UNIVERSITATEA BABEŞ-BOLYAI, CLUJ-NAPOCA

FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

METODE FORMALE ÎN DEZVOLTAREA CORECTĂ A PROGRAMELOR

Rezumatul tezei de doctorat

CONDUCĂTOR ŞTIINŢIFIC:

PROF. DR. MILITON FRENŢIU

DOCTORAND:

MIHIŞ ANDREEA-DIANA

CLUJ-NAPOCA

2011

MULȚUMIRI

Mulţumesc pe această cale tuturor celor care mi-au fost alături în elaborarea acestei teze, în special Doamnei Prof. dr. Tătar Doina, Domnului Prof. dr. Militon Frenţiu, Domnilor Prof. dr. Ştefan Măruşter şi Prof. dr. ing. Ioan Salomie, Familiei Mihăilă, Familiei Mihiş, Colegilor de birou, catedră şi departament.

Această lucrare a fost suportată parţial din: Grant UBB tip TP nr. 2/2006, Grant CNCSIS TD 400/2007, Proiect CNMP 91-037/19.09.2007 CONTO, Proiect CNMP 42-117/2008 I–GLOB şi Proiect IDEI ID_2412/2009.

Cuprinsul rezumatului:

Cuprinsul tezei de doctorat:

I. Introducere

Scopul principal al dezvoltatorilor de soft este de a îndeplini cerinţele clienţilor, indiferent dacă aceste cerinţe se referă la funcţionalitate, timp sau bani. În general, se doreşte ca produsele realizate să fie de calitate superioară, dar principala cerinţă este legată de corectitudinea softului produs. Prin corectitudinea softului se înţelege calitatea lui de a-şi respecta specificaţiile. În anumite domenii, existenţa unui soft care să se comporte exact după cum era prevăzut este esenţială. Din păcate cerinţele clienţilor sunt de cele mai multe ori scrise în limbaj natural, limbaj ambiguu şi dificil de prelucrat.

O soluţie este eliminarea ambiguităţii cerinţelor, fapt posibil datorită metodelor formale. Metodele formale pot fi privite ca un mijloc formal de a descrie problema sau de a modela sistemul [Gab06]. Practic orice aplicare a matematicii în probleme de ingineria sistemelor reprezintă o metodă formală. Metodele formale pornesc de la o bază matematică precisă, din care sunt derivate metode de modelare şi analiză.

Prin metode formale se înţelege utilizarea notaţiilor, tehnicilor şi metodelor matematice în specificarea, proiectarea şi implementarea softului [Bur95, Cla96, Tho95], cu alte cuvinte, „Metodele formale sunt matematica informaticii” [Hol96].

Există mai multe tipuri de formalizări. Cea mai simplă este o utilizare a matematicii (notaţii, metode) pe alocuri la specificare, proiectare, implementare, testare sau întreţinerea produsului.

Următorul nivel de formalizare îl reprezintă limbajele formale de specificare, cum sunt limbajul Z [Abr80], Vienna definition method (VDM) [Jon86] sau Gist [Bal85]. Specificaţiile formale au o formă predefinită, utilizează notaţii matematice şi este posibilă şi verificarea consistenţei lor. Valoarea limbajelor formale de specificare este pe deplin recunoscută, deoarece deşi elaborarea specificaţiilor necesită timp, costul crescut al fazei de specificare, fază timpurie a ciclului de viaţă a unui produs soft, este recuperat ulterior, întrucât costul proiectării, dezvoltării şi întreţinerii scad în mod semnificativ [Sch05].

Totodată se pot deriva programe corecte prin rafinare în paşi succesivi [Dij75, Gri85, Mor90]. Rafinarea constă în aplicarea pas cu pas a unui set de reguli de rafinare. Procesul porneşte de la un program abstract şi se termină la obţinerea codului. Codul astfel obţinut îşi va respecta specificaţiile, nefiind necesare testări ulterioare.

Există însă şi metode formale utilizate în demonstrare automată şi verificarea consistenţei modelelor [Hei96, Rus95]. Acesta este nivelul superior de aplicare a metodelor formale.

Metodele formale s-au aplicat cu succes în specificare, verificare, demonstrare automată dar şi în generarea automată de teste.

