Analiza semantică stocastică în dialogul om-calculator

Proiect de diplomă

Dan Bohuş

Iunie 2000

Departamentul de Calculatoare
Facultatea de Automatică şi Calculatoare
Universitatea "Politehnica" din Timişoara
România

Conducător proiect:
ş.l. ing. Marian Boldea

In order to understand something,
you mustn't try to understand everything.

Weinberg – Lump Law

Cuvinte cheie: sistem de dialog, analiză semantică stocastică, modele Markov, algoritmul Viterbi, estimare de probabilitate maximă, reestimare Katz, antrenare incrementală

Rezumat

Această lucrare prezintă proiectarea, implementarea, dezvoltarea şi evaluarea unui analizor semantic pentru sisteme automate de dialog vocal om-calculator, parte a unui efort mai larg de cercetare în acest domeniu.

Sistemele de dialog constituie un pas important pe drumul spre realizarea unei interfeţe ideale om-calculator, iar în cadrul lor analiza semantică a limbajului natural joacă un rol fundamental: acela de a extrage şi clarifica informaţia transmisă de utilizator şi de a genera o reprezentare bine formalizată a acesteia.

În urma unui studiu al principalelor alternative de proiectare existente, a fost aleasă varianta unui analizor semantic stocatic bazat pe formalismul case-grammar. Pornind de la arhitectura analizorului semantic proiectat, sunt prezentate succesiv componentele acestuia, atât în privinţa suportului teoretic, cât şi a soluţiilor de implementare.

Sunt ilustrate apoi modul în care implementarea rezultată poate fi folosită şi performanţele posibile, prin intermediul unor experimente vizând dezvoltarea şi evaluarea unui analizor semantic ce va fi integrat într-un sistem de dialog pentru informaţii despre orarul Departamentului de Calculatoare.

În final se prezintă concluzii desprinse din proiectarea, implementarea, dezvoltarea şi evaluarea analizorului semantic, precum şi posibile direcţii de continuare a cercetărilor.

Cuprins

Lista tabelelor

Lista figurilor

Capitolul 1

Introducere

De-a lungul scurtei istorii a calculatoarelor, interfeţele dintre acestea şi utilizatori au cunoscut o evoluţie continuă, marcată de diferite paradigme, dar guvernată permanent de necesitatea de a optimiza lucrul cu calculatorul prin eficientizarea unei componente esenţiale – comunicarea cu acesta.

După panourile de comandă şi comutatoarele caracteristice primelor calculatoare, paradigma care s-a impus în domeniul interfeţelor cu utilizatorul a fost cea a liniei de comandă şi a interfeţelor-consolă. Au urmat apoi interfeţele grafice 2D şi paradigma point-and-click care şi în acest moment sunt dominante. În prezent, cercetările în domeniul interfeţelor bazate pe vorbire au condus deja la realizări impresionante, şi migrarea acestor interfeţe de la stadiul experimental la cel comercial a început deja. Numeroase cercetări se fac şi în domeniul interfeţelor 3D, bazate pe imersiune în realitate virtuală, şi chiar a celor controlate prin gândire.

Cu toate acestea suntem încă departe de realizarea unei interfeţe ideale. O astfel de interfaţă ar trebui însă să fie prin excelenţă una multimodală, adaptată perfect la stilul de comunicare uman. Ea ar trebui să accepte intrări direct în limbaj natural, eventual să sesizeze chiar şi gesturile şi mimica utilizatorului, şi să prezinte ieşirile în forma cea mai adecvată şi mai eficientă pentru perceperea lor de către om: grafică, scrisă, vorbită, animaţie, secvenţe video, sunet etc.

Sistemele automate de dialog vocal om-calculator, denumite în cele ce urmează *sisteme de dialog*, constituie un pas important în această direcţie. Multe din problemele, tehnicile şi soluţiile utilizate în dezvoltarea sistemelor de dialog, precum recunoaşterea vorbirii, analiza semantică, modelarea şi controlul dialogului, sinteza vorbirii, vor fi utilizate în proiectarea unor astfel de interfeţe multimodale. În plus, valoarea comercială intrinsecă a sistemelor de dialog justifică şi stimulează cercetările în acest domeniu.

1.1 Sisteme de dialog

Întrucât nu există o definiţie unanim acceptată a noţiunii de *sistem de dialog*, vom porni de la următoarea caracterizare [**?**]:

> Sistemele interactive bazate pe dialog vor fi caracterizate ca sisteme de calcul ce permit oamenilor să îndeplinească cel puţin o parte din sarcini printr-o anumită formă de comunicare verbală.

Această descriere este destul de largă, incluzând o multitudine de sisteme bazate mai mult sau mai puţin pe comunicare verbală.

O primă categorie o reprezintă sistemele de furnizare de informaţii. Aceste sisteme operează de regulă în domenii bine definite, acţionând practic ca interfeţe către diverse baze de date. Ele controlează destul de rigid dialogul, libertatea de formulare a utilizatorului fiind deci redusă. Această abordare duce însă la obţinerea unor sisteme robuste, cu o rată relativ redusă a erorilor, capabile să satisfacă majoritatea cererilor.

Altă categorie foarte importantă este reprezentată de sistemele ce oferă diverse categorii de servicii: rezevări de bilete pentru avioane, trenuri şi alte mijloace publice de transport, servicii telefonice, bancare etc. Numeroase eforturi s-au îndreptat în această direcţie, întrucât astfel de sisteme au şi o pronunţată valoare comercială.

Paleta este completată de diverse sisteme cu comandă vocală şi de sistemele de traducere automată, deşi în aceste cazuri interactivitatea este aproape nulă.

În tabelul **??** sunt prezentate ilustrativ câteva din realizările importante în domeniul sistemelor de dialog.

Pentru a ilustra şi mai bine performanţele de care sunt capabile unele sisteme de dialog la ora actuală, am inclus în figura **??** un exemplu de dialog purtat cu CMU Communicator, un sistem pentru planificarea călătoriilor şi rezervarea de bilete pe liniile aeriene dezvoltat la Carnegie Mellon University.

Sistem	**Tipul sistemului şi domeniul**
WAXHOLM	Furnizare de informaţii în legatură cu traficul navelor în Stockholm (KTH)
TRAINS	Planificarea călătoriilor cu trenul (University of Rochester)
JANUS/ESST	Traducerea automată a dialogurilor spontane om-om pentru planificarea întâlnirilor (Karlsruhe & CMU)
CMU Communicator	Servicii de planificare şi rezervare de bilete pentru călătorii cu avionul (CMU)
TARSAN	Furnizare de informaţii turistice (rute, hoteluri, muzee, adrese, numere telefonice etc) (Canon)
TOSBURG	Preluare de comenzi într-un fast-food (Toshiba)

Tabelul 1.1: Câteva exemple de sisteme de dialog

```
CMC:  Buna ziua, sunt CMU Communicator. Spuneti-mi numele dumneavoastra.
Alex: Sunt Alex Rudnicky.
CMC:  Buna ziua doctor Rudnicky, ma bucur ca ati apelat din nou. Unde ati
      dori sa calatoriti ?
Alex: As vrea sa zbor la San Francisco maine seara.
CMC:  Zburati la San Francisco maine. Un zbor US Airlines pleaca din
      Pittsburgh la 6:10 PM, si ajunge in San Fran...
Alex: OK. Il iau pe asta.
CMC:  Va si intoarceti la Pittsburgh ?
Alex: Da, as vrea sa ma intorc vineri dupa-amiaza.
CMC:  Zburati vineri, 1 octombrie. Exista un zbor ce pleaca la 1:10 PM si
      soseste in Pittsburgh la 8:53 PM. Vreti sa il luati pe acesta ?
Alex: OK. Il iau pe asta.
CMC:  Aveti nevoie de hotel in San Francisco ?
Alex: Fara hotel.
CMC:  Aveti nevoie de o masina...
Alex: Da, te rog.
CMC:  Am facut o rezervare pentru o masina in San Francisco. Doriti un sumar
      al calatoriei ?
Alex: Da.
CMC:  Primul zbor este US Airlines 79 cu plecare din Pittsburgh la 6:10 PM
      si sosire in San Francisco la 8:27 PM. Apoi urmatorul este cu plecare
      din San Francisco la 1:10 PM vineri, 1 octombrie si sosire in
      Pittsburgh la 8:53 PM. Nu am facut rezervare de hotel. Am rezervat o
      masina in San Francisco. Doriti sa platiti cu cartea de credit ?
Alex: Da, te rog.
CMC:  Va multumesc. Va voi trimite si un sumar prin e-mail. Asta s-ar putea
      sa dureze cateva minute. Va multumesc ca ati utilizat Carnegie Mellon
      Communicator.
```

Figura 1.1: Exemplu de dialog cu CMU Communicator

1.2 Structura tipică a unui sistem de dialog

Realizarea unor sisteme de dialog robuste şi cu performanţe ridicate a devenit posibilă relativ recent, datorită creşterii capacităţii de procesare şi a dezvoltărilor din domenii precum modelarea lingvistică şi recunoaşterea şi sinteza vorbirii. Această "tinereţe" a domeniului se traduce însă într-o lipsă de metodologii de dezvoltare consacrate, lucru care face proiectarea sistemelor de dialog o activitate anevoioasă, situată deseori la limita dintre ştiinţă şi artă [**?**].

Cu toate acestea, indiferent de tipul şi domeniul sistemului de dialog, problemele ce trebuie rezolvate sunt în principiu aceleaşi: recunoaşterea vorbirii, analiza semantică, controlul dialogului, generarea răspunsurilor şi sinteza vorbirii. În consecinţă, majoritatea sistemelor sunt proiectate modular, putând fi identificată o arhitectură generică, dictată de aceste probleme.

O structură tipică pentru un sistem de dialog interactiv pentru furnizarea de informaţii este ilustrată în figura **??**. În continuare se va descrie pe scurt rolul fiecăruia din modulele identificate:

- **Modulul de recunoaştere a vorbirii** preia semnalul sonor de la utilizator

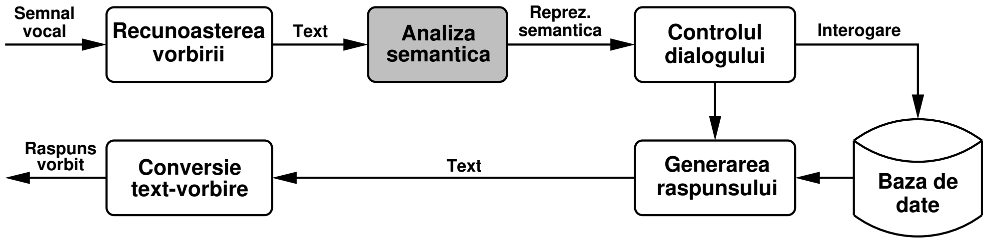


Figura 1.2: Structura unui sistem de dialog pentru obţinerea de informaţii

(de obicei prin microfon sau linie telefonică) şi generează secvenţa de cuvinte cea mai probabilă pornind de la acest semnal, pe baza unor modele acustice şi lingvistice.

- **Modulul de analiză semantică** are ca intrare textul produs de modulul de recunoaştere şi generează o reprezentare formalizată a înţelesului (sensului) acestui text. Analiza semantică are deci rolul de a extrage şi clarifica conţinutul de informaţie utilă al formulărilor utilizatorului.

- **Modulul pentru controlul dialogului** este practic nucleul sistemului de dialog, cu rolul de a realiza gestionarea dialogului şi a interacţiunii cu utilizatorul. În plus, acestui modul îi revine şi rolul de a menţine istoricul dialogului şi de a realiza analiza semantică contextuală.

- **Modulul de generare a răspunsurilor** este comandat de cel de control al dialogului în funcţie de starea curentă a acestuia, şi formulează răspunsul potrivit la un moment dat, folosind eventual şi rezultatele unui acces la o bază de date pentru completarea informaţiilor necesare.

- **Modulul de conversie text-vorbire** funcţionează în sensul invers celui de recunoaştere. El preia ieşirea în formă textuală a generatorului de răspunsuri, şi sintetizează forma sonoră a acesteia.

1.3 Rolul analizorului semantic

Analizorul semantic este o componentă esenţială în orice sistem de dialog. Plasat între modulul de recunoaştere a vorbirii şi cel de control al dialogului (vezi figura **??**), analizorul semantic are rolul de a produce o reprezentare formală a sensului pronunţiilor utilizatorului.

Datorită ambiguităţii şi varietăţii caracteristice limbajului natural, controlul dialogului nu se poate realiza direct pe baza textului obţinut de modulul de recunoaştere a vorbirii. Analiza semantică are deci rolul de a identifica, extrage şi clarifica informaţia utilă existentă în text, transcriind-o într-o formă riguros definită, adecvată procesării prin algoritmii specifici de control al dialogului.

Există mai multe abordări pentru realizarea analizoarelor semantice, dar majoritatea lor se încadrează într-una din două mari categorii: analizoare semantice *bazate pe reguli* şi analizoare semantice *bazate pe principii stocastice*. Lucrarea de faţă se va axa pe a doua categorie menţionată, prezentând teoretic avantajele acesteia şi ilustrându-le practic prin construirea unui analizor semantic stocastic şi evaluarea performanţelor lui într-un domeniu bine definit.

1.4 Organizarea lucrării

În continuare, pentru fundamentarea teoretică a lucrării, capitolul **??** trece în revistă unele aspecte ale analizei semantice în sistemele de dialog, împreună cu principalele alternative de proiectare existente (formalisme de reprezentare a cunoştinţelor şi metode de parsing), şi face o scurtă prezentare a soluţiilor utilizate pentru analiza semantică în câteva sisteme de dialog descrise în literatura de specialitate.

Pornind de la concluziile astfel obţinute, capitolul **??** prezintă realizarea unui analizor semantic stocastic care utilizează un formalism case-grammar pentru reprezentarea cunoştinţelor, detaliind modulele componente şi suportul teoretic pe care se bazează funcţionarea lor.

Experimentele efectuate în vederea integrării acestui analizor într-un sistem de dialog pentru informaţii despre orarul Departamentului de Calculatoare, precum şi rezultatele obţinute, sunt descrise în capitolul 4.

În încheiere sunt prezentate concluzii degajate din dezvoltarea analizorului şi experimentele efectuate, precum şi posibile direcţii de continuare a cercetărilor.

Capitolul 2

Analiza semantică în sistemele de dialog

Acest capitol este dedicat trecerii în revistă a unor aspecte teoretice ale analizei semantice, având ca finalitate alegerea între diferite opţiuni posibile în proiectarea unui analizor semantic utilizabil ca parte componentă a unor sisteme de dialog.

După definirea noţiunii de analiză semantică şi stabilirea rolului analizorului semantic într-un sistem de dialog, se trece la o prezentare a metodelor existente de proiectare a analizoarelor semantice. Sunt prezentate diverse alternative, împreună cu un studiu teoretic al avantajelor şi dezavantajelor fiecăreia.

Capitolul se încheie printr-o scurtă trecere în revistă a unor analizoare semantice din sisteme de dialog prezentate în literatura de specialitate, analizate prin prisma metodelor utilizate şi a performanţelor lor.

2.1 Analiza semantică

2.1.1 Definirea noţiunii de analiză semantică

Dicţionarul explicativ al limbii române furnizează următoarea definiţie pentru noţiunea de *semantică*:

> **semantic, -ă**, *semantici, -ce* s.f., adj. I. S.f. Ramură a lingvisticii care se ocupă cu studierea sensurilor cuvintelor şi a evoluţiei acestor sensuri. [...]

Stabilirea sensului cuvintelor într-un anumit context nu este de loc o operaţie trivială, dificultatea fiind generată de câteva caracteristici esenţiale ale limbajului natural. Pe de o parte, el permite exprimarea aceluiaşi înţeles, transmiterea aceluiaşi conţinut informativ, prin formulări diferite. Pe de altă parte, aceeaşi formulare poate avea semnificaţii diferite în contexte diferite. În plus, mai trebuie remarcat faptul că într-o formulare nu toate cuvintele sunt purtătoare de informaţie utilă.

Aceste caracteristici de ambiguitate ale formei scrise sau vorbite a limbajului natural fac necesară operaţia de analiză semantică. Prin ea se urmăreşte practic extragerea informaţiei utile într-o formă adecvată procesării ulterioare prin diverşi algoritmi. Analiza semantică trebuie deci să îndeplinească următoarele sarcini:

- identificarea cuvintelor purtătoare de informaţie utilă;
- stabilirea sensurilor acestor cuvinte, şi clarificarea lor în cazuri ambigue.

2.1.2 Nivele de analiză semantică

Sunt necesare câteva clarificări suplimentare şi în ceea ce priveşte **nivelul** la care se încearcă identificarea sensului cuvintelor. Încă de la început trebuie precizat că stabilirea sensurilor efective ale cuvintelor în cadrul unor expresii stilistice precum metaforele, metonimiile, exprimările ironice, şi altele de acest gen, depăşesc complet teoriile şi puterea computaţională curente, şi nu fac deci obiectul analizei semantice aşa cum este ea înţeleasă în lucrarea de faţă. Acest gen de exprimări sunt suficient de rare, în special în dialoguri purtate cu sisteme de calcul, astfel încât neluarea lor în considerare nu introduce practic erori semnificative.

Lasând la o parte aceste exprimări, se pot identifica două nivele pe care putem determina sensul cuvintelor. Să ne imaginăm următoarea formulare ipotetică a unui utilizator într-o conversaţie cu un un sistem de dialog:

– **Dar grupa întâi, când are ore tot cu el ?**

Cuvântul **ore**, dacă este considerat izolat, devine ambiguu: nu ştim dacă se referă la noţiunea de timp, sau la cea de oră de curs, laborator etc. La nivelul propoziţiei însă, ambiguitatea dispare, evident sensul fiind cel de oră de curs.

Pe de altă parte, sensul cuvântului **el** nu poate fi determinat doar din propoziţia considerată, pentru aceasta fiind necesare mai multe informaţii. Dacă am cunoaşte şi propoziţia anterioară, care ar putea fi de exemplu:

– **Când are grupa doi curs cu profesorul Vancea?**

atunci ar deveni evident că **el** face de fapt referire la profesorul Vancea.

În consecinţă, putem defini practic două nivele de analiză semantică: la un prim nivel se poate extrage un *înţeles independent de context al propoziţiei* [**?**], caz în care cuvântul **el** este încă ambiguu; la un nivel superior, se poate obţine un *sens complet al propoziţiei*, dependent de contextul în care a fost formulată.

Întrucât modulul de analiză semantică al unui sistem de dialog operează pe propoziţii independente (vezi figura **??**), rolul lui se limitează la primul nivel: obţinerea sensului propoziţiei independent de context şi exprimarea lui într-o formă riguroasă, bine definită. Analiza în context, cea care determină sensul complet al fiecărei propoziţii, se realizează la nivelul modulului de control al dialogului. Acest modul menţine un *istoric al dialogului*, şi poate determina sensul complet pe baza acestui istoric, făcând uz de algoritmi specifici.

În concluzie, într-un sistem de dialog rolul analizorului semantic este de a extrage şi clarifica sensul independent de context al formulărilor utilizatorului.

2.1.3 Legături între analiza semantică şi cea sintactică

Între analiza semantică şi cea sintactică poate exista o strânsă legătură: dacă semantica se ocupă cu stabilirea sensului cuvintelor şi a informaţiei transmise de ele, *sintaxa* defineşte regulile de îmbinare a cuvintelor în propoziţii şi a propoziţiilor în fraze [?]. Ambele folosesc deci cuvântul ca unitate fundamentală de procesare. În plus, majoritatea limbajelor (atât naturale, cât şi formale) sunt *compoziţionale*, adică sensul unei propoziţii derivă din sensurile părţilor ei componente, ceea ce reprezintă un motiv suplimentar pentru realizarea unei legături între ele [?].

Aceasta a făcut ca majoritatea tehnicilor de analiză semantică să îşi aibe originea în cele de analiză sintactică. Principalele instrumente utilizate sunt tot gramaticile, însă cu diverse modificări şi extinderi, necesare pentru a capta aspectele caracteristice analizei semantice.

Atât analiza sintactică cât şi cea semantică sunt foarte utilizate în domeniul limbajelor formale şi al translatoarelor, existând o multitudine de formalisme şi algoritmi specializaţi în acest sens. În acest domeniu, analiza sintactică precede analiza semantică, şi generează informaţii care sunt apoi utilizate pentru analiza semantică. Deci analiza semantică în domeniul limbajelor formale se bazează şi este în general dirijată de cea sintactică.

Extrapolarea acestor tehnici la nivelul limbajului natural se dovedeşte însă dificilă datorită varietăţii şi ambiguităţilor caracteristice lui. În primul rând, construirea unei gramatici acoperitoare pentru sintaxa unei limbi naturale presupune un anumit grad de expertiză lingvistică. Apoi, chiar dacă o astfel de gramatică ar exista, în limbajul natural pot apare formulări incorecte din punctul ei de vedere, dar care să fie totuşi inteligibile, ba chiar perfect acceptabile în uzul curent, datorită caracterului evolutiv al limbii, incompletitudinii gramaticii, dar mai ales datorită diverselor fenomene ce caracterizează vorbirea spontană: repetiţii, starturi false, disfluenţe etc. Într-un sistem de dialog, la acestea se pot adăuga eventuale erori de recunoaştere a vorbirii, care pot deasemeni compromite corectitudinea sintactică a propoziţiei, lăsând-o însă neschimbată din punct de vedere semantic.

În concluzie, deşi majoritatea tehnicilor de analiză semantică actuale sunt extinderi ale unora de analiză sintactică, realizarea prealabilă a unei analize sintactice a limbajului natural nu este nici necesară, nici recomandabilă, putând fi chiar dăunătoare în construirea analizoarelor semantice pentru sisteme de dialog.

2.2 Alternative în proiectarea unui analizor semantic

În cele ce urmează vor fi prezentate principalele alternative de proiectare ce apar în dezvoltarea unui analizor semantic pentru un sistem de dialog.

Primul pas îl constituie alegerea unui formalism adecvat pentru reprezentarea cunoştinţelor, a informaţiei extrase. Aici există mai multe alternative, care vor fi prezentate împreună cu o analiză a avantajelor şi dezavantajelor fiecăreia.

După stabilirea formalismului de reprezentare a cunoştinţelor, a doua problemă ce trebuie atacată este alegerea metodei prin care se va realiza propriu-zis extragerea informaţiei în forma stabilită, deci a algoritmului efectiv de analiză semantică.

2.2.1 Formalisme de reprezentare a cunoştinţelor

De-a lungul timpului au fost definite o multitudine de formalisme gramaticale, şi diverse tehnici de parsing pentru aceste formalisme. Ele au apărut ca urmare a necesităţii de a realiza o analiză sintactică performantă pentru întreaga gamă de limbaje: de la cele puternic formalizate, până la limbajul natural. Datorită legăturilor care se pot stabili între analiza sintactică şi cea semantică, precum şi a necesităţii unui formalism de reprezentare a informaţiei extrase din text, a apărut natural ideea de a extinde gramaticile utilizate în analiza sintactică pentru a capta aspectele specifice analizei semantice. Astfel, pe baza gramaticilor sintactice au aparut noi *formalisme semantice* adaptate pentru analiza semantică.

În continuare se va face o scurtă trecere în revistă a celor mai importante tipuri de gramatici sintactice, urmată de prezentarea principalelor formalisme utilizate în analiza semantică.

Gramatici sintactice

- ***Gramatici Chomsky***

  Noţiunea de gramatică (în sensul utilizat în teoria limbajelor formale şi a translatoarelor) a fost introdusă de Noam Chomsky în anii '50. O gramatică este constituită practic dintr-un set de reguli (producţii) care realizează o descriere formală a structurilor sintactice posibile în limbajul pe care îl descrie. Chomsky a realizat şi o ierarhizare a acestor gramatici în funcţie de tipul producţiilor lor şi, implicit, de capacitatea lor de exprimare. Clasa cea mai largă este cea a gramaticilor de tipul 0, care nu au restricţii în ceea ce priveşte scrierea producţiilor. Urmează clasa gramaticilor de tip 1, sau dependente de context. Şi mai restrictive sunt gramaticile de tipul 2, sau independente de context, iar clasa cea mai restrictivă (şi în acelaşi timp cea mai puţin expresivă) este cea a gramaticilor de tipul 3 sau regulate.

  Gramaticile independente de context (tipul 2) au fost utilizate cu mult succes în descrierea şi analiza sintactică a limbajelor formale. Ele sunt suficient de expresive pentru această sarcină şi, în plus, există numeroşi algoritmi eficienţi (ca timp şi spaţiu de memorie) pentru realizarea parsingului peste aceste gramatici.

  Deşi ele sunt suficient de puternice şi pentru descrierea majorităţii construcţiilor limbajului natural, realizarea unui analizor sintactic bazat pe gramatici independente de context în acest domeniu este o sarcină dificilă. O astfel de gramatică ar conţine un număr mare de reguli, iar scrierea acestora ar necesita un anumit grad de expertiză lingvistică. Dacă în plus dorim captarea robustă a tuturor aspectelor caracteristice limbajului vorbit (disfluenţe, repetiţii, starturi false etc.), aşa cum este necesar într-un sistem de dialog, numărul regulilor ar creşte foarte mult, făcând impractic efortul necesar dezvoltării unei astfel de gramatici [?].

- ***Reţele de tranziţii (Transition Networks)***
  Reţelele de tranziţii constituie un alt formalism sintactic, dar care nu foloseşte reguli, ci grafuri orientate pentru a descrie structurile sintactice acceptate într-un anumit limbaj. O variantă mai puternică a acestora, reţelele de tranziţii recursive (Recursive Transition Networks - RTN) sunt echivalente ca şi capacitate de exprimare cu gramaticile independente de context.

- ***Gramatici extinse (Augmented grammars)***
  Gramaticile extinse sunt obţinute plecând de la cele independente de context, prin ataşarea unor atribute constituenţilor producţiilor lor. Astfel, se pot modela mai uşor anumite aspecte ale limbajului natural precum acordurile între cuvinte, subcategoriile etc. [?]. Din acest punct de vedere, gramaticile extinse beneficiază de o anumită flexibilitate care lipseşte gramaticilor Chomsky, lucru care le face un candidat mai bun pentru analiza sintactică a limbajului natural. Trebuie remarcat că aceeaşi extensie – ataşarea de atribute nodurilor – poate fi aplicată şi reţelelor de tranziţii, rezultând un nou formalism: reţelele de tranziţii extinse.

- ***Gramatici arborescente (Tree adjoining grammars)***
  Gramaticile arborescente reprezintă un alt formalism adaptat la particularităţile analizei sintactice a limbajului natural. Ele descriu structurile admisibile în limbaj prin intermediul unui set de arbori elementari (corespunzători celor mai simple propoziţii ale limbajului), şi a unei operaţii de "alăturare" a arborilor pe baza căreia se pot sintetiza structuri complexe [?]. Capacitatea lor de exprimare o depăşeşte cu puţin pe cea a gramaticilor independente de context, fiind o soluţie mai potrivită pentru analiza sintactică a limbajului natural.

Toate aceste gramatici sunt potrivite pentru analiza sintactică, care are însă un scop diferit de cel al analizei semantice. Pornind de la ele, au fost create noi formalisme, mai potrivite pentru sarcinile şi necesităţile analizei semantice.

