cu diferite

sufixe, în scopul determinării celorlalte forme ale verbului, corespunzătoare altor intrări lexicale, care însă nu mai fac parte ca atare din lexicon. Pentru a exemplifica, în cazul verbelor englezești, vom lua în considerație următoarea regulă care se referă la timpul prezent

(V **RAD** ?r **SUBCAT** ?s **VFORMA** prez **AGR** 3s) →
(V **RAD** ?r **SUBCAT** ?s **VFORMA** baza) (+S)

unde +S este o nouă categorie lexicală, care se referă exclusiv la morfemul pentru sufix *-s*. Această regulă generează constituenții pentru persoana a treia singular a prezentului. Ea reprezintă o regulă generală care transformă

forma de bază a verbului +s

în forma corespunzătoare persoanei a treia, singular a timpului prezent. O altă regulă va genera constituenții pentru timpul prezent în cazul formelor diferite de persoana a treia singular, care la majoritatea verbelor englezești coincid cu rădăcina verbului:

(V **RAD** ?r **SUBCAT** ?s **VFORMA** prez **AGR** {1s 2s 1p 2p 3p}) →
(V **RAD** ?r **SUBCAT** ?s **VFORMA** baza)

Această regulă generală poate însă genera interpretări eronate, ea nefiind corectă în cazul verbelor neregulate. Pentru a trata aceste cazuri, vom introduce o caracteristică binară utilizată cu scopul de a identifica formele neregulate. În mod concret, verbele având caracteristica +PREZ-NEREG au formele de prezent neregulate. Atunci regula anterioară poate fi formulată în mod corect astfel:

(V **RAD** ?r **SUBCAT** ?s **VFORMA** prez **AGR** {1s 2s 1p 2p 3p}) →
(V **RAD** ?r **SUBCAT** ?s **VFORMA** baza PREZ-NEREG–)

O astfel de caracteristică restricționează aplicarea regulilor lexicale standard, formele neregulate fiind adăugate în mod explicit lexiconului. Cele două *reguli lexicale* care ar trebui să existe corespunzător timpului prezent al verbelor englezești sunt, prin urmare:

(V **RAD** ?r **SUBCAT** ?s **VFORMA** prez **AGR** 3s) →
(V **RAD** ?r **SUBCAT** ?s **VFORMA** baza PREZ-NEREG–)+S (6.1)

(V **RAD** ?r **SUBCAT** ?s **VFORMA** prez **AGR** {1s 2s 1p 2p 3p}) →
(V **RAD** ?r **SUBCAT** ?s **VFORMA** baza PREZ-NEREG–) (6.2)

Acesta este doar unul dintre modurile posibile de *tratare a lexiconului*. Exemplul a fost preluat din [4] și își propune, în egală măsură, să ilustreze un alt mod de a nota caracteristicile și valorile acestora. Lăsăm ca exercițiu cititorului găsirea unor reguli similare pentru limba română. Atragem atenția asupra faptului că, în limba română, analiza verbului comportă mult mai multe

dificultăți decât în limba engleză, dar reguli pentru flexiunea substantivelor și a adjectivelor pot fi concepute relativ ușor.

Revenind la utilizarea caracteristicii SEM în cadrul lexiconului, mai precis al **regulilor lexicale**, vom spune că acestea trebuie și ele modificate pentru a putea folosi caracteristica SEM. Variantele revizuite, în acest sens, ale regulilor (6.1) și (6.2) sunt:

(V **VFORMA** prez **AGR** 3s **SEM** <PREZ ?semv>) →
(V **VFORMA** baza **PREZ-NEREG-SEM** ?semv) +S (6.3)

și respectiv

(V **VFORMA** prez **AGR** {1s 2s 1p 2p 3p} **SEM** <PREZ ?semv>) →
(V **VFORMA** baza **PREZ-NEREG-SEM** ?semv) (6.4)

Regulile (6.3) și (6.4) sunt aceleași cu (6.1) și (6.2), cu excepția adăugării caracteristicii SEM.

În ceea ce privește **regulile gramaticii**, pentru a ilustra utilizarea caracteristicii SEM, vom lua în considerație Gramatica 6.1, preluată din [4].

1. (S **SEM** (?semvp ?semnp) → (NP **SEM** ?semnp)(VP **SEM** ?semvp)
2. (VP **VAR** ?v **SEM** (λ a2 (?semv ?v a2))) → (V[_nimic] **SEM** ?semv)
3. (VP **VAR** ?v **SEM** (λ a3 (?semv ?v a3 ?semnp))) →
   (V[_np] **SEM** ?semv) (NP **SEM** ?semnp)
4. (NP CINE - **VAR** ?v **SEM** (PRO ?v ?sempro)) → (PRO **SEM** ?sempro)
5. (NP **VAR** ?v **SEM** (NUME ?v ?semnume)) → (NUME **SEM** ?semnume)
6. (NP **VAR** ?v **SEM** < ?semart ?v (?semnp ?v) >) →
   (ART **SEM** ?semart)(NP **SEM** ?semnp)
7. (NP **SEM** ?semn) → (N **SEM** ?semn)

Caracteristici de tip cap pentru S, VP, NP, NP:VAR

Gramatica 6.1 - utilizează caracteristica **SEM**

Această gramatică acceptă propoziții foarte simple și grupuri verbale, determinând forma lor logică. Se observă că, pe lângă caracteristica SEM, este introdusă o caracteristică suplimentară. Caracteristica VAR are rolul de a memora variabila discursului corespunzătoare constituentului. Ea s-a dovedit în mod special utilă pentru tratarea diverselor tipuri de complemente. Caracteristica VAR este în mod automat generată de către parser atunci când se construiește un constituent lexical pornindu-se de la un cuvânt și este transmisă în sus de-a lungul arborelui de derivare prin tratarea ei ca o *caracteristică de tip*

cap[8] sau *caracteristică principală*. Ea garantează faptul că variabilele discursului sunt întotdeauna unice.

Revenind la regulile gramaticii prezentate, dintre care prima a fost deja comentată, remarcăm faptul că a doua și a treia regulă tratează verbele tranzitive și respectiv intranzitive construind interpretarea adecvată a grupului verbal. Fiecare dintre aceste reguli ia în considerație caracteristica SEM a verbului, ?semv și construiește un predicat unar care se va aplica subiectului. Regula a patra construiește structura SEM adecvată pentru pronume, fiind dat un sens al pronumelui ?sempro, iar regula a cincea face același lucru în cazul numelor proprii. Regula a șasea definește o expresie care presupune o expresie cuantificată constând din cuantificatorul ?semart, variabila discursului ?v și o propoziție care restricționează cuantificatorul, aceasta din urmă fiind construită prin aplicarea predicatului unar ?semcnp variabilei discursului. Spre exemplu, presupunând că variabila discursului ?v este **f1**, grupul nominal *o femeie* ar combina sensul (SEM) lui *o*, adică operatorul O, cu sensul (SEM) lui *femeie*, adică FEMEIE1, pentru a forma expresia <O **f1** (FEMEIE1 **f1**)>. În fine, regula a șaptea și ultima a gramaticii construiește un grup nominal pornind de la un unic substantiv. Întrucât SEM-ul unui substantiv comun este deja un predicat unar, această valoare este pur și simplu preluată și folosită ca SEM al grupului nominal.

Știind în ce fel trebuie modificată o gramatică pentru ca ea să poată folosi în determinarea formei logice, este firesc să ne întrebăm care sunt schimbările ce trebuie aduse unui parser în același scop. Pentru ca analizorul sintactic cu hartă standard să poată trata interpretarea semantică este necesară efectuarea a numai două modificări:

- atunci când o regulă lexicală este instanțiată pentru a fi folosită, caracteristica VAR este setată la o nouă variabilă a discursului;
- ori de câte ori este construit un constituent, caracteristica SEM este simplificată prin efectuarea tuturor lambda reducerilor posibile.

Datorită acestor două modificări, parser-ul cu hartă standard pe care l-am prezentat în capitolul 4 va putea determina forma logică în timpul procesului de analiză sintactică. Pentru a ilustra acest fapt vom lua în considerație un parser de tip bottom-up, care lucrează de la stânga la dreapta și utilizează Gramatica 6.1 și vom analiza o urmă lăsată de acesta asupra propoziției

Ioana mâncase o prăjitură.

al cărei arbore de derivare este cel din Fig. 6.3.

---

[8] Caracteristicile ale căror valori corespunzătoare constituentului părinte și subconstituentului său principal sunt egale se numesc *caracteristici de tip cap* sau *caracteristici principale*. Spre exemplu, în toate regulile referitoare la structura VP, valorile caracteristicilor **VFORMA** și **AGR** corespunzătoare grupului verbal și verbului în jurul căruia se organizează acesta (capul grupului verbal) sunt egale. Prin urmare, **VFORMA** și **AGR** sunt caracteristici de tip cap.

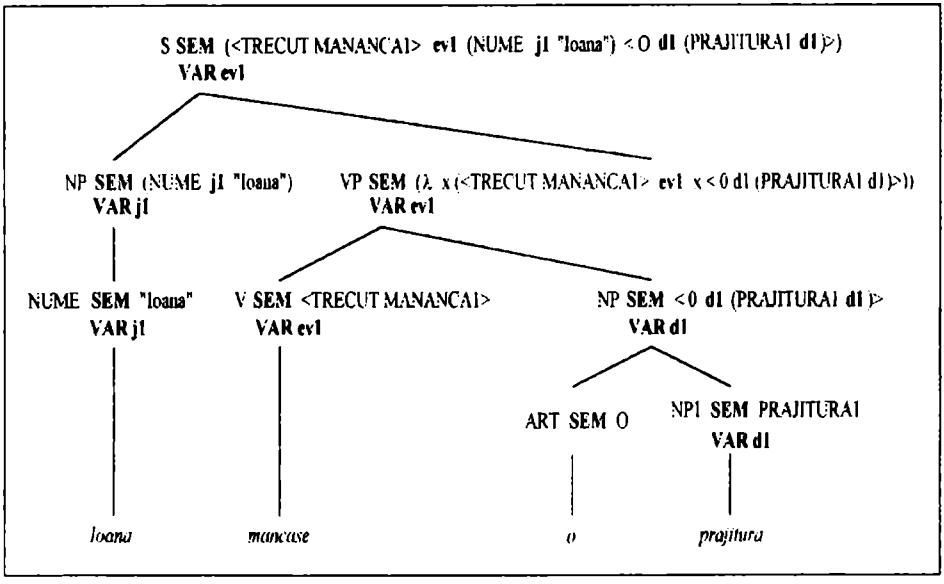


Fig. 6.3. Prezentarea caracteristicilor **SEM** și **VAR** în analiza propoziției *Ioana mâncase o prăjitură.*

Analizorul sintactic începe prin a analiza cuvântul *Ioana* ca pe un nume propriu, utilizând una dintre intrările lexicale. Datorită acestei analize, este generată o nouă variabilă a discursului, **j1**. Caracteristica VAR este setată la **j1**. Acest constituent este apoi folosit de către regula 5 a gramaticii pentru a construi un NP. Întrucât VAR este o caracteristică principală, caracteristica VAR a constituentului NP va fi, de asemenea, **j1**, iar caracteristica SEM corespunzătoare va fi construită din ecuație într-o manieră evidentă. Intrarea lexicală corespunzătoare cuvântului *mâncase* generează un constituent de tip V cu SEM-ul <TRECUT MĂNÂNCĂ1> și o nouă caracteristică VAR, **ev1**. Regula 6 a gramaticii combină caracteristicile SEM ale celor două intrări lexicale corespunzătoare cuvintelor *o* și *prăjitură* cu VAR-ul de la substantiv pentru a construi un constituent de tip NP având SEM-ul <O **d1** (PRĂJITURA1 **d1**)>. Acesta este apoi combinat, prin intermediul regulii 3, cu ambele caracteristici, SEM și VAR, ale verbului, pentru a forma un constituent VP având SEM-ul

(λ x(<TRECUT MĂNÂNCĂ1> **ev1** x <O **d1** (PRĂJITURA1 **d1**)>))

Acesta este apoi combinat cu subiectul NP pentru a se obține forma logică finală a propoziției.

Prin utilizarea celor două caracteristici noi, SEM și VAR, precum și datorită celor două modificări ale parser-ului menționate anterior, devine

posibilă specificarea unei gramatici care va determina *construirea formei logice în timpul procesului de analiză sintactică*. Chiar dacă nu am introdus, până în prezent, decât cele mai simple tehnici de interpretare, putem spune că *procesul de bază al interpretării semantice* este acum complet descris.

În același timp, putem completa ***definiția analizorului sintactic***, prin care înțelegem acel proces care realizează corespondența dintre o propoziție, pe de o parte, și structura sintactică și forma ei logică, pe de altă parte. Un *parser* folosește cunoștințe despre cuvinte și sensurile lor (*lexiconul*), precum și o mulțime de reguli ce definesc structurile admisibile (*gramatica*), cu scopul de a atribui o *structură sintactică* și o *formă logică* unei propoziții constituind intrarea.

6.3. Tratarea ambiguității

Sensurile cuvintelor pot fi legate în diverse moduri, în funcție de *clasele de obiecte* pe care acestea le descriu. Unele sensuri sunt disjuncte (adică nu există nici un obiect care să poată fi în ambele clase în același timp). Spre exemplu, CÂINE1 (sensul tipic al lui *câine*) și PISICA1 (sensul tipic al lui *pisică*) sunt sensuri disjuncte. Există și sensuri care sunt subclase ale altor sensuri. Clasa CÂINE1, de pildă, este o subclasă a clasei MAMIFER1. Unele sensuri se suprapun, cum ar fi sensurile MAMIFER1 și DECASĂ1, desemnând acele animale de casă care sunt mamifere. Toate aceste cunoștințe și informații de natură generală joacă un rol important în dezambiguizarea semantică.

Relația de tip *submulțime* definește o ierarhie a abstracțiunilor relativ la sensurile cuvintelor. Această relație este importantă deoarece ea permite restricțiilor să fie formulate într-un mod concis și intuitiv, în termenii unor clase relativ vaste de obiecte. Un adjectiv desemnând o culoare, de exemplu, nu va avea sens decât dacă se referă la un obiect fizic, întrucât nu se poate vorbi despre gânduri, idei, sentimente, evenimente colorate într-o anumită culoare. Având în vedere relația de tip submulțime se pot crea *ierarhii de tipuri* extrem de utile pentru limbajul natural și prelucrarea acestuia. Menționăm faptul că aceste ierarhii nu reprezintă, de regulă, structuri arborescente. Acest lucru este evident întrucât diferite sensuri pot avea mai multe tipuri, superioare din punct de vedere ierarhic, în care să se încadreze. Un fragment dintr-o astfel de ierarhie a sensurilor ar putea fi cel din Fig. 6.4.