După cum s-a putut observa, pentru a asigura faptul că atât clientul cât şi dezvoltatorul au înţeles acelaşi lucru din documentul de specificare a cerinţelor, acesta ar trebui scris utilizând un limbaj formal. Din păcate nu întotdeauna acest lucru este posibil. Dacă totuşi se doreşte conversia cerinţelor scrise în limbaj natural de către client în specificaţii formale, dezvoltatorii pot apela la ajutorul oferit de diverse unelte capabile să prelucreze limbajul natural.

Structura tezei. Teza este structurată în şase capitole (unul introductiv, unul care prezintă noţiuni fundamentale, trei conţinând contribuţii originale şi ultimul concluzii), are o bibliografie care include un număr de 219 de referinţe şi cinci anexe.

Capitolul 1 introduce contextul, motivaţiile şi scopul tezei, rezumă contribuţiile din aceasta şi furnizează o descriere pe scurt a ei.

Capitolul 2 prezintă succint noţiuni fundamentale de ingineria softului, metode formale, dar şi legătura existentă între metodele de prelucrare a limbajului natural şi metodele formale. Acest capitol conţine şi o scurtă prezentare a limbajului Z, dar şi a metodei de rafinare în paşi succesivi, singura care garantează obţinerea de programe care să-şi respecte specificaţia.

Capitolul 3 descrie contribuţiile în domeniul prelucrării limbajului natural. El este împărţit în trei subcapitole. În primul subcapitol sunt prezentate contribuţiile în dezambiguare şi verificarea relaţiei de implicaţie dintre două texte. Al doilea conţine contribuţiile în domeniul segmentării unui text, iar ultimul cele de extragere a sumarului dintr-un text.

Capitolul 4 realizează trecerea de la limbaj natural la specificaţii formale, prezentând diferite aspecte ale utilizării ontologiilor în acest sens. Sunt cuprinse aici problema potrivirii unei ontologii la un text scris în limbaj natural, a extragerii unei ontologii dintr-un text scris în limbaj natural utilizându-se arborele de analiză sintactică a unei propoziţii. De asemenea este analizat rolul dezambiguării elementelor dintr-o ontologie şi utilizării similarităţii pentru a îmbunătăţi calitatea unei ontologii. În final este folosită o ontologie pentru a prelucra cerinţele cu scopul de a obţine structuri cât mai apropiate de specificaţii formale.

Capitolul 5 prezintă diferite contribuţii în domeniul metodelor formale. Printre acestea se numără o aplicaţie care asistă elaborarea specificaţiilor Z, operarea cu ele şi transformarea lor în programe abstracte, o aplicaţie care asistă procesul de rafinare a programelor abstracte la cod, respectiv o aplicaţie care simplifică structurile condiţionale şi câteva considerente cu privire la reutilizarea specificaţiilor formale şi multiformalism.

Capitolul 6 furnizează concluziile studiilor desfăşurate pe parcursul elaborării tezei.

Cuvinte cheie: specificaţii, metode formale, corectitudine, prelucrarea limbajului natural, ontologii.

Contribuţiile originale introduse în această teză sunt conţinute în capitolele 2, 3, 4 şi 5 şi sunt următoarele:

- O dezbatere a aplicabilităţii metodelor de prelucrare a limbajului natural în prelucrarea cerinţelor [Mih08c] (subcapitolul 2.4).
- Un nou algoritm de dezambiguare [Tăt07a] (subcapitolul 3.1.1).
- Un nou algoritm de dezambiguare şi identificare a părţii de vorbire a cuvintelor [Mih07] (subcapitolul 3.1.2).
- Trei metode direcţionale de evaluare a relaţiei de consecinţă logică [Tăt07b, Tăt09a] (subcapitolul 3.1.3).
- O metodă de folosire a relaţiei de implicaţie logică dintre două texte pentru obţinerea unei ontologii [Mih08b] (subcapitolul 3.1.4).
- Trei metode logice de segmentare a unui text [Tăt08e, Tăt08b] (subcapitolul 3.2.1).
- Alte două variante ale algoritmului de segmentare logică [Mih08d] (subcapitolul 3.2.2).
- O metodă de segmentare genetică cu număr predefinit de segmente [Mih08a] (subcapitolul 3.2.3).
- O metodă de segmentare top-down bazată pe lanţuri lexicale [Tăt08b, Tăt08c, Tăt08d, Tăt09b] (subcapitolul 3.2.4).
- O metodă de sumarizare post segmentare [Tăt08e, Tăt08b] (subcapitolul 3.3.1).