Formalisme pentru analiza semantică

- ***Gramatici semantice***
  Noţiunea de gramatică semantică a fost introdusă de Burton în 1976, extinzând gramaticile Chomsky în sensul integrării informaţiei semantice şi sintactice în acelaşi model [?]. Dezavantajele utilizării acestor gramatici pentru analiza semantică a limbajului natural sunt, pe de o parte, lipsa

de portabilitate între domenii, pe de alta, rigiditatea lor (moştenită de la gramaticile Chomsky) faţă de fenomenele specifice vorbirii spontane.

- ***Formalismul case-grammar (Case-grammar Formalism)***
  Acest formalism este poate cel mai îndepărtat faţă de noţiunea tipică de gramatică (aşa cum a fost introdusă de Chomsky), având totuşi o mulţime de caracteristici care îl fac optim pentru utilizarea în cadrul analizei semantice a limbajului natural.

  Formalismul a fost introdus de Fillmore în 1968, şi extins de Bruce în 1975. La baza lui se află conceptul de *cadru* (frame), introdus de Minsky în 1975 ca o modalitate de reprezentare a cunoştinţelor. Ideea este foarte simplă şi în acelaşi timp puternică, găsindu-şi o largă aplicabilitate în numeroase subdomenii ale inteligenţei artificiale. Componentele esenţiale ale unui cadru sunt un aşa-numit *concept*, fixat, şi o mulţime de *sloturi*, completate cu informaţii ce reprezintă "cunoştinţele" sistemului. Activitatea de completare a sloturilor unui cadru o vom numi *instanţierea cadrului*. Pentru exemplificare, revenim la formularea ipotetică a unui utilizator în discuţia sa cu un sistem de dialog furnizând informaţii despre orar:

  – **Când are grupa doi curs cu profesorul Vancea?**

  Figura **??** ilustrează o instanţă a unui cadru reprezentând conţinutul semantic al acestei formulări. Conceptul <`când`>, specificat în partea superioară a cadrului, ne spune că este vorba despre o întrebare în legătură cu timpul. Restul conţinutului semantic este încapsulat în valorile atribuite sloturilor `grupa`, `tip-materie` şi `profesor`. Nu este obligatoriu ca toate sloturile să fie completate – informaţiile sunt extrase în măsura în care ele există în text.

<`când`>	
`an:`	`-`
`grupa:`	`2`
`subgrupa:`	`-`
`tip-materie:`	`curs`
`profesor:`	`Vancea`

Figura 2.1: Exemplu de cadru

  O a doua noţiune importantă este aceea de *marcaj* (case-marker). Un marcaj este practic un cuvânt din text care realizează o anumită constrângere în ceea ce priveşte completarea sloturilor. Acest gen de constrângeri lexicalizate în instanţierea cadrelor modelează practic sintaxa limbajului, limitând construcţiile interpretabile prin formalismul case-grammar. În exemplul anterior, cuvântul **grupa** este un marcaj indicând faptul că urmează probabil valoarea slotului (*case-value*) `grupa`; similar, cuvântul **profesorul** este un

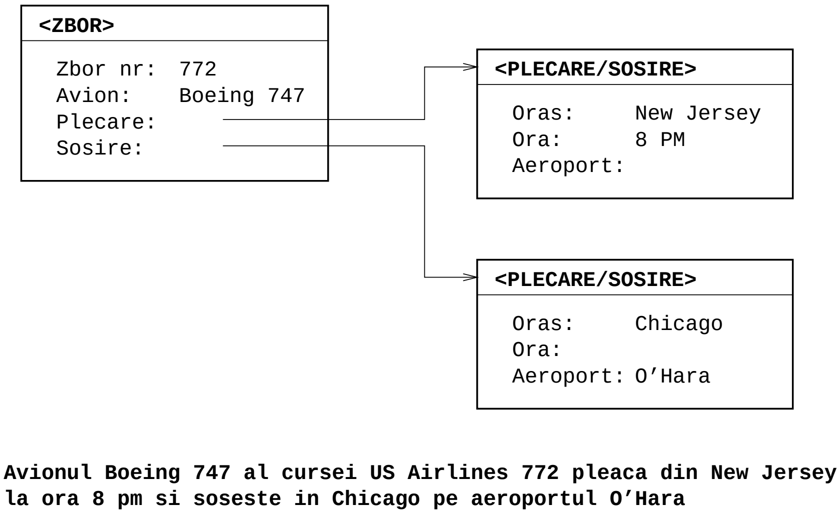


Figura 2.2: Un exemplu de sistem de cadre

marcaj pentru slotul `profesor`. Cuvântul **curs** este ceva mai special, el fiind în acelaşi timp marcaj şi valoare pentru slotul `tip-materie`. Algoritmul de extragere a informaţiei (parsing utilizând formalismul case-grammar) se bazează în mare măsură pe marcaje, acestea indicând unde se află informaţia şi în acelaşi timp clarificând sensul ei.

Pornind de la cadre individuale, se poate realiza un *sistem de cadre* (case-system) care să exprime prin legături interdependenţele semantice dintre cadrele ce îl compun. Legăturile se materializează prin faptul că un slot al unui cadru, în loc să fie completat cu o valoare explicită, este completat cu o legătură spre un alt cadru din sistem. Se formează astfel o reţea de cadre a cărei capacitate de reprezentare a cunoştinţelor este mult mărită faţă de cea a unui cadru individual. Un exemplu este dat în figura **??**.

Prin **formalism case-grammar** se înţelege deci un astfel de sistem de cadre, cu aspecte sintactice incluse prin constrângerile lexicalizate, dictate de marcaje, în instanţierea sloturilor [**?**].

Dintre toate variantele prezentate, formalismul case-grammar reprezintă alternativa cea mai potrivită pentru realizarea analizei semantice într-un sistem de dialog, principalul lui avantaj constând în flexibilitatea mult superioară gramaticilor sintactice şi diverselor variante extinse ale acestora. Spre deosebire de gramaticile clasice, care modelează limbajul şi eventual conţinutul informativ al acestuia prin intermediul unor reguli bine definite, rigide, formalismul case-grammar indică doar structurile semantice acceptabile, prin intermediul cadrelor şi a relaţiilor dintre acestea. Aspectele sintactice sunt incluse, şi dirijează procesul de extragere a informaţiei, însă prin intermediul constrângerilor dictate de marcaje – un mecanism mult mai relaxat decât cel al gramaticilor clasice. Aceasta face ca formalismul case-grammar să poată fi utilizat cu succes şi în cazurile în care gramaticile tipice ar eşua – cele ale formulărilor incorecte sintactic, dar în acelaşi timp valide semantic, formulări destul de des întâlnite în conversaţiile dintre oameni, şi care deci pot fi destul de frecvente şi în conversaţiile purtate cu sisteme de dialog.

Reprezentarea bazată pe cadre este foarte potrivită şi din punctul de vedere al modulului de control al dialogului, deoarece simplitatea reprezentării face ca menţinerea unui istoric semantic al dialogului şi realizarea analizei de context să se poată face relativ uşor.

În concluzie, alegerea formalismului case-grammmar este cea mai potrivită pentru reprezentarea cunoştinţelor în analizorul semantic proiectat.

2.2.2 Metode de parsing

Odată definită modalitatea de reprezentare a cunoştinţelor, următorul pas în proiectarea unui analizor semantic este stabilirea metodei pe care o vom utiliza pentru extragerea efectivă a informaţiei din text. Pentru majoritatea gramaticilor clasice există numeroşi algoritmi de parsing care pot genera reprezentările corespunzătoare. În ceea ce priveşte parsingul bazat pe un formalism case-grammar, soluţiile se grupează în două categorii: *metode bazate pe reguli* şi *metode stocastice.*

Metodele bazate pe reguli, aşa cum le spune şi numele, utilizează un set de reguli care controlează modul în care se face identificarea conceptelor şi completarea sloturilor cu valorile corespunzătoare din text. Figura **??** ilustrează structura unui astfel de analizor.

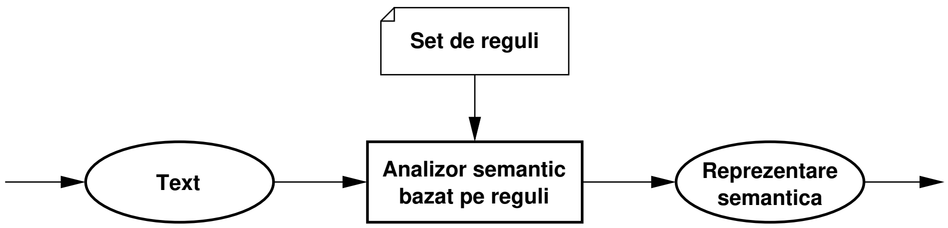


Figura 2.3: Structura unui analizor semantic bazat pe reguli

În general, regulile sunt lexicalizate: se definesc familiile de cuvinte care identifică conceptele/cadrele, şi familiile de cuvinte care joacă rolul de marcaje. În plus, trebuie definite reguli care descriu legăturile dintre marcaje şi valori. Pe baza acestora se pot determina cuvintele care au conţinut semantic şi dau valori sloturilor cadrului identificat. Evident, corectitudinea – în sensul cel mai larg – regulilor de parsing va dicta performanţa analizorului, lucru care face ca partea cea mai mare din proiectarea unui astfel de analizor să stea în definirea regulilor.

Alternativa o reprezintă metodele de parsing stocastic. Ideea fundamentală în aceste metode este de a utiliza un model probabilistic pentru decodarea semantică a formulărilor utilizatorului (figura **??**).

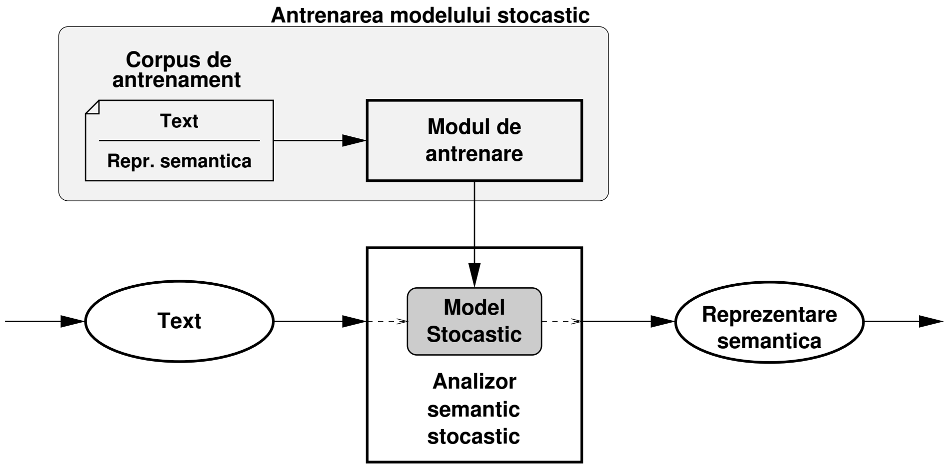


Figura 2.4: Structura unui analizor semantic stocastic

Modelul este creat într-o primă etapă, de antrenare, folosind algoritmi caracteristici tipului de model utilizat (modele Markov, reţele neuronale etc.) Resursa esenţială pentru antrenare este un corpus reprezentativ de formulări utilizator care au fost în prealabil analizate semantic, astfel încât fiecare text are asociată reprezentarea semantică. Prin antrenare, modelul va "învăţa", din perechile (text, reprezentare semantică), cum se generează informaţia semantică din text. Se elimină astfel necesitatea de a defini explicit regulile pe baza cărora se face extracţia, ele fiind practic "captate" automat în faza de antrenare, sub formă de valori ai diverşilor parametri ai modelului [**?**].

Odată etapa de antrenare încheiată, analiza semantică decurge simplu: analizorul primeşte la intrare formulările utilizatorului sub formă de text, iar modelul va furniza reprezentările semantice cele mai probabile pentru intrările respective.

Acest model stocastic este practic un mecanism de învăţare automată care operează în două moduri: în *modul de antrenare*, el primeşte la intrare perechi (text, reprezentare semantică) şi îşi ajustează parametrii interni pe baza unor algoritmi specifici astfel încât, statistic, să capteze corespondenţa dintre text şi reprezentarea semantică. În al doilea mod, de *decodare*, modelul primeşte la intrare doar textul, şi furnizează reprezentarea lui semantică cea mai probabilă.

Comparaţie între parsingul stocastic şi cel bazat pe reguli

Metoda de parsing stocastic prezintă numeroase avantaje faţă de metodele bazate pe reguli, lucru demonstrat în continuare printr-o scurtă analiză comparativă.

După cum s-a subliniat şi anterior, la realizarea unui analizor semantic bazat pe reguli partea cea mai dificilă este proiectarea regulilor. Dezvoltarea unui astfel de set de reguli este costisitoare, şi nu întotdeauna la îndemâna proiectanţilor unui sistem de dialog, întrucât presupune participarea unor experţi lingvişti.

Bazarea parsingului pe un set de reguli are şi dezavantajul că reintroduce o anumită rigiditate în analiza semantică, problemă pe care am încercat să o atenuăm prin utilizarea formalismului case-grammar, iar captarea prin reguli a tuturor fenomenelor caracteristice vorbirii spontane este foarte dificilă, mărind foarte mult numărul de reguli (implicit costul dezvoltării sistemului) şi reducând performanţele.

În plus, metodele bazate pe reguli suferă de o acută lipsă de portabilitate. Fie că este vorba de portarea analizorului semantic de la un domeniu de aplicaţie la altul, fie că este vorba de lărgirea domeniului, de cele mai multe ori este necesară cel puţin o extindere a setului de reguli (pentru o analiză detaliată vezi [**?**]).

În ceea ce priveşte analiza semantică stocastică, singura resursă necesară este un corpus de antrenament analizat semantic (cel mai probabil manual), care să includă diversele tipuri de formulări ce apar în dialogurile în care sistemul va fi implicat. Obţinerea unui astfel de corpus se poate face iniţial prin metoda Vrăjitorului din Oz [**?**], [**?**], sau pe parcursul funcţionării unui sistem existent.

De remarcat că problema analizării semantice manuale a acestui corpus nu este foarte mare, întrucât dezvoltarea modelului se poate face *incremental* (*bootstrapping*): se analizează semantic manual o porţiune a corpusului, se antrenează o primă variantă a modelului cu această porţiune, modelul este utilizat pentru a decoda semantic o nouă porţiune din corpus, decodare care se corectează manual, după care pe baza ei şi a primei porţiuni se face o nouă antrenare şamd.

În plus, metodele stocastice elimină practic toate celelalte dezavantaje ale sistemelor bazate pe reguli. În cazul lor, nu trebuie dezvoltat un set de reguli, acestea fiind practic învăţate automat în faza de antrenare a modelului. Dispare deci în mare măsură necesitatea expertizei lingvistice. Analiza semantică stocastică are un grad sporit de flexibilitate, modelul fiind "reglat" prin antrenare pe tipurile de formulări care apar în dialoguri. În fine, în cazul portării analizorului la alt domeniu (sau al extinderii domeniului), singurul lucru necesar este o reantrenare a modelului cu un corpus de formulări adecvat.

În concluzie, pentru sistemele automate de dialog vocal om-calculator o abordare stocastică a parsingului bazat pe formalismul case-grammar este mult mai potrivită decât cele bazate pe reguli, iar lucrarea de faţă se va axa în continuare pe dezvoltarea unui astfel de analizor semantic.

2.3 Stadiul actual

Vom încheia prezentarea teoretică a alternativelor de proiectare a unui analizor semantic pentru sisteme de dialog printr-o scurtă trecere în revistă a soluţiilor utilizate de câteva sisteme existente. După cum vom vedea, o analiză comparativă a metodelor folosite şi a performanţelor lor reprezintă încă o justificare a alegerii soluţiei unui analizor semantic stocastic bazat pe formalismul case-grammar.

Sisteme cu analizoare semantice bazate pe reguli

- **CMU-PHOENIX**
  Acest sistem de analiză semantică, dezvoltat la Carnegie Mellon University [?], foloseşte un formalism case-grammar pentru reprezentarea cunoştinţelor, iar parsingul se face la nivel de expresii (nu propoziţii), pe bază de reguli implementate sub forma unor reţele de tranziţii recursive (vezi **??**). PHOENIX este utilizat în diverse sisteme de dialog, obţinând în domeniul ATIS (Air Travel Information Services) o rată a erorilor de 3.8% (cea mai bună performanţă a unui analizor semantic în acest domeniu la evaluările DARPA din 1994).

- **JANUS**
  JANUS [?] este un sistem de traducere automată dezvoltat în colaborare de ATR Interpreting Telephony Research Laboratories (Japonia), CMU, Universitatea din Karlsruhe şi Siemens. Sistemul efectuează traduceri automate în domeniul programării de întâlniri, şi utilizează în paralel două metode de analiză semantică: sistemul CMU-PHOENIX, şi un parser LR generalizat. Rata erorilor sistemului este de cca 24.5% pentru traduceri din germană în engleză, şi 18.6% pentru traduceri din spaniolă în engleză.

- **LIMSI-L'ATIS**
  LIMSI-L'ATIS este versiunea franceză de sistem de dialog în domeniul ATIS, dezvoltat la LIMSI-CNRS [?], utilizând şi el reprezentarea prin cadre şi parsingul pe bază de reguli. Pentru descrierea sistemului de cadre şi a setului de reguli de parsing a fost creat un limbaj declarativ dedicat. Întrucât s-a observat că regulile eşuează în anumite situaţii caracteristice vorbirii spontane, s-a introdus o fază de preprocesare a intrării analizorului semantic, în care se realizează o normalizare a textului. Rata de eroare măsurată a acestui sistem a fost de cca. 8.7%

- **LIMSI-MASK**
  LIMSI-MASK este un sistem similar cu LIMSI-L'ATIS, dezvoltat tot la LIMSI-CNRS [?], dar care operează în domeniul rezervării de bilete şi obţinerii de informaţii referitoare la mersul trenurilor. Abordarea analizorului semantic este identică cu cea din LIMSI-L'ATIS.

Sisteme cu analizoare semantice stochastice

- **AT&T CHRONUS**
  AT&T CHRONUS [?] este un analizor semantic bazat pe parsing stocastic, dezvoltat la AT&T şi testat pe domeniul ATIS. Analizorul utilizează un model Markov, analiza semantică reducându-se la decodarea stărilor acestuia pentru o secvenţă de simboluri observate care corespunde intrării. Sistemul are performanţe foarte bune, fiind comparabil cu CMU-PHOENIX pe acelaşi domeniu – rata erorii de 3.8%

- **BBN-HUM (Hidden Understanding Model)**
  Acest sistem, dezvoltat la firma BBN, utilizează tehnici de învăţare automată statistică atât pentru analiza semantică la nivelul propoziţiei, cât şi pentru analiza semantică în context [**?**]. În teste neoficiale pe domeniul ATIS a fost raportată o rată a erorilor de cca. 9.5%.

- **PHILIPS Train-Timetable Inquiry System**
  Aşa cum îi spune şi numele, acesta este un sistem de dialog dezvoltat de Philips Research Laboratories [**?**], care operează în domeniul furnizării de informaţii despre mersul trenurilor. Sistemul utilizează tehnici stocastice de parsing peste o gramatică semantică (vezi **??**). În această abordare, deşi producţiile gramaticii sunt elaborate manual de un grup de experţi, probabilităţile acestor producţii sunt "învăţate" automat dintr-un corpus de antrenament. Performanţele acestui sistem nu se ridică însă la nivelul celorlalte: rata raportată a erorilor este de cca. 25%.

Capitolul 3

Un analizor semantic stocastic

În capitolul anterior au fost prezentate principalele alternative de proiectare, şi s-a decis utilizarea formalismului case-grammar şi a parsingului stocastic pentru realizarea unui analizor semantic pentru sisteme de dialog.

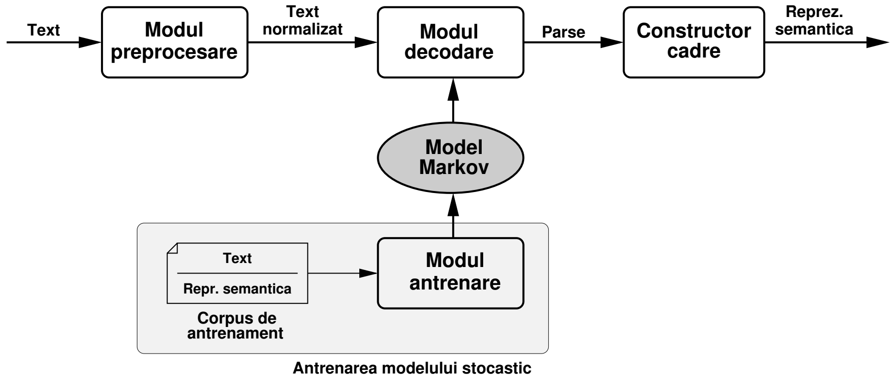
În acest capitol, pornind de la schema bloc a analizorului semantic, vor fi detaliate componentele lui, prezentându-se atât suportul teoretic cât şi soluţiile de implementare alese pentru fiecare modul.

Deşi analizorul este independent de domeniul sistemului de dialog în care va fi utilizat, pe parcursul acestui capitol vor fi utilizate ca exemple formulări specifice unui sistem dedicat furnizării de informaţii despre orarul Departamentului de Calculatoare, anticipând astfel experimentele şi rezultatele din capitolul 4.

3.1 Structura analizorului semantic

Figura **??** prezintă arhitectura analizorului semantic împreună cu diversele reprezentări prin care trece textul obţinut din pronunţia utilizatorului pe parcursul extragerii informaţiei şi generării reprezentării semantice. Elementul central al analizorului proiectat îl constituie un model Markov (vezi **??**) pe baza căruia se realizează analiza semantică. Pe lângă modulele de decodare şi antrenare a modelului Markov, în componenţa sistemului mai intră două module: un modul pentru preprocesarea intrării şi un modul pentru construirea efectivă a cadrelor. În continuare va fi prezentat pe scurt rolul fiecărei componente în cadrul arhitecturii ilustrate.

- **Modulul de preprocesare**
  Modulul de preprocesare este situat la graniţa dintre modulul de recunoaştere automată a vorbirii şi decodorul semantic propriu-zis. Rolul principal al acestuia este de a realiza o normalizare a textului primit de la modulul de recunoaştere. Preprocesarea constă în eliminarea evenimentelor non-lexicale, normalizarea numerelor, reducerea flexiunilor, unificarea expresiilor, înlocuiri de aliasuri, unificarea categoriilor şi eliminarea cuvintelor din afara domeniului sistemului de dialog.




```
Text:            (aa) ce profesor tine laboratorul de s e la grupa (aa) doi
Text normalizat: ce-profesor laborator [MATERIA:"Sisteme expert"] grupa [NR:"2"]
Parse:           <ce-profesor> (m:laborator) (v:laborator) (m:grupa) (v:grupa)
Repr. semantica: <ce-profesor>
                   <identificare>
                     grupa = 2
                   <specif-materie>
                     laborator = "Sisteme expert"
```

Figura 3.1: Arhitectura analizorului semantic

- **Modelul Markov**
  Modelul stocastic utilizat pentru realizarea parsingului în analizorul semantic proiectat este un *model Markov cu stări ascunse*, care acţionează practic ca un mecanism de învăţare (vezi **??**). Crearea modelului se realizează printr-o etapă premergătoare de antrenare în care acesta "captează" regulile de decodare semantică sub forma unor parametri interni. Odată creat, modelul poate fi folosit într-un decodor care va genera cea mai probabilă secvenţă de etichete semantice (parse) corespunzătoare secvenţei de intrare.

- **Modulul de decodare**
  Pornind de la textul normalizat prin preprocesare, considerat ca o succesiune de simboluri observate ale modelului Markov, modulul de decodare determină cea mai probabilă succesiune de etichete semantice corespunzătoare prin aplicarea algoritmului Viterbi (vezi **??**).

- **Modulul de antrenare**
  Acest modul este utilizat doar în etapa premergătoare de creare/antrenare a modelului stocastic, el neavând un rol direct în analiza semantică efectivă. Modulul realizează antrenarea modelului printr-o serie de algoritmi specifici (vezi **??**), pe baza unui corpus de formulări utilizator care pot fi considerate tipice pentru domeniul de aplicaţie al sistemului de dialog, şi care în prealabil au fost etichetate semantic.

- **Constructorul de cadre**
  Constructorul de cadre este ultimul modul al analizorului semantic. El primeşte de la modulul de decodare succesiunea de etichete semantice (parse), pornind de la care construieşte reprezentarea pe bază de cadre a interpretării semantice a formulării utilizatorului.

3.2 Modulul de preprocesare

Întrucât forma în care modulul de recunoaştere a vorbirii furnizează textul nu este întotdeauna cea mai potrivită pentru realizarea analizei semantice, apare necesitatea introducerii unui modul de preprocesare între decodorul semantic şi modulul de recunoaştere a vorbirii.

Rolul principal al preprocesorului este de a realiza o normalizare a textului de intrare. În plus, el realizează şi o reducere a vocabularului de intrare al decodorului semantic propriu-zis, fără a afecta însă conţinutul semantic al formulărilor. Această reducere a vocabularului este necesară întrucât dimensiunile modelului stocastic sunt direct dependente de dimensiunile vocabularului de intrare, iar prin reducerea dimensiunii sale se măresc şansele de a obţine un model mai performant pornind de la un acelaşi corpus de antrenament.

Operaţiile care compun etapa de preprocesare sunt sintetizate în tabelul **??**, împreună cu exemplificări ale efectelor pe care le au asupra textului de intrare. În continuare vom detalia fiecare dintre aceşti paşi:

1. **Eliminarea evenimentelor acustice non-lexicale**
   Unele sisteme de recunoaştere automată a vorbirii pot transcrie şi evenimente acustice non-lexicale care apar în semnalul sonor. Fie că este vorba de evenimente acustice produse de utilizator (tuse, ezitări etc.), fie că este vorba de diverse alte zgomote din mediu, aceste evenimente nu transportă nici un fel de informaţie utilă, deci pot fi eliminate fără riscuri. În această lucrare am presupus că ele sunt furnizate în paranteze de modulul de recunoaştere a vorbirii; în exemplul din tabelul **??**, au fost eliminate în această fază două ezitări (aa).

2. **Normalizarea numerelor**
   În acest al doilea pas al preprocesării se realizează transformarea numerelor din forma scrisă furnizată de modulul de recunoaştere în reprezentarea cu cifre arabe corespunzătoare. În exemplul dat, “doi” se transformă în 2.

3. **Reducerea flexiunilor**
   Spre deosebire de alte limbi (e.g. engleza), limba română este una puternic flexionară, astfel încât poate fi utilă introducerea unui mecanism de reducere a flexiunilor. Astfel se realizează o normalizare a intrării şi o reducere a vocabularului domeniului, fără însă a afecta conţinutul semantic al formulării. În exemplificarea dată, “laboratorul” se transformă în forma nearticulată **laborator**, iar “grupa” în **grupă**.

Operaţie	**Exemplificare rezultat**
Recunoaştere vorbire	(aa) ce profesor ţine laboratorul de s e la grupa (aa) doi
Eliminare even. non-lexicale	ce profesor ţine laboratorul de s e la grupa doi
Normalizare numere	ce profesor ţine laboratorul de s e la grupa **2**
Reducere flexiuni	ce profesor ţine **laborator** de s e la **grupă** 2
Unificare expresii	**ce-profesor** ţine laborator de s e la grupă 2
Înlocuiri aliasuri	ce-profesor ţine laborator de **"Sisteme expert"** la grupă 2
Unificare categorii	ce-profesor ţine laborator de **[MATERIA:"Sisteme expert"]** la grupă **[NR:"2"]**
Elim. cuv. din afara domeniului	ce-profesor laborator [MATERIA:"Sisteme expert"] grupă [NR:"2"]

Tabelul 3.1: Paşii de preprocesare

4. **Unificarea expresiilor**
   În această etapă se realizează unificarea sub formă de expresii unitare a unor grupuri de cuvinte cu un corespondent semantic unic (e.g. formularea "ce profesor" se transformă în **ce-profesor**.)