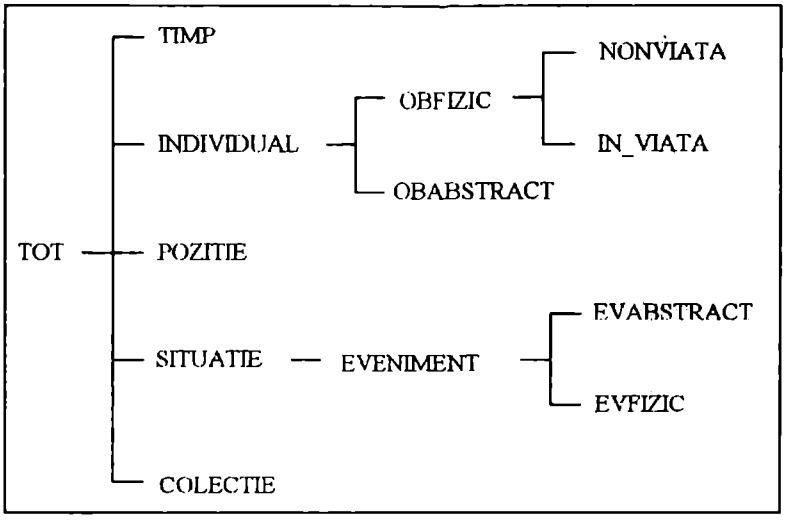


Fig. 6.4 O ierarhie de sensuri ale cuvintelor

Fiind dată o asemenea ierarhie de sensuri pentru limbajul natural, se pot analiza *restricțiile pe care predicatele le impun argumentelor lor.* Exemplificăm, în cele ce urmează, cu verbul *a citi.* Acesta are două argumente principale: agentul, care trebuie să fie un obiect capabil de a citi (i.e. un argument de tip PERSOANA) și tema, care trebuie să fie un obiect conținând text (cum ar fi o carte). Pentru a manevra în mod corect astfel de verbe, s-ar putea introduce în ierarhie un nou tip, cum ar fi OBTEXT, care s-ar afla într-o relație de tip submulțime cu NONVIAȚĂ. Acest nou tip ar putea, la rândul lui, include tipurile CARTE, ARTICOL, REVISTA etc. Constrângerile impuse de astfel de structuri ierarhice permit alegerea sensurilor adecvate pentru cuvintele unei propoziții, alegere care nu este evidentă pe baza studiului exclusiv al formei logice.

Întrucât multe cuvinte sunt ambigue, o propoziție dată poate avea mai multe înțelesuri distincte, corespunzătoare unor combinații diferite ale sensurilor cuvintelor. Un interpretor semantic poate realiza dezambiguizarea folosind așa-numitele **restricții de selecție** sau **restricții selecționale**, prin care înțelegem acele specificații referitoare la combinațiile admise ale sensurilor cuvintelor ce pot fi folosite pentru a elimina eventualele formulări incoerente construite de către un parser. Găsirea combinațiilor de sensuri admisibile poate fi privită ca reprezentând o problemă de satisfacere a unor *constrângeri.* În acest caz, constrângerile includ relațiile semantice dintre predicate unare (subclasă, disjunct etc.), care sunt exprimate prin ierarhii de tipuri (cum ar fi cele din Fig. 6.4) și restricții selecționale asupra tipurilor semantice ale argumentelor relațiilor binare. În cazul verbului *a citi,* de pildă, restricțiile selecționale sunt exprimate după cum urmează:

(AGENT CITEŞTE1 PERSOANA) - agentul trebuie să fie o persoană
(TEMA CITEŞTE1 OBTEXT) - tema trebuie să fie un obiect de tipul OBTEXT
(6.5)

Pentru a implementa tehnica dezambiguizării într-un interpretor semantic este nevoie de un mecanism de extragere din forma logică a informației referitoare la tipuri, informație care este proprie formei logice. Vom exemplifica această tehnică luând în considerație propoziția

Storcătorul citește un articol.

Această propoziție conține două cuvinte ambigue, și anume *storcătorul*, care poate însemna fie aparatul cu care se storc fructe (STORCT/MAS1), fie o persoană care stoarce idei, informații etc. (STORCT/PERS) și *articolul*, care poate însemna fie o lucrare (ARTICOL/TEXT), fie o parte de vorbire (ARTICOL1). Datorită prezenței celor două cuvinte ambigue, propoziția dată ar putea avea, teoretic vorbind, patru înțelesuri semantice, câte unul pentru fiecare combinație posibilă a sensurilor acestor cuvinte. Dar numai una dintre interpretările propoziției date are sens, și anume:

(CITEŞTE1 [AGENT <ART_L **d1** STORCT/PERS>]
[TEMA <UN **p1** ARTICOL/TEXT>]) (6.6)

Atunci când dorim să extragem din forma logică informația referitoare la tipuri, trebuie să examinăm fiecare termen al formei logice relativ la predicatele unare și binare care îl folosesc pe acesta. Spre exemplu, forma logică a propoziției menționate, înaintea aplicării restricțiilor selecționale (6.5), este următoarea:

(CITEŞTE1 **r1** [AGENT <ART_L **d1** {STORCT/MAS1 STORCT/PERS}>]
[TEMA <UN **p1** {ARTICOL/TEXT ARTICOL1}>])

Despachetând această notație, punem în evidență următoarele relații unare și binare:

(CITEŞTE1 **r1**)
({STORCT/MAS1 STORCT/PERS}**d1**)
({ARTICOL/TEXT ARTICOL1}**p1**)
(AGENT **r1 d1**)
(TEMA **r1 p1**)

Având în vedere restricțiile selecționale (6.5), putem elimina câteva dinte posibilitățile luate în considerație referitor la termenii formei logice. Astfel, pentru ca (AGENT **r1 d1**) să fie validat, **d1** trebuie să fie o persoană. Prin urmare, constrângerile unare asupra lui **d1** pot fi simplificate de la ({STORCT/MAS1 STORCT/PERS}**d1**) la (STORCT/PERS **d1**). În mod similar, interpretarea lui **p1** este simplificată la (ARTICOL/TEXT **p1**).

Transferând aceste constrângeri înapoi în forma logică, obținem o unică interpretare neambiguă (6.6).

Desigur, există situații mult mai complexe. Întreaga discuție de până acum, spre exemplu, s-a făcut pentru un anumit sens al verbului *a citi*, anume sensul CITEȘTE1. Dar verbul *a citi* ar putea avea două sensuri, CITEȘTE1 și CITEȘTE2, cel de-al doilea corespunzând unor contexte de tipul *a citi gândurile cuiva*. Restricțiile selecționale corespunzătoare sensului CITEȘTE2 ar putea fi exprimate sub următoarea formă:

(AGENT CITEȘTE2 PERSOANA)
(TEMA CITEȘTE2 STARE_MENTALA) (6.7)

Luând în considerație acest sens suplimentar, forma logică inițială a propoziției *Storcătorul citește un articol* devine:

({CITEȘTE1 CITEȘTE2} **r1**
[AGENT <ART_L **d1** {STORCT/MAS1 STORCT/PERS}>]
[TEMA <UN **p1** {ARTICOL/TEXT ARTICOL1}>])

Această ambiguitate suplimentară nu afectează însă rezultatul final deoarece sensul CITEȘTE2 presupune ca TEMĂ o STARE MENTALĂ și nici unul dintre sensurile lui *articol* nu este o subclasă a tipului STARE_MENTALĂ. Prin urmare, nu există nici o combinație de interpretări care să permită sensul CITEȘTE2.

Această tehnică trebuie extinsă asupra *substantivelor proprii*, a *pronumelor* și a *adjectivelor*. Va trebui să presupunem că există informație asupra tipului pentru acele obiecte la care se referă substantivele proprii. Spre exemplu, tipul lui *Ion* ar putea fi MASC (de la "masculin"). Cu alte cuvinte, *Ion* ar desemna un "obiect animat" de gen "masculin", adică de tip MASC. Tipul grupurilor pronominale ar putea fi definit de către sensul pronumelui, astfel încât EA1 ar trebui să fie o subclasă a lui FEMININ. O discuție detaliată pe marginea acestui subiect, referitor la limba engleză, poate fi găsită în [4]. Urmând acest model [4], se pot face adaptări corespunzătoare fiecărei limbi naturale în parte, adaptări care presupun analizarea particularităților sintactico-morfologice ale diferitelor limbi, introducerea de noi relații tematice etc.

De fiecare dată când este produsă o formă logică, mulțimea relațiilor semantice unare și binare pe care aceasta le conține poate fi verificată prin confruntarea cu toate constrângerile selecționale. Dacă nu există nici o interpretare care să satisfacă constrângerile, atunci forma logică respectivă reprezintă o interpretare anormală și constituentul corespunzător poate fi înlăturat. Altfel, forma logică simplificată, din care au fost eliminate sensurile imposibile, poate fi folosită ca reprezentând SEM-ul constituentului.

Un algoritm relativ simplu de satisfacere a constrângerilor este **Algoritmul 6.1**, prezentat în [4]. Pentru a implementa acest algoritm este necesară o procedură care să realizeze împerecherea sensurilor. O împerechere eșuează

dacă intersecția celor două tipuri este vidă. Altfel, procedura de împerechere returnează un tip care reprezintă intersecția acestora. Dacă unul dintre tipuri este un subtip al celuilalt, atunci rezultatul este subtipul respectiv. Luând în considerație exemplul anterior și presupunând că procedura de împerechere a sensurilor se numește *Perche*, putem spune, de pildă, că rezultatul apelării lui Pereche (PERSOANA, STORCT/PERS) va fi STORCT/PERS. În alte cazuri tipurile se pot suprapune, rezultatul întors de procedură fiind un sens care este un subtip al ambelor sensuri.

Algoritmul de satisfacere a constrângerilor începe prin a crea o listă a tuturor sensurilor posibile ale fiecărei variabile a discursului, după care iterează prin mulțimea relațiilor binare pentru a elimina acele constrângeri unare care nu pot satisface cel puțin o constrângere binară. Dacă este efectuată vreo schimbare, atunci constrângerile binare sunt din nou verificate. Procedeul continuă până când nu mai este operată nici o modificare. Algoritmul este următorul:

Algoritmul 6.1

Pasul de inițializare

Atribuie tipuri (variabila$_i$) listei sensurilor posibile ale variabilei *i*.

Pasul de iterare

Iterează în cadrul fiecărei relații binare (rel variabila$_1$ variabila$_2$) astfel:

1. Pentru fiecare sens$_1$ în tipuri (variabila$_1$):
   a. găsește toate restricțiile selecționale (rel sens$_1$ sens$_2$) unde sens$_2$ se intersectează cu un sens din tipuri (variabila$_2$);
   b. dacă nu se găsește nici una, înlătură sens$_1$ din tipuri (variabila$_1$).
2. Elimină din tipuri (variabila$_2$) orice sens care nu s-a împerecheat cu cel puțin o restricție la pasul 1.

Pasul de încheiere

Dacă, în cadrul ultimei iterații, s-a produs vreo schimbare asupra tipurilor variabilelor, atunci execută din nou pasul de iterare.
Altfel, dacă tip (variabila$_i$) este vidă pentru toți *i*, comunică eșecul.

Restricțiile selecționale furnizează o tehnică de dezambiguizare extrem de importantă și, din această cauză, sunt utilizate într-o formă sau alta în aproape toate sistemele computaționale. Există cel puțin două *moduri în care restricțiile selecționale pot fi adăugate unui parser*. **Modelul secvențial** presupune execuția programului care reprezintă implementarea parser-ului, urmată de verificarea tuturor interpretărilor găsite de acesta corespunzător unor constituenți de tip S compleți. **Modelul incremental** presupune verificarea

fiecărui constituent sugerat de către parser urmată de înlăturarea lui în cazul în care este incorect format din punct de vedere semantic. Literatura de specialitate [4] consemnează faptul că așa-numita metodă incrementală poate fi în mod semnificativ mai eficientă decât metoda secvențială.

6.4. Definirea structurii semantice. Semantică și teoria modelelor. Implementarea în Prolog

Termenul de semantică a fost până în prezent folosit pentru a desemna și a reflecta **reprezentarea înțelesului propozițiilor**. Această din urmă operație a fost exprimată în *limbajul formei logice*. Există însă și un alt sens al termenului de semantică, folosit în teoria limbajelor formale, sens care furnizează un **înțeles al însuși limbajului formei logice**.

Acest al doilea sens al termenului de semantică se concentrează asupra posibilității de a distinge diferitele *clase de obiecte semantice* prin explorarea proprietăților acestora care țin de un model teoretic, adică definind unități semantice în termenii corespondențelor acestora cu elementele teoriei mulțimilor. Aceste tehnici sunt tipice logicii matematice, dar pot fi aplicate cu succes și în studiul limbajului natural.

În acest context, construcția de bază pentru definirea proprietăților semantice este aceea de **model**, care intuitiv poate fi privit ca fiind alcătuit dintr-o mulțime de obiecte, împreună cu proprietățile și relațiile lor, însoțite de o specificare a legăturii dintre limbajul studiat și aceste obiecte și relații. Un model poate fi privit ca reprezentând un context particular în care are loc evaluarea unei propoziții. Evident, pot fi impuse diverse constrângeri asupra proprietăților pe care acest model le poate avea.

Teoria modelelor reprezintă o excelentă metodă pentru a studia *sensul independent de context*, întrucât sensurile propozițiilor nu sunt definite în raport cu un anumit model ci, mai degrabă, prin modul în care ele se raportează la orice model posibil. Cu alte cuvinte, sensul unei propoziții este definit în termenii proprietăților pe care acesta le are cu privire la un model arbitrar.

Din punct de vedere formal, un model **m** este un tuplu $\langle D_m, I_m \rangle$, unde D_m este **domeniul de interpretare** (i.e. o mulțime de obiecte primitive), iar I_m este **funcția de interpretare**.

O *formulă* de tipul *mănâncă(fata,cireșe)* ("Fata mănâncă cireșe") face parte dintr-un limbaj logic. Un **model m** pentru acest limbaj constă dintr-un **domeniu** D_m (care este mulțimea persoanelor și a obiectelor individuale despre care se poate vorbi) și o **funcție de interpretare** I_m care realizează o corespondență între elementele limbajului și cele ale domeniului. (Fiecărui element al limbajului îi corespunde un element al domeniului.) Mai exact:

- I_m face corespondența dintre constante logice (nume proprii) și elemente individuale ale lui D_m. Spre exemplu, I_m *(ana)* este Ana.

- I_m face corespondența dintre predicate și mulțimi de tupluri de elemente ale lui D_m. Spre exemplu, I_m *(galben)* selectează toate 1-tuplurile constând din elemente ale lui D_m care sunt galbene. În mod analog, I_m *(iubește)* alege toate perechile $<x,y>$ de elemente ale lui D_m astfel încât x iubește pe y; unui predicat cu trei argumente îi corespunde, prin I_m, o mulțime de triplete ordonate etc.

În acest context, se poate defini *ce înseamnă ca o formulă să fie adevărată*, după cum urmează:

- O formulă de tipul *predicat*$(arg_1, arg_2, ...)$ este adevărată dacă și numai dacă $<I_m(arg_1), I_m(arg_2), ...> \in I_m$ *(predicat)*.
  Spre exemplu, *aleargă(câine,pisică)* este adevărată atunci când $<I_m$ *(câine)*$, I_m$ *(pisică)*$> \in I_m$ *(aleargă)*.

- O formulă care conține cuantificatori sau conectori cum ar fi $\neg$, $\wedge$, $\vee$, $\forall$, $\exists$ este adevărată dacă îndeplinește condițiile impuse de definițiile conectorilor și cuantificatorilor așa cum sunt ele cunoscute din logica matematică.