- Trei metode de extragere a unui sumar de lungime variabilă [Tăt08d] (subcapitolul 3.3.2).
- O metodă de sumarizare pe lanţuri lexicale [Tăt08b, Tăt08c, Tăt08d, Tăt09b] (subcapitolul 3.3.3).
- Mai multe metrici pentru evaluarea gradului în care o ontologie corespunde unui text de limbaj natural [Mih10b] (subcapitolul 4.1).
- Două metode de prelucrare a arborelui de analiză sintactică a unei propoziţii şi de extragere automată a tripletelor ontologice [Mih10c] (subcapitolul 4.2).
- Un algoritm de identificare a elementelor plasate greşit dintr-o ontologie [Mih10d] (subcapitolul 4.3).
- O dezbatere cu privire la necesitatea utilizării dezambiguării în evaluarea ontologiilor [Mih10e] (subcapitolul 4.4).
- O metodă de extragere a specificaţiilor formale dintr-un text reprezentând cerinţele [Mih10f] (subcapitolul 4.5).
- Un studiu privind continuitatea activităţii de asigurare a calităţii softului [Şer05] (subcapitolul 5.1).
- O dezbatere pe marginea necesităţii abordării multiformale a specificării produselor soft complexe [Cio04] (subcapitolul 5.2).
- O analiză a refolosirii specificaţiilor formale [Mih05] (subcapitolul 5.3).
- O unealtă care permite rafinarea asistată a schemelor Z [Mih10a] (subcapitolul 5.4).
- O unealtă care asistă procesul de rafinare a programelor abstracte în programe cod [Mih06a] (subcapitolul 5.5).
- O unealtă care asistă în mod automat simplificarea expresiilor condiţionale [Mih06b, Lup08, Lup09] (subcapitolul 5.6).

II. Metode formale în dezvoltarea programelor

2.1. Ciclul de viaţă a unui produs soft

Procesul de dezvoltare a softului parcurge un „ciclu de viaţă” caracterizat de trei momente importante, numite faze ale ciclului: definiţia, dezvoltarea şi exploatarea.

2.2. Specificarea cerinţelor

Specificarea cerinţelor este ultima fază a procesului de analiză a cerinţelor. Rezultatul analizei cerinţelor este **Documentul de specificare a cerinţelor softului** care reprezintă contractul client-dezvoltator.

2.2.1 Specificaţii informale

În multe proiecte de dezvoltare, documentul de specificare constă în pagină după pagină scrise în limbaj natural, care însă este ambiguu.

2.2.2 Tehnici semiformale de specificare

Există mai multe tipuri de tehnici semiformale de specificare: **tehnici grafice orientate pe acţiuni** [DeM78, Gan79, You79], **tehnici grafice orientate pe date** [Che76] şi **alte tehnici semiformale de specificare a cerinţelor** (PLS/PLA [Tei77], SADT [Ros85], SREM [Alf85]).

2.2.3 Tehnici formale de specificare

Dintre tehnicile formale de specificare se pot aminti: **Maşini cu stări finite** [Kam87], **Reţele Petri** [Pet62] şi **limbajul Z** [Abr80]. O specificaţie Z este compusă din patru părţi: mulţimea datelor de intrare, tipurile datelor şi constantele, definiţii de stări, starea iniţială şi operaţiile. **Alte tehnici formale** sunt: Anna [Luc85], Gist [Bal85], VDM [Jon86] şi CSP [Hoa85].

2.3. Rafinarea în paşi succesivi

Primul care a propus o metodă care asigură corectitudinea programelor a fost Floyd [Flo67]. În acelaşi timp s-a considerat mult mai important să se scrie programe corecte prin construcţie. În această direcţie un rol deosebit l-au avut Dijkstra [Dij75], Gries [Gri81], Dromey [Dro89] şi Morgan [Mor90]. Ideea lui Dijkstra, de a deriva în mod formal programe din specificaţii a fost continuată apoi de Gries [Gri81], care consideră că este mai important să se dezvolte programe corecte decât să se demonstreze ulterior corectitudinea lor [Fre06]. Pentru a se obţine programe corecte, specificaţiile trebuie transformate în cod folosind un set de reguli bine definite.