5. **Înlocuiri de aliasuri**
   Această operaţie este dependentă de domeniul sistemului de dialog, şi prin ea se realizează înlocuirea aliasurilor (prescurtări, sinonime etc.) unor cuvinte sau expresii (e.g. "s e" devine **"Sisteme expert"**.)

6. **Unificarea categoriilor**
   Operaţia de unificare a categoriilor are ca scop gruparea cuvintelor în categorii semantice pentru a facilita analiza ulterioară şi a reduce dimensiunile vocabularului decodorului. În continuare decodorul va lucra cu categoriile respective ca simboluri de intrare, revenindu-se la valorile concrete doar în faza de construire a reprezentării pe bază de cadre.

   Putem vorbi despre existenţa a două tipuri de categorii. Pe de o parte, se pot defini categorii generice în care să se unifice cuvintele de acelaşi tip sau diverse sinonime. De exemplu, toate numerele care apar sunt grupate în categoria [NR] (în exemplul nostru, "2" devine [NR:"2"]).

   Pe de altă parte, se pot defini şi categorii specifice domeniului sistemului de dialog proiectat. În acest sens, se poate defini câte o categorie pentru fiecare câmp din baza de date interogată de sistem ale cărui valori pot să apară în formulările utilizatorului. În exemplul ilustrat, "Sisteme expert", care este o valoare a câmpului MATERIA din baza de date, este transformată prin unificarea categoriilor în [MATERIA:"Sisteme expert"]. Se observă însă că o categorie similară nu a putut fi definită şi pentru câmpul GRUPA întrucât valorile acestui câmp (1, 2, 3 ...) sunt ambigue – când apare un număr în text, preprocesorul nu poate decide dacă el desemnează o grupă, o subgrupă, sau o oră. Din această cauză valorile numerice au fost unificate în categoria generică [NR], urmând ca decodorul semantic să stabilească efectiv sensul acestor numere.

În concluzie, modul în care se face definirea categoriilor este în mare măsură dictat de particularităţile domeniului sistemului de dialog.

7. **Eliminarea cuvintelor din afara domeniului**
Această ultimă operaţie implică eliminarea tuturor cuvintelor care nu sunt relevante în contextul domeniului stabilit al sistemului de dialog. În exemplul ilustrat, au fost eliminate verbul "ţine" şi prepoziţiile "de" şi "la". Aceste eliminări se realizează pe baza unui dicţionar al domeniului, dicţionar care trebuie să conţină toate cuvintele relevante din punct de vedere semantic pentru domeniul respectiv. Construirea acestui dicţionar se realizează în momentul analizei corpusului de antrenament al modelului stocastic (vezi capitolul 4.)

Trebuie remarcat că necesitatea utilizării tuturor acestor etape de preprocesare depinde de la un domeniu la altul, şi de la un sistem de dialog la altul, fiind dictată în mare măsură şi de particularităţile modulului de recunoaştere automată a vorbirii utilizat. De exemplu, unele sisteme de recunoaştere nu transcriu evenimente acustice non-lexicale, caz în care etapa 1 nu mai este necesară. Similar, anumite sisteme pot furniza numerele direct în forma cu cifre arabe, ceea ce ar face şi etapa 2 inutilă. Deasemeni, funcţie de domeniu, etapa de înlocuire de aliasuri poate apare ca oportună sau nu.

Pentru generalitate însă, în analizorul semantic proiectat în aceasta lucrare toate etapele de preprocesare prezentate sunt considerate necesare.

3.3 Modelul Markov

Modelele Markov pot fi considerate actualmente o metodă consacrată în recunoaşterea automată a vorbirii şi prelucrarea limbajului natural. Există cel puţin două aspecte care au condus la impunerea lor [?]: un prim avantaj vine din faptul că sunt bazate pe o teorie matematică riguroasă, cercetările fiind deci susţinute de numeroase rezultate matematice acumulate anterior. Un al doilea avantaj este faptul că aceste modele pot fi antrenate pe baza unor date culese în condiţii "reale", lucru care face ca sistemele construite pe baza lor să prezinte o robusteţe sporită.

În continuare se va face o scurtă prezentare teoretică a modelelor Markov, după care se va trece la descrierea algoritmilor şi tehnicilor specifice utilizate în contextul antrenării modelului şi al realizării decodării semantice pe baza lui.

3.3.1 Definiţie. Probleme fundamentale

Definiţia 1 *Prin model Markov cu n stări ascunse şi m simboluri observate vom înţelege un cvintuplu $\mathcal{M} =< \Omega, S, A, B, \Pi >$, unde:*

- $\Omega = \{o_1, o_2, \ldots, o_m\}$ *este o mulţime de simboluri observate sau generate;*
- $S = \{s_1, s_2, \ldots, s_n\}$ *este mulţimea stărilor modelului;*
- $A = \{a_{ij}\}$, $a_{ij} = P(s_j^{t+1}|s_i^t)$, $i, j = \overline{1,n}$, *reprezintă matricea probabilităţilor tranziţiilor între stări, în care fiecare linie este o distribuţie de probabilitate corespunzătoare unei stări;*
- $B = \{b_{ik}\}$, $b_{ik} = P(o_k^t|s_i^t)$, $i = \overline{1,n}$, $k = \overline{1,m}$, *reprezintă distribuţiile de probabilitate ale simbolurilor în stările modelului;*
- $\Pi = \{\pi_i\}$, $\pi_i = P(s_i^1)$, $i = \overline{1,n}$, *reprezintă distribuţia de probabilitate a stării iniţiale.*

În esenţă un astfel de model este format dintr-o mulţime de stări S între care tranziţiile se fac cu anumite probabilităţi, date prin matricea A: $a_{ij} = P(s_j^{t+1}|s_i^t)$ defineşte probabilitatea ca modelul să ajungă în starea s_j la momentul $t+1$ dacă se afla în starea s_i la momentul t, sau cu alte cuvinte, probabilitatea ca modelul să efectueze o tranziţie din starea s_i în starea s_j. În plus, $\pi_i = P(s_i^1)$ reprezintă probabilitatea ca modelul să se afle în starea s_i la momentul iniţial, deci Π modelează probabilistic starea iniţială.

În fiecare din aceste stări, modelul consumă sau generează, funcţie de modul în care este utilizat, un simbol din mulţimea Ω, cu o probabilitate dată de distribuţia de probabilitate asociată stării respective – $b_{ik} = P(o_k^t|s_i^t)$ reprezintă deci probabilitatea în starea s_i a simbolului o_k.

Figura **??** ilustrează un model Markov cu 4 stări şi 3 simboluri. După cum se observă, unele probabilităţi de tranziţie sau ale simbolurilor pot fi nule. Mecanismul de funcţionare al unui astfel de model este următorul: la momentul iniţial, modelul intră într-o stare iniţială conform distribuţiei de probabilitate a stării iniţiale Π. În acea stare are loc emiterea unui simbol din mulţimea Ω conform distribuţiei de probabilitate a simbolurilor asociate stării respective. În continuare, se efectuează o tranziţie spre o nouă stare în conformitate cu distribuţia de probabilitate a tranziţiilor, urmată de o nouă emitere a unui simbol, şi aşa mai departe. Specificarea unui model constă deci în definirea stărilor, a simbolurilor (în esenţă a numărului acestora), şi a matricilor A, B, şi Π.

Se poate spune deci că un model Markov cu stări ascunse (în continuare vom utiliza prescurtarea engleză HMM – Hidden Markov Model) este un model dublu

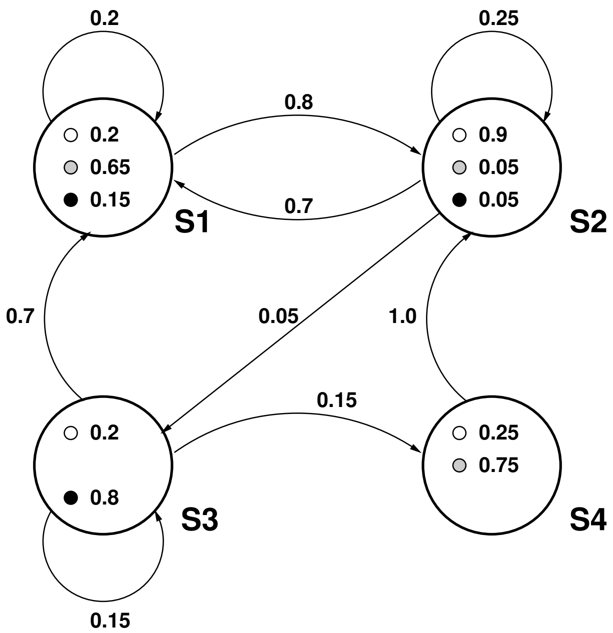


Figura 3.2: Exemplu de model Markov

stocastic [?]: el este format dintr-un proces stocastic ascuns (tranziţiile între stări), care nu poate fi observat decât prin intermediul unul alt set de procese stocastice, care generează simbolurile observate.

Utilizarea modelelor HMM în modelarea efectivă a unor procese presupune rezolvarea a trei probleme fundamentale [?]:

1. **Evaluarea HMM -** Determinarea probabilităţii $P(\mathcal{O}|\mathcal{M})$ ca un anumit model $\mathcal{M}$ să fi generat o secvenţă de simboluri $\mathcal{O} = (q_1, q_2, ...q_T)$.

2. **Decodarea stărilor unui HMM -** Dată fiind o succesiune de simboluri $\mathcal{O} = (q_1, q_2, ...q_T)$ şi un model $\mathcal{M}$, se cere determinarea secvenţei de stări în care simbolurile ar fi putut fi generate, $\mathcal{S}$, optimă într-un anumit sens[1].

3. **Antrenarea HMM -** Ajustarea parametrilor unui model $\mathcal{M}$ astfel încât să se maximizeze probabilitatea ca o secvenţă de simboluri $\mathcal{O} = (q_1, q_2, ...q_T)$ să fi fost generată de $\mathcal{M}$.

Pentru toate aceste probleme există soluţii eficiente [?], care au impus HMM ca tehnică de modelare stocastică.

În continuare vom expune modul în care un astfel de model poate fi utilizat pentru decodarea semantică.

[1] Se pot defini mai multe criterii de optim [?].

3.3.2 Modelele Markov în analiza semantică

Problema realizării analizei semantice este în esenţă o problemă de decodare. Scopul analizorului semantic este de a obţine corespondentul semantic (într-un anumit formalism) al textului de intrare. Practic, în cazul sistemului proiectat, se cere ca, plecând de la textul de intrare normalizat, să determinăm şirul de etichete semantice corespunzatoare (parse).

În acest sens, analiza semantică privită ca o decodare a textului de intrare poate fi modelată prin intermediul unui HMM. Cuvintele din textul normalizat vor juca rolul de simboluri observate ale modelului, iar etichetele semantice vor corespunde stărilor. Astfel, dat fiind un model $\mathcal{M}$ şi o secvenţă de intrare $\mathcal{O}$ (textul normalizat) se poate determina (problema 2) cea mai probabilă succesiune de stări interne $\mathcal{S}$ care generează intrarea $\mathcal{O}$, cu alte cuvinte, cea mai probabilă succesiune de etichete semantice corespondente (vezi figura **??**).

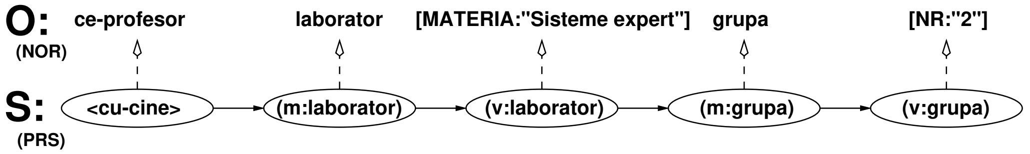


Figura 3.3: Utilizarea modelelor Markov în decodarea semantică

Rămâne problema creării modelului $\mathcal{M}$ astfel încât decodarea să se desfăşoare corect, adică respectând semantica domeniului şi a limbii în discuţie. Aşa cum s-a menţionat anterior, avantajul principal al metodei stocastice de analiză semantică este chiar faptul că modelul se crează prin antrenare pe baza unui corpus de formulări caracteristice domeniului de aplicaţie al sistemului. În mod evident, dimensiunile modelului (numărul de stări şi numărul de simboluri observate) sunt date de numărul de etichete semantice şi respectiv dimensiunea vocabularului normalizat. Odată dimensiunile stabilite, parametrii A, B şi Π se pot determina pe baza corpusului de antrenament. Matricea A, reprezentând probabilităţile de tranziţie între stări, va capta practic modurile posibile în care etichetele semantice se pot succeda. În mod similar, matricea B, reprezentând probabilităţile simbolurilor observate, va capta corespondenţa dintre etichetele semantice şi lexemele textului, iar împreună, cele două matrice vor modela corespondenţa dintre textul normalizat şi reprezentarea sa semantică.

3.3.3 Antrenarea modelului

Antrenarea modelului Markov are ca scop determinarea parametrilor lui astfel încât să modeleze cât mai corect (în sens statistic) semantica formulărilor, şi se realizează pe baza unui corpus de antrenament compus din formulări normalizate şi secvenţele de etichete semantice corespunzătoare. Cuvintele ce apar în formulările normalizate vor constitui simbolurile observate ale modelului, iar etichetele semantice vor corespunde stărilor.

Dimensiunile modelului sunt determinate anterior începerii antrenării: numărul etichetelor semantice distincte ce apar în corpus va fi numărul de stări (n), iar dimensiunea vocabularului formulărilor normalizate dă numărul de simboluri (m). În aceste condiţii, antrenarea se reduce la determinarea parametrilor a_{ij}, b_{ik} şi π_i.

Antrenarea prin estimare de probabilitate maximă

Estimarea parametrilor A, B şi Π se poate face pe baza numărării apariţiilor evenimentelor corespunzătoare (tranziţiile între stări şi corespondenţele stări-simboluri) în corpusul de antrenament. Introducem următoarele notaţii:

- ω_{ij} - reprezintă o tranziţie între stările s_i şi s_j, altfel spus o succesiune de două etichete semantice sau un *bigram* (s_i, s_j);

- $c(\omega_{ij})$ = numărul de apariţii în corpus a unei tranziţii între stările s_i şi s_j (de câte ori apare în corpus bigram-ul (s_i,s_j));

- C_i = numărul de apariţii în corpus a stării (etichetei semantice) s_i ca primă stare a unui bigram:

$$C_i = \sum_{j=1}^{n} c(\omega_{ij}) \tag{3.1}$$

- o_{ik} = numărul de generări ale simbolului observat o_k din starea s_i (de câte ori apare în corpus o corespondenţă între eticheta semantică s_i şi cuvântul normalizat o_k);

- O_i = numărul total de simboluri generate în starea s_i:

$$O_i = \sum_{k=1}^{m} o_{ik} \tag{3.2}$$

- p_i = numărul de apariţii în corpus a stării (etichetei) s_i pe prima poziţie a unei formulări;

- P = numărul de formulări din corpus:

$$P = \sum_{i=1}^{n} p_i \tag{3.3}$$

Cu aceste notaţii, parametrii A, B şi Π se calculează astfel:

$$a_{ij} = P(s_j^{t+1}|s_i^t) = \frac{c(\omega_{ij})}{C_i}\ , \qquad i,j=\overline{1,n} \tag{3.4}$$

$$b_{ik} = P(o_k^t|s_i^t) = \frac{o_{ik}}{O_i}\ , \qquad i=\overline{1,n}\ ,\ k=\overline{1,m} \tag{3.5}$$

$$\pi_i = P(s_i^1) = \frac{p_i}{P}\ , \qquad i=\overline{1,n} \tag{3.6}$$

formulele ??, ?? şi ?? asigurând faptul că a_{ij}, b_{ik} şi π_i satisfac condiţia de normalizare a distribuţiilor de probabilitate : $\sum_{j=1}^{n} a_{ij} = 1$, $\sum_{k=1}^{m} b_{ik} = 1$ şi $\sum_{i=1}^{n} \pi_i = 1$.

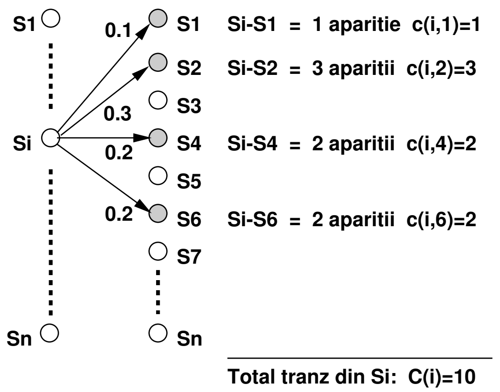


Figura 3.4: Estimarea de probabilitate maximă

Coeficienţii $c(\omega_{ij})$, o_{ik} şi p_i pot fi uşor determinaţi prin numărare dintr-o singură parcurgere a corpusului de antrenament, antrenarea având deci complexitatea O(n) în raport cu dimensiunea acestuia. Aceast mod de estimare a parametrilor modelului poartă numele de *estimare de probabilitate maximă* (Maximum Likelihood Estimation), datorită faptului că maximizează probabilitatea ca simbolurile observate să fi fost generate de modelul rezultat. Aşa cum ilustrează şi figura ??, probabilitatea de tranziţie între două stări este proporţională cu numărul de apariţii în corpus a tranziţiei între etichetele semantice corespunzătoare. Similar, distribuţiile de probabilitate ale simbolurilor pentru fiecare stare reflectă proporţiile corespondenţelor etichetă-cuvânt din corpusul de antrenament.

Observaţie: Dacă o anumită succesiune de etichete semantice nu apare în corpusul de antrenament, parametrul $c(\omega_{ij})$ respectiv va fi zero, şi în consecinţă şi probabilitatea calculată a_{ij} a tranziţiei între stările respective va fi nulă (fig. ??).

Ameliorarea modelului prin metoda Katz

Observaţia anterioară introduce şi o problemă în ceea ce priveşte utilizarea cu succes a unui model Markov în decodarea semantică. În condiţiile unei dimensiuni relativ reduse a corpusului de antrenament, este foarte probabil ca acesta să nu surprindă toate succesiunile valide de etichete semantice. În consecinţă, modelul va conţine o mulţime de parametri nuli, care grevează faza de decodare (vezi

??), în sensul excluderii totale a succesiunilor de etichete semantice care nu au fost observate. Evenimentele slab reprezentate, sau care nu sunt reprezentate în corpus, nu vor fi deci modelate corect dacă utilizăm metoda de antrenare prin estimare de probabilitate maximă.

Pentru ameliorarea situaţiilor de acest tip se pot utiliza o serie de *metode de netezire*, prin care se realizează o ameliorare a modelului în condiţii de raritate a datelor, astfel încât să fie admisibile şi evenimente neobservate. În analizorul proiectat am utilizat în acest scop metoda de reestimare Katz [**?**], metodă al carei suport teoretic pentru modelele la nivel de bigrame va fi prezentat în continuare (pentru varianta generalizată la m-grame vezi [**?**]).

Ideea fundamentală a reestimării Katz este de a micşora într-o anumită măsură probabilitatea bigramelor observate, şi de a redistribui masa de probabilitate astfel eliberată între bigramele neobservate în corpusul de antrenament.

În cele ce urmează vom nota cu n_k numărul bigramelor distincte care apar în corpusul de antrenament de k ori şi cu N numărul total de bigrame din corpus. Atunci, este satifăcută relaţia

$$N = \sum_k k \cdot n_k \tag{3.7}$$

Reestimarea Katz înlocuieşte estimările de probabilitate maximă ale probabilităţilor bigramelor calculate la nivelul întregului corpus:

$$P(\omega_{ij}) = \frac{c(\omega_{ij})}{N} \tag{3.8}$$

cu estimările Turing corespunzătoare, definite prin relaţia:

$$P_T(\omega_{ij}) = \frac{c^*(\omega_{ij})}{N} \tag{3.9}$$

unde:

$$c^*(\omega_{ij}) = (c(\omega_{ij}) + 1) \cdot \frac{n_{c(\omega_{ij})+1}}{n_{c(\omega_{ij})}} \tag{3.10}$$

Raportul $\frac{c^*(\omega_{ij})}{c(\omega_{ij})}$ poartă denumirea de *coeficient de reducere Turing* (Turing discount coefficient), şi se notează cu $d_{c(\omega_{ij})}$. Cu aceste notaţii este evident că:

$$P_T(\omega_{ij}) \quad = \quad d_{c(\omega_{ij})} \cdot P(\omega_{ij}) \tag{3.11}$$

Atunci, pe baza relaţiilor de mai sus şi ţinând cont că:

$$\sum_{\omega_{ij}:c(\omega_{ij})>0} P(\omega_{ij}) \quad = \quad 1, \tag{3.12}$$

se poate arăta că masa de probabilitate F eliberată prin utilizarea estimărilor Turing este:

$$
\begin{aligned}
F &= \sum_{\omega_{ij}:c(\omega_{ij})>0} \Big(P(\omega_{ij}) - P_T(\omega_{ij})\Big) \\
&= \sum_{r>0} n_r \cdot (1-d_r) \cdot \frac{r}{N} = \frac{n_1}{N} \qquad (3.13)
\end{aligned}
$$

În acelaşi timp:

$$
\sum_{\omega_{ij}:c(\omega_{ij})=0} P_T(\omega_{ij}) = 1 - \sum_{\omega_{ij}:c(\omega_{ij})>0} P_T(\omega_{ij}) = \frac{n_1}{N} = F \qquad (3.14)
$$

Masa de probabilitate eliberată F poate fi deci redistribuită bigramelor care nu au apărut în corpusul de antrenament. Redistribuirea se poate face proporţional cu probabilităţile Turing de apariţie în corpus a stărilor spre care se efectuează tranziţia. Atunci, dacă a_{ij} sunt probabilităţile tranziţiilor calculate conform (**??**), ele se modifică astfel:

$$
c(\omega_{ij}) > 0: \quad a'_{ij} = d_{c(\omega_{ij})} \cdot a_{ij} \qquad (3.15)
$$

$$
c(\omega_{ij}) = 0: \quad a'_{ij} = \frac{1 - \sum_{\omega_{ik}:c(\omega_{ik}>0)} d_{c(\omega_{ij})} \cdot a_{ij}}{\sum_{\omega_{ik}:c(\omega_{ik})=0} P_T(\omega_k)} \cdot P_T(\omega_j) \qquad (3.16)
$$

În cele prezentate anterior, reducerea probabilităţii prin înlocuirea cu estimările Turing corespunzătoare s-a făcut pentru toate bigramele reprezentate în corpus, pentru care $c(\omega_{ij}) > 0$ – vezi ecuaţia (**??**).

Metoda de reestimare Katz propune însă lăsarea intactă a probabilităţilor bigramelor care apar de un număr relativ mare de ori în corpus, întrucât pentru ele se poate presupune că estimările sunt de încredere (reliable) [**?**]. Reducerea probabilităţilor prin înlocuirea lor cu estimările Turing se va face doar pentru bigramele slab reprezentate în corpus, de un număr de ori cel mult egal cu un parametru fixat K (vezi figura **??**). În acest caz, pentru a păstra masa de probabilitate eliberată şi distribuită bigramelor inexistente în corpus la aceeaşi valoare dată de formula (**??**), este necesară o recalculare a coeficienţilor de reducere:

$$
c(\omega_{ij}) > K \quad : \quad d'_{c(\omega_{ij})} = 1, \qquad (3.17)
$$

$$
1 \le c(\omega_{ij}) \le K \quad : \quad d'_{c(\omega_{ij})} = \frac{d_{c(\omega_{ij})} - \dfrac{(K+1)\cdot n_{K+1}}{n_1}}{1 - \dfrac{(K+1)\cdot n_{K+1}}{n_1}} \qquad (3.18)
$$

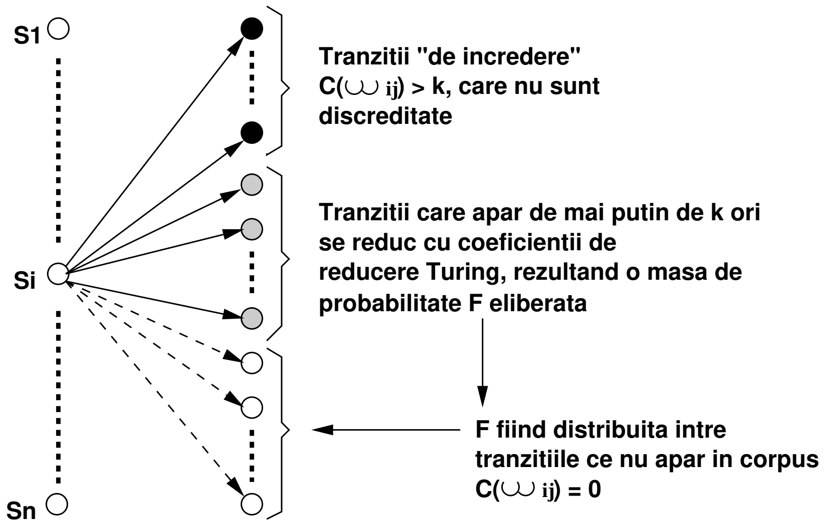


Figura 3.5: Principiul reestimării Katz

reestimarea făcându-se tot după formulele (**??**) şi (**??**), dar cu coeficienţii de reducere d'.

Practic, algoritmul de reestimare Katz porneşte de la estimările de probabilitate maximă a_{ij}, determină coeficienţii de reducere Turing d şi d', după care pe baza formulelor (**??**) şi (**??**) reestimează matricea A.

În mod similar, algoritmul de reestimare Katz se poate aplica şi distribuţiei de probabilitate a stării iniţiale Π prin ajustarea estimărilor de probabilitate maximă π_i cu un set corespunzător de coeficienţi de reducere.

Se elimină astfel valorile nule din A şi Π, mărind robusteţea şi fiabilitatea modelului în faţa unor evenimente (structuri semantice) care nu au apărut în corpusul de antrenament.

3.3.4 Decodarea semantică

Problema decodării semantice bazată pe modelul Markov revine la a determina succesiunea de stări $\mathcal{S}$ cea mai probabilă să fi generat o secvenţă de simboluri $\mathcal{O}$.

Fie $\mathcal{O} = (q_1, q_2, \ldots, q_T)$ o secvenţă de simboluri observate şi $\mathcal{S} = (t_1, t_2, \ldots, t_T)$ o succesiune de stări de lungime T. Atunci probabilitatea ca un model $\mathcal{M}$ să fi generat secvenţa $\mathcal{O}$ trecând prin stările $\mathcal{S}$ este:

$$\begin{aligned} P(\mathcal{O}|\mathcal{S},\mathcal{M}) &= \pi_{t_1} b_{t_1 q_1} a_{t_1 t_2} b_{t_2 q_2} a_{t_2 t_3} b_{t_3 q_3} \ldots a_{t_{T-1} t_T} b_{t_T q_T} \\ &= \pi_{t_1} b_{t_1 q_1} \prod_{k=1}^{T-1} a_{t_k t_{k+1}} b_{t_{k+1} q_{k+1}} \end{aligned} \tag{3.19}$$

Problema decodării se reduce deci la găsirea succesiunii de stări $\mathcal{S}$ care maximizează această probabilitate:

$$\mathcal{S} = \arg\max_{S}\{P(\mathcal{O}|S, \mathcal{M})\}$$

Din ecuaţiile de mai sus se poate observa cu uşurinţă că o căutare exhaustivă în spaţiul succesiunilor de stări $\mathcal{S}$ nu este practică, având o complexitate $O(Tn^T)$, unde n este numărul de stări ale modelului şi T este lungimea secvenţei observate. Un algoritm mai eficient pentru determinarea secvenţei $\mathcal{S}$ este algoritmul Viterbi [**?**], prezentat în continuare.