Semantica cuantificatorilor în FOPC este relativ simplă. Pentru limbajul natural există cuantificatorii generalizați la care ne-am referit anterior (§6.1.1). Condițiile de adevăr corespunzătoare fiecărui cuantificator de acest tip specifică relația cerută între obiectele care satisfac cele două propoziții. Spre exemplu, să considerăm propoziția

(MAJORITATEA1 x (Px) (Qx))

Aceasta ar putea fi definită ca fiind adevărată *relativ la un model* m dacă și numai dacă peste jumătate dintre obiectele din $I_m(P)$ sunt în $I_m(Q)$. De pildă, propoziția

(MAJORITATEA1 x (CAINE1 x) (LATRĂ1 x))

este adevărată relativ la un model **m** dacă și numai dacă mai mult de jumătate dintre elementele mulțimii {x | (CAINE1 x)} fac parte, în egală măsură, din mulțimea {y | (LATRĂ1 y)}, adică numai dacă peste jumătate dintre câinii din **m** latră.

În cele ce urmează, pentru a simplifica notația, nu vom mai face referire la un model **m**. Rămâne însă valabil faptul că întotdeauna trebuie să ne raportăm la un anumit model.

Cadrul de lucru în care ne-am plasat aici este, prin urmare, cel al *teoriei modelelor*. Prin comparație cu alte modalități de a evalua formulele logice, teoria modelelor prezintă două avantaje importante. Mai întâi, este de notat faptul că ea nu atribuie valori de adevăr unor propoziții întregi, ci atribuie înțelesuri tuturor părților fiecărei formule. În al doilea rând, teoria modelelor

lucrează cu baze de cunoştinţe (modele), fără a face presupuneri sau revendicări asupra lumii reale în ansamblul ei. Acest fapt este deosebit de important, întrucât el corespunde îndeaproape modului în care se realizează gestionarea de către calculator a unei baze de date.

6.4.1. Cuvinte şi grupuri sintactice simple

Tabelul 6.1 oferă exemple relativ la modul în care unele cuvinte şi grupuri sintactice simple pot fi reprezentate sub formă de formule logice, precum şi o modalitate de a reprezenta aceste formule logice în Prolog. Sunt folosite lambda notaţia şi faptul că ^ este asociativ la dreapta, i.e. avem Y^X^formula = Y^(X^formula).

TABELUL 6.1 Reprezentări ale unor cuvinte şi grupuri sintactice simple

Tipul constituentului	Reprezentare logică	Transcrierea în Prolog
Substantiv propriu *Andrei*	Constantă logică *andrei*	`andrei`
Substantiv comun *câine* *profesor*	Predicat unar *(λx)câine(x)* *(λx)profesor(x)*	`X^caine(X)` `X^profesor(X)`
Adjectiv *roşu* *mare*	Predicat unar *(λx)roşu(x)* *(λx)mare(x)*	`X^rosu(X)` `X^mare(X)`
Substantiv însoţit de adjective *câine mare*	Predicate unare legate prin "şi" *(λx)(câine(x)∧mare(x))*	`X^(mare(X),caine(X))`
Grup verbal *latră* *mănâncă pâine*	Predicat unar *(λx)latră(x)* *(λx)mănâncă(x,pâine)*	`X^latra(X)` `X^mananca(X,paine)`
Verb tranzitiv *mănâncă*	Predicat binar *(λy) (λx)mănâncă(x,y)*	`Y^X^mananca(X,Y)`
Grup prepoziţional *cu trăsura*	Predicat unar *(λx)cu(x,trăsura)*	`X^cu(X,trasura)`
Prepoziţie *cu*	Predicat binar *(λy) (λx)cu(x,y)*	`Y^X^cu(X,Y)`

Analizând acest tabel se observă că substantivele proprii sunt constante logice ("Andrei" = *andrei*), în timp ce substantivele comune, ca și adjectivele, sunt predicate ("câine" = *(λx)câine(x)*). A fi un câine, ca și a avea o anumită culoare, reprezintă o *proprietate*. Această diferență în tratarea substantivelor proprii și a celor comune este legată de deosebirea dintre **sens** și **referire**. Astfel, un nume se poate referi la un *unic individ* și, prin urmare, este tradus direct într-o constantă logică. Dar un substantiv comun, cum ar fi *câine*, se poate referi la o *multitudine de ființe diferite*. Traducerea sa, din această cauză, este *proprietatea* pe care indivizii respectivi o au în comun. Referința la *câine* într-un anumit enunț este acea valoare a lui *x* pentru care *câine(x)* devine adevărat.

Remarcăm, de asemenea, faptul că diferite verbe cer un număr diferit de argumente, după cum reprezintă verbe tranzitive, intranzitive etc., fenomen care a fost deja analizat atunci când au fost menționate posibilitățile de subcategorizare ale verbelor. Verbul intranzitiv *latră* este tradus în predicatul unar *(λx)latră(x)*. Un verb tranzitiv este tradus sub forma unui predicat binar (cazul verbului *mănâncă* din tabel). Argumentele acestor verbe sunt completate, pas cu pas, pe măsură ce se progresează de la verb la un constituent de tip VP și apoi la unul de tip S ca în exemplele următoare:

Verb	*mănâncă*	*(λy) (λx)mănâncă(x,y)*
Grup verbal	*mănâncă pâine*	*(λx)mănâncă(x,pâine)*
Propoziție	*Ana mănâncă pâine*	*mănâncă(ana,pâine)*

sau

Verb	*dă*	*(λz) (λy) (λx)dă(x,y,z)*
Grup verbal1	*dă cartea*	*(λy) (λx)dă(x,y,cartea)*
Grup verbal2	*dă cartea elevului*	*(λx)dă(x,elevului,cartea)*
Propoziție	*Ana dă cartea elevului*	*dă(ana,elevului,cartea)*

Unor propoziții le corespund formule care au în structura lor conectori logici, ca în exemplele următoare:

Cuțu și Grivei sunt animale	*animal(cuțu) ∧ animal(grivei)*
Ana nu mănâncă pâinea	*¬mănâncă(ana,pâinea)*

Observăm, de asemenea, că reprezentarea pentru *este un câine* este aceeași ca pentru *câine*. Remarcăm, în final, faptul că logica de ordinul întâi nu este suficientă pentru a reprezenta întreaga bogăție a limbajului natural. Unele situații care nu pot fi reprezentate în maniera descrisă până acum sunt, spre exemplu, cele în care un predicat are ca argument un alt predicat sau cele în care un predicat are ca argument o întreagă propoziție. Astfel de observații

referitoare la limba engleză se fac în [19]. Preluând acele exemple, prezentăm în continuare situațiile cele mai semnificative în care calculul predicatelor de ordinul întâi nu este suficient pentru a prezenta fragmente de text din limbajul natural, cu specială referire la limbile engleză și română:

- predicat care are ca argument un alt predicat

  Max has an unusual property.
  $(\exists p)(p(max) \wedge unusual(p))$

  Max are o proprietate neobișnuită.
  $(\exists p)(p(max) \wedge neobișnuit(p))$

- predicat care are o întreagă propoziție ca argument

  Fido believes Felix is human.
  believs(fido,human(felix))

  Fido crede că Felix este uman.
  crede(fido,uman(felix))

- context în care referirea este blocată

  John is looking for a book.
  respectiv
  Ion caută o carte.
  (Aici nu este clar cum trebuie reprezentat *o carte*. Orice formulă care conține *$(\exists x)$...carte(x)* nu este corectă, întrucât nu este obligatoriu ca această carte să existe.)

Analiza unui text mai amplu în limbaj natural va întâlni, fără îndoială, astfel de siutații. Dacă însă ne limităm la texte al căror conținut (în ceea ce privește informația) poate fi reprezentat sub forma unei baze de date electronice, logica de ordinul întâi va fi, de regulă, suficientă pentru a realiza toate reprezentările necesare.

Pentru a exemplifica cu o *implementare Prolog* relativ simplă, în cele ce urmează ne vom concentra asupra semanticii unui constituent alcătuit dintr-un substantiv comun însoțit de zero sau mai multe adjective, dar fără nici un determinant, pe care îl vom numi *constituentul N'*. (Precizăm că prin *determinant* înțelegem un cuvânt de tipul *o, niște, acei, acele* etc., adică un articol nehotărât ca în *o frumoasă fată săracă*, un articol nehotărât cu funcție adjectivală ca în *niște frumoase fete sărace*, un pronume cu funcție adjectivală

ca în *acele frumoase fete sărace* ș.a.m.d.. În această accepție, *determinantul*[9] desemnează o clasă închisă de cuvinte și/sau de forme gramaticale, a căror caracteristică sintactică este ocurența obligatorie într-un grup nominal, în vecinătatea substantivului, și a căror funcție semantică este cea de determinare, adică de precizare a referentului unui substantiv ca definit, identificabil de către vorbitor, în raport cu unul nedefinit, general.) Un exemplu de constituent de tip N' este

frumoasa fată săracă.

În implementarea semanticii constituentului N' vom începe prin a ține cont de faptul că adjectivele și substantivele proprii sunt traduse sub forma unor predicate unare. Toate predicatele din structura constituentului N' vor fi combinate prin utilizarea simbolului logic $\wedge$ ("și"). În cazul de față, pornind de la

frumoasa = (λx)*frumoasa(x)*
fată = (λx)*fata(x)*
săracă = (λx)*săracă(x)*

dorim să obținem:

frumoasa fată săracă = (λx)(*frumoasa(x)*$\wedge$*fata(x)* $\wedge$*săracă(x)*)

În *notație Prolog*, dorim să combinăm

```
X^frumoasa(X) , X^fata(X) și X^saraca(X)
```

pentru a obține

```
X^(frumoasa(X),fata(X),saraca(X)).
```

În mod crucial, pentru ca implementarea să fie corectă, trebuie să se efectueze *unificarea variabilelor*. Cu alte cuvinte, nu trebuie să se obțină o structură de tipul `(frumoasa(X),fata(Y),saraca(Z))`.
Acest lucru se realizează prin intermediul argumentelor regulilor gramaticilor DC. Iată *intrările lexicale* corespunzătoare unor cuvinte particulare:

```
adj(X^frumoasa(X)) --> [frumoasa].

adj(X^saraca(X)) --> [saraca].

n(X^fata(X)) --> [fata].
```

[9] Pentru detalii, vezi [9], s.v. *determinant, referent*.

În ceea ce privește *regulile PS*, o primă regulă este $N' \rightarrow N$, care este ușor de tratat, întrucât, în acest caz, semantica constituentului **N'** este aceeași cu cea a substantivului:

```
n1(Sem) --> n(Sem).
```

La rândul ei, regula care combină un adjectiv cu un **N'** este următoarea:

```
n1(X^(P,Q)) --> adj(X^P), n1(X^Q).
```

Se observă faptul că această a doua regulă este puternic recursivă. (După combinarea unui adjectiv cu un constituent de tip **N'**, regula poate combina un alt adjectiv cu constituentul **N'** rezultat ș.a.m.d.). Recursivitatea acestei reguli este vizibilă și în Fig. 6.5.

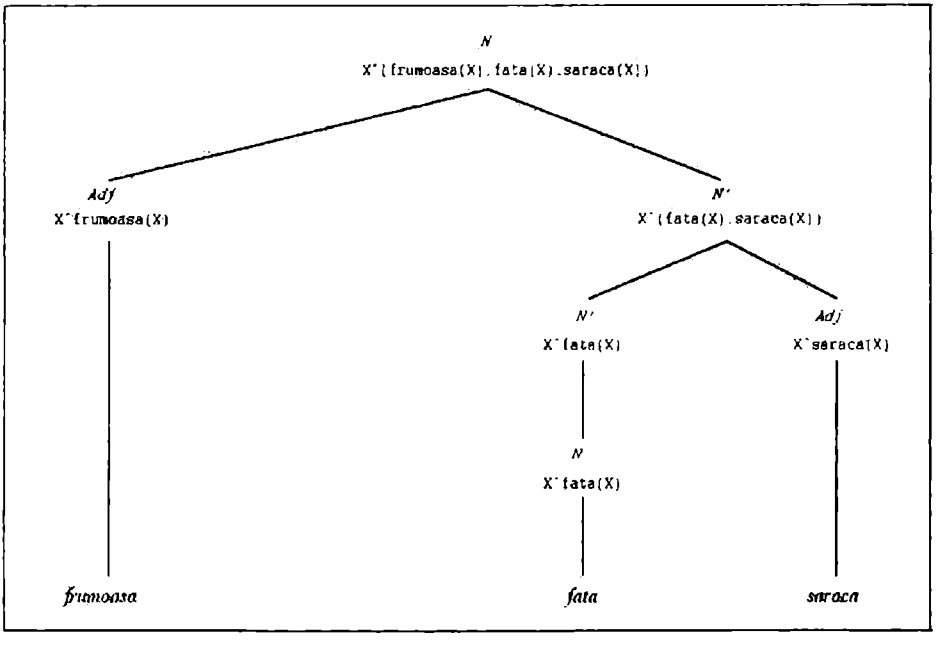


Fig. 6.5 Construirea semanticii unui constituent **N'**

6.4.2. Cuantificatori (Determinanți)

Determinanții din limbajul natural, la care ne-am referit anterior (§6.4.1), corespund **cuantificatorilor** din logica formală. De aceea, *în accepția prezentă*, vom folosi acești termeni ca reprezentând sinonime. Tabelul 6.2 prezintă câteva propoziții conținând cuantificatori, precum și reprezentările lor semantice.

6.4.2.1. Cuantificatori în limbă, în logică și în Prolog

TABELUL 6.2 Structura semantică de ansamblu a propoziției determinată de cuantificatori

Propoziție	Reprezentări
Andrei plânge.	*plânge(andrei)* `plange(andrei)`
Un băiat plânge.	*(∃x)(băiat(x)∧plânge(x))* `un(X,baiat(X),plange(X))`
Fiecare băiat plânge. (cu sensul "*toți* băieții plâng")	*(∀x)(băiat(x)→plânge(x))* `toti(X,baiat(X),plange(X))`
Grivei aleargă o pisică.	*(∃x)(pisica(x)∧aleargă(grivei,x))* `o(X,pisica(X),alearga(grivei,X))`
Grivei aleargă fiecare pisică.	*(∀x)(pisica(x)→aleargă(grivei,x))* `toate(X,pisica(X),alearga(grivei,X))`
Un profesor ceartă un student.	*(∃x)(profesor(x) ∧(∃y)(student(y) ∧ ceartă(x,y)))* `un(X,profesor(X),un(Y,student(Y), cearta(X,Y)))`
Un profesor ceartă fiecare student.	*(∃x)(profesor(x) ∧(∀y)(student(y) → ceartă(x,y)))* `un(X,profesor(X),toti(Y,student(Y), cearta(X,Y)))`
Fiecare profesor ceartă un student.	*(∀x)(profesor(x) → (∃y)(student(y) ∧ ceartă(x,y)))* `toti(X,profesor(X),un(Y,student(Y), cearta(X,Y)))`
Fiecare profesor ceartă fiecare student.	*(∀x)(profesor(x) → (∀y)(student(y) → ceartă(x,y)))* `toti(X,profesor(X),toti(Y,student(Y), cearta(X,Y)))`

După cum se știe și se poate observa și în tabelul anterior, cuantificatorul ∃ de obicei se cuplează cu conectorul ∧, iar cuantificatorul ∀ se cuplează, de regulă, cu conectorul →. În cadrul transpunerilor în Prolog ale formulelor logice conectorii sunt impliciți. O "traducere" de tipul `toti(X,Scop1,Scop2)`

înseamnă că toate valorile lui **x** care satisfac **Scop1** satisfac, în egală măsură, **Scop2**. Un *predicat* care să testeze acest lucru în cadrul interogărilor bazelor de date poate fi definit după cum urmează:

```
toti(_,Scop1,Scop2):-
   \+ (Scop1, \+ Scop2).
```

(i.e. nu există nici o modalitate de a satisface **Scop1** care să nu poată fi extinsă - prin instanțierea mai multor variabile - pentru a satisface și **Scop2**). Se observă că argumentul **x** nu este folosit în mod efectiv. El este inclus aici numai pentru ca **toti** să aibă aceeași structură a argumentelor pe care o vor avea alți cuantificatori, ce urmează a fi definiți în viitor.