2.4. Metode de procesare a limbajului natural utilizate în analiza cerinţelor

La conceperea unui produs nou, clienţii şi dezvoltatorii se întâlnesc pentru a specifica cerinţele noului sistem. Pentru a nu omite anumite cerinţe, se folosesc diferite **tehnici de colectare a cerinţelor**. Astfel, se generează un volum mare de cerinţe exprimate în limbaj natural. Prelucrarea lor poate beneficia de pe urma folosirii unor **mecanisme de prelucare a limbajului natural**, după cum se poate observa din figura 1:

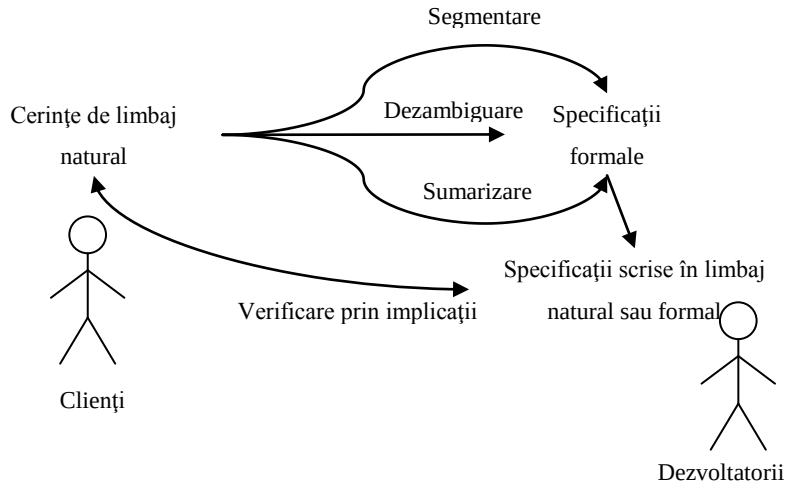

Figura 1. Schema utilizării mecanismelor de prelucrare a limbajului natural în analiza cerinţelor

2.5. Ontologiile

O ontologie este „O specificare explicită a unei conceptualizări” [Gru93]. Astăzi coloana vertebrală a Internetului Semantic se consideră a fi OWL (Web Ontology Language) şi RDF (Resource Description Framework) [Pol09]. Ele reprezintă cele mai bune limbaje de modelare a unei ontologii. Cel mai general mod de definire a unei ontologii este utilizând triplete [Bra85]. Un triplet e compus dintr-un *concept*, un *predicat* care reprezintă o relaţie direcţională spre *obiect*, care de cele mai multe ori este o caracteristică a conceptului. Pentru uniformitate, toate componentele ontologiei conceptele, relaţiile şi obiectele sunt reprezentate de un URI (Uniform Resource Identifiers) [Ber98].

III. Contribuţii în domeniul prelucrării limbajului natural

3.1. Contribuţii în dezambiguarea automată a unui text

Pentru a putea dezambigua rapid specificaţiile scrise în limbaj natural de către clienţi, este nevoie de o unealtă capabilă de a identifica automat sensul cuvintelor. Cum este de aşteptat, când se urmăreşte identificarea sensului corect al cuvintelor, cea mai de încredere sursă este dicţionarul.

3.1.1 Identificarea sensului cuvintelor

Una dintre cele mai cunoscute metode de dezambiguare bazată pe dicţionar este cea a lui Lesk [Les86]. În [Ban03] algoritmul lui Lesk a fost dezvoltat utilizând dicţionarul de limbă engleză WordNet [***WNt]. Algoritmul dezvoltat de Banerjee şi Pedersen adaugă definiţiei din WordNet sinonime, exemple şi definiţii ale cuvintelor înrudite. Totodată se introduce o nouă măsură „overlap" care favorizează intersecţiile multiple de cuvinte.

Algoritmul original CHAD [Tăt07a] seamănă cu algoritmul prezentat în [Ban03], doar că dezambiguează toate cuvintele unui text, printr-o singură parcurgere a textului şi nu doar un cuvânt ţintă. Algoritmul se aplică în mod repetat pe grupuri de trei cuvinte, dintre care primele două au fost deja dezambiguate, iar al treilea urmează să fie dezambiguat. Astfel putem privi algoritmul ca pe un lanţ, ale cărui ochiuri sunt reprezentate de grupurile de trei cuvinte. Primele trei cuvinte sunt însă dezambiguate simultan.