Algoritmul Viterbi este un algoritm de programare dinamică care calculează iterativ, pentru subşirurile prefix ale secvenţei observate de lungimi $1, 2, \ldots, T$, valorile maxime ale probabilităţii P, precum şi succesiunile de stări corespunzătoare. Astfel, succesiunea de stări $\mathcal{S}$ este construită incremental, pornind de la prefixele $(q_1), (q_1, q_2), (q_1, q_2, q_3)$, până la secvenţa completă $\mathcal{O} = (q_1, \ldots, q_T)$.

Se defineşte variabila $\delta_k(j)$ care exprimă probabilitatea maximă ca modelul $\mathcal{M}$ să fi generat prefixul de lungime k – $(q_1, q_2, ...q_k)$. Atunci δ poate fi exprimat recursiv astfel:

$$\delta_1(i) = \pi_i \cdot b_{iq_1} \qquad i = \overline{1, n} \tag{3.20}$$

$$\delta_k(j) = \max_{1\le i\le n}\{\delta_{k-1}(i) \cdot a_{ij}\} \cdot b_{jq_k} \qquad j = \overline{1, n} \tag{3.21}$$

caz în care probabilitatea maximă căutată este:

$$\max_{S}\{P(\mathcal{O}|S, \mathcal{M})\} = \max_{1\le i\le n}\{\delta_T(i)\} \tag{3.22}$$

Scopul algoritmului de decodare Viterbi fiind determinarea succesiunii de stări corespunzătoare acestei valori maxime, este necesară şi reţinerea succesiunii stărilor care maximizează δ. În acest scop, se defineşte:

$$\psi_k(j) = \arg\max_{1\le i\le n}\{\delta_{k-1}(i) \cdot a_{ij}\} \qquad j = \overline{1, n} \tag{3.23}$$

Algoritmul este prezentat în pseudocod în figura **??** şi ilustrat în figura **??**. El începe cu o etapă de iniţializare în care sunt determinate valorile δ_{1i}. Apoi, în etapa iterativă se ia în considerare pe rând câte un nou simbol din secvenţa $\mathcal{O}$. Pentru fiecare stare, se determină probabilitatea cu care modelul poate ajunge în starea respectivă şi genera secvenţa observată până la simbolul curent (se calculează valoarea δ_k pe baza valorilor deja calculate δ_{k-1}). În acelaşi timp se reţine pentru fiecare stare care este starea anterioară care a realizat maximul probabilităţii (valorile ψ).

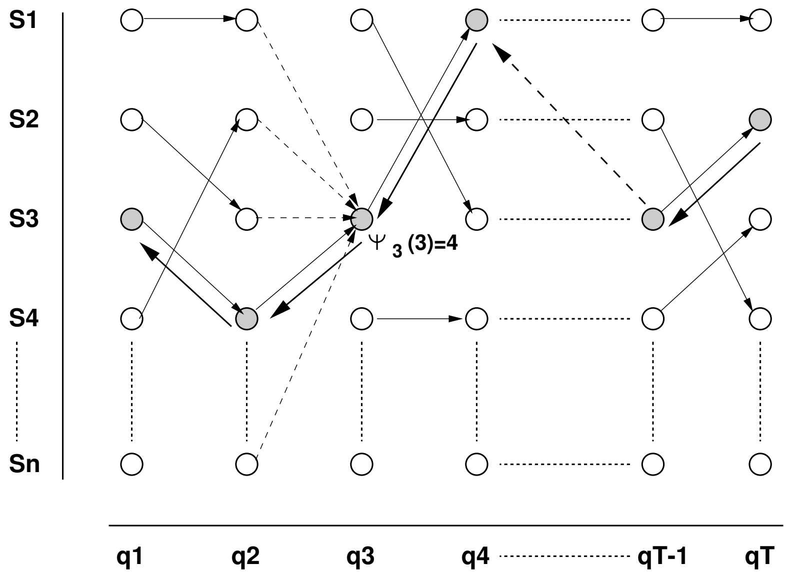


Figura 3.6: Algoritmul Viterbi

`/* -- iniţializare --------- */`
`pentru i de-la 1 la n`
  $\delta_1(i) = \pi_i \cdot b_{iq_1}$
  $\psi_1(i) = 0$
`/* -- iteraţii ------------- */`
`pentru k de-la 2 la T`
  `pentru j de-la 1 la n`
    $\delta_k(j) = \max\limits_{1 \leq i \leq n} \{\delta_{k-1}(i) \cdot a_{ij}\} \cdot b_{jq_k}$
    $\psi_k(j) = \arg\max\limits_{1 \leq i \leq n} \{\delta_{k-1}(i) \cdot a_{ij}\}$
`/* -- terminare ------------ */`
$PMAX = \max\limits_{1 \leq i \leq n} \{\delta_T(i)\}$
$t_T = \arg\max\limits_{1 \leq i \leq n} \{\delta_T(i)\}$
`/* -- parcurgere inversă --- */`
`pentru k de-la T-1 la 1`
  $t_k = \psi_{k+1}(t_{k+1})$

Figura 3.7: Pseudocod Viterbi

După ce toate simbolurile secvenţei observate au fost prelucrate, valoarea probabilităţii maxime este dată, conform (**??**), de maximul valorilor δ_{Ti}. Starea care realizează acest maxim este practic ultima stare t_T din $\mathcal{S}$. Pornind de la ea, structura descrisă în figură este parcursă în sens invers pe baza indicilor ψ, identificându-se stările ce compun secvenţa decodată $\mathcal{S}$.

O analiză simplă a complexităţii în timp a algoritmului Viterbi arată că aceasta este de ordinul $O(Tn^2)$, deci polinomială şi mult redusă faţă de căutarea exhaustivă care are complexitatea $O(Tn^T)$. Algoritmul Viterbi oferă deci o soluţie eficientă pentru decodarea stărilor unui model Markov pornind de la o secvenţă observată dată. Împreună cu algoritmii de antrenare a modelului, el constituie baza pentru realizarea analizei semantice în sistemul proiectat.

3.4 Constructorul de cadre

Constructorul de cadre este ultimul modul din analizorul semantic. Întrucât decodorul furnizează doar secvenţa de etichete semantice corespunzătoare textului de intrare, este necesar acest modul suplimentar pentru a realiza translatarea în reprezentarea prin cadre (figura **??**).

Text normalizat	cu-cine grupă [NR:"2"] [ZI-SAPT:"luni"] [PERIOADA:"dimineaţă"] laborator [MATERIA:"Inteligenţă artificială"]
Etichete semantice	<cu-cine> (m:grupa) (v:grupa) (v:zi-sapt) (v:perioada-zi) (m:laborator) (v:laborator)
Cadru	<cu-cine> <identificare> grupa = 2 <specif-timp> zi-sapt = luni perioada-zi = dimineaţă <specif-materie> laborator = "Inteligenţă artificială"
Cadru normalizat	<cu-cine> <identificare> grupa = 2 <specif-timp> zi-sapt = 1 perioada-zi = DIM <specif-materie> laborator = C024

Figura 3.8: Corespondenţa succesiune de etichete semantice – cadru

Formalismul case-grammar implică trei tipuri de etichete semantice: concepte – reprezentate aici sub forma <concept>, marcaje – reprezentate sub forma (m:marcaj), şi valori – reprezentate sub forma (v:valoare). Pentru a realiza instanţierea cadrelor, constructorul se bazează însă numai pe concepte şi valori. Pe baza conceptelelor sunt determinate cadrele care trebuie instanţiate, iar valorile sunt utilizate pentru completarea sloturilor. Marcajele nu au un rol direct în construirea cadrelor, ele modelând limbajul doar la nivelul decodării semantice.

Algoritmul utilizat de constructorul de cadre este relativ simplu, însă are câteva particularităţi importante. Într-un prim pas se identifică printr-o parcurgere a succesiunii de etichete semantice toate conceptele care apar. Pe baza lor, cunoscând întregul sistem de cadre utilizat, constructorul instanţiază cadrele corespunzătoare, împreună cu eventualele subcadre ale acestora. Trebuie remarcat că, datorită posibilităţii apariţiei unor erori de recunoaştere şi modalităţii stocastice de realizare a decodării semantice, nu există nici o garanţie în privinţa faptului că succesiunea de etichete semantice va conţine un concept: într-o astfel de succesiune pot exista nici una, una, sau mai multe etichete-concept. Primul caz prezintă anumite particularităţi, şi va fi tratat separat.

După identificarea unuia sau mai multor concepte şi instanţierea cadrelor respective, constructorul realizează o nouă parcurgere a succesiunii de etichete semantice, de această dată urmărind numai valorile. Fiecare valoare este ataşată la toate cadrele instanţiate care au un slot corespunzător; cu alte cuvinte, dacă există mai multe concepte în formulare, se încearcă completarea fiecăruia din cadrele instanţiate cu toate valorile ce apar în formulare.

Un ultim pas îl constituie normalizarea cadrelor. Această operaţie este similară normalizării realizate prin preprocesare. Rolul ei este de a adapta cadrul pentru utilizarea lui în continuare de către modulul de control al dialogului spre care va fi transmis. În principiu această operaţie unifică diverse sinonime care pot apărea ca valori ale sloturilor, şi eventual poate realiza conversii ale acestor valori pentru a uşura utilizarea lor ulterioară, de exemplu valorile sloturilor "laborator" sunt transformate din valori textuale în coduri de acces la baza de date utilizată.

Formulări fără concept

Un caz aparte îl constituie formulările utilizator care după decodare nu conţin nici o etichetă semantică concept. Evident, această situaţie necesită o tratare specială prin care să se determine ce cadru trebuie instanţiat.

Pentru exemplificare, vom porni de la următoarea cerere de detalii ipotetică pe care un sistem de dialog o adresează unui utilizator:

- Dacă sunteţi student, precizaţi anul, grupa, subgrupa.

Două posibile răspunsuri sunt ilustrate în figura **??**. Deşi ambele formulări sunt valide semantic, şi decodorul realizează cu succes obţinerea succesiunii de etichete semantice corecte în ambele cazuri, al doilea răspuns nu poate fi transformat direct în reprezentarea prin cadre datorită lipsei unui concept.

Sistem	Dacă sunteţi student, precizaţi anul, grupa, subgrupa.
Utilizator 1	sunt student în anul întâi grupa doi subgrupa doi
Parse 1	<identificare> (m:anul) (v:anul) (m:grupa) (v:grupa) (m:subgrupa) (v:subgrupa)
Utilizator 2	anul întâi grupa doi subgrupa doi
Parse 2	(m:anul) (v:anul) (m:grupa) (v:grupa) (m:subgrupa) (v:subgrupa)

Figura 3.9: Formulări fară concept

Această lipsă a conceptului se manifestă practic datorită lipsei unui cuvânt în formularea utilizatorului care să declanşeze un concept (cum este cuvântul "sunt" din prima formulare). Sensul propoziţiei este însă clar dacă se ia în considerare şi întrebarea sistemului. Soluţia utilizată în aceste situaţii este ca modulul de control al dialogului să transmită analizorului semantic care sunt conceptele aşteptate de la utilizator în momentul respectiv.

Rolul modulului de control al dialogului este de a realiza gestionarea dialogului şi a interacţiunii cu utilizatorul, şi, în acest sens, el este capabil să precizeze (în cel mai rău caz cu o anumită probabilitate) care este conceptul aşteptat de la utilizator. Trebuie remarcat că majoritatea formulărilor fără concept apar chiar ca urmare a unor întrebări de clarificare din partea sistemului, deoarece în aceste cazuri întrebarea sistemului creează contextul în care apare răspunsul: în toate aceste situaţii modulul de control al dialogului a lansat întrebarea, deci el "ştie" în principiu care este conceptul aşteptat de la utilizator (de exemplu, după întrebarea de mai sus, este clar că sistemul aşteaptă conceptul <identificare>). Ceva mai delicate sunt situaţiile în care modulul de control se află într-un punct în care este rândul utilizatorului să preia iniţiativa dialogului. În acest caz, el poate furniza doar probabilităţi asociate conceptelor aşteptate.

Pentru a unifica situaţiile am presupus că, la analiza semantică a fiecărei formulări utilizator, modulul de control al dialogului furnizează o listă de concepte aşteptate, însoţită de probabilităţile asociate fiecăruia. Această listă este utilizată în cazul în care decodarea semantică nu reuşeşte doar pe baza formulării utilizatorului (nu este detectat un concept). În această situaţie se reia decodarea introducând pe rând câte un concept din listă în faţa succesiunii observate (astfel de secvenţe extinse sunt utilizate şi în antrenarea modelului – vezi anexa **??**). De această dată conceptul va fi identificat, iar în final este aleasă varianta care maximizează produsul dintre probabilitatea conceptului şi probabilitatea de observare a succesiunii de simboluri extinsă rezultată din decodare.

Dacă $\mathcal{L} = (c_1, c_2, \ldots, c_t)$ este lista de concepte primite, cu probabilităţile corespondente $(p_1, p_2, \ldots, p_t)$, iar $\mathcal{O} = (q_1, q_2, \ldots, q_T)$ este succesiunea de simboluri observate care prin decodare nu duc la apariţia unui concept, atunci practic se încearcă decodarea tuturor secvenţelor $\mathcal{O}_i = (c_i, q_1, q_2, \ldots, q_T)$. Dintre acestea este aleasă $\mathcal{O}_k$, unde $k = \arg\max_{1\le i\le t}\{p_i \cdot P(\mathcal{O}_i|\mathcal{M})\}$.

Observaţie: Se poate spune că, într-o anumită măsură, prin acest mecanism de transmitere a conceptelor aşteptate de către modulul de control al dialogului, analizorul semantic realizează şi o analiză contextuală parţială.

3.5 Detalii de implementare

În cele ce urmează vom prezenta diverse aspecte privind implementarea analizorului semantic proiectat în secţiunile anterioare.

Implementarea efectivă a fost realizată în limbajul C++, ales în principal datorită eficienţei, a capabilităţilor de programare orientată pe obiecte şi a portabilităţii codului scris în acest limbaj. Dezvoltarea propriu-zisă a fost făcută pentru sistemul de operare Linux, utilizând mediul integrat de dezvoltare KDevelop [?].

Forma finală în care analizorul semantic este distribuit este cea a unei biblioteci statice de clase, pentru a fi cu uşurinţă inclus într-un sistem de dialog proiectat într-un cadru mai larg. Pe baza acestei biblioteci pot fi realizate implementări ale analizorului semantic sub diverse forme: de la programe filtru care lucrează în pipe, până la servere la nivel sockets, RPC (Remote Procedure Call) sau RMI (Remote Method Invocation). Deşi în figurile care urmează nu sunt ilustraţi constructorii şi destructorii claselor, menţionăm că toate sunt scrise în forma ortodox-canonică (pentru o descriere completă a membrilor şi interfeţelor vezi anexa **??**).

În continuare prezentăm principalele clase, relaţiile dintre ele, precum şi algoritmii utilizaţi în realizarea funcţionalităţii lor. Se poate face o grupare a lor în două categorii: clase care modelează datele specifice problemei analizei semantice pe baze stocastice, şi clase care implementează algoritmii utilizaţi.

3.5.1 Modelarea datelor

Prima categorie, a claselor care modelează datele, cuprinde clasele CSequence, CStringSequence, CCaseFrame, CCaseFrameSystem, CUtterance, CUtteranceCorpus şi CDictionary , ilustrate în diagrama UML (Unified Modeling Language) din figura **??**. Figura prezintă relaţiile existente între aceste clase, precum şi membrii şi metodele lor[2].

- CStringSequence - modelează o succesiune de şiruri de caractere de lungime variabilă, fiind utilizată în principal pentru reprezentarea formei normalizate a unei formulări utilizator şi a succesiunii de etichete semantice corespondente. Implementarea se bazează pe tablouri alocate dinamic care se redimensionează, când este necesar, prin creşterea cu o cuantă fixată DELTA.

- CSequence - similară cu CStringSequence, modelează o secvenţă de numere. Este utilizată de algoritmii de antrenare şi decodare, care lucrează cu indecşii şirurilor de caractere într-un dicţionar în locul şirurilor respective.

- CCaseFrame - modelează cadrele specifice formalismului case-grammar. În esenţă, fiecare obiect cadru are un concept (membrul Concept) şi o listă de atribute şi valori (membrii Attributes şi Values). Este de remarcat că valorile pot fi şiruri de caractere sau alte cadre, situaţie modelată prin unionul CCaseValue. Clasa oferă metode pentru citirea definiţiei cadrului dintrun fişier, citirea şi scrierea instanţei cadrului din/într-un buffer, setarea şi obţinerea valorilor atributelor.

- CCaseFrameSystem - modelează un sistem de cadre conform formalismului case-grammar. Un astfel de sistem constituie practic o mulţime de cadre (CaseFrames), între care pot exista relaţii de tip părinte-fiu. Clasa permite

[2] Sunt reprezentate doar membrii şi metodele considerate semnificative.

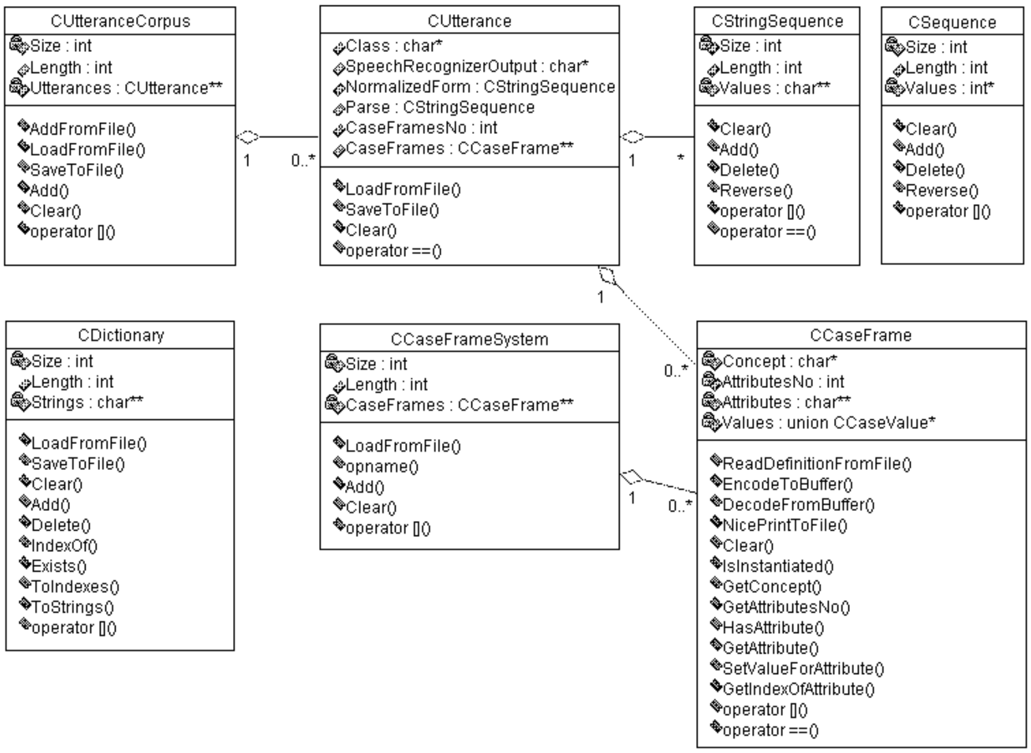


Figura 3.10: Clase pentru modelarea datelor

citirea specificaţiei unui sistem de cadre dintr-un fişier de control, de tipul celui ilustrat în anexa **??**.

- CUtterance - modelează o formulare utilizator, cu toate reprezentările prin care trece pe parcursul realizării analizei semantice: ieşire a sistemului de recunoaştere automată a vorbirii (SpeechRecognizerOutput), text normalizat (NormalizedForm), succesiune de etichete semantice (Parse) şi reprezentare prin cadre (CaseFrames). Sunt implementate metode pentru citirea şi scrierea formulării din/într-un fişier, utilizând formatul de corpus din anexa **??**.

- CUtteranceCorpus - modelează un corpus de formulări utilizator, oferind metode pentru citirea şi scrierea lui din/într-un fişier. Implementarea se bazează pe tablouri alocate dinamic de obiecte de tipul CUtterance.

- CDictionary - modelează un dicţionar, implementând metodele caracteristice acestei structuri de date. Implementarea este bazată pe un tablou alocat dinamic, iar căutarea este liniară[3].

[3]Dimensiunile mici ale dicţionarelor implicate fac suficientă o astfel de implementare.

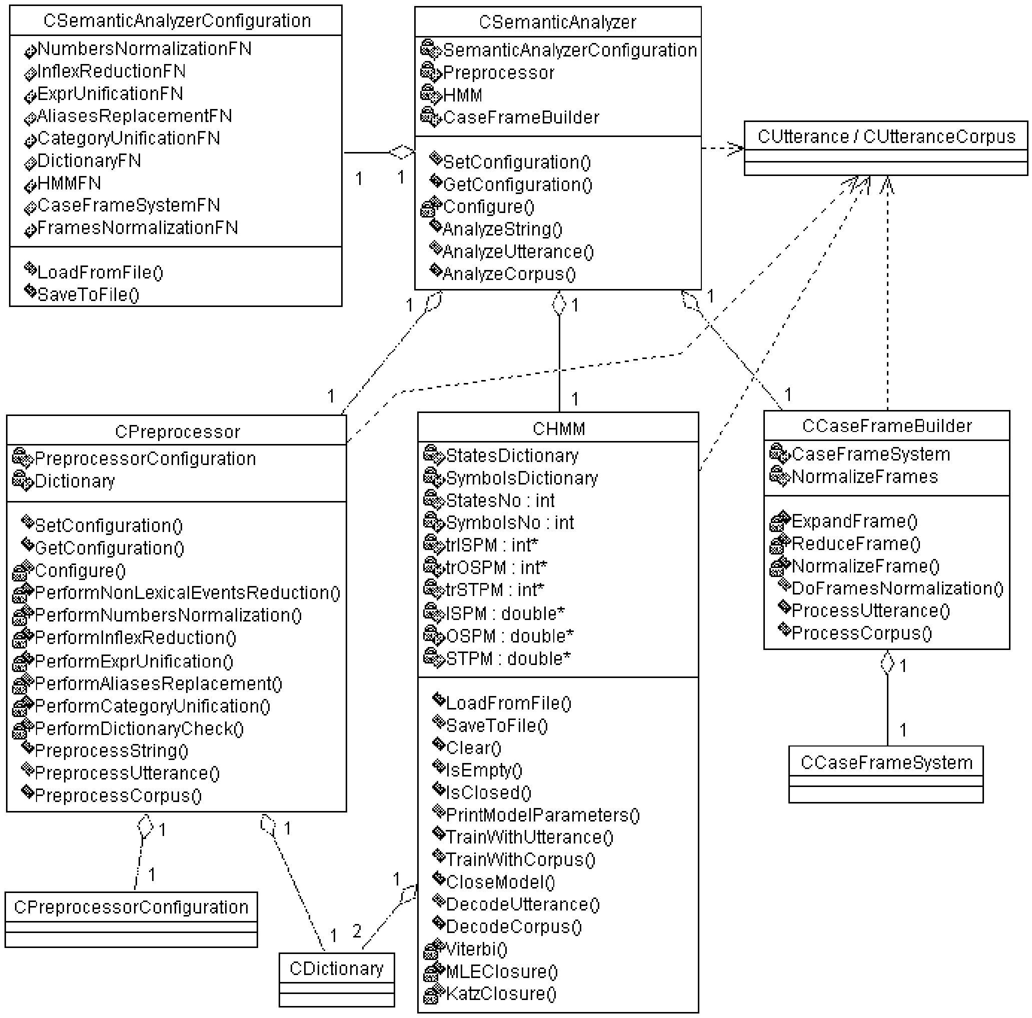


Figura 3.11: Clase pentru implementarea algoritmilor

3.5.2 Implementarea algoritmilor

Clasele care implementează algoritmii implicaţi în rezolvarea problemelor analizei semantice pe baze stocastice sunt prezentate în diagrama UML din figura **??**. Pe lângă acestea, mai există şi o serie de clase auxiliare, introduse pentru o mai bună modularizare şi pentru creşterea lizibitaţii codului.

- CPreprocessor - implementează funcţionalitatea modulului de preprocesare. Controlul preprocesorului se realizează pe baza unui obiect membru de tip CPreprocessorConfiguration prin care se specifică paşii de preprocesare ce trebuie efectuaţi şi fişierele care îi controlează. Clasa oferă metode pentru preprocesarea unui şir de caractere – PreprocessString, a unei formulări utilizator – PreprocessUtterance, sau a unui întreg corpus – PreprocessCorpus. Realizarea efectivă a preprocesării se face prin metodele private Perform. Eliminarea evenimentelor non-lexicale se realizează prin simpla parcurgere a textului de intrare şi eliminarea oricărui text în paranteze. Etapele de normalizare a numerelor, reducere a flexiunilor, unificare a expresiilor, înlocuire a aliasurilor, şi unificare a categoriilor, se realizează prin înlocuiri de şiruri de caractere, specificate prin intermediul unor fişiere de control (vezi anexa **??**). A fost construit un analizor lexical şi sintactic pentru aceste fişiere, şi prin analizarea lor se generează tabelele de înlocuiri care vor fi utilizate pentru efectuarea fiecăruia din paşii de preprocesare enumeraţi.

- CHMM - implementează modelul Markov, incluzând parametrii acestuia şi algoritmii de antrenare şi decodare utilizaţi. În componenţa modelului intră dicţionarele de simboluri şi etichete semantice (SymbolsDictionary şi StatesDictionary), precum şi distribuţiile de probabilitate A (STPM – State Transition Probability Matrix), B (OSPM – Observation Symbol Probability Matrix)) şi Π (ISPM – Initial State Probability Matrix). Variantele trISPM, trOSPM, trSTPM sunt utilizate în timpul antrenării modelului pentru a număra evenimentele respective. Clasa CHMM este serializabilă, existând metode pentru salvarea şi citirea modelelor în/din fişiere. Clasa oferă metode pentru antrenarea modelului folosind o formulare sau un întreg corpus (TrainWithUtterance şi TrainWithCorpus). Metoda CloseModel realizează închiderea modelului şi calculează matricile de probabilitate ISPM, OSPM şi STPM prin estimare de probabilitate maximă sau reestimare Katz (pe baza metodelor private MLEClosure şi KatzClosure). Deasemeni clasa oferă metode pentru realizarea decodării semantice (DecodeUtterance şi DecodeCorpus) pe baza algoritmului Viterbi.

- CCaseFrameBuilder - această clasă corespunde modulului de construcţie a cadrelor. Ea utilizează un sistem de cadre (CaseFrameSystem) care specifică toate cadrele posibile, şi pune la dispoziţie o metodă pentru construcţia reprezentării prin cadre a unei formulări utilizator (ProcessUtterance).