Așa cum se arată în [19], problema care se ridică este aceea a tratării situațiilor în care nu există nici o modalitate de a satisface **Scop1**. Într-un astfel de caz, **toti(X,Scop1,Scop2)** reușește, acesta nefiind, de regulă, rezultatul dorit. *În logică* $(\forall x)(p(x) \rightarrow q(x))$ este adevărat atunci când $p(x)$ este întotdeauna fals. Dar, *în limbajul natural*, nu dorim să susținem că *toate merele sunt necoapte* este adevărat atunci când nu există nici un măr în baza de cunoștințe. În consecință, cuantificatorul **toti** trebuie modificat astfel încât să poată testa că există într-adevăr o soluție a lui **Scop1**, după care să urmeze un *cut* care să prevină backtracking-ul inutil. Prin urmare, definiția predicatului **toti** este următoarea:

```
toti(_,Scop1,Scop2):-
     \+ (Scop1, \+ Scop2),
     Scop1,
     !.
```
(6.8)

Prin urmare, *definiția în Prolog a cuantificatorului* $\forall$ *din logică*, căruia îi corespunde *cuantificatorul* ***toți*** al limbajului natural, este dată de (6.8), respectiv de *predicatul* **toți**, definit anterior (6.8).

Definiția în Prolog a *cuantificatorului* ***un*** (corespunzător cuantificatorului $\exists$ din logica matematică) este și mai simplu de stabilit. În acest caz trebuie găsită *o singură soluție* a lui **Scop1** și **Scop2**. Din nou este folosit un *cut* pentru a preveni backtracking-ul inutil. Prin urmare, definiția *predicatului* corespunzător **un** este următoarea:

```
un(_,Scop1,Scop2):-
     Scop1,
     Scop2,
     !.
```
(6.9)

6.4.2.2. Restrictor și domeniu

Așa cum s-a putut observa (6.8), (6.9), în Prolog nu apar conectorii $\wedge$ și $\rightarrow$ (asociați lui $\exists$ și respectiv $\forall$). Implementarea în Prolog tratează fiecare cuantificator ca reprezentând o legătură între o variabilă cuantificată și două scopuri Prolog.

Această idee este dezvoltată în [19], unde sunt luate în considerație următoarele formule:

$(\exists x)(pisica(x) \wedge animal(x))$	"Cel puțin o pisică este un animal".
$(\forall x)(pisica(x) \rightarrow animal(x))$	"Orice pisică este un animal".

Aceste formule pot fi scrise într-un alt mod, dacă adoptăm următoarea *convenție*: $(\forall x{:}pisica(x))$ va însemna "pentru toți x astfel încât x este o pisică". Utilizând această notație, formulele anterioare pot fi scrise sub forma care urmează (și în care conectorii $\wedge$ și $\rightarrow$ nu mai intervin):

$(\exists x{:}pisica(x))animal(x)$	"Cel puțin o pisică este un animal".
$(\forall x{:}pisica(x))\ animal(x)$	"Orice pisică este un animal".

Un exemplu similar îl constituie formulele:

$(\exists x{:}concurent(x))câștigător(x)$	"Cel puțin un concurent este un câștigător".
$(\forall x{:}\ concurent(x))câștigător(x)$	"Orice concurent este un câștigător".

În această nouă notație, *pisica(x)* și respectiv *concurent(x)* se numesc **restrictorul** cuantificatorului $\exists x$ sau $\forall x$. Restrictorul unui cuantificator are rolul de a restricționa mulțimea valorilor lui x pe care cuantificatorul le poate alege. Restul formulei, în cazul nostru *animal(x)* și respectiv *câștigător(x)*, reprezintă **domeniul** cuantificatorului i.e. propoziția care trebuie să fie adevărată pentru valorile adecvate ale lui x.

Din această perspectivă, fiecare *cuantificator* poate fi privit ca reprezentând *o relație* între

- variabilă cuantificată
- domeniu
- restrictor

Mai precis, pentru orice variabilă cuantificată x, **cuantificatorul** este o *relație* între mulțimea valorilor lui x care satisfac restrictorul și mulțimea valorilor lui x care satisfac atât restrictorul, cât și scopul.

Plecând de la această definiție a cuantificatorului se pot concepe diverși cuantificatori generalizați, esențiali în analiza limbajului natural, cum ar fi:

- *(doi x)*, care înseamnă că exact două dintre valorile lui *x* care satisfac restrictorul satisfac și domeniul;
- *(majoritatea x)*, care înseamnă că mai mult de jumătate dintre valorile lui *x* care satisfac restrictorul satisfac și domeniul.

Cuantificatorii generalizați pot fi reprezentați *în Prolog* prin intermediul unor predicate cu trei argumente cum ar fi `doi(X,Scop1,Scop2)` `majoritatea(X,Scop1,Scop2)` ș.a.m.d. Observăm că, în aceste cazuri, X trebuie să fie argument, chiar dacă, anterior, el nu era folosit de predicatele **toti** și **un**. Motivul este evident: cuantificatori precum ***doi*** și ***majoritatea*** trebuie să numere valorile lui X care satisfac scopurile și trebuie să se asigure că numără într-adevăr valorile lui X și nu pe cele ale altei variabile.

6.4.2.3. Construcția structurilor cuantificate

Determinanții au o importanță structurală extrem de mare. Ei afectează mult mai mult din structura semantică a unei propoziții decât sugerează structura sintactică a acesteia. Spre exemplu, propoziția

Fiecare profesor îndrumă un student.

are următoarea reprezentare:

```
toti(X,profesor(X),un(Y,student(Y),indruma(X,Y)))
```

Aici constituentul *fiecare profesor* dă naștere unei structuri de tipul `toti`, care conține nu numai reprezentarea grupului nominal (în care intervine *fiecare*), ci și reprezentarea grupului verbal. La nivel sintactic *fiecare* modifică numai *profesor*, pe când la nivel semantic același *fiecare* are un domeniu care se extinde asupra întregii propoziții, fapt ce apare ca evident având în vedere structura de tipul `toti` care este generată. Cuantificatorul ***toți*** domină întreaga reprezentare semantică, chiar dacă, din punct de vedere sintactic, el aparține exclusiv unui NP.

În ceea ce privește *implementarea*, asupra căreia ne concentrăm în cele ce urmează, vom începe prin a remarca faptul că, în ciuda complexității lor, majoritatea structurilor semantice de bază pot fi construite prin unificarea argumentelor în cadrul regulilor gramaticilor DC.

Reprezentarea verbelor și a substantivelor comune va fi aceeași de până acum, ceea ce duce la existența unor intrări lexicale de tipul:

```
n(X^profesor(X)) --> [profesor].
n(X^student(X)) --> [student].
```

```
v(X^indruma(X)) --> [indruma].
```

Reprezentarea semantică a unui determinant va avea forma `(X^Res)^(X^Dom)^Formula`, întrucât, în accepția prezentă, un determinant cuplează un domeniu și un restrictor. Prin urmare, reprezentarea anterioară are sensul următor: fiind date un restrictor și un domeniu, se creează o formulă pe baza acestora. Variabila X apare aici în mod explicit pentru ca variabilele corespunzătoare ale diferiților termeni să poată fi unificate. În acest fel se obțin intrări lexicale corespunzătoare determinanților de tipul următor:

```
det((X^Res)^(X^Dom)^toti(X,Res,Dom)) --> [fiecare].
det((X^Res)^(X^Dom)^un(X,Res,Dom)) --> [un];[o].
```

Reprezentarea semantică a unui determinant fiind cunoscută, se poate trece la ***reprezentarea grupului nominal***. Grupul nominal este privit ca reprezentând același lucru cu un determinant, cu excepția faptului că restrictorul a fost furnizat, ca în structura din Fig. 6.6.

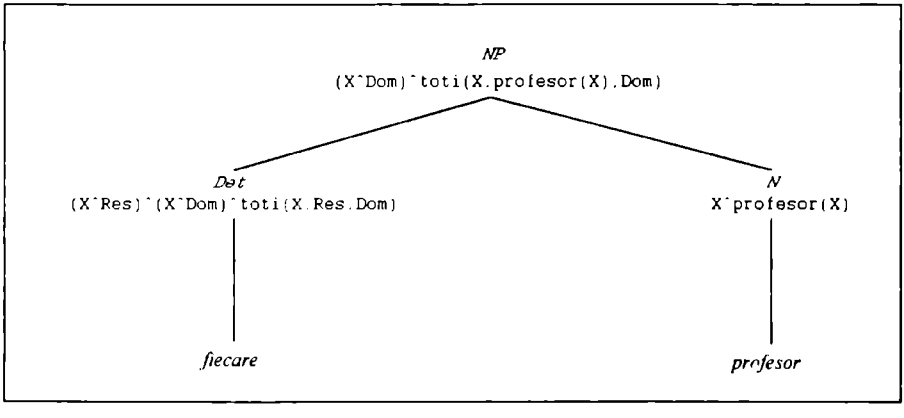


Fig. 6.6 Semantica grupului nominal

Se observă aici că cei doi constituenți Det și NP au reprezentări asemănătoare, cu excepția faptului că un NP așteaptă să-i fie furnizat numai domeniul, nu și restrictorul. Regula gramaticii pentru constituentul NP este:

```
np((X^Dom)^Pred) --> det((X^Res)^(X^Dom)^Pred),n(X^Res).
```

sau, mai concis,

```
np(Sem) --> det((X^Res)^Sem),n(X^Res).
```

În ceea ce privește ***grupul verbal***, există două reguli pentru constituentul VP, după cum verbul în jurul căruia se construiește acesta nu are obiect direct, respectiv există un obiect de tip NP în cadrul grupului verbal.

Dacă verbul nu are obiect direct, atunci semantica grupului verbal este aceeași cu a verbului corespunzător:

```
vp(Sem) --> v(Sem).
```

Dacă în structura grupului verbal există un obiect direct NP, atunci regula corespunzătoare a gramaticii este:

```
vp(X^Pred) --> v(Y^X^Dom),np((Y^Dom)^Pred).
```

Aceasta legitimează structuri de tipul celei din Fig. 6.7.

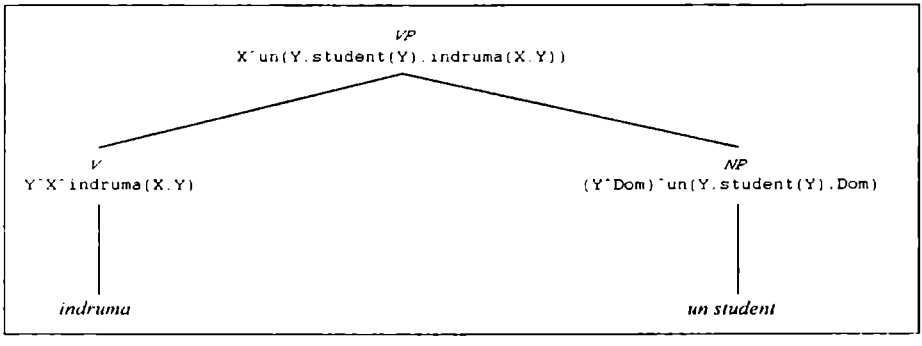


Fig. 6.7 Semantica grupului verbal

Regula pentru construcția unei întregi propoziții preia un grup nominal (care necesită un domeniu) și un grup verbal (care necesită un argument) și le îmbină în felul următor:

```
s(Sem) --> np((X^Dom)^Sem),vp(X^Dom).
```

Aceasta validează structuri de tipul celei din Fig. 6.8.

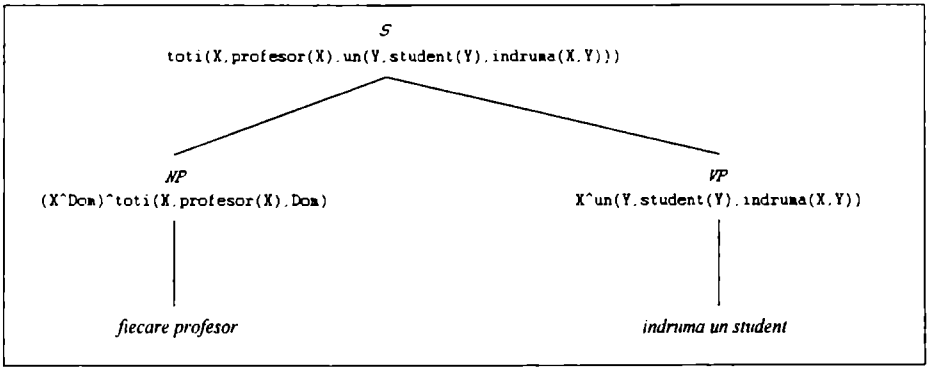


Fig. 6.8 Structura semantică a unui întregi propoziții

Se observă că, în această implementare, preluată din [19], *determinanții* (și respectiv grupurile nominale care îi conțin) controlează structura semantică în ansamblul ei.

O ultimă problemă pe care o ridică această implementare este aceea că numele proprii nu mai pot fi reprezentate sub forma unor constante. Ele trebuie să fie niște structuri care acceptă ca domeniu un VP. Astfel, în loc de

```
np(andrei) --> [andrei].
```

în program se va scrie

```
np((andrei^Dom)^Dom) --> [andrei].
```

Aici `(andrei^Dom)^Dom` este o expresie care primește `X^Dom` de la grupul verbal, instanțiază variabila lambda la valoarea `andrei` și întoarce `Dom` fără nici o altă modificare.

Construcția detaliată a *structurii semantice* a propoziției

Fiecare profesor îndrumă un student.

este înfățișată în Fig. 6.9.

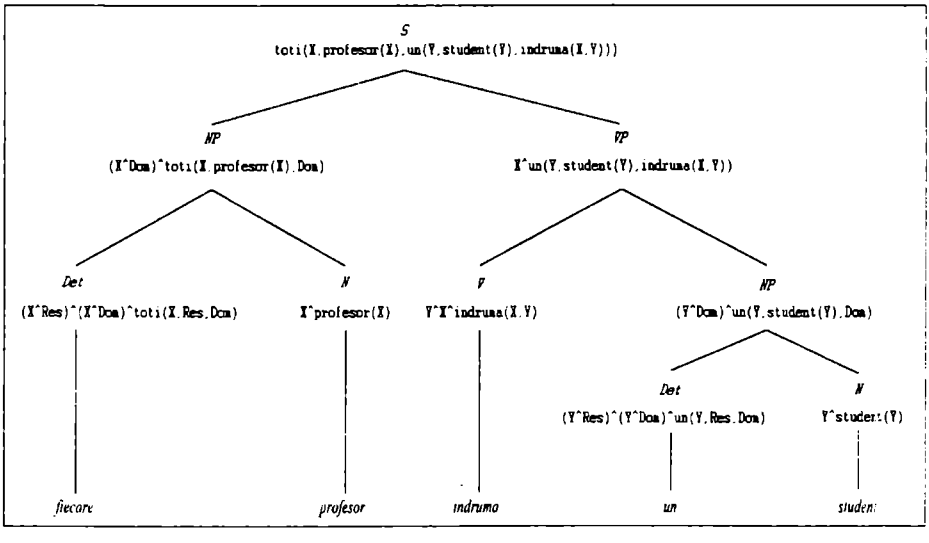


Fig. 6.9. Construirea detaliată a structurii semantice

6.4.2.4. Ambiguități ale domeniului

Propoziția *Un profesor îndrumă fiecare student* este ambiguă. Iată cele două interpretări posibile ale ei:

Un profesor îndrumă fiecare student =

(1) `un(X,profesor(X),toti(Y,student(Y),indruma(X,Y)))`
"Există un profesor care îndrumă toți studenții."