Algoritmul foloseşte trei măsuri pentru calculul scorului tripletului de sensuri $s^i_{w_1}, s^j_{w_2}, s^k_{w_3}$ al celor trei cuvinte w_1, w_2, w_3: Dice [Dic45], overlap şi Jaccard [Jac901]. Aceste trei măsuri evaluează în mod diferit numărul de cuvinte comune existente în definiţiile corespunzătoare sensurilor $s^i_{w_1}, s^j_{w_2}, s^k_{w_3}$ ale celor trei cuvinte w_1, w_2, respectiv w_3, definiţii notate cu $D_{w_1}, D_{w_2}, D_{w_3}$:

$$\text{Dice: scor}(s^i_{w_1}, s^j_{w_2}, s^k_{w_3}) = 3*\frac{|D_{w_1} \cap D_{w_2} \cap D_{w_3}|}{|D_{w_1}|+|D_{w_2}|+|D_{w_3}|}$$

$$\text{overlap: scor}(s^i_{w_1}, s^j_{w_2}, s^k_{w_3}) = \frac{|D_{w_1} \cap D_{w_2} \cap D_{w_3}|}{\min(|D_{w_1}|,|D_{w_2}|,|D_{w_3}|)}$$

$$\text{Jaccard: scor}(s^i_{w_1}, s^j_{w_2}, s^k_{w_3}) = \frac{|D_{w_1} \cap D_{w_2} \cap D_{w_3}|}{|D_{w_1} \cup D_{w_2} \cup D_{w_3}|}$$

Incluzând subalgoritmul de dezambiguare a unui triplet de cuvinte, se obţine algoritmul CHAD prezentat în figura 2.

Deoarece deseori intersecţia definiţiilor celor trei cuvinte este egală cu 0, există cuvinte ale căror scoruri sunt egale cu 0 pentru toate sensurile lor. Acestor cuvinte li s-a atribuit primul sens din WordNet, dat fiind faptul că în WordNet sensurile cuvintelor sunt ordonate după frecvenţa lor.

Algoritmul a fost testat pe zece texte alese în mod aleator din corpusul Brown [***BrC, Nel79] (A01, A02, A11, A12, A13, A14, A15, B13, B20 şi C01) care au deja identificate părţile de vorbire ale cuvintelor. Astfel rezultatele au putut fi comparate cu cele

din corpusul SemCor [***Sem], unde pe lângă partea de vorbire, este indicat şi sensul cuvântului. Algoritmul CHAD a fost rulat prima dată pentru substantive, apoi doar pentru verbe şi a treia oară pentru substantive, verbe, adjective şi adverbe.

Algoritmul *CHAD (W, n, S)* ***este:***
date *W {un text format din cuvintele* $w_1,w_2,...,w_n$*}, n {numărul cuvintelor din text}*
@Dezambiguează_triplet (w_1,w_2,w_3, $s^*_{w_1}, s^*_{w_2}, s^*_{w_3}$ *)*
Pentru $i \leftarrow 4,n$ ***execută***
$p \leftarrow 1$
$max \leftarrow scor(\ s^*_{w_{i-2}}, s^*_{w_{i-1}}, s^1_{w_i}\ )$
{sensurile identificate deja pentru cuvintele w_{i-2} *şi* w_{i-1} *sunt* $s^*_{w_{i-2}}, s^*_{w_{i-1}}$ *}*
Pentru *fiecare sens* $s^m_{w_i}$ ***execută***
Dacă $max < scor(\ s^*_{w_{i-2}}, s^*_{w_{i-1}}, s^m_{w_i}\ )$
atunci $p \leftarrow m$
$max \leftarrow scor(\ s^*_{w_{i-2}}, s^*_{w_{i-1}}, s^m_{w_i}\ )$
sfd
sfp
$s^*_{w_i} \leftarrow s^p_{w_i}$
sfp
rezultate *S {sensurile* $s^*_{w_i}, i = \overline{1,n}$ *}*
sf *CHAD*

Figura 2. Algoritmul CHAD

Chiar dacă precizia nu depăşeşte precizia primului sens din WordNet (diferenţa medie pentru substantive fiind 0,0338, pentru verbe de 0,0340, iar pentru toate cuvintele 0,0491), aceste rezultate se încadrează în rezultatele obţinute până acum unde la a doua competiţie Senseval [***Sen], doar două din şapte echipe (cu metode nesupervizate) au obţinut rezultate mai bune faţă de primul sens din WordNet.

Algoritmul CHAD nu este dependent de limba în care e scris textul, el putând fi aplicat cu succes şi pentru limba română. Acest algoritm poate fi aplicat pentru a îmbunătăţi traducerea [Tăt07a], dar şi pentru a identifica părţile de vorbire [Mih07] sau pentru a verifica relaţia de implicaţie dintre două texte.