- CSemanticAnalyzer - modelează întreaga linie de analiză semantică, formată din modulele de preprocesare, decodare semantică şi construcţie a cadrelor. Configurarea analizorului se realizează prin intermediul unui obiect serializabil de tipul CSemanticAnalyzerConfiguration, care specifică parametrii necesari pentru fiecare din modulele implicate. Clasa oferă metode pentru analiza semantică a unei formulări utilizator dată ca şir de caractere sau ca obiect CUtterance. Aceste metode (AnalyzeString şi AnalyzeUtterance) au ca parametru opţional, pe lângă textul sau formularea utilizatorului, o listă de concepte cu probabilităţile corespunzătoare, utilizată conform algoritmului descris în secţiunea **??** pentru tratarea formulărilor fără concept. Deasemeni, clasa CSemanticAnalyzer oferă o metodă (AnalyzeCorpus) pentru realizarea analizei semantice a unui întreg corpus de formulări.

3.5.3 Programe utilitare

Pe baza bibliotecii menţionate au fost dezvoltate o serie de programe ce pot fi utilizate pe parcursul dezvoltării unui sistem de dialog pentru procesarea corpusurilor şi antrenarea şi evaluarea modelelor Markov implicate. În continuare se va face o scurtă descriere a fiecăruia (anexa **??** prezintă sumar instrucţiunile de utilizare, afişate la apelul fără parametri în linia de comandă).

- hmm_build
  Acest program constituie practic modulul de antrenare al modelului Markov. El operează în trei etape: crearea, antrenarea şi închiderea modelului. La creare este necesară specificarea vocabularului textului normalizat şi a mulţimii etichetelor semantice. Etapa de antrenare se realizează pe baza unui corpus de antrenament (pentru structura corpusului, vezi anexa **??**). În această etapă se realizează practic doar numărarea evenimentelor. În etapa de închidere a modelului sunt calculaţi efectiv parametrii A, B şi Π prin estimarea de probabilitate maximă, eventual cu aplicarea reestimării Katz (caz în care trebuie specificaţi şi parametrii corespunzători).

- hmm_print
  Afişează parametrii A, B şi Π ai unui model Markov într-o formă lizibilă pentru proiectanţi. Dacă modelul nu este închis, sunt afişate numărul tranziţiilor şi al observaţiilor, împreună cu distribuţiile bigramelor şi stărilor iniţiale (utile pentru studiul reestimării Katz). Pentru modelele închise sunt listate distribuţiile de probabilitate din A, B şi Π.

- hmm_decode
  Realizează decodarea semantică a unui corpus de formulări utilizator preprocesate (incluzând forma NOR - vezi anexa **??**). Trebuie specificat fişierul conţinând modelul Markov pe baza căruia se realizează decodarea.

- preproc
  Acesta este modulul de preprocesare a intrării. Programul operează asupra unui corpus de formulări utilizator în forma furnizată de modulul de recunoaştere a vorbirii (SRO), şi realizează preprocesarea lor în conformitate cu instrucţiunile transmise prin linia de comandă. Se pot specifica etapele de preprocesare care să fie efectuate, iar controlul lor se realizează prin intermediul unor fişiere de configurare (vezi anexa **??**).

- frame_build
  Operează asupra unui corpus de formulări decodate (în forma PRS - vezi anexa **??**), şi generează reprezentarea prin cadre pentru fiecare formulare. Specificaţiile sistemului de cadre utilizat sunt date prin intermediul unui fişier de control a cărui structură este ilustrată în anexa **??**. Se poate specifica şi un fişier pentru controlul normalizării cadrelor.

- corpus_stats
  Calculează diverse statistici ale corpusurilor de formulări utilizator: numărul de formulări şi distribuţiile lor pe clase sau funcţie de formele (SRO, NOR, PRS şi FRM) disponibile ale fiecăreia. Deasemeni calculează numărul de bigrame din corpus şi distribuţia etichetelor semantice.

- corpus_filter
  Acesta este un utilitar cu ajutorul căruia pot fi efectuate diverse operaţii asupra corpusurilor de formulări utilizator. Se pot specifica prin expresii regulate ce clase de formulări să se păstreze sau să se şteargă din corpus. Deasemeni, pot fi şterse anumite reprezentări (NOR, PRS, FRM) ale formulărilor utilizator dintr-un corpus.

- corpus_diff
  Compară două corpusuri, identifică şi eventual marchează diferenţele dintre ele. Este utilizat în etapele de dezvoltare şi evaluare a modelului stocastic, când se realizează comparaţii între decodările manuale şi automate ale corpusurilor.

- dict_build
  Este utilizat pentru construirea dicţionarelor de simboluri observate şi stări ale unui model Markov, plecând de la corpusul de antrenament. O altă facilitate este posibilitatea de a înregistra evoluţia dimensiunii dicţionarului pe măsura parcurgerii corpusului, lucru util pentru a observa eventuala ei stabilizare.

Capitolul 4

Experimente şi rezultate

Acest capitol va ilustra modul în care implementarea software descrisă în capitolul anterior poate fi utilizată efectiv pentru dezvoltarea unui analizor semantic dedicat pentru un sistem de dialog cu un anumit domeniu de aplicaţie, precum şi performanţele care se pot obţine.

După descrierea domeniului şi capabilităţilor sistemului ales ca exemplu, şi a colectării şi pregătirii datelor folosite pentru dezvoltarea şi evaluarea analizorului semantic, sunt prezentate etapele succesive de dezvoltare a acestuia, urmate de evaluarea performanţelor sale.

4.1 Domeniul şi capabilităţile sistemului de dialog

Prima etapă în construcţia unui sistem de dialog îl constituie analiza domeniului în care acesta va opera şi delimitarea capabilităţilor sale. Unul din rezultatele acestei etape va fi o definire clară a noţiunilor cu care va lucra analizorul semantic.

Lucrarea de faţă a urmărit integrarea analizorului într-un sistem care furnizează informaţii despre orarul Departamentului de Calculatoare. Aceasta alegere este justificată de faptul că domeniul este suficient de complex pentru a permite evaluări realiste ale tehnicilor utilizate în diversele module ale sistemului (recunoaşterea vorbirii, analiza semantică, controlul dialogului), şi în acelaşi timp abordabil în condiţiile resurselor limitate disponibile.

Analiza domeniului de aplicaţie a pornit de la baza de date din care se va face extragerea informaţiilor despre orar, care trebuie să conţină practic toate informaţiile disponibile în legătură cu orarul (o posibilă structură a unei astfel de baze de date este dată în anexa **??**). Au fost astfel identificate principalele tipuri de informaţii cu care va opera sistemul de dialog (materii, profesori, săli de clasă, ani de studii etc.), şi care stabilesc o primă delimitare a domeniului de aplicaţie.

Pentru a defini încă de la început capabilităţile sistemului, au fost stabilite următoarele caracteristici ale acestuia:

- sistemul furnizează informaţii despre orar;

- baza de date conţine orarul de pe semestrul curent;
- sistemul nu tratează interogări care implică prelucrări suplimentare ale informaţiilor din baza de date (exemplu: câte ore am în total luni?);
- sistemul lucrează cu noţiuni precum: materii, tipuri de ore (cursuri, seminarii, proiecte, laboratoare), profesori (nume şi grad), săli de clasă, ore, ani de studii, specializări, grupe, subgrupe;
- sistemul nu cunoaşte numele studenţilor, identificarea acestora făcându-se doar la nivel de an, specializare, grupă, subgrupă;
- sistemul acceptă majoritatea specificatorilor de timp: zile ale săptămânii, perioade ale zilei (dimineaţa, după-amiza, seara), ore exacte, intervale orare, exprimări relative ale zilei (azi, mâine, ieri);
- sistemul lansează subdialoguri de clarificare/obţinere de detalii suplimentare în momentele în care acest lucru este necesar.

Câteva dialoguri considerate tipice pentru acest sistem, înregistrate folosind mediul specializat pentru dezvoltarea sistemelor de dialog MDOWZ [?], [?], sunt prezentate în anexa ??.

După definirea domeniului şi a capabilităţilor sistemului de dialog în care va fi inclus, a doua etapă în dezvoltarea unui analizor semantic stocastic o constituie colectarea datelor ce vor fi utilizate în acest scop.

4.2 Colectarea şi pregătirea datelor

Resursa principală utilizată în dezvoltarea unui analizor semantic stocastic este un corpus de formulări utilizator, tipice sistemului de dialog în care va fi integrat, pe baza căruia se elaborează fişierele de control şi se antrenează modelul Markov.

Datele folosite pentru experimentele descrise în continuare au fost colectate folosind mediul specializat de dezvoltare a sistemelor de dialog MDOWZ [?], [?]. Au fost obţinute astfel patru corpusuri totalizând 1375 formulări utilizator. Primele trei, notate woz-i, woz-ii şi woz-iii, au fost utilizate pentru dezvoltarea analizorului, iar ultimul (woz-test) a fost păstrat pentru evaluarea acestuia.

Cele patru corpusuri au fost colectate în patru etape succesive, în perioada martie–iunie 2000. Pentru a asigura atât o acoperire minimă a domeniului de aplicaţie, cât şi o varietate cât mai mare a formulărilor utilizatorilor, au fost înregistrate dialoguri bazate pe scenarii (tabelul ??) şi dialoguri libere. În primul caz, fiecare utilizator a interpretat 2-3 scenarii. În dialogurile libere, utilizatorii au putut formula orice fel de interogări, fiind informaţi doar asupra domeniului sistemului. De menţionat că dacă în cazul scenariilor utilizatorii au fost constrânşi să rezolve doar problema dată, în dialogurile libere aceştia au putut formula un număr mai mare de interogări diverse în cadrul aceluiaşi dialog.

Scenariul 1	Încercaţi să aflaţi **când** aveţi un anumit curs/seminar/proiect/laborator
Scenariul 2	Încercaţi să aflaţi **unde** aveţi un anumit curs/proiect/seminar/laborator
Scenariul 3	Încercaţi să aflaţi **cu cine** aveţi un anumit curs/seminar/proiect/laborator
Scenariul 4	Încercaţi să aflaţi **ce ore** aveţi într-o anumită **zi**
Scenariul 5	Încercaţi să aflaţi **ce ore** aveţi într-o anumită **perioadă** a unei zile
Scenariul 6	Încercaţi să aflaţi **unde** are ore o anumită **grupă** dintr-un an
Scenariul 7	Încercaţi să aflaţi **ce ore** are o anumită **grupă** dintr-un an într-un anumit **interval**
Scenariul 8	Încercaţi să aflaţi **ce ore** se ţin într-o anumită **sală** într-un anumit **interval**

Tabelul 4.1: Scenarii utilizate pentru colectarea datelor

Au fost astfel înregistrate dialoguri purtate de 42 de subiecţi, majoritatea studenţi şi cadre didactice de ambele sexe din cadrul Facultăţii de Automatică şi Calculatoare. Vârsta acestora a avut o medie de 23 ani, iar majoritatea lor nu utilizaseră anterior un sistem de dialog pentru obţinerea de informaţii, putând fi deci consideraţi un grup potrivit pentru obţinerea unor date pe baza cărora să se poată realiza dezvoltarea şi evaluarea analizorului.

Dialogurile înregistrate au fost analizate folosind diverse criterii. Pe de o parte, dialogurile au fost grupate în dialoguri finalizate şi întrerupte (prin aceasta înţelegând terminarea prematură a dialogului, cu diverse cauze: renunţarea utilizatorului, instabilitatea MDWOZ etc.) Pe de altă parte, dialogurile au fost apreciate ca reuşite sau eşuate – un dialog eşuat fiind unul în care sistemul nu a reuşit să furnizeze răspunsurile corecte, deşi întrebările utilizatorului respectau domeniul şi capabilităţile sistemului, iar în baza de date existau informaţiile cerute. Rezultatele acestor clasificări şi alte statistici referitoare la dialogurile din care au fost extrase datele folosite pentru dezvoltarea analizorului sunt cuprinse în tabelul **??**.

Pentru ca dezvoltarea analizorului semantic (inclusiv antrenarea modelului Markov) să nu fie afectată de erorile care pot apare într-un sistem de dialog la ieşirea modulului de recunoaştere automată a vorbirii, mediul MDWOZ permite înregistrarea formulărilor utilizatorilor sub forma unor fişiere audio care pot fi ulterior transcrise manual, evitându-se astfel erorile de recunoaştere menţionate.

Următorul pas a constat deci în transcrierea manuală a formulărilor utilizatorilor înregistrate în fişiere audio într-un format (SRO – prescurtare de la Speech Recognizer Output) care ar fi putut fi generat de un modul ideal de recunoaştere automată a vorbirii. Au rezultat astfel 3 fişiere corpus de tip text (mai multe detalii despre structura lor sunt date în anexa **??**).

Corpus	**woz-i**	**woz-ii**	**woz-iii**	**TOTAL**
Dimensiuni corpus				
Număr de dialoguri	59	62	61	182
Număr de formulări utilizator	371	377	340	1088
Distribuţia dialogurilor după sexul utilizatorilor				
Bărbaţi	47	52	55	154 (85%)
Femei	12	10	6	28 (15%)
Distribuţia dialogurilor după ocupaţia utilizatorilor				
Studenţi	40	54	52	146 (80%)
Profesori	16	8	5	29 (16%)
Alţii	3	0	4	7 (4%)
Distribuţia dialogurilor pe tipuri (scenarii/libere)				
Scenariul 1	8	6	2	16
Scenariul 2	3	8	5	16
Scenariul 3	5	13	0	18
Scenariul 4	6	3	7	16
Scenariul 5	11	1	4	16
Scenariul 6	6	2	8	16
Scenariul 7	2	8	6	16
Scenariul 8	0	7	9	16
Total scenarii	41	48	41	130 (71%)
Dialoguri libere	18	14	20	52 (29%)
Dialoguri finalizate/întrerupte				
Finalizate	56	60	58	174 (96%)
Întrerupte	3	2	3	8 (4%)
Dialoguri reuşite/eşuate				
Reuşite	44	46	55	145 (80%)
Eşuate	15	16	6	37 (20%)
Distribuţia dialogurilor reuşite/eşuate după tip				
Scenarii reuşite	32	34	36	102 (78%)
Scenarii eşuate	9	14	5	28 (22%)
Libere reuşite	12	12	19	43 (83%)
Libere eşuate	6	2	1	9 (17%)

Tabelul 4.2: Statistici referitoare la datele de antrenament

4.3 Dezvoltarea analizorului semantic

Dezvoltarea analizorului semantic pentru un sistem de dialog având un anumit domeniu de aplicaţie urmăreşte, pe de o parte, construirea fişierelor de configurare care controlează preprocesarea şi construcţia cadrelor (secţiunile **??** şi **??** şi anexa **??**), pe de alta, definirea şi antrenarea modelului Markov (secţiunea **??**).

Pentru a putea însa antrena modelul pe baza unui corpus, este necesar ca acesta să fie deja etichetat semantic (secţiunea **??**), ceea ce impune realizarea manuală cel puţin a decodării semantice (presupunând că preprocesarea poate fi făcută automat pe baza unor fişiere de configurare generate manual). În continuare vom prezenta metoda utilizată pentru a trata această problemă.

4.3.1 Metoda incrementală (bootstrap)

Etichetarea semantică manuală a celor 1088 formulări utilizator din corpusul de antrenament ar fi fost o activitate de lungă durată, monotonă şi predispusă la erori (şi costisitoare, dacă ar fi fost plătită), astfel că pentru a atenua aceste probleme am folosit o abordare *incrementală* (aşa-numita metodă *bootstrap*).

Ideea principală a metodei este de a construi versiuni succesive ale analizorului (fişiere de configurare şi model Markov) pe baza unor porţiuni din ce în ce mai mari din corpusul de antrenament, şi de a le utiliza apoi pentru a eticheta automat noi porţiuni din acesta. Astfel, în etapa de *iniţializare*, se generează prima versiune a fişierelor de configurare şi se etichetează semantic manual prima porţiune din corpusul de antrenament, pe baza căreia se crează un prim model. Folosind fişierele de configurare şi modelul astfel obţinute, se analizează semantic o nouă porţiune din corpus (de această dată automat), după care se realizează corecţia manuală a etichetării respectivei porţiuni, precum şi eventuale actualizări ale fişierelor de configurare. Cu porţiunea deja etichetată din corpus (prima parte manual, a doua automat şi corectată manual), se construieşte o nouă versiune a modelului, utilizată în continuare la etichetarea automată a unei noi porţiuni din corpusul de antrenament şamd. Procesul continuă astfel în *iteraţii* până la etichetarea întregului corpus, pe baza căruia se construieşte versiunea finală a analizorului.

Avantajele metodei incrementale sunt multiple: în primul rând, ea elimină în mare măsură munca de etichetare manuală a unui număr foarte mare de formulări şi reduce problema la etichetarea manuală a unei prime porţiuni din corpus şi doar corectarea manuală a etichetărilor restului formulărilor, ceea ce reduce timpul necesar dezvoltării modelului Markov. În plus, această metodă permite evaluări intermediare ale performanţelor modelului şi ale întregului analizor, prin care se pot urmări îmbunătăţirile obţinute pe baza corpusului de antrenament. Metoda permite deci şi detectarea momentului în care performanţele se stabilizează, moment din care este puţin probabil ca utilizarea unor noi formulări să le îmbunătăţească spectaculos.

4.3.2 Erori şi evaluarea performanţelor

Înainte de a trece la prezentarea în detaliu a etapelor dezvoltării analizorului, sunt necesare câteva precizări cu privire la tipurile de erori de care acesta poate fi afectat şi evaluarea performanţelor sale.

În primul rând, trebuie menţionat că în corpusurile de antrenament pot exista şi formulări invalide, în sensul că acestea fie sunt incorect formulate semantic, fie depăşesc domeniul prestabilit al sistemului de dialog propus (pentru câteva exemple, vezi anexa **??**). Acest gen de formulări nu sunt utilizate nici în fazele de antrenare (ar influenţa negativ modelul creat) şi nici în realizarea evaluărilor.

Lăsând la o parte aceste formulări invalide, în cadrul analizei semantice pot să apară erori la nivelul fiecăruia dintre modulele componente (preprocesor, decodor, constructor de cadre). De cele mai multe ori, aceste erori se vor propaga prin lanţul de analiză (e.g. o eroare de preprocesare va genera în continuare o eroare de decodare şi apoi de construcţie a cadrelor), însă pot exista şi situaţii în care o eroare apărută la nivelul unui modul este "corectată" de unul din modulele următoare. Cel mai des acest comportament apare la nivelul constructorului de cadre: întrucât corespondenţa dintre succesiunile de etichete semantice şi cadrele corespunzătoare nu este biunivocă, aceeaşi reprezentare poate fi obţinută pornind de la succesiuni de etichete semantice diferite, de unde posibilitatea "corecţiei" unor erori de etichetare semantică.

În evaluările efectuate în continuare vor fi prezentate atât erorile care apar la nivelul fiecărui modul (evaluare glass-box), ele fiind utile pentru a aprecia robusteţea şi performanţa fiecărei componente, cât şi cele globale (evaluare black-box), ele determinând în final performanţele analizorului dezvoltat.

4.3.3 Etape şi evaluări intermediare

Dezvoltarea analizorului s-a realizat în trei etape succesive, utilizând corpusurile woz-i, woz-ii şi woz-iii transcrise în format SRO. Rezultatul fiecărei etape este o versiune a analizorului semantic definită prin fişierele de configurare (vezi anexa **??**) şi modelul Markov corespunzătoare.

Cele trei etape au fost:

- **Iniţializarea**
  - Crearea versiunii I pe baza corpusului woz-i
  - Evaluarea pe corpusul de antrenament (woz-i)
  - Evaluarea pe un nou corpus (woz-ii)
- **Iteraţia 1**
  - Crearea versiunii II pe baza corpusurilor woz-i şi woz-ii
  - Evaluarea pe corpusul de antrenament (woz-i şi woz-ii)
  - Evaluarea pe un nou corpus (woz-iii)

- **Iteraţia 2**
  - Crearea versiunii III (finală) pe baza woz-i, woz-ii şi woz-iii
  - Evaluarea pe corpusul de antrenament (woz-i, woz-ii şi woz-iii)

În continuare vom detalia fiecare din aceste etape şi vom prezenta rezultatele evaluărilor intermediare.

Iniţializarea

- **Crearea versiunii I pe baza corpusului woz-i**
  - S-a realizat preprocesarea manuală a corpusului woz-i. Simultan, ţinând cont şi de analiza preliminară a domeniului, au fost elaborate fişierele de control al preprocesării – fişierele pentru normalizarea numerelor, reducerea flexiunilor, unificarea expresiilor, înlocuirea aliasurilor şi unificarea categoriilor (vezi anexa **??**). În ceea ce priveşte unificarea categoriilor, în această fază au fost identificate 8 categorii, din care 3 generice (ZI-RELATIV, PERIOADA-ZI, NR) şi 5 corespunzătoare câmpurilor din baza de date (MATERIA, ZI-SAPT, GRUPA, PROFESOR, SPECIALITATEA). Pe baza formei preprocesate a corpusului woz-i, s-a construit şi dicţionarul formei normalizate (dicţionarul de simboluri observate ale modelului Markov).
  - S-a realizat etichetarea semantică manuală a corpusului woz-i. Pe parcursul acestei etichetări, au fost marcate formulările fără concept (NC). Deasemeni, au fost identificate 19 (5.1%) formulări invalide (NEG), care au fost eliminate din corpusul de antrenament, dar au fost totuşi păstrate separat pentru o eventuală analiză a extinderii domeniului şi capabilităţilor sistemului (vezi anexa **??**). Tot în baza etichetării semantice manuale a fost construită mulţimea de etichete semantice utilizate.
  - Pe baza algoritmului de estimare de probabilitate maximă, s-a antrenat un prim model Markov (h-i) folosind corpusul etichetat semantic manual woz-i (352 formulări). Modelul rezultat are 40 de stări şi 46 de simboluri observate (dimensiuni dictate practic de numărul de etichete semantice şi dimensiunile dicţionarului normalizat).
  - Pe baza analizei preliminare a domeniului şi a formulărilor apărute în primul corpus de antrenament, a fost conceput un sistem de cadre pentru reprezentarea semantică a formulărilor utilizator, format în primă instanţă din 13 cadre: <da>, <nu>, <identificare>, <specif-timp>, <specif-materie>, <ce-materie>, <ce-curs>, <ce-laborator>, <ce-seminar>, <ce-proiect>, <cand>, <unde> şi <cu-cine> (pentru forma finală a sistemului de cadre utilizat, vezi anexa **??**).

La sfârşitul acestei etape de iniţializare s-a obţinut o primă versiune (I) de analizor semantic, definită de fişierele de control al preprocesării, modelul Markov h-i, şi fişierul de descriere a sistemului de cadre.

- **Evaluarea versiunii I pe corpusul de antrenament (woz-i)**

  În continuare, a fost realizată o evaluare a performanţelor analizorului pe setul de formulări de antrenament (woz-i). În mod evident, acest gen de evaluare nu va identifica nici erori de preprocesare şi nici de construcţie a cadrelor, deoarece fişierele de control respective au fost elaborate pornind chiar de la acest corpus. Pe de altă parte, datorită naturii stocastice a modelului Markov, pot apărea erori de decodare semantică, deci putem realiza o primă evaluare a modelului Markov şi pe baza setului de antrenament.

  Calitatea preprocesării poate fi deasemeni evaluată prin măsura în care se realizează o stabilizare a dicţionarului normalizat. Aşa cum am precizat în secţiunea **??**, unul din rolurile preprocesorului este de a realiza o reducere a dicţionarului domeniului, fără a pierde informaţii importante. În ultima etapă a preprocesării toate cuvintele care nu fac parte din dicţionarul normalizat sunt eliminate din formulare. În concluzie, dacă această etapă nu este atent controlată prin fişierul de configurare aferent, cuvinte semnificative pot fi eliminate din formulare în mod eronat. Printr-un studiu al evoluţiei dicţionarului normalizat pe măsura parcurgerii corpusului de antrenament, se poate estima calitatea preprocesării efectuate.

  În consecinţă:

  - S-a realizat o statistică a reducerii dicţionarului domeniului prin cele 7 etape succesive de preprocesare, ilustrată prin graficul din figura **??**. După cum se observă pe acest grafic, cele mai pronunţate reduceri ale dicţionarului sunt realizate în ultimele două etape, de unificare a categoriilor şi de eliminare a cuvintelor necuprinse în dicţionar. Deşi se observă o tendinţă de stabilizare a dimensiunilor dicţionarului normalizat (curba cea mai de jos), nu putem vorbi încă despre o stabilizare completă a acestuia.
  - S-a realizat etichetarea semantică automată a corpusului woz-i folosind versiunea I a analizorului. Verificarea etichetării semantice a evidenţiat 14 erori de decodare, dar după construcţia cadrelor s-a constatat o reducere a numărului acestora la numai 6 (vezi tabelul **??**).

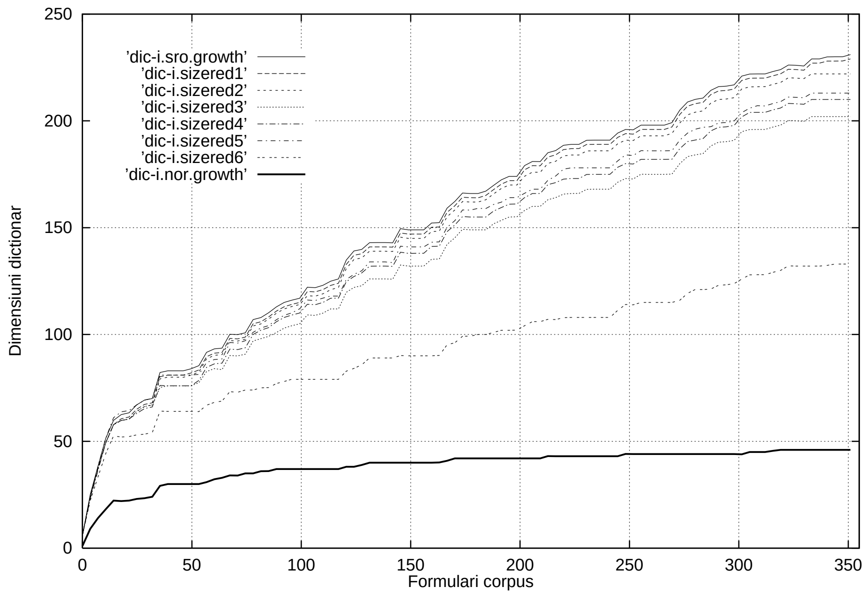


Figura 4.1: Reducerea dicţionarului prin preprocesare – versiunea I

EVALUARE PE CORPUS DE ANTRENAMENT (woz-i)	
Număr formulări de test	**352**
Erori de decodare	14 4.0 %
Erori globale (black-box)	6 1.7 %
EVALUARE PE CORPUS NOU (woz-ii)	
Număr total de formulări	377
Formulări invalide (NEG)	8 2.1 %
Număr formulări de test	**369**
Erori de preprocesare	35 9.5 %
Erori de decodare	44 11.9 %
Erori de construcţie a cadrelor	1 0.3 %
Erori globale (black-box)	80 **21.7 %**

Tabelul 4.3: Evaluări performanţe – versiunea I

- **Evaluarea versiunii I pe un corpus nou (woz-ii)**
  - S-a realizat analiza semantică (preprocesarea, decodarea şi construcţia cadrelor) a corpusului woz-ii folosind versiunea I a analizorului.
  - Rezultatele au fost corectate şi totodată s-a realizat evaluarea performanţelor modelului. În acest scop, prin inspecţia manuală a rezultatelor analizei semantice au fost identificate şi marcate corespunzător erorile de preprocesare, erorile de decodare, erorile de construcţie a cadrelor şi formulările invalide (NEG). Evident, în majoritatea cazurilor are loc o propagare a erorilor, adică erorile de preprocesare vor genera în continuare erori de decodare şi de construcţie a cadrelor. Per total, rata erorii obţinută în această evaluare a fost de 21.7 %, statisticile concludente în acest sens fiind cuprinse în partea a II-a a tabelului **??**.