(2) `toti(Y,student(Y),un(X,profesor(X),indruma(X,Y)))`
"Fiecare student este îndrumat de un profesor (nu neapărat de același profesor)."

Prima dintre aceste structuri reprezintă ceea ce generează regulile discutate până în prezent. Cea de-a doua poate fi derivată din prima[10], prin intermediul unei transformări cunoscute în literatura de specialitate [19] sub numele de **ridicarea cuantificatorilor**, în felul următor

```
un(X,profesor(X),toti(Y,student(Y),indruma(X,Y)))
   ⇓
toti(Y,student(Y),un(X,profesor(X),indruma(X,Y)))
```

sau, mai general, conform următoarei scheme de *ridicare a cuantificatorului din domeniu*:

Q1(V1,R1,Q2(V2,R2,S2)) ⇓ Q2(V2,R2,Q1(V1,R1,S2))

(Ridicarea cuantificatorilor din domeniu)

---

[10] Transpunând aceste structuri în limbajul calculului predicatelor, al cărui aspect semantic înseamnă, în esență, studiul semnificațiilor posibile și al valorilor de adevăr ale diverselor afirmații, rezultă că avem de demonstrat următoarea implicație:

$$\exists X_{profesor(X)}\ \forall Y_{student(Y)}\ \text{îndrumă}(X,Y) \Rightarrow \forall Y_{student(Y)}\ \exists X_{profesor(X)}\ \text{îndrumă}(X,Y)$$

Utilizând o notație prescurtată, avem de demonstrat implicația

$$\exists X_{prof(X)}\ \forall Y_{st(Y)}\ \text{înd}(X,Y) \Rightarrow \forall Y_{st(Y)}\ \exists X_{prof(X)}\ \text{înd}(X,Y)$$

Demonstrație:

$\exists X_{prof(X)}\ \forall Y_{st(Y)}\ \text{înd}(X,Y) \Leftrightarrow \exists X(prof(X) \wedge \forall Y(st(Y) \rightarrow \text{înd}(X,Y))) \Leftrightarrow$

$\Leftrightarrow \exists X \forall Y(prof(X) \wedge (st(Y) \rightarrow \text{înd}(X,Y))) \Rightarrow \forall Y \exists X(prof(X) \wedge (\neg st(Y) \vee \text{înd}(X,Y))) \Leftrightarrow$

$\Leftrightarrow \forall Y \exists X(prof(X) \wedge (\neg st(Y) \vee prof(X) \wedge \text{înd}(X,Y))) \Rightarrow \forall Y \exists X(\neg st(Y) \vee prof(X) \wedge \text{înd}(X,Y))$

$\Leftrightarrow \forall Y(\neg st(Y) \vee \exists X prof(X) \wedge \text{înd}(X,Y)) \Leftrightarrow \forall Y(st(Y) \rightarrow \exists X(prof(X) \wedge \text{înd}(X,Y))) \Leftrightarrow$

$\Leftrightarrow \forall Y_{st(Y)}\ \exists X_{prof(X)}\ \text{înd}(X,Y)$ c.c.t.d.

Demonstrații similare se pot face și în cazul ridicării unui cuantificator din restrictor.

unde Q1 și Q2 semnifică cuantificatorii, iar R2 nu conține nici o variabilă cu excepția lui V2.

Este posibilă și ridicarea unui cuantificator din restrictor, ca în exemplul următor:

Cineva care îndrumă fiecare student obosește =

(1) `un(X,toti(Y,student(Y),indruma(X,Y)),oboseste(X))`
"Există cineva care îndrumă toți studenții și obosește."

(2) `toti(Y,student(Y),un(X,indruma(X,Y),oboseste(X)))`
"Pentru fiecare student există cineva care îl îndrumă și obosește."

Schema generală de ridicare a cuantificatorului în acest caz se referă la *ridicarea cuantificatorului din restrictor* și este următoarea:

Q1(V1,Q2(V2,R2,S2),S1) ⇓ Q2(V2,R2,Q1(V1,S2,S1))

(Ridicarea cuantificatorilor din restrictor)

Cele două scheme de ridicare a cuantificatorilor prezentate aici reprezintă de fapt instanțieri ale unui model mult mai general. S-a arătat că aproape orice cuantificator, plasat oriunde în cadrul domeniului sau al restrictorului, poate fi ridicat pentru a-și extinde domeniul asupra întregii propoziții. **Schema generală de ridicare a cuantificatorilor**, preluată din [19], este următoarea:

- Se alege structura cuantificată care trebuie ridicată; fie aceasta `Q(V,R,S)`.
- Se înlocuiește `Q(V,R,S)` cu `S`. Formula rezultată se notează cu `F`.
- Rezultatul operației de ridicare a cuantificatorului este atunci `Q(V,R,F)`.

Această schemă generală descrie cum se face ridicarea **unui** cuantificator aflat oriunde în cadrul propoziției. Dar operația de ridicare a cuantificatorilor este recursivă. Prin urmare, pot fi ridicați mai mulți cuantificatori. Literatura de specialitate înregistrează numeroase controverse legate de răspunsul la următoarea întrebare: cât de departe ar trebui să i se permită acestei recursivități să acționeze?

În principiu, o propoziție având *n* cuantificatori poate fi citită (interpretată) în *n!* moduri. În practică nu vor exista niciodată toate aceste interpretări ale aceleiași propoziții, din diverse motive (unele sunt logic echivalente, altele sunt blocate de diverse principii structurale etc.). Majoritatea studiilor întreprinse se referă la limba engleză, problema rămânând una deschisă în cazul marii majorități a limbilor naturale. Printre rezultatele notabile în acest sens,

referitoare la limba engleză, se numără cele ale lui Hobbs și Schieber (1987). Acești autori demonstrează faptul că, în limba engleză, nu este permis ca un cuantificator să fie luat din afara unui NP și plasat între doi cuantificatori care își au originea în cadrul acelui NP [36].

Există, de asemenea, diferențe relativ mari în ușurința cu care diverși cuantificatori pot fi ridicați. Așa cum se remarcă în [19], în limba engleză, pentru care s-au făcut majoritatea studiilor, cuantificatorul *each* poate aproape oricând să fie ridicat; *some, all* și *every* pot fi ridicați, dar de obicei acest lucru nu este necesar; *any* are tendința de a nu putea fi ridicat; numeralele, în general, nu sunt ridicate.

Putem face acum observația că operația de ridicare a cuantificatorului a fost deja folosită, începând cu momentul discutării *importanței structurale a determinanților*. În Fig. 6.9, de pildă, cuantificatorul *toți* domină reprezentarea semantică a întregii propoziții, chiar dacă, din punct de vedere sintactic, el aparține doar primului grup nominal.

Fig. 6.9 se bazează pe ideea că *un cuantificator trebuie ridicat din structura unui grup nominal individual pentru a-și extinde domeniul asupra întregii propoziții*. Așa cum se observă în [19], are, prin urmare, sens construirea unei reprezentări semantice inițiale în care cuantificatorii se găsesc în interiorul grupurilor nominale, construcție urmată de ridicarea cuantificatorilor până la un nivel la care dau naștere unor formule logice corect formate, eventualele operații ulterioare de ridicare a cuantificatorilor fiind opționale.

Un rezultat fundamental privitor la ridicarea cuantificatorilor din interiorul grupurilor nominale, referitor la limba engleză, este cel obținut de Hobbs și Schieber (1987). Aceștia propun [36] un algoritm care realizează ridicarea cuantificatorilor din interiorul grupurilor nominale generând *toate* și *numai* acele posibilități de existență a domeniului care sunt structural posibile în limba engleză.

6.4.3. Răspunsul la interogări

Ca *o aplicație* la implementările în Prolog de natură semantică realizate până în prezent, ne vom referi la modalitatea de a răspunde unor întrebări simple de tipul da/nu, atunci când este dată o bază de cunoștințe. Este vorba despre întrebări precum

A îndrumat fiecare profesor un student?

Pentru a putea răspunde unor astfel de întrebări este nevoie doar să schimbăm regulile gramaticii astfel încât întrebările să poată fi analizate sintactic. În acest scop, regula referitoare la constituentul S

```
s(Sem) --> np((X^Dom)^Sem),vp(X^Dom).
```

se va înlocui cu una adecvată pentru întrebări. Pentru limba engleză, o astfel de regulă ar putea fi:

```
s(Sém) --> [did],np((X^Dom)^Sem),vp(X^Dom).
```

Ne propunem să răspundem întrebării englezeşti

*Did every dog **chase** a cat?*
(A alergat fiecare câine o pisică?)

printr-o formulare ca

*Yes, every dog **chased** a cat.*
(Da, fiecare câine a alergat o pisică.)

Pentru aceasta, este nevoie şi de adăugarea formelor verbale de tipul

```
v(Y^X^chased(X,Y)) --> [chased];[chase].
```

pentru verbele tranzitive sau de tipul

```
v(X^meowed(X)) --> [meowed];[ meow].
```

în cazul verbelor intranzitive.

În acest cadru, propoziţia

Did every dog chase a cat?

se traduce direct în interogarea

```
?-all(X,dog(X),some(Y,cat(Y),chased(X,Y))).
```

căreia i se poate da răspuns în cazul existenţei unei *baze de cunoştinţe* adecvate.

Pentru a răspunde unor întrebări care conţin pe **care** sau **câţi** (**which, how many**) este necesară o modalitate de a obţine o *listă de soluţii* corespunzătoare unei interogări în Prolog. În acest sens, sunt posibile două abordări [19], la care ne vom referi în continuare.

O primă abordare constă în folosirea predicatului **setof**. Predicatul predefinit **setof** întoarce o listă cu toate soluţiile unei interogări. Mai exact,

```
?-setof(X,Scop,L).
```

va instanţia `L` la o listă a tuturor valorilor lui `X` care intervin în soluţiile referitoare la `Scop`. Această listă este sortată alfabetic, iar duplicatele sunt înlăturate din cadrul ei. Spre exemplu:

```
student(andrei).
```

```
student(maria).

?-setof(X,student(X),L).
L=[andrei,maria]
```

Problema acestei abordări rezidă în aceea că, dacă **Scop** conține o variabilă diferită de **X**, atunci **setof** va întoarce numai soluțiile în care acea variabilă are o anumită valoare. În cursul procesului de backtracking, **setof** va permite apoi acelei variabile să primească o altă valoare ș.a.m.d.. Spre exemplu, având baza de cunoștințe

```
f(a,c).
f(b,c).
f(c,d).
f(d,d).
```

obținem:

```
?-setof(X,f(X,Y),L).
L=[a,b]
Y=c?;
L=[c,d]
Y=d?;
no
```

Ceea ce se dorește este obținerea tuturor valorilor lui **X** într-o unică listă. Altfel, vor exista interogări la care nu se va putea răspunde, cum ar fi: câți câini aleargă după orice pisică (nu neapărat aceeași pisică). Rezolvarea acestei probleme este următoarea: dacă se scrie

```
?-setof(X,Term^Scop,L).    % aici ^ nu desemnează lambda
```

atunci **setof** lucrează ca și înainte, cu excepția faptului că toate variabilele din **Term** iau toate valorile posibile. În cazul bazei de cunoștințe anterioare, avem:

```
?-setof(X,Y^f(X,Y),L).
L=[a,b,c,d]?;
no
```

În mod evident, dorim ca toate variabilele lui **Scop** să fie tratate în acest fel, din care cauză vom scrie **Scop^Scop**. Vom îngloba această tehnică în definiția unui nou predicat, numit **soluții**:

```
% solutii(X,Scop,Lista)
% Intoarce, in Lista, toate valorile lui X care satisfac Scop.
% Variabilele libere din Scop iau toate valorile  posibile.
% Lista nu contine duplicate.
```

```
Solutii(X,Scop,Lista):-
      setof(X,Scop^Scop,Lista).          % aici ^ nu desemnează lambda
```

În cazul exemplului anterior, interogarea Prologului s-ar face astfel:

```
?-setof(X,f(X,Y)^f(X,Y),L).
L=[a,b,c,d]?;
no
```

În cazul acelor variante de Prolog care nu conțin predicatul predefinit **setof** sau care nu acceptă tehnica folosirii lui `Scop^Scop`, există o *a doua abordare* posibilă. Predicatul încorporat **findall** lucrează ca și **setof**, dar permite variabilelor libere să ia toate valorile posibile, astfel:

```
?-findall(X,f(X,Y),L).
L=[a,b,c,d]?;
no
```

Problema pe care o ridică predicatul **findall** constă în faptul că el nu înlătură duplicatele din lista soluțiilor. (Spre exemplu, dacă numărăm câinii care aleargă după pisici, nu dorim să îl numărăm pe Grivei de două ori doar fiindcă el aleargă după două pisici diferite.) De aceea, vom defini predicatul **soluții** ca mai jos,

```
solutii(X,Scop,Lista):-
      findall(X,Scop,L),
      sterge_duplicate(L,Lista).
```

urmând ca predicatul **sterge_duplicate** să fie la rândul său definit în mod corespunzător de către programator.

Fiind în posesia listei de soluții corespunzătoare unei interogări în Prolog, ne putem ocupa de o clasă aparte de întrebări, și anume cele care încep cu cuvinte precum **care, cine, ce, câți**, cuvinte ce vor fi tratate ca niște *cuantificatori*. (În limba engleză întrebările care încep cu *who, what, which* și *how many* constituie, de asemenea, o categorie aparte, cunoscută și sub denumirea de "WH-questions".) În cele ce urmează, ne vom concentra asupra lui **care** (*which*) și **câți** (*how many*), care au rol de determinanți.

Am menționat deja faptul că vom trata **care** și **câți** ca pe niște cuantificatori. Singura diferență față de analiza anterioară a cuantificatorilor este faptul că, atunci când aceste cuvinte intervin într-o interogare, sistemul Prolog va afișa valorile pe ecran, nu se va limita la testarea valorii de adevăr a unei propoziții:

```
care(X,Res,Dom):-
```

```
        solutii(X,(Res,Dom),L),
        write(L),
        nl.

cati(X,Res,Dom):-
        solutii(X,(Res,Dom),L),
        length(L,N),
        write(N),
        nl.
```

Aceste cuvinte - şi deci întrebările de acest tip - prezintă, de asemenea, *ambiguităţi ale domeniului*. Spre exemplu, propoziţia

Care profesor îndrumă un student?

poate însemna:

- care profesor îndrumă un student anume;
- care profesor îndrumă, în general, un student (fără să intereseze dacă este vorba de un student anume).

În tratarea acestor ambiguităţi se vor folosi aceleaşi tehnici de la tratarea ambiguităţii generate de cuantificatorii **toţi** şi **un**, analizaţi anterior.