Iteraţia 1

- **Crearea versiunii II pe baza corpusurilor woz-i şi woz-ii**
  - S-a realizat analiza erorilor de preprocesare apărute la analiza semantică a corpusului woz-ii (vezi subsecţiunea anterioară). Pe baza acestei analize au fost îmbunătăţite fişierele de control al preprocesării pentru a elimina respectivele erori. În esenţă au fost adăugate noi expresii, aliasuri, şi chiar o nouă categorie – SALA – corespunzătoare câmpului respectiv din baza de date. Întrucât s-a observat că numeroase erori au apărut datorită incompletitudinii fişierelor de configurare, a fost realizat un studiu detaliat al acestora, şi au fost aduse şi alte modificări pentru completitudine şi omogenitate. În continuare, la dicţionarul normalizat au fost adăugate câteva noi cuvinte apărute pe parcursul preprocesării celui de-al doilea corpus.
  - S-a realizat etichetarea semantică a variantei deja preprocesate a corpusului woz-ii utilizând versiunea I a analizorului. Erorile de decodare semantică au fost corectate şi, pe baza celor 721 formulări etichetate semantic din primele două corpusuri (woz-i şi woz-ii) a fost antrenată prin algoritmul de estimare de probabilitate maximă o a doua variantă a modelului Markov – h-ii (46 stări şi 56 simboluri observate).
  - Eroarea de construcţie a cadrelor apărută la evaluarea versiunii I a condus la o reevaluare a sistemului de cadre utilizat, modificat pentru a trata şi întrebări despre existenţa unor anumite cursuri/laboratoare etc. (e.g.: E laborator de SDA astăzi?).

Noile variante ale fişierelor de configurare a preprocesării, modelului Markov şi descrierii sistemului de cadre utilizat definesc o nouă versiune de analizor semantic – versiunea II, care a fost evaluată în continuare atât pe corpusul său de antrenament, cât şi pe un nou corpus de test.

- **Evaluarea versiunii II pe corpusul de antrenament (woz-i şi woz-ii)**

  – Calitatea preprocesării a fost din nou analizată prin curba de evoluţie a dimensiunilor dicţionarului normalizat, ilustrată în figura **??**. În urma modificărilor aduse fişierelor de control al preprocesării şi a analizei corpusului woz-ii, dimensiunea vocabularului normalizat a crescut de la 46 la 56 de cuvinte, deci încă nu putem vorbi despre o stabilizare a acestuia.

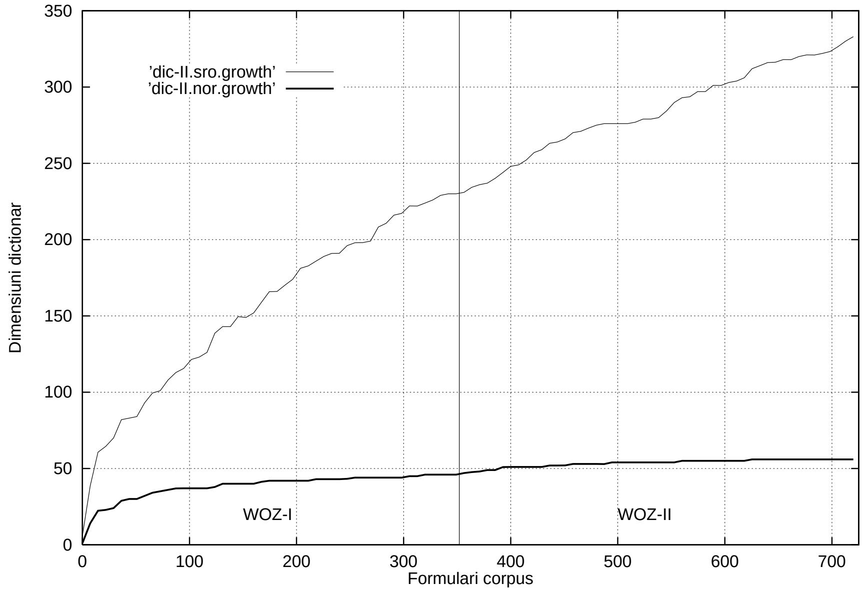


Figura 4.2: Reducerea dicţionarului prin preprocesare – versiunea II

  – S-a realizat analiza semantică automată a corpusului woz-ii folosind versiunea II a analizorului. Verificarea manuală a acesteia a pus în evidenţă 22 de erori de etichetare semantică, rezultând în 14 erori de reprezentare pe bază de cadre. O statistică mai detaliată este prezentată în prima parte a tabelului **??**.

- **Evaluarea versiunii II pe un corpus nou (woz-iii)**

  – S-a realizat analiza semantică (preprocesarea, decodarea şi construcţia cadrelor) a corpusului woz-iii folosind versiunea II a analizorului.

  – După corectarea rezultatelor s-a realizat evaluarea performanţelor. Este de remarcat că de această dată nu au mai apărut erori de preprocesare

EVALUARE PE CORPUS DE ANTRENAMENT (woz-i+ii)	
Număr formulări de test	**721**
Erori de decodare	22 3.1 %
Erori globale (black-box)	14 1.9 %
EVALUARE PE CORPUS NOU (woz-iii)	
Număr total de formulări	340
Formulări invalide (NEG)	10 2.9 %
Număr formulări de test	**330**
Erori de preprocesare	0 0 %
Erori de decodare	46 13.9 %
Erori de construcţie a cadrelor	0 0 %
Erori globale (black-box)	45 **13.6 %**

Tabelul 4.4: Evaluări performanţe – versiunea II

şi nici de construcţie a cadrelor, lucru care constituie un prim indiciu în direcţia stabilizării fişierelor de control şi a sistemului de cadre utilizate. Au fost identificate 46 erori de decodare, din care 45 s-au propagat şi după construcţia cadrelor. Per total, rata erorii versiunii II a fost de 13.6%, semnificativ mai redusă decât a versiunii I (rezultatele sunt prezentate în detaliu în partea a doua a tabelului **??**).

Iteraţia 2

- **Crearea versiunii III pe baza corpusurilor woz-i, woz-ii şi woz-iii**
  - S-a preprocesat corpusul woz-iii folosind versiunea II a analizorului. Întrucât nu au apărut erori (vezi subsecţiunea anterioară), nu a fost necesară intervenţia manuală şi nici efectuarea de modificări în fişierele de control respective.
  - S-a realizat etichetarea semantică a variantei preprocesate a corpusului woz-iii utilizând versiunea II. Erorile de decodare semantică au fost corectate şi, pe baza celor 1051 formulări etichetate semantic din primele trei corpusuri (woz-i, woz-ii şi woz-iii), au fost antrenate două variante (finale) de model Markov. Prima (h-iii.mle) a fost antrenată utilizând ca şi până acum estimarea de probabilitate maximă. La a doua variantă

(h-iii.katz) a fost aplicată reestimarea Katz atât pentru probabilitaţile tranziţiilor între stări ($k = 4$ – au fost discreditate bigramele care au apărut de maximum 4 ori), cât şi pentru probabilitaţile stărilor iniţiale ($k = 2$).

– Întrucât nu au existat erori de construcţie a cadrelor la analiza semantică a corpusului woz-iii (vezi secţiunea anterioară), fişierul de descriere al sistemului de cadre a rămas nemodificat.

În urma acestei etape au fost construite versiunile finale ale analizorului semantic stocastic proiectat: versiunea III-MLE (cu modelul Markov construit prin estimare de probabilitate maximă) şi versiunea III-Katz (cu modelul Markov construit aplicând şi reestimarea Katz).

- **Evaluarea versiunii III pe corpusul de antrenament (woz-i+ii+iii)**

– Calitatea preprocesării a fost din nou analizată prin intermediul curbei de evoluţie a dimensiunilor dicţionarului normalizat, ilustrată în figura ??. De această dată, la parcurgerea ultimului corpus de antrenament (woz-iii) nu au apărut erori de preprocesare şi nici cuvinte noi în dicţionarul normalizat, acesta rămânând deci la dimensiunea de 56 cuvinte. În concluzie, putem spera că din acest moment avem o stabilizare a dicţionarului simbolurilor observate.

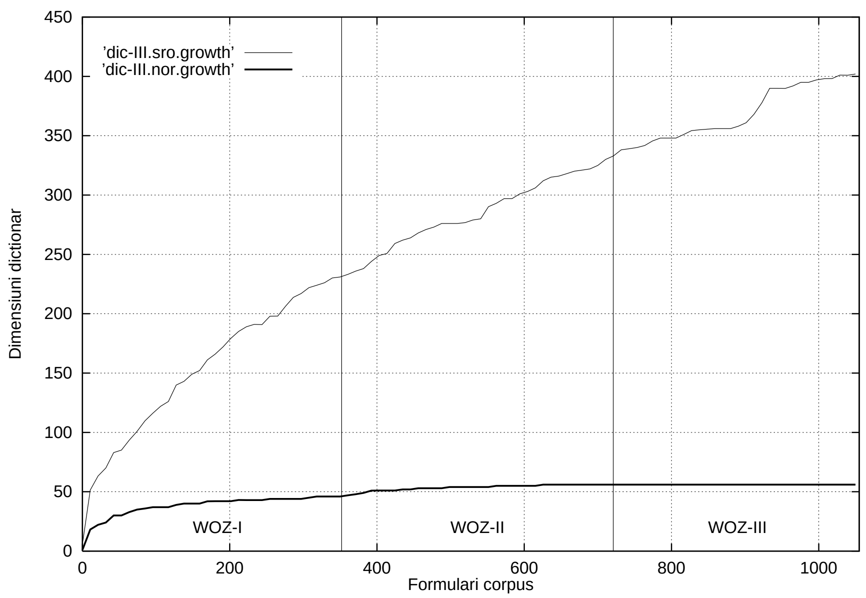


Figura 4.3: Reducerea dicţionarului prin preprocesare – versiunea III

– S-a realizat analiza semantică automată a primelor 3 corpusuri folosind versiunile III-MLE şi III-Katz ale analizorului. Din statistica din tabelul

EVALUARE PE CORPUS DE ANTRENAMENT (woz-i+ii+iii)		
Număr formulări de test		**1051**
MLE	Erori de decodare	31 2.9 %
	Erori globale (black-box)	20 1.9 %
Katz	Erori de decodare	38 3.6 %
	Erori globale (black-box)	26 2.5 %

Tabelul 4.5: Performanţele versiunilor finale (III) pe corpusul de antrenament

??, rezultă că modelul reestimat prin metoda Katz a produs sensibil mai multe erori de etichetare semantică decât modelul MLE. Acest lucru se explică prin faptul că testul a fost făcut pe corpusul de antrenament. Întrucât scopul reestimării Katz este de a modela mai bine evenimentele care nu apar în corpusul de antrenament, este clar că reestimarea nu va aduce îmbunătăţiri ale performanţei în cazul unui astfel de test. Mai mult, datorită faptului că metoda Katz discreditează anumite evenimente ce au apărut în corpusul de antrenament, performanţa modelului reestimat poate fi chiar mai slabă decât a celui iniţial la o testare pe formulările din corpusul de antrenament.

Această etapă încheie dezvoltarea incrementală a analizorului semantic, urmând ca în continuare să prezentăm evaluările performanţelor versiunilor finale.

4.4 Evaluarea finală a analizorului

În scopul evaluării comparative a performanţelor celor două versiuni de analizor dezvoltate (III-MLE şi III-Katz), a fost utilizat cel de-al patrulea corpus de formulări utilizator (woz-test). Câteva statistici privind acest corpus de test sunt prezentate în tabelul **??**. El totalizează 287 de formulări, însă înainte de a trece la realizarea analizei lui semantice au fost identificate şi eliminate 4 formulări invalide (NEG), testarea urmând a se efectua pe 283 de formulări.

După efectuarea analizei semantice a corpusului woz-test folosind cele două versiuni de analizor, a fost realizată inspecţia manuală a rezultatelor şi marcarea diverselor tipuri de erori apărute (o statistică a lor este prezentată în tabelul **??**).

În testele efectuate nu a mai apărut nici o eroare la nivelul modulelor de preprocesare şi de construcţie a cadrelor. La nivelul decodorului semantic, modelul Markov obţinut după aplicarea reestimării Katz s-a comportat semnificativ mai bine decât cel antrenat prin estimare de probabilitate maximă, generând numai 19 erori, faţă de cele 26 generate de al doilea. Construcţia cadrelor a eliminat însă

Corpus	**woz-test**	
Dialoguri	45	
Dialoguri după scenarii	23	(51%)
Dialoguri libere	22	(49%)
Număr total de formulări	287	
Formulări invalide (NEG)	4	(1.4%)
Număr formulări de test	**283**	

Tabelul 4.6: Statistici referitoare la datele de test

Versiune	**III-MLE**	**III-Katz**
Număr formulări de test	**283**	**283**
Erori de preprocesare	0	0
Erori de decodare (procentaj)	26 9.18%	19 6.71%
Erori de decodare corectate la construcţia cadrelor	3	1
Erori de construcţie a cadrelor	0	0
Total erori (black-box) **(procentaj)**	**23** **8.12%**	**18** **6.36%**

Tabelul 4.7: Rezultatele evaluării finale a performanţelor

câteva erori de decodare, astfel încât per total rata erorii versiunii III-Katz a fost de 6.36%, faţă de 8.12% pentru versiunea III-MLE.

În concluzie, aplicând reestimarea Katz, numărul de erori de decodare a fost redus de la 26 la 19, ceea ce reprezintă o îmbunătăţire relativă cu cca. 27% a ratei erorilor de decodare.

Rata globală a erorilor de 6.36% clasifică analizorul semantic dezvoltat pe domeniul propus ca unul de performanţă ridicată (reamintim că performanţele cele mai bune menţionate în literatură sunt de 3.8%) şi confirmă oportunitatea abordării stocastice în dezvoltarea lui.

Observaţie: Întrucât nu s-a observat încă o stabilizare a performanţelor (rata erorilor a scăzut de la 21.7% la 13.6%, 8.12%, şi în cele din urmă la 6.36%), se poate încerca o îmbunătăţire în continuare a acestora prin utilizarea unor noi seturi de formulări. În acest sens se poate apela într-o primă instanţă chiar la ultimul corpus utilizat în evaluarea analizorului (`woz-test`).

Capitolul 5

Concluzii şi continuări

În capitolele anterioare au fost descrise în detaliu proiectarea, implementarea software, dezvoltarea şi evaluarea performanţelor unui analizor semantic stocastic pentru sisteme automate de dialog vocal om-calculator.

Acest ultim capitol este dedicat prezentării concluziilor desprinse din acestă experienţă şi identificării câtorva posibile direcţii de continuare a cercetărilor.

5.1 Concluzii

Acestă lucrare a prezentat dezvoltarea unui modul de analiză semantică pentru sisteme de dialog. Etapele parcurse în acest sens au fost:

- realizarea unui studiu al diverselor alternative de proiectare, pe baza căruia a fost luată decizia utilizării formalismului case-grammar pentru reprezentarea cunoştinţelor şi a unei abordări stocastice bazate pe modele Markov pentru realizarea parsingului;
- proiectarea arhitecturii analizorului semantic şi a funcţionalităţii fiecăruia dintre modulele componente;
- implementarea software efectivă a analizorului proiectat;
- dezvoltarea resurselor (fişiere de configurare şi modele Markov) necesare pentru integrarea analizorului într-un sistem de dialog pentru furnizarea de informaţii despre orarul Departamentului de Calculatoare;
- evaluarea performanţelor analizorului semantic construit.

Pe baza etapelor sintetizate mai sus se pot desprinde următoarele concluzii:

- Deşi nu se pot face comparaţii directe (datorită diferenţelor existente între domeniile de dialog şi datele utilizate), rata erorilor de 6.36%, obţinută la evaluarea finală a analizorului, este totuşi apropiată de cele mai bune (3.8%) prezentate anterior în literatură.

- Abordarea stocastică aleasă a permis dezvoltarea rapidă şi eficientă a unui analizor semantic pentru sisteme automate de dialog vocal om-calculator.

- Utilizarea unor algoritmi specifici de îmbunătăţire a modelului stocastic a permis construirea unui analizor cu o robusteţe sporită în faţa diverselor fenomene specifice vorbirii spontane, concretizată într-o reducere relativă a ratei erorilor de circa 27%.

- Analizorul semantic proiectat în acestă lucrare este unul general, independent atât de domeniul cât şi de limba sistemului de dialog în care urmează a fi integrat. Adaptarea analizorului la un anumit sistem de dialog (ilustrată în capitolul 4) presupune un efort mult mai redus decât elaborarea unui eventual set de reguli pentru modelarea analizei semantice. La schimbarea sau extinderea domeniului de dialog trebuie doar înregistrat un nou corpus de antrenament, caracteristic noului sistem, şi repetate etapele de dezvoltare prin care se construiesc modelul Markov şi setul de fişiere de configurare a preprocesării şi construcţiei cadrelor. Analizorul proiectat este deci caracterizat de un grad ridicat de portabilitate, oferind o soluţie generală pentru problema analizei semantice în sistemele de dialog.

Toate acestea arată că abordarea stocastică, deja dominantă în recunoaşterea automată a vorbirii, este pe deplin justificată şi în analiza semantică a limbajului natural pentru sisteme automate de dialog vocal om-calculator, pentru care oferă o soluţie performantă, robustă, şi cu un grad ridicat de portabilitate.

5.2 Direcţii de continuare

Continuările cercetărilor prezentate în această lucrare se pot canaliza pe două direcţii principale: pe de o parte, îmbunătăţiri ale implementării şi performanţelor analizorului, pe de alta, integrarea acestuia într-un sistem de dialog complet.

În privinţa îmbunătăţirii implementării şi performanţelor:

- La nivelul implementării pot fi îmbunătăţite unele dintre soluţiile alese, în special în ceea ce priveşte modulul de preprocesare. Este vorba în primul rând despre normalizarea numerelor, care în faza curentă se realizează pe baza unor simple înlocuiri de şiruri de caractere, soluţie acceptabilă în sistemul de dialog proiectat, dar insuficientă într-un caz general. Aceasta presupune scrierea unei gramatici a numerelor şi realizarea normalizării lor pe baza ei. Etapa de reducere a flexiunilor poate fi deasemeni realizată într-un cadru general pe baza unui dicţionar morfo-sintactic conţinând atât formele flexionare ale cuvintelor, cât şi lemele respectivelor forme [**?**]. Deasemeni, se pot realiza optimizări ale structurilor de date şi algoritmilor din biblioteca de clase (e.g. implementare cu hash-table a dicţionarelor etc.)

- O altă extensie utilă ar fi dezvoltarea unui mecanism pentru detectarea şi semnalarea erorilor apărute pe parcursul analizei semantice. În momentul de faţă analizorul nu dispune de un astfel de mecanism, cadrele construite fiind transmise către modulul de control al dialogului chiar dacă reprezentarea obţinută nu este cea corectă (din punct de vedere al semanticii limbajului natural). Realizarea unui astfel de mecanism este însă problematică şi presupune o analiză în contextul modulelor adiacente de recunoaştere a vorbirii şi control al dialogului, constituind în sine un domeniu interesant de studiu pe viitor.

- În cadrul domeniului de dialog propus, se poate încerca îmbunătăţirea performanţei analizorului prin continuarea antrenării modelului Markov pe baza unor noi seturi de formulări. Întrucât pe parcursul etapelor prezentate nu s-a observat încă o stabilizare a performanţei, este posibil să obţinem astfel o nouă creştere a ei.

O a doua direcţie în care se pot focaliza eforturile de cercetare viitoare o constituie integrarea analizorului semantic dezvoltat într-un sistem de dialog complet pentru furnizarea de informaţii despre orar. În acest sens, menţionăm ca lucrarea de faţă este parte integrantă a unui efort pe scară mai largă de dezvoltare a unor sisteme automate de dialog vocal om-calculator [**?**], [**?**], [**?**].

Bibliografie

[1] J. Allen. *Natural Language Understanding.* Benjamin/Cummings Publishing Company, 1995.

[2] H. Aust şi M. Oerder. The Philips automatic train timetable information system. *Speech Communication*, 17:249–62, 1995.

[3] S.K. Bennacef, H. Bonneau-Maynard, J.L. Gauvin, L.F. Lamel, şi W. Minker. A Spoken Language System for Information Retrieval. În *Proceedings ICSLP'94*, pag. 1271–74, 1994.

[4] D. Bohuş şi M. Boldea. A Web-based Text Corpora Development System. În *Proceedings Second International Conference on Language Resources and Evaluation*, Atena, Grecia, 2000.

[5] M. Boldea. SADVOC: Sisteme Automate de Dialog Vocal Om-Calculator. Propunere de proiect major de cercetare înaintată CNCSIS, noiembrie 1999.

[6] I. Coteanu, L. Seche, şi M. Seche. *Dicţionarul explicativ al limbii române.* Editura Academiei, 1975.

[7] J.L. Gauvain, S. Bennacef, L. Devillers, L.F. Lamel, şi S. Rosset. The Spoken Language Component of the Mask Kiosk. În K. Varghese şi S. Pfleger, *Human Comfort and Security of Information Systems*, pag. 93–103. Springer, 1997.

[8] S.M. Katz. Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 1987.

[9] KDevelop. www.kdevelop.org.

[10] W. Minker, A. Waibel, şi J. Mariani. *Stochastically-Based Semantic Analysis.* Kluwer Academic Publishers, 1999.

[11] R. De Mori, editor. *Spoken Dialogues with Computers.* Academic Press, 1998.

[12] C. Munteanu. Mediu pentru dezvoltarea sistemelor de dialog prin metoda Vrăjitorului din Oz. Lucrare de diplomă, Universitatea "Politehnica" din Timişoara, iunie 1999.

[13] C. Munteanu. Modelarea şi controlul dialogului om-calculator. Dizertaţie de studii aprofundate, Departamentul de Calculatoare, Universitatea "Politehnica" din Timişoara, iunie 2000.

[14] C. Munteanu şi M. Boldea. MDWOZ: A Wizard of Oz Environment for Dialog Systems Development. În *Proceedings Second International Conference on Language Resources and Evaluation*, Atena, Grecia, 2000.

[15] L.R. Rabiner şi B.H. Juang. An Introduction to Hidden Markov Models. *IEEE Acoustics, Speech and Signal Processing Magazine*, 1986.

[16] S. Russell şi P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[17] N.O. Bernsen şi H. Dybkjaer şi L. Dybkjaer. *Designing Interactive Speech Systems: From First Ideas to User Testing.* Springer, 1997.

[18] E. Tzoukermann, R. Pieraccini, şi Z. Gorelov. Natural Language Processing in the CHRONUS System. În *Proceedings ICSLP'92*, 1992.

[19] W. Ward. Understanding Spontaneous Speech: the PHOENIX System. În *Proceedings ICASSP'91*, pag. 365–68, 1991.

[20] M. Woszczyna et al. Recent Advances in JANUS: A Speech Translation System. În *Proceedings EUROSPEECH'93*, pag. 1295–98, Berlin, 1993.

Anexa A

Exemple de dialoguri

În continuare sunt prezentate câteva dialoguri considerate tipice pentru sistemul de dialog la care se referă experimentele din capitolul 4. Prefixul "S:" indică formulările sistemului, iar "U:" – formulările utilizatorilor.

Dialog 1 (scenariul 1)

```
S: Buna ziua, ati mai utilizat acest sistem ?
U: Nu
S: Acest sistem furnizeaza informatii despre orar. Va rog formulati
   intrebarile si raspunsurile dumneavoastra cat mai clar si nu
   interveniti in timp ce sistemul va raspunde.
   Daca sunteti student precizati anul, grupa, subgrupa. Daca sunteti
   cadru didactic precizati numele. In caz contrar precizati ca nu
   doriti sa va identificati.
U: Sunt student in anul doi grupa doi subgrupa doi unu.
S: Va rog formulati intrebarea dumneavoastra.
U: As vrea sa aflu cand am laboratorul de SDA.
S: La anul 2 grupa 2 subgrupa 1, laboratorul de ''Structuri de date si
   algoritmi'' se tine marti de la 16 la 18 in sala A305. Mai doriti
   alte informatii ?
U: Nu mersi ok, asta e tot.
S: Va multumesc ca ati folosit acest sistem. La revedere.
```

Dialog 2 (scenariul 5)

```
S: Buna ziua, ati mai utilizat acest sistem ?
U: Da
S: Daca sunteti student precizati anul, grupa, subgrupa. Daca sunteti
   cadru didactic precizati numele. In caz contrar precizati ca nu
```

```
   doriti sa va identificati.
U: Student anul intai grupa trei subgrupa unu, Cosmin.
S: Va rog formulati intrebarea dumneavoastra.
U: As dori sa aflu ce ore am intre ... vineri intre ora 8 si 12
S: La anul 1, cursul de ``Matematici speciale'' se tine vineri de la 8
   la 10 in sala CA2. Mai doriti alte informatii ?
U: Nu multumesc.
S: Va multumesc ca ati folosit acest sistem. La revedere.
```

Dialog 3 (liber)

```
S: Buna ziua, ati mai utilizat acest sistem ?
U: Da.
S: Daca sunteti student precizati anul, grupa, subgrupa. Daca sunteti
   cadru didactic precizati numele. In caz contrar precizati ca nu
   doriti sa va identificati.
U: Anul intai.
S: Va rog formulati intrebarea dumneavoastra.
U: Cand are grupa doi semigrupa unu seminar de Analiza Matematica 2 ?
S: La anul 1 grupa 2 subgrupa 1, seminarul de ``Analiza Matematica 2''
   se tine miercuri de la 16 la 18 in sala A314. Mai doriti alte
   informatii ?
U: Da.
S: Va rog formulati intrebarea dumneavoastra.
U: Cand am timp liber marti, dimineata de la opt pana seara la opt ?
S: Imi pare rau, nu am inteles. Va rog reformulati.
U: Ce ore am marti ?
S: Exista mai multe raspunsuri, sistemul le va preciza pe toate daca nu
   specificati criterii suplimentare pentru clarificarea intrebarii. Doriti
   sa specificati astfel de criterii ?
U: Da.
S: Precizati grupa si subgrupa.
U: Grupa 2 semigrupa 1.
S: De la 8 la 10 laboratorul de ``Analiza si sinteza disp. numerice''.
   De la 10 la 12 seminarul de ``Matematici speciale''.
   De la 14 la 16 seminarul de ``Sport''.
   Mai doriti alte informatii ?
U: Da.
S: Va rog formulati intrebarea dumneavoastra.
U: Cu cine si unde se tine laboratorul de la 8 la 10 ?
S: La anul 1 grupa 2 subgrupa 1, laboratorul de ``Analiza si sinteza disp.
   numerice'' se tine marti de la 8 la 10 in sala B624 cu sl. Mihaescu.
   Mai doriti alte informatii ?
U: Nu.
S: Va multumesc ca ati folosit acest sistem. La revedere.
```

Dialog 4 (liber)

```
S: Buna ziua, ati mai utilizat acest sistem ?
U: Da.
S: Daca sunteti student precizati anul, grupa, subgrupa. Daca sunteti
   cadru didactic precizati numele. In caz contrar precizati ca nu
   doriti sa va identificati.
U: Anul 4 grupa 4 subgrupa 1.
S: Va rog formulati intrebarea dumneavoastra.
U: Ce curs este de la ora 11 in sala A109 ?
S: La anul 4, cursul de ``Sisteme de intrare-iesire'' se tine luni de la
   11 la 14 in sala A109.
   La anul 4, cursul de ``Retele locale de calculatoare'' se tine joi
   de la 11 la 14 in sala A109.
   Mai doriti alte informatii ?
U: Da.
S: Va rog formulati intrebarea dumneavoastra.
U: Cine tine cursurile respective ?
S: La anul 4, cursul de ``Sisteme de intrare-iesire'' se tine cu prof.
   Strugaru.
   La anul 4, cursul de ``Retele locale de calculatoare'' se tine cu prof.
   Strugaru. Mai doriti alte informatii ?
U: Da.
S: Va rog formulati intrebarea dumneavoastra.
U: Ce laboratoare are in cursul zilei de marti grupa patru ?
S: Exista mai multe raspunsuri, sistemul le va preciza pe toate daca nu
   specificati criterii suplimentare pentru clarificarea intrebarii. Doriti
   sa specificati astfel de criterii ?
U: Da.
S: Precizati la ce subgrupa.
U: Patru unu.
S: La anul 4 grupa 4 subgrupa 1, laboratorul de ``Sisteme de intrare-
   iesire'' se tine marti de la 8 la 10 in sala B413.
   La anul 4 grupa 4 subgrupa 1, laboratorul de ``Ingineria sistemelor
   industriale'' se tine marti de la 12 la 14 in sala SPM.
   Mai doriti alte informatii ?
U: Da.
S: Va rog formulati intrebarea dumneavoastra.
U: As vrea sa stiu daca Radu Marinescu are laborator de IP marti.
S: Exista mai multe raspunsuri, sistemul le va preciza pe toate daca nu
   specificati criterii suplimentare pentru clarificarea intrebarii. Doriti
   sa specificati astfel de criterii ?
U: Da.
S: Precizati grupa.
U: Grupa trei.
S: La anul 4 grupa 3 subgrupa 2, laboratorul de ``Ingineria programarii''
   se tine marti de la 16 la 18 in sala B528 cu asistent Radu Marinescu.
   La anul 4 grupa 3 subgrupa 1, laboratorul de ``Ingineria programarii''
```

```
   se tine marti de la 18 la 20 in sala B528 cu asistent Radu Marinescu.
   Mai doriti alte informatii ?
U: As vrea sa stiu daca exista totusi laborator de IP de la 12 marti.
S: La anul 4 grupa 4 subgrupa 2, laboratorul de ``Ingineria programarii''
   se tine marti de la 12 la 14 in sala B528. Mai doriti alte informatii ?
U: Nu.
S: Va multumesc ca ati folosit acest sistem. La revedere.
```

Anexa B

Structura unui corpus

Dezvoltarea fişierelor de configurare şi a modelului Markov utilizate în analiza semantică se fac pe baza unui corpus de formulări utilizator tipice pentru sistemul de dialog proiectat. În continuare vom prezenta formatul în care formulările sunt stocate într-un fişier conţinând un asemenea corpus.

Fiecare formulare utilizator trece prin mai multe reprezentări de-a lungul lanţului de analiză semantică: text de la modulul de recunoaştere, text normalizat, succesiune de etichete semantice şi reprezentare pe bază de cadre. Evident, toate aceste forme trebuie reprezentate în corpusul de antrenament, drept pentru care a fost stabilită următoarea structură pentru o formulare utilizator:

```
%<clasă formulare>
SRO:<forma obţinută de la modulul de recunoaştere>
NOR:<forma normalizată, obţinută după preprocesare>
PRS:<succesiunea de etichete semantice, obţinută după decodare>
FRM:<reprezentare pe bază de cadre>
$
```

Clasa formulării (e.g. NEG – formulare invalidă, NC – formulare fără concept etc.) este folosită pentru operaţii selective asupra corpusului şi poate fi vidă. Formele SRO (Speech Recognizer Output), NOR (NORmalized), PRS (PaRSe) şi FRM (FRaMe) deasemeni pot să existe sau nu pentru o anumită formulare.

Pentru exemplificare, prezentăm în continuare o porţiune dintr-un corpus.

```
%
SRO:da am mai utilizat acest sistem
NOR:da
PRS:<da>
FRM:(<da>)
$
```

```
%
SRO:(aa) sunt student în anul doi grupa trei
NOR:sunt student an [NR:"2"] grupă [NR:"3"]
PRS:<identificare> (null) (m:anul) (v:anul) (m:grupa) (v:grupa)
FRM:(<identificare> (anul "2") (grupa "3"))
$

%
SRO:(aa) aş vrea să ştiu ce ore am marţi
NOR:ce-ore [ZI-SAPT:"marţi"]
PRS:<ce-materie> (v:zi-sapt)
FRM:(<ce-materie> (<specif-timp> (zi-sapt "marţi")))
$

%
SRO:în ce zile şi la ce ore îl găsesc pe domnul profesor Jian
NOR:ce-zi şi la ce-ore profesor [PROFESOR:"Ionel Jian"]
PRS:<când> (null) (null) (null) (m:profesor) (v:profesor)
FRM:(<când> (<identificare> (profesor "Ionel Jian")))
$

%NC
SRO:anul cinci grupa management
NOR:<identificare> an [NR:"5"] grupă management
PRS:<identificare> (m:anul) (v:anul) (m:grupa) (v:grupa)
FRM:(<identificare> (anul "5") (grupa "management"))
$

%
SRO:(aa) sunt în anul întâi grupa a doua semigrupa întâi şi aş dori să-mi
    spui când am laborator de electronică doi
NOR:sunt an [NR:"1"] grupă [NR:"2"] subgrupă [NR:"1"] şi când
    laborator [MATERIA:"Electrotehnică II"]
PRS:<identificare> (m:anul) (v:anul) (m:grupa) (v:grupa) (m:subgrupa)
    (v:subgrupa) (null) <cand> (m:laborator) (v:laborator)
FRM:(<identificare> (anul "1") (grupa "2") (subgrupa "1")) (<când>
    (<identificare> (anul "1") (grupa "2") (subgrupa "1"))
    (<specif-materie> (laborator "Electrotehnică II")))
$

%
SRO:în ce zile sunt laboratoare de circuite integrate la anul trei
NOR:ce-zi sunt laborator [MATERIA:"Circuite integrate"] la an [NR:"3"]
PRS:<când> (null) (m:laborator) (v:laborator) (null) (m:anul) (v:anul)
FRM:(<când> (<identificare> (anul "3")) (<specif-materie> (laborator
    "Circuite integrate")))
$

%
```

```
SRO:unde se ţine cursul de proiectarea translatoarelor
NOR:unde curs [MATERIA:"Proiectarea translatoarelor"]
PRS:<unde> (m:curs) (v:curs)
FRM:(<unde> (<specif-materie> (curs "Proiectarea translatoarelor")))
$

%
SRO:ce predă profesorul Popa în anul trei
NOR:ce profesor [PROFESOR:"Mircea Popa"] an [NR:"3"]
PRS:<ce-materie> (m:profesor) (v:profesor) (m:anul) (v:anul)
FRM:(<ce-materie> (<identificare> (anul "3") (profesor "Mircea Popa")))
$

%
SRO:în ce sală ţine grupa a treia din anul patru calculatoare laboratorul
    de inteligenţă artificială
NOR:ce-sală grupă [NR:"3"] an [NR:"4"] [SPECIALITATEA:"calculatoare"]
    laborator [MATERIA:"Inteligenţă artificială"]
PRS:<unde> (m:grupa) (v:grupa) (m:anul) (v:anul) (v:specialitatea)
    (m:laborator) (v:laborator)
FRM:(<unde> (<identificare> (anul "4") (specialitatea "calculatoare")
    (grupa "3")) (<specif-materie> (laborator "Inteligenţă artificială")))
$

%
SRO:am cursuri în ziua de marţi între ora opt şi zece
NOR:curs zi [ZI-SAPT:"mari"] între oră [NR:"8"] şi [NR:"10"]
PRS:<ce-curs> (m:zi-sapt) (v:zi-sapt) (m:ora-start) (null) (v:ora-start)
     (m:ora-stop) (v:ora-stop)
FRM:(<ce-curs> (<specif-timp> (zi-sapt "marţi") (ora-start "8")
    (ora-stop "10")))
$

%
SRO:aş vrea să ştiu cu cine se ţine laboratorul de programare în limbaj
    de asamblare
NOR:cu-cine laborator [MATERIA:"Programarea în limbaj de asamblare"]
PRS:<cu-cine> (m:laborator) (v:laborator)
FRM:(<cu-cine> (<specif-materie> (laborator "Programarea în limbaj de
    asamblare")))
$

%
SRO:ce ore am duminică noaptea
NOR:ce-ore [ZI-SAPT:"duminică"] [PERIOADA-ZI:"noapte"]
PRS:<ce-materie> (v:zi-sapt) (v:perioada-zi)
FRM:(<ce-materie> (<specif-timp> (zi-sapt "duminică") (perioada-zi
    "noapte")))
$
```

```
%
SRO:sunt student în anul şase (aa) la grupa de soft
NOR:sunt student an [NR:"6"] la grupă
PRS:<identificare> (null) (m:anul) (v:anul) (null) (m:grupa)
FRM:(<identificare> (anul "6"))
$

%
SRO:mă numesc Pop Traian
NOR:<identificare> [PROFESOR:"Traian Pop"]
PRS:<identificare> (v:profesor)
FRM:(<identificare> (profesor "Traian Pop"))
$
```

Prezentăm şi câteva formulări invalide (din clasa NEG), eliminate atât din antrenamentul cât şi din testrea analizorului. Ele fie depăşesc domeniul sistemului de dialog, fie sunt greşit formulate, nereprezentând intrări valide pentru analizor.

```
%NEG
SRO:ce înseamnă a se de ne
NOR:ce [MATERIA:"Analiza şi sinteza dispozitivelor numerice"]
$

%NEG
SRO:în ce zi am curs dimineaţa şi laborator după masă cu acelasi profesor
    în săli diferite
NOR:ce-zi curs [PERIOADA-ZI:"dimineaţă"] şi laborator [PERIOADA-ZI:
    "după-amiază"] profesor sală
$

%NEG
SRO:de ce se face pedagogie
NOR:ce [MATERIA:"Pedagogie"]
$

%NEG
SRO:care este primul laborator din săptămână
NOR:care laborator săptămână
$

%NEG
SRO:care este următorul laborator
NOR:care laborator
$
```

Anexa C

Structura bazei de date orar

Această anexă ilustrează o posibilă structură a unei baze de date conţinând informaţii despre orarul facultăţii noastre, stabilită în cursul definirii domeniului sistemului de dialog.

SALI		
I	ID_SALA	NUMERIC
	NUME	STRING
	LOCATIE	STRING

Tabelul C.1: Tabela de săli

MATERII		
I	ID_MATERIE	NUMERIC
	NUME	STRING
	TIP	CHAR
	DESCRIERE	STRING

Tabelul C.2: Tabela de materii

PROFESORI		
I	ID_PROFESOR	NUMERIC
	NUME	STRING
	GRAD	STRING

Tabelul C.3: Tabela de profesori

ORAR		
I	ID_MATERIE	NUMERIC
I	ID_PROFESOR	NUMERIC
I	ID_SALA	NUMERIC
	ORA_START	NUMERIC
	ORA_STOP	NUMERIC
	ZI	NUMERIC
	PARITATE_SAPT	NUMERIC
	AN	NUMERIC
	SPECIALIZARE	CHAR
	GRUPA	NUMERIC
	SUBGRUPA	NUMERIC

Tabelul C.4: Tabela orar

Anexa D

Fişierele de configurare

Comportamentul fiecăruia din cele 3 module ale analizorului semantic proiectat este controlat prin intermediul unor fişiere: modulul de preprocesare se bazează pe o serie de fişiere care controlează modalitatea în care se realizează etapele de preprocesare, decodorul semantic se bazează pe modelul Markov, iar constructorul de cadre – pe un fişier cu descrierea sistemului de cadre utilizat pentru generarea reprezentării semantice, şi eventual unul pentru controlul normalizării cadrelor. Structura acestor fişiere de control este descrisă în continuare.

D.1 Controlul preprocesării

Cinci din cele şapte etape de preprocesare (normalizarea numerelor, reducerea flexiunilor, unificarea expresiilor, înlocuirea aliasurilor şi unificarea categoriilor) se realizează prin înlocuiri de şiruri de caractere specificate în fişiere de configurare. Acestea sunt simple fişiere text, compuse din intrări de forma:

```
<şir_înlocuitor>: <şir_1>,<şir_2>, ...,<şir_N>;
```

unde `şir_1`, ..., `şir_N` vor fi înlocuite cu `şir_înlocuitor`.

Dacă şirurile de caractere conţin spaţii, ele vor fi delimitate prin ghilimele. În fişiere sunt permise şi comentarii, acestea fiind introduse prin caracterul `#` şi terminându-se la sfârşitul liniei curente.

În continuare sunt prezentate porţiuni din cele 5 fişiere utilizate pentru controlul etapelor de preprocesare menţionate în varianta finală a analizorului.

numbers.pp - normalizarea numerelor

```
# Fisier control preprocesare - normalizarea numerelor

1:  unu, întâi, "a întâia";
```

```
2:  doi, două, "al doilea", "a doua";
3:  trei, "al treilea", "a treia";
4:  patru, "al patrulea", "a patra";
5:  cinci, "al cincilea", "a cincea";
6:  şase, "al şaselea", "a şasea";
7:  şapte, "al şaptelea", "a şaptea";

# ...  şi aşa mai departe ...  (până la 24 + numerele de săli)
```

inflx.pp - reducerea inflexiunilor

```
# Fisier control preprocesare - reducerea inflexiunilor

an:  anul, anii;
grupă:  grupa, grupe, grupele;
subgrupă:  subgrupa, subgrupe, subgrupele, semigrupă,
   semigrupa, semigrupe, semigrupele;
serie:  seria, seriile, serii;
secţie:  secţia, secţiile, secţii;

laborator:  laboratorul, laboratoarele, laboratoare;
curs:  cursul, cursuri, cursurile;
seminar:  seminarul, seminarele, seminare, seminarii, seminariile;
proiect:  proiectul, proiectele, proiecte;

sală:  sala, sălile, săli, sale;

student:  studentul, studentă, studenta, studenţi, studenţii;
profesor:  profesorul, profesoară, profesoara, profesori, profesorii,
   "cadru didactic", "cadrul didactic";
asistent:  asistentul, asistentă, asistenta, asistenţi, asistenţii;

zi:  zile, ziua, zilele, zilei;
dimineaţă:  dimineaţa, dimineţile;
seară:  seara, serile, deseară;
noapte:  noaptea, nopţile;
semestru:  semestrul, semestre, semestrele;

astăzi:  azi;

orar:  orarul, orare, orarele;
ore:  orele;
oră:  ora;

duminică:  dumineca, duminica;
luni:  lunea;
marţi:  marţea;
```

```
miercuri: miercurea;
joi: joia;
vineri: vinerea;
sâmbătă: sâmbăta;

impar: impară, impare;
```

expr.pp - unificarea expresiilor

```
# Fisier control preprocesare - unificarea expresiilor

ce-ore: "ce ore", "ce orar";
ce-oră: "ce oră";
ce-curs: "ce curs";
ce-laborator: "ce laborator";
ce-seminar: "ce seminar";
ce-proiect: "ce proiect";
ce-profesor: "ce profesor", "ce asistent";

ce-sală: "ce sală";
cu-cine: "cu cine", "cu ce profesor", "cu ce asistent";

de-la: "de la oră", "de la";
la: "la oră";

după-amiază: "după masă", "după amiază", "după masa", "după amiaza";
ce-zi: "ce zi";
ce-am: "ce am";
```

alias.pp - înlocuirea aliasurilor

```
# Fisier control preprocesare - inlocuiri alias-uri

# Aliasuri denumiri materii ------------------------------------

"Dispozitive şi circuite electronice":
   "d c e",
    "de ce e",
    "dispozitive şi circuite electronice",
    "circuite electronice";

"Matematici speciale":
    "matematici speciale",
    "me se",
```

```
    "m s",
    "matematici";

"Analiza şi sinteza dispozitivelor numerice":
    "analiza şi sinteza dispozitivelor numerice",
    "a s d n",
    "a se de ne";

# ...  şi aşa mai departe

# Aliasuri sali de clasa -------------------------------------
A101:  "a 101", "a 1 0 1", "101";
A106:  "a 106", "a 1 0 6", "106";
A109:  "a 109", "a 1 0 9", "109";
A110:  "a 110", "a 1 1 0", "110";

# ...  şi aşa mai departe

# Aliasuri specialitati--------------------------------------
H1:  "hard 1", "h 1";
H2:  "hard 2", "h 2";
S1:  "soft 1", "s 1";
S2:  "soft 2", "s 2";

# Aliasuri profesori -- probabil nu foarte acurate-----------
"Sorin Babii":
    "Babii",
    "Babii Sorin";

"Dan Cosma":
    "Cosma",
    "Cosma Dan";

"Vladimir Creţu":
    "Creţu",
    "Creţu Vladimir";

# ...  şi aşa mai departe
```

categ.pp - unificarea categoriilor

```
# Fisierul control preprocesare - unificarea categoriilor

MATERIA:
    "Dispozitive şi circuite electronice",
    "Matematici speciale",
    "Sport",
    "Engleza",
    "Analiza şi sinteza dispozitivelor numerice",
    "Programarea calculatoarelor",
    "Electrotehnică II",

# ...  şi aşa mai departe ...
ZI-SAPT:
    luni,
    marţi,
    miercuri,
    joi,
    vineri,
    sâmbătă,
    duminică;

ZI-RELATIV:
    astăzi,
    mâine,
    poimâine,
    răspoimâine,
    ieri,
    alaltăieri;

GRUPA:
    H1, H2, S1, S2;

NR:
    1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24;

PROFESOR:
    "Sorin Babii",
    "Nicolae Bogoevici",
    "Radu Boraci",
    "Carmen de Sabata",
    "Horia Ciocârlie",
    "Dan Cosma",

# ...  şi aşa mai departe

SPECIALITATEA:
```

```
    calculatoare,
    automatică,
    colegiu;

PERIOADA-ZI:
    dimineaţă,
    după-amiază,
    seară,
    prânz,
    noapte;

SALA:
    A101, A106, A109, A110, A117, A204, A212,
    A305, A314, A315, ASPC, B011, B020, B028,
    B303, B305, B413, B414, B419, B425, B513,
    B514, B520, B528, B528B, B529, B623, B624,
    C312, CA2, SPM;
```

D.2 Controlul constructorului de cadre

Modulul de construcţie a cadrelor citeşte definiţia sistemului de cadre utilizat pentru realizarea analizei semantice dintr-un fişier de configurare. Structura acestui fişier este asemănănătoare cu cea a fişierelor de control al preprocesării, fiind de tip text şi având intrări de forma:

```
<concept>:  <slot_1>,<slot_2>, ...,<slot_N>;
```

unde fiecare concept defineşte un cadru. Sloturile pot fi şiruri de caractere, caz în care valorile lor sunt simple, sau pot fi la rândul lor concepte (încadrate între caracterele “<” şi “>”), caz în care valoarea slotului respectiv indică o legătură spre un subcadru dat de respectivul concept.

Pentru exemplificare, prezentăm în continuare fişierul de descriere a sistemului de cadre utilizat în varianta finală a analizorului semantic proiectat.

frames.fb - construcţia cadrelor

```
<da>:; # cadru fără sloturi

<nu>:; # cadru fără sloturi

<identificare>:
    anul,
```

```
    specialitatea,
    seria,
    grupa,
    subgrupa,
    profesor;

<specif-timp>:
    zi-relativ,
    zi-sapt,
    perioada-zi,
    ora-start,
    ora-stop;

<specif-materie>:
    tip-materie,
    materie,
    laborator,
    seminar,
    proiect,
    curs;

<ce-materie>:
    <identificare>, # subcadru
    <specif-materie>, # subcadru
    <specif-timp>, # subcadru
    sala; # cadru simplu

<ce-curs>:
    <identificare>,
    <specif-materie>,
    <specif-timp>,
    sala;

<ce-laborator>:
    <identificare>,
    <specif-materie>,
    <specif-timp>,
    sala;

<ce-seminar>:
    <identificare>,
    <specif-materie>,
    <specif-timp>,
    sala;
```

```
<ce-proiect>:
    <identificare>,
    <specif-materie>,
    <specif-timp>,
    sala;

<cand>:
    <identificare>,
    <specif-materie>,
    <specif-timp>,
    sala;

<unde>:
    <identificare>,
    <specif-materie>,
    <specif-timp>,
    sala;

<cu-cine>:
    <identificare>,
    <specif-materie>,
    <specif-timp>,
    sala;
```

Anexa E

Extrase de cod C++

E.1 Antete de clase

CStringSequence

```
/************************************************************
                          stringsequence.h
 ************************************************************
  Models a sequence of strings
 ************************************************************
    begin              : Wed Apr 5 2000
    copyright          : (C) 2000 by Dan Bohus
    email              : danbo@ear.utt.ro
 ************************************************************/

#ifndef STRINGSEQUENCE_H
#define STRINGSEQUENCE_H

/**
  *@author Dan Bohus
  */

#define STRINGSEQUENCE_DEFAULT_SIZE 10
#define STRINGSEQUENCE_DELTA 10

// CStringSequence class - models a sequence of strings data
// structure. The strings are represented as a dynamically
// allocated array that grows with STRINGSEQUENCE_DELTA
// when necessary.
class CStringSequence {
private:
  int Size;       // allocated array size
  char **Values;  // strings array
```

```
public:
  int Length;      // number of strings
public:
  // Orthodox-canonical class construction
  CStringSequence();
  CStringSequence(const CStringSequence& aStringSequence);
  CStringSequence& operator=(const CStringSequence&
                             aStringSequence);
  bool operator==(const CStringSequence& aStringSequence);
  ~CStringSequence();

  // string sequence manipulation & access routines
  void Clear();
  int Add(const char* aValue);
  int Insert(int aIndex, const char* aValue);
  void Delete(int aIndex);
  void Reverse();
  char*& operator[](const int aIndex);
};

#endif
```

CSequence

```
/***************************************************************
                          sequence.h
 ***************************************************************
  Models a sequence of integers
 ***************************************************************
    begin                : Tue Apr 4 2000
    copyright            : (C) 2000 by Dan Bohus
    email                : danbo@ear.utt.ro
 ***************************************************************/

#ifndef SEQUENCE_H
#define SEQUENCE_H

/**
  *@author Dan Bohus
  */

#define SEQUENCE_DEFAULT_SIZE 20
#define SEQUENCE_DELTA 10

// CSequence class - models a sequence of integers data
// structure. The integers are stored as a dynamically
// allocated array that grows with SEQUENCE_DELTA when
// necessary.
class CSequence {
private:
  int Size;      // allocated array size
  int* Values;   // integers array
public:
  int Length;    // number of integers
public:
  // Orthodox-canonical class construction
  CSequence();
  CSequence(const CSequence& aSequence);
  CSequence& operator=(const CSequence& aSequence);
  ~CSequence();

  // sequence manipulation & access routines
  void Clear();
  int Add(const int aValue);
  void Delete(int aIndex);
  void Reverse();
  int& operator[](const int aIndex);
};

#endif
```

CCaseFrame

```
/*********************************************************
                        caseframe.h
 *********************************************************
  Models the case-frame data structure.
 *********************************************************
    begin                : Wed May 3 2000
    copyright            : (C) 2000 by Dan Bohus
    email                : danbo@ear.utt.ro
 *********************************************************/

#ifndef CASEFRAME_H
#define CASEFRAME_H

/**
  *@author Dan Bohus
  */

#include <stdio.h>

// forward CCaseFrame class declaration
class CCaseFrame;

// union used to encompass String and SubFrame values for
// case-values
typedef union TCaseValue {
  char* String;
  CCaseFrame* SubFrame;
};

// result type for case-frame definition reading routine
typedef enum CReadFrameDefinitionResult {rfdrOK,
      rfdrError, rfdrEndOfFile};

// CCaseFrame class - models a case frame.
class CCaseFrame {
private:
  char* Concept;                // frame concept
  int AttributesNo;             // number of attributes
  char** Attributes;            // attributes array
  TCaseValue* Values;           // values array
public:
  // Orthodox-canonical Class Construction
  CCaseFrame();
  CCaseFrame(CCaseFrame& aCaseFrame);
  CCaseFrame& operator=(CCaseFrame& aCaseFrame);
  bool operator==(CCaseFrame& aCaseFrame);
  ~CCaseFrame();
```

```
  // case-frame clearing routines
  void Clear();                    // clears the entire frame
  void ClearValues();              // clears the case-values

  // seralization routines
  CReadFrameDefinitionResult ReadDefinitionFromFile(FILE*
                                                     fileP);
  bool DecodeFromBuffer(const char* buffer, int& pointer);
  void EncodeToBuffer(char* buffer);
  void NicePrintToFile(FILE* fileP, bool printVoidValues,
                       int tab=0);

  // frame manipulation & access routines
  bool IsInstantiated();
  char* GetConcept();
  int GetAttributesNo();
  bool HasAttribute(char* attribute);
  char* GetAttribute(int aIndex);
  int GetIndexOfAttribute(char* attribute);
  TCaseValue* operator[](char* attribute);
  TCaseValue* operator[](int aIndex);

  bool SetValueForAttribute(char* attribute, char* value);
};

#endif
```

CCaseFrameSystem

```
/***************************************************************
                        caseframesystem.h
 ***************************************************************
  Models a case-frame system
 ***************************************************************
    begin                : Wed May 3 2000
    copyright            : (C) 2000 by Dan Bohus
    email                : danbo@ear.utt.ro
 ***************************************************************/

#ifndef CASEFRAMESYSTEM_H
#define CASEFRAMESYSTEM_H

/**
  *@author Dan Bohus
  */

#include "caseframe.h"

#define CASEFRAMESYSTEM_DEFAULT_SIZE 10
#define CASEFRAMESYSTEM_DELTA 3

// CCaseFrameSystem class - models a set of case frames
// forming a case-frame system. The set is implemented as a
// dynamically allocated array of CCaseFrame objects
class CCaseFrameSystem {
private:
  int Size;                   // allocated array size
  CCaseFrame** CaseFrames;    // case-frames
public:
  int Length;            // number of case-frames in the system
public:
  // Orthodox-canonical class construction
  CCaseFrameSystem();
  CCaseFrameSystem(CCaseFrameSystem& aCaseFrameSystem);
  CCaseFrameSystem& operator=(CCaseFrameSystem&
                              aCaseFrameSystem);
  ~CCaseFrameSystem();

  // read definition from file
  bool LoadFromFile(const char* fileName);
  // case frame system acces & manipulation routines
  int Add(CCaseFrame& aCaseFrame);
  CCaseFrame& operator[](const int aIndex);
  void Clear();
};

#endif
```