Răspunsul la interogări reprezintă numai una dintre aplicaţiile implementărilor în Prolog de natură semantică descrise în §6.4.1 şi §6.4.2. Evident, există numeroase alte aplicaţii. Putem progresa de la a răspunde întrebărilor (atunci când există o bază de cunoştinţe adecvată), până la a construi efectiv o bază de cunoştinţe plecând de la input în limbaj natural. Numeroase alte probleme, a căror analiză este realizată dincolo de sfera teoriei modelelor, se ridică în semantica computaţională (traducerea asistată de calculator, înţelegerea evenimentelor, înţelegerea prin predicţie, structura discursului, generarea limbajului ş.a.). Toate acestea nu fac însă obiectul lucrării de faţă, care are un caracter introductiv. Ele se regăsesc în numeroase lucrări actuale de semantică şi pragmatică computaţionale, majoritatea reprezentând domenii încă deschise în lingvistica computaţională.

6.5. Reţele semantice

Una dintre cele mai utile reprezentări ale cunoştinţelor lexicale este cea sub formă de *reţea semantică*, prezentată aici foarte pe scurt şi numai în linii extrem de generale.

O *reţea semantică* este un graf orientat ale cărui arce leagă vârfuri etichetate. Vârfurile sau nodurile grafului reprezintă sensuri ale cuvintelor sau clase abstracte de sensuri, în timp ce arcele reprezintă legături stabilite între aceste sensuri. Astfel de legături semantice pot fi de diverse tipuri, cum ar fi

cele de natură *subtip* sau cele de tip *parte din* (sau *parte a*), în care obiectele sunt puse în legătură cu părțile lor componente. Există numeroase astfel de ierarhii de tipuri, asupra cărora vom reveni în cele ce urmează. Ierarhia din Fig. 6.10, spre exemplu, prezintă un fragment dintr-o rețea semantică simplă.

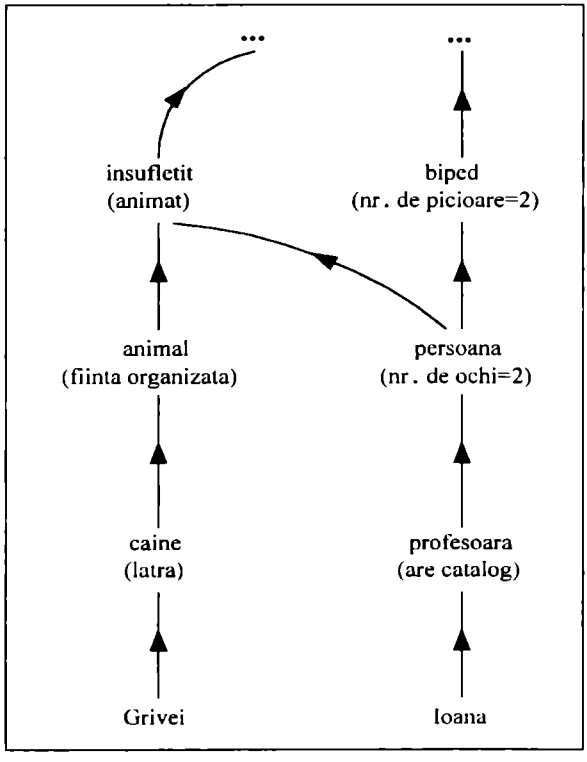


Fig. 6.10 O rețea semantică

Pentru a aborda această metaforă a rețelelor semantice din punct de vedere computațional și pentru a ne putea referi la implementarea în Prolog corespunzătoare, vom începe prin a remarca faptul că diversele informații referitoare la proprietățile și atributele unui individ sau obiect pot fi adesea obținute luându-se în considerație *clasele* cărora individul sau obiectul respectiv le aparține.

Referindu-ne în mod special la implementarea Prolog vom face observația că este mult mai economicoasă, din punctul de vedere al faptelor și al regulilor necesare, exprimarea informației la nivelul claselor (un biped are două picioare) decât în termeni de indivizi (Ioana are două picioare, Andrei are două picioare etc.), ori de câte ori acest lucru este posibil. În plus, este mult mai economicoasă exprimarea informațiilor generale referitoare la modul în care lumea înconjurătoare se organizează în clase de obiecte (toate animalele sunt

însuflețite) decât specificarea tuturor claselor cărora un individ le aparține (Ioana este profesoară, Ioana este o persoană, Ioana reprezintă un biped etc.). Prin urmare, un sistem care face raționamente legate de lumea înconjurătoare va folosi reguli având următoarea formă generală:

```
insufletit(X) daca animal(X)
animal(X) daca caine(X)
biped(X) daca persoana(X)

numar_de_picioare(X,2) daca biped(X)
numar_de_ochi(X,2) daca persoana(X)
are_catalog(X) daca profesoara(X)
```

Faptele și regulile de primul tip descriu *ierarhia* indivizilor și a claselor care alcătuiesc lumea înconjurătoare. Literatura de specialitate numește aceasta o "ierarhie de tip **isa**". *Relațiile semantice* corespunzătoare, adică de tip **isa** se referă la relația dintre un element având o sferă mai largă sau element supraordonat (*hiperonim*) și un element subordonat în sensul de inclus (*hiponim*). Astfel, cuvântul *feroviar* se află într-o relație de tip **isa** cu *ceferist*, iar *paricid* se află în aceeași relație cu *patricid*. Regulile de cel de-al doilea tip din exemplele considerate exprimă felul în care diverse atribute și proprietăți pot să decurgă din apartenența la o anumită clasă. Cel mai adesea este convenabil ca ierarhia de tipul **isa** să fie reprezentată sub forma unui graf orientat, atașându-se informații de cel de-al doilea tip claselor care o alcătuiesc. Rezultatul acestui mod de reprezentare este ceea ce vom numi *rețea semantică*.

O caracteristică esențială a rețelelor semantice este aceea a *moștenirii proprietăților*. În mod tipic aceasta este implementată ca o procedură de căutare în graf, căutare care începe din nodul cel mai caracteristic. Fiind dată, spre exemplu, rețeaua din Fig. 6.10, putem afla câte picioare are Ioana efectuând o căutare ascendentă (de-a lungul săgeților), în cadrul ierarhiei de tip **isa**, până la găsirea unei clase căreia îi este atașată informația **numar_de_picioare**. Acesta este un proces de inferență foarte direcționat, care implică (necesită) căutarea. Stabilirea punctului până la care un sistem de procesare a limbajului natural poate evolua cu succes utilizând această inferență de tip limitat, bazată pe existența claselor, rămâne o problemă deschisă.

În special datorită acestei caracteristici a moștenirii proprietăților rețelele semantice ușurează construcția lexiconului. Spre exemplu, definițiile următoare

```
Macro sin_Vi:
        <cat> = V
        <arg0 cat> = NP
        <arg0 caz> = nom.

Macro sin_Vt:
```

syn_Vi
<arg1 cat> = NP
<arg1 caz> = acc.

Lexem mănâncă:
sin_Vt.

pot fi privite ca introducând rețeaua pentru unități lexicale din Fig. 6.11.

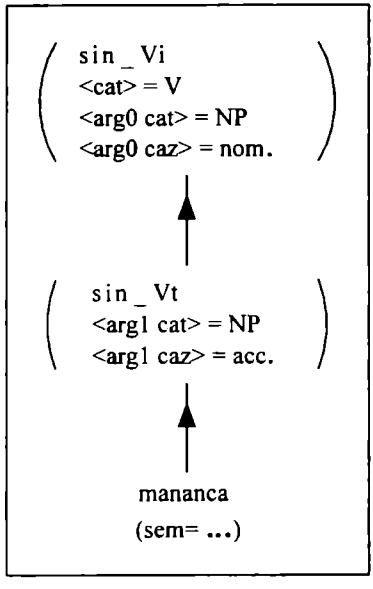


Fig. 6.11 O rețea pentru unități lexicale

O altă caracteristică pe care o prezintă sistemele bazate pe rețele semantice este indexarea informației după *obiectele* care intervin și nu în funcție de relațiile existente, această idee a organizării informației în jurul obiectelor fiind cea care se află și la baza paradigmei atât de cunoscute a programării orientate pe obiecte. După cum se știe, multe sisteme bazate pe logică își organizează informația în funcție de relațiile care intervin, astfel încât, de pildă, toate regulile care permit tragerea unor concluzii referitoare la **numar_de_picioare** sunt păstrate într-o anumită zonă, cele referitoare la **numar_de_ochi** într-o alta ș.a.m.d. În plus, este prevăzută o modalitate eficientă de a ajunge de la numele unei relații la grupul de informații referitoare la respectiva relație. Această manieră de organizare a informației este adecvată inferenței care trebuie să răspundă unor întrebări de tipul "Cine are două picioare?", întrucât întreaga informație relevantă referitoare la acest subiect este grupată și accesibilă, dar în mod evident nu este convenabilă atunci când se pun întrebări de genul "Ce știți despre Maria?"

Implementarea în Prolog a unei rețele semantice simple ca cea din Fig. 6.10 este directă. Pentru a realiza această implementare vom introduce predicatul **atr(Entitate, Atribut, Valoare)** prin intermediul căruia vom stipula ce valoare au diverse atribute particulare în cazul unor entități particulare. În cadrul acestui sistem simplificat nu vom face nici o distincție între indivizi și clase, ambele fiind considerate entități. Atunci când un atribut reprezintă o proprietate pe care o entitate s-ar putea să o aibă sau nu, vor fi folosite în mod corespunzător valorile **da** și **nu**, după cum urmează:

```
atr(membru_club,sex,feminin).
atr(asociat,membru_asociat,da).
atr(asociat,cetatenie,non_ROM).
atr(membru_pe_viata,membru_pe_viata,da).
atr(membru_pe_viata,cetatenie,'ROM').
atr(ioana,peste_30,nu).
atr(andrei,peste_30,da).
```

Efectul predicatului **atr** va fi augmentat prin introducerea predicatului **isa**, care leagă entitățile existente într-o ierarhie, în felul următor:

```
isa(asociat, membru_club).
isa(membru_pe_viata,membru_club).
isa(ioana,asociat).
isa(andrei,asociat).
```

Componenta finală este predicatul **are_atr**, care definește când o entitate poate fi considerată ca având asociată o anumită pereche de tipul (*atribut,valoare*). Acest fapt poate avea loc în una din următoarele două situații:

- o pereche de tipul (*atribut,valoare*) este indicată în mod direct prin intermediul predicatului **atr**:

```
are_atr(Entitate,Atribut,Valoare):-
    atr(Entitate,Atribut,Valoare).
```

- entitatea în cauză este legată, prin intermediul predicatului **isa**, de o altă entitate, care, la rândul ei, are asociată respectiva pereche (*atribut,valoare*):

```
are_atr(Entitate1,Atribut,Valoare):-
    isa(Entitate1,Entitate2),
```

```
are_atr(Entitate2,Atribut,Valoare).
```

Programul complet, scris în SICStus Prolog și furnizând o rețea pentru testare, este cel care urmează și face referire la un al doilea program, **arata_re.pl**, ce expune totalitatea teoremelor referitoare la rețea. Lăsând în seama cititorului scrierea corectă a codului **arata_re.pl**, vom spune că, pe baza Programului 6.1, se pot trage concluzii referitoare la *Ioana*, constând în faptul că aceasta este membru asociat, este de sex feminin, nu are peste 30 de ani și nu are cetățenie română. Concluzii similare se pot trage referitor la *Maria*: are peste 30 de ani, este membru asociat, este de sex feminin, nu este de cetățenie română.

Programul 6.1

```
%
?-reconsult('arata_re.pl').
%
% Membru_club
   atr(membru_club,sex,feminin).

% Asociat
   isa(asociat,membru_club).
      atr(asociat,membru_asociat,da).
      atr(asociat,cetatenie,non_ROM).

% Membru_pe_viata
   isa(membru_pe_viata,membru_club).
      atr(membru_pe_viata,membru_pe_viata,da).
      atr(membru_pe_viata,cetatenie,'ROM').

% Ioana
   isa(ioana,asociat).
      atr(ioana,peste_30,nu).

% Maria
   isa(maria,asociat).
      atr(maria,peste_30,da).

are_atr(Entitate,Atribut,Valoare):-
   atr(Entitate,Atribut,Valoare).
are_atr(Entitate1,Atribut,Valoare):-
   isa(Entitate1,Entitate2),
```

```
are_atr(Entitate2,Atribut,Valoare).
```

Implementarea discutată aici a fost propusă în [29] şi lucrează corect atunci când este furnizată o reţea care respectă următoarele reguli:

(1) Nu există cicluri de tip **isa** i.e. reţeaua este un DAG.
(2) Nu există nici o entitate care să aibă specificată, corespunzător unui atribut, o valoare specificată şi în cazul unui vârf ancestral din cadrul DAG-ului.
(3) Ori de câte ori există o pereche de entităţi astfel încât nici una nu reprezintă un descendent al celeilalte, dar având un descendent comun, nu există nici un atribut pentru care ambele entităţi să specifice o valoare.
(4) Nu există nici o entitate care să fie asociată local cu mai mult de o valoare pentru un atribut dat.

Existenţa reţelelor semantice (şi nu numai aceasta) a dat naştere la numeroase discuţii privind aşa-numita inferenţă plauzibilă sau implicită. *Inferenţele plauzibile* pot fi privite ca reprezentând acele reguli de inferenţă care sunt verificate în mod implicit. Aşa cum se remarcă în [29], sistemele bazate pe reţele semantice implementează în mod tradiţional un anumit tip de *inferenţă implicită*, prin faptul că ele permit ca informaţia specificată în mod direct relativ la o anumită entitate să prevaleze asupra oricărei informaţii pe care acea entitate ar putea-o moşteni.

Implementarea Prolog prezentată anterior este a unei reţele semantice care se bazează pe moştenirea proprietăţilor. Ea poate fi uşor modificată astfel încât informaţia corespunzătoare nodurilor aflate la niveluri inferioare să prevaleze asupra celei moştenite de la nodurile corespunzătoare unor niveluri superioare. În acest sens, trebuie modificată definiţia predicatului **are_atr**, astfel încât să se verifice faptul că nu a fost disponibilă pentru atribut nici o informaţie provenind de la nodul aflat la un nivel inferior (informaţie care ar prevala asupra celei moştenite):

```
are_atr(Entitate,Atribut,Valoare):-
    atr(Entitate,Atribut,Valoare).
are_atr(Entitate1,Atribut,Valoare):-
    isa(Entitate1,Entitate2),
    are_atr(Entitate2,Atribut,Valoare),
    nelocal(Entitate1,Atribut).
```

Verificarea la nivel local este realizată de predicatul **nelocal**, care, în cazul existenţei unei valori locale, trebuie să eşueze şi să rămână în această stare:

```
nelocal(Entitate,Atribut):-
    atr(Entitate,Atribut,_),!,
    fail.
```

În caz contrar, predicatul trebuie să înregistreze succes:

```
nelocal(_,_).
```

Aceasta completează modificările necesare pentru a converti *moştenirea "absolută"* în ceea ce am numit *moştenire implicită*. În mod evident, condiţia (2), enunţată anterior corespunzător reţelelor în care moştenirea este strictă sau absolută, nu se mai aplică în acest caz. În schimb, condiţiile (3) şi (4) rămân valabile pentru a garanta unicitatea valorilor atributelor.