CUtterance

```
/*********************************************************
                         utterance.h
 *********************************************************
  Models a user utterance, with all possible forms through
  decoding: speech recognizer output, normalized form,
  parse & case-frame representation
 *********************************************************
    begin                 : Wed Apr 5 2000
    copyright             : (C) 2000 by Dan Bohus
    email                 : danbo@ear.utt.ro
 *********************************************************/

#ifndef UTTERANCE_H
#define UTTERANCE_H

/**
  *@author Dan Bohus
  */

#include <stdio.h>

#include "stringsequence.h"
#include "caseframe.h"

// CUtterance class - models an utterance
class CUtterance {
public:
  char* Class;                      // utterance class
  char* SpeechRecognizerOutput;     // speech recognizer output
  CStringSequence NormalizedForm;   // normalized form
  CStringSequence Parse;            // parse
  int CaseFramesNo;                 // number of case-frames
  CCaseFrame* CaseFrames;           // case-frames
public:
  // Orthodox-canonical class construction
  CUtterance();
  CUtterance(char* aClass,
       char* aSpeechRecognizerOutput,
       CStringSequence* aNormalizedForm = NULL,
       CStringSequence* aParse = NULL,
       int aCaseFrameNo = 0,
       CCaseFrame* someCaseFrames = NULL);
  CUtterance(FILE* fileP);
  CUtterance(CUtterance& aUtterance);
  CUtterance& operator=(CUtterance& aUtterance);
  bool operator==(CUtterance& aUtterance);
  ~CUtterance();
```

```
    // serialization routines
    bool LoadFromFile(FILE* fileP);
    bool SaveToFile(FILE* fileP);

    // utterance access & manipulation routines
    bool HasClass();
    bool HasSpeechRecognizerOutput();
    bool HasNormalizedForm();
    bool HasParse();
    bool HasCaseFrames();

    void Clear();
    void ClearCaseFrames();
    void DeleteCaseFrame(const int aIndex);
    void SetClass(char* aClass);
    void SetSpeechRecognizerOutput(char*
                                   aSpeechRecognizerOutput);
};

#endif
```

CUtteranceCorpus

```
/*************************************************************
                         utterancecorpus.h
 *************************************************************
  Models a corpus of utterances
 *************************************************************
    begin                : Thu Apr 6 2000
    copyright            : (C) 2000 by Dan Bohus
    email                : danbo@ear.utt.ro
 *************************************************************/

#ifndef UTTERANCECORPUS_H
#define UTTERANCECORPUS_H

/**
  *@author Dan Bohus
  */

#include "utterance.h"

#define UTTERANCECORPUS_DEFAULT_SIZE 500
#define UTTERANCECORPUS_DELTA 50

// CUtteranceCorpus class - models a corpus of utterances.
// The utterances are stored as a dynamically allocated
// array that grows with UTTERANCECORPUS_DELTA whenever
// necessary
class CUtteranceCorpus {
private:
  int Size;                    // allocated array size
  CUtterance** Utterances;     // utterances array
public:
  int Length;                  // number of utterances
public:
  // Orthodox-canonical class construction
  CUtteranceCorpus();
  CUtteranceCorpus(const char* fileName);
  CUtteranceCorpus(CUtteranceCorpus& aUtteranceCorpus);
  CUtteranceCorpus& operator=(CUtteranceCorpus&
                             aUtteranceCorpus);
  ~CUtteranceCorpus();

  // serialization routines
  bool AddFromFile(const char* fileName);
  bool LoadFromFile(const char* fileName);
  bool SaveToFile(const char* fileName);

  // utterance corpus access & management routines
  int Add(CUtterance& aUtterance);
```

```
  void Clear();
  CUtterance& operator[](const int aIndex);
};

#endif
```

CDictionary

```
/***********************************************************
                    dictionary.h
 ***********************************************************
  Models a dictionary data structure
 ***********************************************************
    begin                : Tue Apr 4 2000
    copyright            : (C) 2000 by Dan Bohus
    email                : danbo@ear.utt.ro
 ***********************************************************/

#ifndef DICTIONARY_H
#define DICTIONARY_H

/**
  *@author Dan Bohus
  */

#include "sequence.h"
#include "stringsequence.h"

#define DICTIONARY_DEFAULT_SIZE 500
#define DICTIONARY_DELTA 50

// CDictionary class - models a typical dictionary data
// structure. The words are represented as a dynamically
// allocated array that grows with DICTIONARY_DELTA when
// necessary
class CDictionary {
private:
  int Size;     // allocated size
  char** Strings;              // strings array
public:
  int Length;                  // number of strings in dictionary
public:
  // Orthodox-canonical class construction
  CDictionary();
  ~CDictionary();
  CDictionary(const CDictionary& aDictionary);
  CDictionary& operator=(const CDictionary& aDictionary);

  // serialization routines
  bool LoadFromFile(const char* fileName);
  bool SaveToFile(const char* fileName);

  // dictionary access routines
  char* operator[](const int aIndex);
  int IndexOf(const char* aString);
  bool Exists(const char* aString);
```

```
  // dictionary manipulation routines
  int Add(const char* aString);
  void Delete(const char* aString);
  void Clear();

  // dictionary-based conversion routines
  bool ToIndexes(CStringSequence& ss, CSequence& s);
  bool ToStrings(CSequence& s, CStringSequence& ss);
};

#endif
```

CPreprocessor

```
/*********************************************************
                         preprocessor.h
 *********************************************************
  Implements the preprocessing module
 *********************************************************
    begin                 : Sat Apr 8 2000
    copyright             : (C) 2000 by Dan Bohus
    email                 : danbo@ear.utt.ro
 *********************************************************/

#ifndef PREPROCESSOR_H
#define PREPROCESSOR_H

/**
  *@author Dan Bohus
  */

#include "preprocessorconfiguration.h"
#include "utterancecorpus.h"
#include "sequence.h"
#include "stringsequence.h"
#include "dictionary.h"
#include "repltable.h"

// preprocessor stage type definition
typedef enum TReplType {rtNumbersNormalization,
                        rtInflexReduction,
                        rtExprUnification,
                        rtAliasesReplacement,
                        rtCategoryUnification};

// CPreprocessor class - models semantic preprocessing
class CPreprocessor {
private:
  CPreprocessorConfiguration PreprocessorConfiguration;
  TReplTable ReplTables[5];       // replacement tables
  CStringSequence Replacements;   // replacements
  CDictionary Dictionary;         // normalized dictionary
private:
  // internal configuration routines
  bool Configure();
  bool ConfigureType(TReplType type, const char* fileName);
  void StringToSequence(char* from, CStringSequence& sequence);
public:
  // Orthodox-canonical class construction
  CPreprocessor();
  CPreprocessor(CPreprocessor& aPreprocessor);
  CPreprocessor&  operator=(CPreprocessor& aPreprocessor);
```

```
  ~CPreprocessor();

  // set / get preprocessor configuration routines
  bool SetConfiguration(CPreprocessorConfiguration&
                         aPreprocessorConfiguration);
  CPreprocessorConfiguration& GetConfiguration();

  // preprocessing routines
  CStringSequence PreprocessString(char* string);
  void PreprocessUtterance(CUtterance& aUtterance);
  void PreprocessCorpus(CUtteranceCorpus& aUtteranceCorpus);

  // preprocessing stages routines
  void PerformNonLexicalEventsReduction(char* from, char* to);
  void PerformNumbersNormalization(char* from, char* to);
  void PerformInflexReduction(char* from, char* to);
  void PerformExprUnification(char* from, char* to);
  void PerformAliasesReplacement(char* from, char* to);
  void PerformCategoryUnification(char* from, char* to);
  void PerformDictionaryCheck(char* from,
                              CStringSequence& sequence);
};

#endif
```

CHMM

```
/*********************************************************
                        hmm.h
 *********************************************************
  Models the Hidden Markov Model. The A, B and Pi probability
  distributions are represented as matrices. Training is
  accomplished by counting events in the training corpus.
  Routines for Katz reestimation and Viterbi decoding
  are implemented.
 *********************************************************
    begin               : Thu Apr 20 2000
    copyright           : (C) 2000 by Dan Bohus
    email               : danbo@ear.utt.ro
*********************************************************/

#ifndef HMM_H
#define HMM_H

/**
  *@author Dan Bohus
  */

#include "dictionary.h"
#include "utterancecorpus.h"

// type of HMM closing used
typedef enum TClosingType {ctNone,
                           ctMaximumLikelihood,
                           ctKatzReestimation};

// CHMM class
class CHMM {
private:
  int StatesNo;   // number of internal (hidden) states
  int SymbolsNo;  // number of symbols

  int* trISPM;    // training initial state prob. matrix
  int* trOSPM;    // training observation symbol prob. matrix
  int* trSTPM;    // training state transition prob. matrix

  double* ISPM;   // initial state probabilities matrix
  double* OSPM;   // observation symbol probabilities matrix
  double* STPM;   // state transition probabilities matrix

  CDictionary SymbolsDictionary; // obs. symbols dictioanry
  CDictionary StatesDictionary;  // states dictionary

public:
  // Orthodox-canonical class construction
```

```
    CHMM();
    CHMM(const char* aSymbolsDictionaryFileName,
         const char* aStatesDictionaryFileName);
    CHMM(char* fileName);
    CHMM(CHMM& aHMM);
    CHMM& operator=(CHMM& aHMM);
    ~CHMM();

    // serialization routines
    bool LoadFromFile(const char* fileName);
    bool SaveToFile(const char* fileName);

    // HMM model management routines
    void Clear();
    bool IsEmpty();
    bool IsClosed();

    // public training routines
    int TrainWithUtterance(CUtterance& aUtterance, FILE* logFP);
    void TrainWithCorpus(CUtteranceCorpus& aCorpus, FILE* logFP);

    // public closure routines
    void CloseModel(TClosingType ct, int k=0);

    // public state decoding routines
    double DecodeUtterance(CUtterance& aUtterance);
    void DecodeCorpus(CUtteranceCorpus& aCorpus);

    // printing routine
    void PrintModelParameters(FILE* fp);

private:
    // closure routines:
    void MLEClosure();       // maximum likelihood estimation
    void KatzClosure(int k);  // Katz re-estimation

    // decoding (Viterbi) algorithm
    double Viterbi(CSequence& obs, CSequence& states);
};

#endif
```

CCaseFrameBuilder

```
/*******************************************************
                    caseframebuilder.h
 *******************************************************
  Models the case-frame builder module
 *******************************************************
    begin                : Thu May 4 2000
    copyright            : (C) 2000 by Dan Bohus
    email                : danbo@ear.utt.ro
 *******************************************************/

#ifndef CASEFRAMEBUILDER_H
#define CASEFRAMEBUILDER_H

/**
  *@author Dan Bohus
  */

#include "caseframesystem.h"
#include "utterancecorpus.h"
#include "stringsequence.h"
#include "sequence.h"
#include "repltable.h"

// result type for case frame building routine
typedef enum TFrameBuilderResult {fbrError,
                                  fbrNoConcept,
          fbrOneConcept,
                                  fbrMultipleConcept};

// CCaseFrameBuilder class
class CCaseFrameBuilder {
private:
  CCaseFrameSystem CaseFrameSystem; // case frame system
  bool NormalizeFrames;             // normalize frames flag
  TReplTable ReplTable;             // norm. replacement table
  CStringSequence Replacements;     // norm. replacement values
private:
  // internal frame building routines
  void ExpandFrame(CCaseFrame& aCaseFrame);
  bool ReduceFrame(CCaseFrame& aCaseFrame);
  void NormalizeFrame(CCaseFrame& aCaseFrame);
public:
  // Orthodox-canonical class construction
  CCaseFrameBuilder();
  CCaseFrameBuilder(CCaseFrameSystem& aCaseFrameSystem);
  CCaseFrameBuilder(CCaseFrameBuilder& aCaseFrameBuilder);
  CCaseFrameBuilder& operator=(CCaseFrameBuilder&
                               aCaseFrameBuilder);
```

```
  ~CCaseFrameBuilder();

  // case-frame builder settings control
  void SetCaseFrameSystem(CCaseFrameSystem& aCaseFrameSystem);
  CCaseFrameSystem& GetCaseFrameSystem();
  void DoFramesNormalization(const char* fileName);

  // public frame construction routines
  TFrameBuilderResult ProcessUtterance(CUtterance& aUtterance);
  void ProcessCorpus(CUtteranceCorpus& aUtteranceCorpus);
};

#endif
```

CSemanticAnalyzer

```
/***********************************************************
                        semanticanalyzer.h
 ***********************************************************
  Models the semantic analyzer by aggregating separate
  preprocessing, decoding and frame-building modules
 ***********************************************************
    begin                : Fri May 19 2000
    copyright            : (C) 2000 by Dan Bohus
    email                : danbo@ear.utt.ro
 ***********************************************************/

#ifndef SEMANTICANALYZER_H
#define SEMANTICANALYZER_H

/**
  *@author Dan Bohus
  */

#include <stdio.h>

#include "semanticanalyzerconfiguration.h"
#include "utterance.h"
#include "utterancecorpus.h"
#include "conceptarray.h"
#include "preprocessor.h"
#include "hmm.h"
#include "caseframebuilder.h"

// CSemanticAnalyzer class
class CSemanticAnalyzer {
private:
  CSemanticAnalyzerConfiguration SemanticAnalyzerConfiguration;
  CPreprocessor Preprocessor;          // preprocessor module
  CHMM HMM;                            // HMM decoding module
  CCaseFrameBuilder CaseFrameBuilder;  // frame build. module

private:
  // internal configuration routine
  bool Configure();

public:
  // Orthodox-canonical class construction
  CSemanticAnalyzer(const char* fileName = NULL);
  CSemanticAnalyzer(CSemanticAnalyzer& aSemanticAnalyzer);
  CSemanticAnalyzer&  operator=(CSemanticAnalyzer&
                                aSemanticAnalyzer);
  ~CSemanticAnalyzer();
```

```
  // get / set analyzer configuration
  bool SetConfiguration(CSemanticAnalyzerConfiguration&
                        aSemanticAnalyzerConfiguration);
  CSemanticAnalyzerConfiguration& GetConfiguration();

  // semantic analysis routines
  char* AnalyzeString(char* string,
                      CConceptArray* aConceptArray = NULL);
  void AnalyzeUtterance(CUtterance& aUtterance,
                        CConceptArray* aConceptArray = NULL);
  void AnalyzeCorpus(CUtteranceCorpus& aUtteranceCorpus);
};

#endif
```

E.2 Algoritmi

Reestimarea Katz

```
// D: perform Katz Closure
//    Katz discounts all events occuring less than k times,
//    considering them unreliable. The freed probability mass
//    is redistributed among unseen events.
//    LittleK - used for reestimation of initial state prob.
//    BigK    - used for reestimation of state transition prob.

void CHMM::KatzClosure(int LittleK, int BigK){
  if(!IsClosed()) {
    // allocate floating point matrices
    ISPM = new double[StatesNo];
    OSPM = new double[StatesNo*SymbolsNo];
    STPM = new double[StatesNo*StatesNo];

    // normalize trOSPM - just like MLE (Katz only works on
    // state transition and initial state probabilities)
    for(int i=0; i<StatesNo; i++)
      for(int j=0; j<SymbolsNo; j++)
        if(trOSPM[i*(SymbolsNo+1)+j]==0)
          OSPM[i*SymbolsNo+j] = 0;
        else
          OSPM[i*SymbolsNo+j] = trOSPM[i*(SymbolsNo+1)+j]/
                                (double)trOSPM[i*(SymbolsNo+1)
                                +SymbolsNo];

    // if LittleK=0, perform simple MLE
    if(LittleK == 0) {
      // normalize ISPM
      for(int i=0; i<StatesNo; i++)
        if(trISPM[i]==0) ISPM[i]=0;
        else ISPM[i] = trISPM[i]/(double)trISPM[StatesNo];
    } else {
      // else perform Katz reestimation for initial state
      // probability distribution

      // first compute initial state density distribution
      // find out most occurent initial state
      int maxValue = -1;
      int maxI = 0;
      for(int i=0; i<StatesNo; i++)
        if(trISPM[i]>maxValue) {
          maxValue = trISPM[i];
```

```
    maxI = i;
  }

// allocate space and initialize density distribution
int *nis = new int[maxValue+1];
for(int i=0; i<=maxValue; nis[i++] = 0);

// compute initial state density distribution
for(int i=0; i<StatesNo; i++)
  nis[trISPM[i]]++;

// compute normalizing factor for discount coefficients
double fis = 1 - ((LittleK+1)*nis[LittleK+1])/
                  (double)(nis[1]);

// allocate initial state discount coefficients
double* dcis = new double[LittleK];

// precompute intial state discount parameters
for(int i=1; i<=LittleK; i++)
  // compute discount parameter for i, store it at
  // dcis[i-1]
  dcis[i-1] = ((((i+1)*nis[i+1])/(double)(i*nis[i]))+
               fis-1)/fis;

// now for the initial states
// compute A = freed probability mass
// compute B = normalization factor for freed probability
//             mass redistribution
double A = 1; // start from full probability
double B = 0; // start from zero
bool noDiscount = 1;
for(int i=0; i<StatesNo; i++) {
  if(trISPM[i]>0) {
    // positive initial state occurence
    if(trISPM[i]<=LittleK) { // discounted
      A -= dcis[trISPM[i]-1]*trISPM[i]/
           (double)trISPM[StatesNo];
      noDiscount = 0;
    } else  // not discounted
      A -= trISPM[i]/(double)trISPM[StatesNo];
  }
  else B += trSTPM[i*(StatesNo+1)+StatesNo];
}

// if no discount occured, A must be at or pretty
// close to 0, so manually normalize it to avoid small
// non-zero values
if(noDiscount) A = 0;
```

```
    // now, finally we can compute the reestimated Katz
    // probabilities for the initial state distribution
    for(int i=0; i<StatesNo; i++) {
      if(trISPM[i]>0) {
        if(trISPM[i]<=LittleK)
          ISPM[i] = dcis[trISPM[i]-1]*trISPM[i]/
                    (double)trISPM[StatesNo];
        else ISPM[i] = trISPM[i]/(double)trISPM[StatesNo];
      }
      else ISPM[i] = (A/B)*trSTPM[i*(StatesNo+1)+StatesNo];
    }

    // deallocate dcis and nis
    delete[] nis;
    delete[] dcis;
  }

  // now go to trSTPM - state transition probabilities
  // if no BigK, then perform simple MLE
  if(BigK == 0) {
    // normalize trSTPM
    for(int i=0; i<StatesNo; i++) {
      int total = 0;
      for(int j=0; j<StatesNo; j++)
        total += trSTPM[i*(StatesNo+1)+j];
      for(int j=0; j<StatesNo; j++)
        if(trSTPM[i*(StatesNo+1)+j]==0)
          STPM[i*StatesNo+j] = 0;
        else STPM[i*StatesNo+j] = trSTPM[i*(StatesNo+1)+j]/
                                  (double)total;
    }
  } else {
    // else perform Katz reestimation on state transition
    // probabilities

    // first determine bigram density distribution
    // first find out most occurent bigram
    int maxValue = -1;
    int maxI = 0, maxJ = 0;
    for(int i=0; i<StatesNo; i++)
      for(int j=0; j<StatesNo; j++)
        if(trSTPM[i*(StatesNo+1)+j]>maxValue) {
          maxValue = trSTPM[i*(StatesNo+1)+j];
          maxI = i;
          maxJ = j;
        }

    // allocate space and initialize bigram density distr.
```

```
int *n = new int[maxValue+1];
for(int i=0; i<=maxValue; n[i++] = 0);

// compute bigram density distribution
for(int i=0; i<StatesNo; i++)
  for(int j=0; j<StatesNo; j++)
    n[trSTPM[i*(StatesNo+1)+j]]++;

// compute normalizing factor for discount coefficients
double f = 1 - ((BigK+1)*n[BigK+1])/(double)(n[1]);

// allocate discount coefficients
double* dc = new double[BigK];

// precompute discount parameters
for(int i=1; i<=BigK; i++)
  // compute discount parameter for i, store it
  // at dc[i-1]
  dc[i-1] = ((((i+1)*n[i+1])/(double)(i*n[i]))+f-1)/f;

// now for each state
for(int i=0; i<StatesNo; i++) {
  // ci - total occurences of state i as the first state
  //      in a transition
  int ci = 0;
  for(int j=0; j<StatesNo; j++)
    ci += trSTPM[i*(StatesNo+1)+j];

  // A - freed probability mass
  // B - normalizing constant
  double A = 1; // start from full probability
  double B;     // start from zero

  bool noDiscount = 1;

  // for each transition compute freed probability mass
  for(int j=0; j<StatesNo; j++) {
    int cij = trSTPM[i*(StatesNo+1)+j];
    if(cij>0) {
      if(cij<=BigK) {
        A -= dc[cij-1]*cij/(double)ci;
        noDiscount = 0;
      } else A -= cij/(double)ci;
    }
    else B += trSTPM[j*(StatesNo+1)+StatesNo];
  }

  // if no discount occured, A must be at or pretty
  // close to 0, so manually normalize it to avoid small
```

```
        // non-zero values
        if(noDiscount) A = 0;

        // now, finally we can compute the reestimated Katz
        // probabilities
        for(int j=0; j<StatesNo; j++) {
          int cij = trSTPM[i*(StatesNo+1)+j];
          int cj = 0;
          for(int l=0; l<StatesNo; l++)
            cj += trSTPM[j*(StatesNo+1)+l];
          if(cij>0) {
            if(cij<=BigK)
              STPM[i*StatesNo+j] = dc[cij-1]*cij/(double)ci;
            else
              STPM[i*StatesNo+j] = cij/(double)ci;
          } else
          STPM[i*StatesNo+j] = (A/B)*
                              trSTPM[j*(StatesNo+1)+StatesNo];
        }
      }

      // don t forget to deallocate n and dc
      delete[] n;
      delete[] dc;
    }

    // finally deallocate training matrices
    delete[] trISPM;
    delete[] trOSPM;
    delete[] trSTPM;
    trISPM = trOSPM = trSTPM = NULL;
  }
}
```

Algoritmul Viterbi

```
// D: performs Viterbi decoding on the specified
//    observation sequence
//    returns the maximum path probability obtained

double CHMM::Viterbi(CSequence& obs, CSequence& states){
  // get observation sequence length
  int m = obs.Length;
  int n = StatesNo;

  // allocate delta(symbol, state), phi(state, nextstate)
  double* delta = new double[n*m];
  int* phi = new int[n*m];

  // INITIALIZATION
  for(int state=0; state<n; state++) {
    delta[/*(n*0)+*/state] = ISPM[state]*
                             OSPM[state*SymbolsNo+obs[0]];
    phi[/*(n*0)+*/state] = 0;
  }

  // RECURSION
  for(int symbol=1; symbol<m; symbol++) {
    for(int state=0; state<n; state++) {
      double maxvalue = -1;
      int maxarg = -1;
      for(int prevstate=0; prevstate<n; prevstate++) {
        double newvalue = delta[(n*(symbol-1))+prevstate]*
                          STPM[prevstate*StatesNo+state];
        if(newvalue>maxvalue) {
          maxvalue = newvalue;
          maxarg = prevstate;
        }
      }
      delta[(n*symbol)+state] = maxvalue *
                          OSPM[state*SymbolsNo+obs[symbol]];
      phi[(n*symbol)+state] = maxarg;
    }
  }

  // TERMINATION
  double maxvalue = -1;
  int maxarg = -1;
  for(int state=0; state<n; state++) {
    double newvalue = delta[n*(m-1)+state];
    if(newvalue>maxvalue) {
      maxvalue = newvalue;
      maxarg = state;
```

```
    }
  }

  // PATH BACKTRACKING
  states.Clear();
  for(int i=0; i<m; i++) states.Add(i);

  states[m-1] = maxarg;
  for(int i=m-2; i>=0; i--)
    states[i] = phi[(n*(i+1))+states[i+1]];

  return maxvalue;
}
```

Anexa F

Invocarea programelor utilitare

hmm_build

```
hmm_build       [-create|-train|-close filename] [-l|-la [filename]]
                <specific_options>
Options:
 -create filename - creates a new HMM model to filename
 -train filename  - trains an existing HMM model
 -close filename  - closes the training of an existing HMM model
 -l filename      - create log file -> filename
 -la filename     - append to log file -> filename

Specific CREATE options:
 -sym filename    - symbols dictionary filename
 -st filename     - states dictionary filename

Specific TRAIN options:
 -corpus filename - training corpus filename

Specific CLOSE options:
 -mle             - close for maximum likelihood estimations
 -katz k1         - close by Katz re-estimation of state transition
                    probabilities with parameter k1
 -katzI k2        - close by Katz re-estimation of initial state
                    probabilities with parameter k2
```

hmm_print

```
hmm_print       filename [-o filename]
Options:
 filename         - HMM model to print
 -o filename      - file to direct output to
```

hmm_decode

```
hmm_decode      hmm_filename -i filename [-o filename]
Options:
 hmm_filename     - HMM filename
 -i filename      - utterance corpus filename to decode
 -o filename      - utterance corpus filename to write to
                     if absent, output is directed to input file
```

preproc

```
preproc         filename [-o filename] <tasks>
Options:
 filename         - utterance corpus filename to process
 -o filename      - utterance corpus filename to write to
                     if absent, output is directed to input file
Tasks:
 -nler            - do non-lexical events reduction
 -nn filename     - do numbers normalization as specified by filename
 -ir filename     - do inflexions reduction as specified by filename
 -eu filename     - do expression unification as specified by filename
 -a filename      - do aliases replacement as specified by filename
 -cu filename     - do category unification as specified by filename
 -dc filename     - do dictionary check as specified by filename
```

frame_build

```
frame_build     frames_filename [-nor filename] -i filename [-o filename]
Options:
 frames_filename  - filename containing frame-system description
 -nor filename    - normalize frames according to filename
 -i filename      - utterance corpus filename to process
 -o filename      - utterance corpus filename to write to
                     if absent, output is directed to input file
```

corpus_stats

```
corpus_stats    filename [-o filename] [-sld dic_filename]
Options:
 filename          - utterance corpus filename to process
 -o filename       - statistics filename to write to
                      if absent, output is directed to stdout
 -sld dic_filename - compute semantic labels density (dic_filename
                       is the semantic labels dictionary)
```

corpus_filter

```
corpus_filter   filename [-o filename] [-keep|-discard regexp] [-nonor]
                [-noprs] [-nofrm]
Options:
 filename         - utterance corpus filename to filter
 -o filename      - utterance corpus filename to write to
                     if absent, output is directed to input file
 -keep regexp     - keep only utterances with names matching regexp
 -discard regexp  - discard utterances with names matching regexp
 -nonor           - eliminate NOR form from utterances
 -noprs           - eliminate PRS form from utterances
 -nofrm           - eliminate FRM form from utterances
```

corpus_diff

```
corpus_diff     -c1 filename -c2 filename -utt|-sro|-nor|-prs|-frm
                [-markdiff mark]
Options:
 -c1 filename     - first corpus filename
 -c2 filename     - second corpus filename
 -utt             - compare whole utterances
 -sro             - compare SRO forms
 -nor             - compare NOR forms
 -prs             - compare PRS forms
 -frm             - compare FRM forms
 -markdiff mark   - mark utterances that are different by appending
                       -<mark> to the name of the utterance
```

dict_build

```
dict_build      [-sro|-nor|-prs filename] [-i|-c filename] [-s filename]
Options:
 -sro filename    - extracts words from speech recognizer output
 -nor filename    - extracts words from normalized form
 -prs filename    - extracts words from parse
 -i filename      - increment dictionary
 -c filename      - create new dictionary
 -s filename      - log dictionary statistics
```