Referindu-ne la implementarea în Prolog a unor reţele semantice simple am luat în consideraţie ierarhia clasică de tip **isa**. O altă ierarhie importantă este cea de tip **parte din** (sau **parte a**), în care obiectele sunt puse în legătură cu părţile lor componente. Pentru a recunoaşte astfel de legături semantice şi pentru a realiza dezambiguizarea pe baza lor, este necesară codificarea informaţiei referitoare la structura obiectelor care fac parte din reţeaua semantică. Aceasta se realizează prin introducerea în graf a unei noi legături, care codifică acest tip de relaţie. Arcele grafului orientat pot fi, prin urmare, la rândul lor etichetate, indicând diferite tipuri de legături semantice. Un sistem de reprezentare complet trebuie să poată indica dacă o parte componentă reprezintă un obiect unic (mânerul unei uşi) ori o mulţime de obiecte (camerele unei case). În plus, sistemul trebuie să reprezinte diverse tipuri de relaţii care pot exista între părţile componente, cum ar fi relaţiile de tip spaţial. Toate aceste aspecte sunt strâns legate de un alt domeniu, acela al reprezentării cunoştinţelor. Datorită diverselor ierarhii de tipuri existente, se pot imagina reprezentări alternative [4] ale unor reţele semantice mai complexe, cum ar fi cea din Fig. 6.12. Aici reţeaua semantică rămâne un graf orientat, dar în care a fost introdus un nou tip de nod, aşa-numitul **nod existenţial**, desemnat printr-un pătrat. Nodul existenţial reprezintă o valoare particulară. Tipul nodului existenţial nu este indicat sub formă de etichetă ci este codificat prin utilizarea unui arc ce desemnează o legătură semantică de tip **isa**. Celelalte legături semantice care intervin sunt de tip **parte din** (notate **PD**).

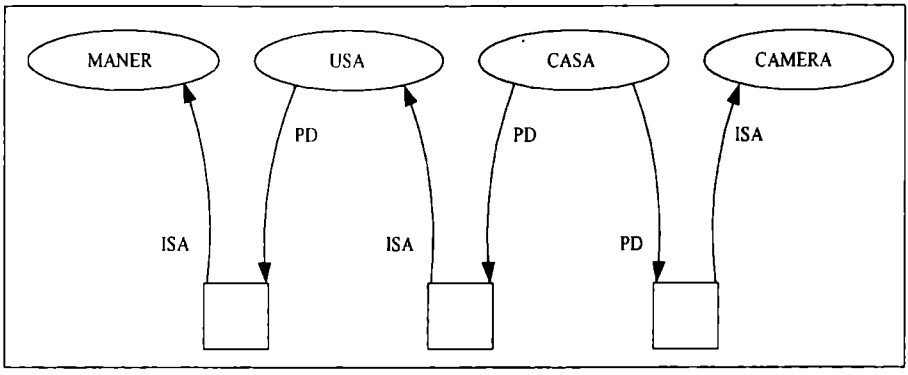


Fig. 6.12 Câteva legături de tip **parte din (PD)**

Avantajele oferite de rețelele semantice în reprezentarea cunoștințelor lexicale sunt multiple. Printre acestea remarcăm din nou faptul că rețelele semantice ușurează *construcția lexiconului*, permițând moștenirea proprietăților. În același timp, ele furnizează o mulțime foarte bogată de legături între sensurile cuvintelor, ceea ce facilitează *dezambiguizarea*. Aceasta din urmă este, de regulă, realizată printr-o reprezentare ierarhică a sensurilor cuvintelor, cel mai adesea captată prin intermediul rețelelor semantice. Folosindu-se astfel de ierarhii pot fi definite restricții selecționale și pot fi utilizate aceste constrângeri pentru a reduce numărul de sensuri posibile ale unui cuvânt. Un exemplu celebru de rețea semantică este WordNet[11], datorită căreia *procesarea cunoștințelor* (în limba engleză) a dobândit noi dimensiuni.

---

[11] WordNet reprezintă în primul rând o *bază de date lexicală interactivă*, dezvoltată în ultimii 15 ani, pentru limba engleză, la Universitatea Princeton, de către un grup de cercetători condus de profesorul George Miller. În același timp, WordNet poate fi privită ca un *dicționar semantic*, deoarece cuvintele sunt localizate pe baza *afinităților conceptuale* cu alte cuvinte, spre deosebire de cazul dicționarelor clasice, unde cuvintele sunt ordonate alfabetic. Deși este similară unui tezaur, WordNet este mult mai utilă inteligenței artificiale, întrucât este înzestrată cu o bogată mulțime de relații între cuvinte și sensuri ale cuvintelor.

WordNet conține majoritatea substantivelor, verbelor, adjectivelor și adverbelor limbii engleze, organizate în mulțimi de sinonime numite **synset**-uri. Fiecare *synset* reprezintă un *concept*.

Prin urmare, spre deosebire de dicționarele alfabetice standard, care organizează vocabularul folosind similarități morfologice, WordNet structurează informația lexicală în termeni de sensuri ale cuvintelor. WordNet face corespondența dintre formele tip ale cuvintelor și sensurile acestora utilizând categoria sintactică ca parametru. Astfel, cuvintele aparținând aceleiași categorii sintactice care pot fi folosite pentru a exprima același înțeles sunt grupate într-un același synset. Cuvintele polisemantice aparțin mai multor synset-uri. Spre exemplu, cuvântul englezesc *computer* are două sensuri definite

în WordNet, ceea ce face ca el să aparţină la două synset-uri diferite, după cum urmează:

(1) {computer, data processor, electronic computer, information processing system}

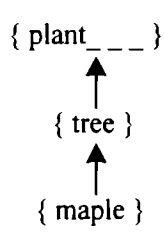
şi

(2) {calculator, reckoner, figurer, estimator, computer}.

În versiunea sa curentă (versiunea 1.6), WordNet conţine 129509 cuvinte organizate în 99643 synset-uri, reţeaua utilizând un număr de 229152 noduri. Cuvintele şi conceptele sunt legate între ele prin *relaţii semantice*. Există în total 299711 asemenea relaţii.

Relaţiile semantice se stabilesc între cuvinte, între cuvinte şi synset-uri, precum şi între synset-uri. Fiecare cuvânt ţinteşte către unul sau mai multe synset-uri, fiecare dintre acestea corespunzând unui anumit sens al cuvântului respectiv. Prin urmare, diferite cuvinte pot ţinti către un acelaşi sens (synset). Bogăţia mulţimii de relaţii stabilite între synset-uri este ceea ce face ca reţeaua semantică WordNet să fie atât de puternică şi de interesantă pentru diverse tipuri de aplicaţii. Exemple de relaţii semantice existente în WordNet sunt **sinonimia** (*synonymy*), folosită pentru a forma synset-urile, **hiperonimia** (*hypernymy*) şi **hiponimia** (*hyponymy*), corespunzând relaţiei de tip *isa* şi respectiv relaţiei inverse (*reverse isa*), **meronimia** (*meronymy*), corespunzând relaţiei *parte-din*, relaţia **cauzală** referitoare la verbe şi altele.

O importanţă deosebită este ataşată relaţiilor de hiperonimie şi hiponimie ca relaţii între synset-uri. Un exemplu de o astfel de relaţie semantică în WordNet este următorul:



isa (sau hiperonimie)

Cu ajutorul relaţiei de hiperonimie (sau de tip isa) conceptele de substantiv şi de verb sunt structurate sub formă de ierarhii. Cele de adjectiv şi de adverb au o structură diferită (*cluster*). În WordNet există 11 ierarhii substantivale şi 512 ierarhii verbale. Semantica relaţiei de tip isa permite unui concept să moştenească toate proprietăţile hiperonimelor sale. În plus, proprietăţile tipice ale unui concept sunt enunţate sub formă de glosă ataşată fiecărui concept în parte. Fiecare glosă include o definiţie, una sau mai multe explicaţii suplimentare şi unul sau mai multe exemple.

WordNet reprezintă o bază de date lexicală a limbii engleze care a fost adoptată pe scară largă pentru o întreagă varietate de *aplicaţii practice* atât din domeniul inteligenţei artificiale, cât şi din cel al procesării limbajului natural. Mulţi cercetători care utilizează WordNet, în special în domeniul inteligenţei artificiale, consideră că aceasta reprezintă o bază de cunoştinţe lexicală şi o valorifică ca atare. *Procesarea cunoştinţelor* a dobândit noi dimensiuni în S.U.A. datorită existenţei WordNet. În acelaşi timp, comunitatea ştiinţifică internaţională se arată extrem de interesată de dezvoltarea unor baze de date lexicale de tip WordNet pentru cât mai multe limbi, în încercarea de a crea *o infrastructură ontologică uniformă*. Astfel, întrucât mulţimea de bază a relaţiilor care

leagă între ele conceptele rămâne aceeaşi, indiferent de limbă, algoritmii de inferenţă pentru extragerea informaţiei pot rămâne aceiaşi.

Posibilele aplicaţii ale WordNet în cele mai variate domenii (regăsirea informaţiei, extragerea informaţiei, dezambiguizarea, generarea limbajului natural, învăţarea, dicţionarele electronice, achiziţia de cunoştinţe ş.a.) sunt citate în peste 300 de lucrări ştiinţifice. În ultimii ani a apărut şi interesul pentru efectuarea de inferenţă statistică pe baza WordNet. Reţeaua WordNet este public disponibilă:

http://www.cogsci.princeton.edu/~wn/

Este de menţionat faptul că, la mijlocul anilor `90, datorită multiplelor aplicaţii dezvoltate pe baza WordNet, a fost puternic resimţită nevoia de a se crea baze de date asemănătoare şi pentru alte limbi, în special pentru cele europene. Un imens efort ştiinţific şi financiar a fost lansat în Europa Occidentală, pentru a se crea aşa-numita **EuroWordNet**, utilizând varianta americană WordNet ca model. Acest efort ştiinţific s-a concretizat în anul 1996, în cadrul proiectului de cercetare - dezvoltare "EuroWordNet", sub conducerea Universităţii din Amsterdam. În prezent există câte o bază de date lexicală de tip WordNet pentru limbile daneză, italiană şi spaniolă (fiecare aflată în continuă îmbunătăţire) şi se lucrează la unele similare pentru limbile germană, franceză şi estoniană. Tot în prezent se pune problema creării unor astfel de baze de date lexicale interactive pentru limbile din Europa Centrală şi de Est, folosindu-se varianta WordNet a limbii engleze ca model şi adaptând-o specificului fiecărei limbi în parte.

ANEXA1. Limbajul de bază al formei logice

- **Atomii** sau **constantele** reprezentării sunt date de sensurile cuvintelor.

- **Termenii** sunt acele constante care descriu obiecte, inclusiv obiecte abstracte, cum ar fi evenimente sau situații.

- **Predicatele** sunt acele constante care descriu relații și proprietăți.
  Exemplu: Fido este un câine.
  CAINE1 - predicat
  FIDO1 - termen
  (CAINE1 FIDO1)
- Predicatele care pot avea un singur argument se numesc *predicate unare* sau *proprietăți.*
- Predicatele cu două argumente se numesc *predicate binare.*
- Predicatele cu *n* argumente se numesc *predicate n-are.*
  Exemplu: predicat binar
  Maria iubește pe Ion.
  (IUBEȘTE1 MARIA1 ION1)

Observații:
- Substantivele proprii pot fi numai termeni.
- Substantivele comune sunt predicate unare.
- Verbele corespund, de regulă, unor predicate n-are.

- **Operatorii logici** sunt o clasă specială de constante cu care se construiesc propoziții complexe. *Exemple* de operatori logici sunt:
  Operatori logici în FOPC: V, &, →
  Operatori logici în limba română: sau, și, dacă, numai dacă
  Operatori logici în limba engleză: or, and, if, only if, not
  Exemple:
  Sue does not love Jack.
  (NOT(LOVES1 SUE1 JACK1))
  Jack loves Sue or Jack loves Mary.
  (OR(LOVES1 JACK1 SUE1)(LOVES1 JACK1 MARY1))

- **Cuantificatori** - *Exemple*:
  Cuantificatori în FOPC: ∀ și ∃ *Exemplu*: ∀xP(x) înseamnă "P(x) adevărat pentru orice obiect posibil x"; aceasta se verifică arareori în limbajul natural, care folosește **cuantificatori generalizați**.
  Cuantificatori în limba română: toți, unii, majoritatea, mulți, câțiva

Cuantificatori în limba engleză: all, some, most, many, a few, the
The dog barks.
(THE **x** : (DOG1 **x**)(BARKS1 **x**))
Most dogs bark.
(MOST **d1** : (DOG1 **d1**)(BARKS1 **d1**))
Most barking things are dogs.
(MOST **d2** : (BARKS1 **d2**)(DOG1 **d2**))

- **Operatorii predicat** primesc un predicat ca argument și produc un nou predicat. Sunt folosiți pentru tratarea formelor de plural. *Exemplu*:
The dogs bark.
(THE **x** : ((PLUR DOG1) **x**)(BARKS1 **x**))

- **Operatorii modali** sunt folosiți în reprezentarea sensului unor verbe cum ar fi *a crede, a dori. Exemplu*:
Ion crede că Anca este fericită.
(CREDE ION1 (FERICIT ANCA1))

- **Operatorii de timp** (care indică timpul verbal) reprezintă o clasă de operatori modali pentru limbajul natural. *Exemplu*:
Ion o va vedea pe Ana.
(VIITOR(VEDE1 ION1 ANA1))

BIBILIOGRAFIE

1. ABRAMSON, H., DAHL, V., *Logic Grammars*. New York, Springer, 1989.
2. AHO, A. V., SETHI, R., ULLMAN, J. D., *Compilers: Principles, Techniques, and Tools*. Addison - Wesley, 1986.
3. AHO, A. V., ULLMAN, J. D., *The Theory of Parsing, Translation and Compiling*. Englewood Cliffs, NJ, Prentice - Hall, 1972.
4. ALLEN, J. F., *Natural Language Understanding*. Menlo Park, California, Benjamin - Cummings, 1987.
5. ANDERSEN, P., Translation Tools for the CEEC Candidates for EU Membership - an Overview, în *Terminologie et Traduction*, 1, 1998, p. 140 - 166.
6. ATANASIU, A., *Curs de lingvistică matematică*. Editura Universității din București, 1998.
7. AVRAM, M., *Gramatica pentru toți*. Ediția a II-a revăzută și adăugită. București, Editura Humanitas, 1997.
8. BARTON, E. G. Jr., BERWICK, R. C., SVEN RISTAD, E., *Computational Complexity and Natural Language*. Cambridge, MA, MIT Press, 1987.
9. BIDU-VRÂNCEANU, A., CĂLĂRAȘU, C., IONESCU-RUXĂNDOIU, L., MANCAȘ, M. și PANĂ DINDELEGAN, G., *Dicționar general de științe. Științe ale limbii*. București, Editura Științifică, 1997.
10. BLOOMFIELD, L., *Language*. New York, Holt, Rinehart and Winston, 1933.
11. BRATKO, I., *Prolog Programming for Artificial Intelligence*. Second edition. Workingham, Addison Wesley, 1990.
12. BUNT, H., Parsing with Discontinuous Phrase Structure Grammar, în *Current Issues in Parsing Technology*. Edited by Masaru Tomita, Boston, Kluwer Academic Publishers, 1995.
13. CHARNIAK, E., *Statistical Language Learning*. Cambridge, MA, MIT Press, 1993.
14. CHOMSKY, N., *Syntactic Structures*. Haga, Monton, 1957.
15. CHOMSKY, N., *Aspects of the Theory of Syntax*. Cambridge, MA, MIT Press, 1965.
16. CHOMSKY, N., *Lectures on Government and Binding*. Cinnaminson, NJ, Foris Publications, 1981.
17. CORNILESCU, A., *Concept of Modern Grammar*. Editura Universității din București, 1955.

18. COVINGTON, M. A., *GULP 2.0: an extension of Prolog for unification -based grammar*. Research Report AI - 1989 - 01, Artificial Intelligence Programs, The University of Georgia, 1989.
19. COVINGTON, M. A., *Natural Language Processing for Prolog Programmers*. Englewood Cliffs, NJ, Prentice Hall, 1994.
20. *Current Trends in Romanian Linguistics*. Edited by A. Rosetti and S. Golopenția Eretescu, București, Editura Academiei, 1978.
21. DIACONESCU, P., Evoluția noțiunii de morfem și stadiul actual al analizei morfematice, în *Elemente de lingvistică structurală*, București, Editura Științifică, 1967.
22. *DICȚIONARUL EXPLICATIV AL LIMBII ROMÂNE* (DEX). Ediția a II-a, București, Editura Univers Enciclopedic, 1996.
23. EARLEY, J., An efficient context-free parsing algorithm. *Commun. of the ACM* 13, 1970, p. 94 - 102. Retipărit în RNLP.
24. EISNER, J. M., Three new probabilistic models for dependency parsing: an exploration. *Proceedings of COLING-96*, Copenhagen, 1996.
25. GAL, A., LAPALME, G., SAINT-DIZIER, P., SOMERS, H., *Prolog for Natural Language Processing*. Chichester, England, Wiley, 1991.
26. GAZDAR, G., Review article: Finite State Morphology. *Linguistics*, 23, 1985, p. 597 - 607.
27. GAZDAR, G., PULLUM, G. K., Computationally relevant properties of natural languages and their grammars. *New Generation Computing*, 3, 1985, p. 273 - 306.
28. GAZDAR, G., KLEIN, E., PULLUM, G., SAG, I., *Generalized Phrase Structure Grammar*. Oxford, Blackwell, 1985.
29. GAZDAR, G., MELLISH, C., *Natural Language Processing in Prolog: an Introduction to Computational Linguistics*. Wokingham, Addison - Wesley, 1989.
30. *Gramatica limbii române*. Ediția a II-a, revăzută și adăugită (vol. I și II), București, Editura Academiei, 1966.
31. HARBUSCH, K., A Polynomial Parser for Contextual Grammars with Linear, Regular and Context-Free Selectors. *Proceedings of MOLG - Sixth Meeting on the Mathematics of Language*, Orlando, Florida, 1999, p. 323 - 335.
32. HARABAGIU, M. S., MOLDOVAN, D. I., Knowledge Processing on an Extended WordNet, în *WORDNET: An Electronic Lexical Database*. Edites by Christiane Fellbaum, Cambridge, MA, The MIT Press, 1998.
33. HRISTEA, F., On WG syntactic analysis with special reference to Romanian. *Analele Universității București, matematică-informatică*, 1, 1998, p. 59 - 69.
34. HRISTEA, F., POPESCU, M., A Word Grammar approach to syntactic analysis with special reference to Romanian. *Analele Universității București, matematică-informatică, Special Issue*, 1998, p. 101 - 113.

35. HOBBS, J. R., Resolving pronoun references. *Lingua*, 44, 1978, B11-338. Retipărit în RNLP.
36. HOBBS, J. R., SHIEBER, S. M., An algorithm for generating quantifier scopings. *Computational Linguistics*, 13, 1987, p. 47 - 63.
37. HOBBS, J. R. STICKEL, M., APPELT, D., MARTIN, P., Interpretation as abduction. *Artificial Intelligence* 63, 1-2, 1993, p. 69 - 142.
38. HUDSON, R., *Word Grammar*. Oxford, Blackwell, 1984.
39. HUDSON, R., *English Word Grammar*. Oxford, Blackwell, 1990.
40. HUDSON, R., *English Grammar*. London, Rontledge, 1998.
41. IONESCU, E., *Manual de lingvistică generală*. Bucureşti, Editura ALL, 1992.
42. JACKENDOFF, R. S., *X Syntax: a Study of Phrase Structure*. Cambridge, MA, MIT Press, 1977.
43. JACKENDOFF, R. S., *Semantic Structures*. Cambridge, MA, MIT Press, 1990.
44. JOHNSON, M., Features and formulae. *Computational Linguistics* 17, 1991, p. 131 - 153.
45. JOSHI, A., Tree - adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions, în *Natural Language Parsing*. Edited by D. R. Dowty, L. Karttunen, and A. Zwicky, New York, Cambridge U. Press, 1985.
46. KAC, M., Review of Marcus (1980). *Language* 58, 1982, p. 447 - 455.
47. KAY, M., The MIND System, în *Natural Language Processing*. Edited by Randall Rustin, New York, Algorithmics Press, 1973.
48. KAY, M., Algorithm schemata and data structures in syntactic processing. *CSL - 80 - 12*, Xerox Corporation, 1980. Retipărit în RNLP.
49. KAY, M., Parsing in functional unification grammar, în *Natural Language Parsing*. Edited by D. R. Dowty, L. Karttunen, and A. Zwicky, New York, Cambridge U. Press, 1982. Retipărit în RNLP.
50. KIMBALL, J., Seven principles of surface structure parsing in natural language. *Cognition* 2, 1973, p. 15 - 47.
51. KING, M., *Parsing Natural Language*. London, Academic Press, 1983.
52. *Limbaj şi tehnologie*. Editat de Dan Tufiş, Bucureşti, Editura Academiei Române, 1996.
53. LYONS, J., *Introducere în lingvistica teoretică* (traducere de Alexandra Cornilescu şi Ioana Ştefănescu), Bucureşti, Editura Ştiinţifică, 1995.
54. MANECA, C., *Lexicologie statistică romanică*, Editura Universităţii din Bucureşti, 1978.
55. MARCUS, M. P., A computational account of some constraints on language, în *Theoretical Issues in Natural Language Processing - 2*. Edited by D. Waltz, Urbana, Illinois, Association for Computational Linguistics, 1978.
56. MARCUS, M. P., *A Theory of Syntactic Recognition for Natural Language*, Cambridge, MA, MIT Press, 1980.

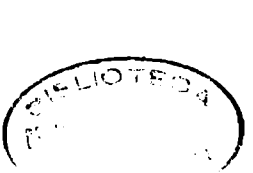
57. MARCUS, S., *Gramatici şi automate finite*. Bucureşti, Editura Academiei, 1964.
58. MARCUS, S., Analytique et génératif dans le linguistique algébrique, în *To Honour Jakobson, Essays on the Occasion of his Seventieth Birthday II*, The Hague, Mouton, 1967.
59. Marcus, S., Linguistique générative, modèles analytiques et linguistique générale, *RRL* 14, 4, 1969, p. 313 - 326.
60. MARCUS, S., Deux types nouveaux de grammaires génératives, *CLTA*, 6, 1969, p. 69 - 74.
61. MARCUS, S., Contextual grammars, *RRMPA* 14, *10*, 1969, p. 1473 - 1482.
62. MARCUS, S., *Poetica matematică*, Bucureşti, Editura Academiei, 1970.
63. MARCUS, S., *Mathematische Poetik* (Bucureşti, Editura Academiei; Frankfurt am Main, Athenäum Verlag), 1973.
64. MARCUS, S., Steps of abstraction in contemporary science, *Noesis* 2, 1974, p. 83 - 87.
65. MARCUS, S., Mathematical and computational linguistics and poetics, în *Current Trends in Romanian Linguistics*. Edited by A. Rosetti and S. Golopenţia Eretescu, Bucureşti, Editura Academiei, 1978.
66. MARCUS, S., Linguistics for programming languages. *RRL - Cahiers de linguistique théorique et appliquée*, 16 (1), 1979, p. 29 - 38.
67. MARCUS, S., Contextual grammars and natural languages, în *Handbook of Formal Languages*, volume 2. Edited by Grzegorz Rozenberg and Arto Salomaa, 1997, p. 215 - 235.
68. MARCUS, S., MARTIN-VIDE, C., PĂUN, Gh., Contextual Grammars as Generative Models of Natural Languages. *Computational Linguistics* 24 (2), 1998, p. 245 - 274.
69. MARTIN-VIDE, C., MATEESCU, A., MIQUEL-VERGÉS, J., PĂUN, Gh., Internal Contextual Grammars: Minimal, Maximal and Scattered Use of Selectors, în Proceedings of the Fourth A Bar-Ilan Symposium on Foundations of Artificial Intelligence, *(BISFAI'95)*, edited by M. Koppel and E. Shamir, 1995, p. 132 - 142.
70. MATSUMOTO, Y., TANAKA, H., HIRAKAWA, H., MIYOSHI, H., YASUKAWA, H., BUP: a bottom-up parser embedded in Prolog. *New Generation Computing* 1, 1983, p. 145 - 158.
71. MEL'ČUK, I. A., *Dependency Syntax: Theory and Practice*. Buffalo, Suny Press, 1987.
72. MOISIL, Gr. C., Probleme puse de traducerea automată. Conjugarea verbelor în limba română scrisă. *SCL* 11, *1*, 1960, p. 7 - 29.
73. MOISIL, Gr. C., Preliminariile traducerilor automate. *LR* 9, *1*, 1960, p. 3 - 27.
74. MOISIL, Gr. C., Problèmes posés par la traduction automatique. La declinaison en roumain écrit. *CLTA* 1, 1962, p. 123 - 134.

75. MOISIL, Gr. C., Problèmes posés par la traduction automatique. La microsyntaxe du verbe roumain. *CLTA* 2, 1965, p. 165 - 189.
76. *Natural Language Parsing*. Edited by David R. Dowty, Lauri Karttunen, and Arnold M. Zwicky, Cambridge, Cambridge University Press, 1985.
77. *Natural Language Understanding and Logic Programming, I*. Edited by Veronica Dahl and Patrick Saint-Dizier, Amsterdam, North-Holland, 1985.
78. *Natural Language Understanding and Logic Programming, II*. Edited by Veronica Dahl and Patrick Saint-Dizier, Amsterdam, North-Holland, 1988.
79. NISTOR-DOMONKOS, E., Les informations grammaticales qui interviennent dans le processus de traduction automatique. *CLTA* 2, 1965, p. 53 - 62.
80. PARTEE, B., MEULEN, A., WALL, R., *Mathematical Methods in Linguistics (corrected 1st ed.)*. Boston, Kluwer Academic Publishers, 1993.
81. PĂUN, Gh., Asupra gramaticilor contextuale. *SCMatem* 26, *8*, 1974, p. 1111 - 1129.
82. PĂUN, Gh., Complexitatea limbajelor contextuale selective. *SCMatem* 27, *5*, 1975, p. 559 - 569.
83. PĂUN, Gh., Contextual Grammars with Restrictions in Derivation. *RRMPA* 22, *8*, 1977, p. 1147 - 1154.
84. PĂUN, Gh., *Gramatici contextuale*. Bucureşti, Editura Academiei, 1982.
85. PĂUN, Gh., On some open problems about Marcus contextual grammars. *International Journal of Computer Mathematics*, 17, 1985, p. 9 - 23.
86. PĂUN, Gh., Marcus contextual grammars. After 25 years. *Bulletin of the EATCS*, 52, 1994, p. 263 - 273.
87. PĂUN, Gh., ROZENBERG, G., SALOMAA, A., Marcus contextual grammars: modularity and leftmost derivation, în *Mathematical Aspects of Natural and Formal Languages*, editat de Gh. Păun, Singapore, World Scientific, 1994.
88. PĂUN, Gh., *Marcus Contextual Grammars*. Boston, Dordrecht, Kluwer, 1997.
89. RATNAPARKHI, A., REYNAR, J., ROUKOS, S., A Maximum Entropy Model for Prepozitional Phrase Attachment, în Proceedings of the Human Language Technology Workshop, *ARPA*, 1994.
90. ROUNDS, W. C., LFP: A logic for linguistic description and an analysis of its complexity. *Computational Linguistics* 14, 1988, p. 1 - 10.
91. SALOMAA, A., *Formal Languages*, New York, Academic Press, 1973.
92. SALOMAA, A., *Theory of Automata*. Pergamon Press, 1969.
93. SHIEBER, S. M., The design of a computer language for linguistic information. *Proc. COLING*, 1984, p. 362 - 366.

94. SHIEBER, S. M., Criteria for designing computer facilities for linguistic analysis. *Linguistics*, 23, 1985, p. 189 - 211.
95. SHIEBER, S. M., Evidence against the non-context-freeness of natural language. *Linguistics and Philosophy*, 8, 1985, p. 333 - 43.
96. SHIEBER, S. M., Using restriction to extend parsing algorithms for complex-feature-based formalisms. *ACL Proceedings, 23rd Annual Meeting*, 1985, p. 145 - 52.
97. SHIEBER, S. M., *An Introduction to Unification-Based Approaches to Grammar*. Chicago, Chicago University Press, 1986.
98. SLATOR, B. M., WILKS, Y., PREMO: Parsing by conspicuous lexical consumption, în *Current Issues in Parsing Technology*. Edited by Masaru Tomita, Boston, Kluwer Academic Publishers, 1995.
99. TESNIERE, L., *Elements de syntaxe structurale*. Paris, Klincksieck, 1959.
100. TUFIŞ, D., Resurse lingvistice computaţionale pentru limba română: trecut, prezent şi viitor, în *Limbaj şi tehnologie*, Editura Academiei Române, Bucureşti, 1996.
101. WILKS, Y., An intelligent analyzer and understander of English. *Communications of the ACM* 18, 1975, p. 264 - 274. Retipărit în GROSZ ET AL. (1986), p. 193 - 203.
102. WINOGRAD, T., *Language as a Cognitive Process: Syntax*. Addison - Wesley, 1983.
103. *WORDNET: An Electronic Lexical Database*. Edited by Christiane Fellbaum, Cambridge, MA, The MIT Press, 1998.

Addenda la BIBLIOGRAFIE

104. GRAMATOVICI, R., An efficient parser for a class of contextual languages. *Fundamenta Informaticae*, 33, 1998, p. 211-238.
105. MARCUS, S., Contextual Grammars and Natural Languages, în *Handbook of Formal Languages, Vol. 2*, G. Rozenberg, A. Salomaa (Eds.), Springer-Verlag Berlin Heidelberg, 1997.



Tiparul executat sub c-da nr. 742/2000 la
Tipografia Editurii Universităţii din Bucureşti

Dincolo de analiza tehnicilor fundamentale ale lingvisticii computaționale şi ale prelucrării limbajului natural, lucrarea îşi propune clarificarea unor concepte şi a unor termeni de bază, precum şi crearea unei culturi generale în domeniu, prezentarea unui scurt istoric al contribuțiilor româneşti şi formarea unei viziuni de ansamblu asupra unui domeniu relativ controversat, în special datorită caracterului său interdisciplinar.

Sunt prezentate teoriile şi tehnicile de bază ale lingvisticii computaționale, în marea lor majoritate concepute şi testate pentru limba engleză şi se fac o serie de observații şi de comentarii privitoare la adaptarea acestora în cazul altor limbi şi, în mod special, al limbii române.

Structurată în şase capitole, lucrarea tratează concepte fundamentale clasice, dar, în egală măsură, abordează câteva dintre cele mai noi teme ale ligvisticii computaționale, aducând discuția în planul celor mai recente preocupări ale domeniului. Unele dintre aplicațiile moderne pe care le aminteşte sau le descrie, în special în cadrul notelor de subsol (proiectul DBR-MAT şi parsing statistic pentru limba română, un parser bazat pe gramatici contextuale, WordNet ca bază de date lexicală interactivă şi ca rețea semantică), constituie subiecte de reflecție şi eventuale teme de cercetare cu vaste posibilități de aprofundare, extindere şi generalizare.

ISBN 973-575-493-2