3.1.2 Identificarea părţii de vorbire a cuvintelor

În momentul în care o persoană întâlneşte un cuvânt necunoscut, mai ales într-o limbă pe care nu o cunoaşte suficient a cărei gramatică nu şi-a însuşit-o încă, va încerca să caute cuvântul respectiv în dicţionar, să-i afle sensul corespunzător contextului. Dar unele cuvinte constituie mai multe părţi de vorbire diferite, depinzând de context. În cazul în care cuvântul căutat are astfel de caracteristici, i se va identifica şi partea de vorbire corespunzătoare contextului. Acest fapt a stat la baza unui nou algoritm de identificare simultană a părţilor de vorbire şi a sensului cuvintelor derivat din algoritmul CHAD aflat în figura 3 [Mih07].

Experimentele s-au efectuat pe aceleaşi zece texte din corpusul Brown. Deoarece în SemCor, care de asemenea a fost folosit ca standard pentru evaluare, există cuvinte a căror parte de vorbire nu este identificată (Notag), acele cuvinte nu au fost considerate în calculul preciziei metodei. Valorile minime, maxime şi medii se află în tabelul 1.

Metoda este însă promiţătoare în ciuda rezultatelor care se află sub rezultatele obţinute utilizând mai multe resurse şi reguli gramaticale, deoarece ea poate fi aplicată pentru mai multe limbi şi se realizează în acelaşi timp şi dezambiguarea cuvintelor.

Algoritmul *PV_CHAD (W, n, S, P)* ***este***
date *W {un text format din cuvintele $w_1,w_2,...,w_n$}, n {numărul de cuvinte din text}*
@Dezambiguează_şi_identifică_PV_triplet (w_1, w_2, w_3, $s^{*}_{1\ p_1^{*w_1}}$, $s^{*}_{2\ p_2^{*w_2}}$, $s^{*}_{3\ p_3^{*w_3}}$)
Pentru *i ← 1,n* ***execută***
r ← prima parte de vorbire din dicţionar corespunzătoare cuvântului w_i
k ← 1 {primul sens al primei părţi de vorbire a cuvântului w_i}
max ← scor($s^{*}_{i-2\ p^{*}_{i-2}{}^{w_{i-2}}}$, $s^{*}_{i-1\ p^{*}_{i-1}{}^{w_{i-1}}}$, $s^{1}_{i\ r^{w_i}}$)
Pentru *fiecare parte de vorbire q a cuvântului w_i* ***execută***
Pentru *fiecare sens $s^{m}_{i\ q^{w_i}}$ al cuvântului w_i cu partea de vorbire q* ***execută***
Dacă *max< scor($s^{*}_{i-2\ p^{*}_{i-2}{}^{w_{i-2}}}$, $s^{*}_{i-1\ p^{*}_{i-1}{}^{w_{i-1}}}$, $s^{m}_{i\ q^{w_i}}$)*
atunci *r ← q*
k ← m
max ← scor($s^{*}_{i-2\ p^{*}_{i-2}{}^{w_{i-2}}}$, $s^{*}_{i-1\ p^{*}_{i-1}{}^{w_{i-1}}}$, $s^{m}_{i\ q^{w_i}}$)
sfd
sfp
sfp
$p^{*}_{i} \leftarrow r$
$s^{*}_{i\ p_i^{*w_i}} \leftarrow s^{k}_{i\ r^{w_i}}$
sfp
rezultate *S {sensurile $s^{*}_{i}, i \leftarrow 1,n$}, P { părţile de vorbire $p^{*}_{i}, i \leftarrow 1,n$ }*
sf *PV_CHAD*

Figura 3. Algoritmul PV_CHAD

Precizia	Valoarea minimă	Valoarea maximă	Valoarea medie
Identificarea tuturor părţilor de vorbire	20,16%	64,13%	42,78%
Dezambiguarea tuturor cuvintelor	7,00%	41,43%	21,78%
Substantive	53,04%	82,83%	66,12%
Verbe	67,83%	92,02%	84,45%
Adjective	29,55%	75,34%	56,39%
Adverbe	46,87%	80,65%	69,41%

Tabelul 1. Valorile minimă, maximă şi medie ale preciziei în funcţie de părţile de vorbire.

Acest algoritm poate fi folosit cu succes pentru identificarea automată a părţilor de vorbire din cerinţe, astfel se pot identifica mai uşor entităţile şi relaţiile pentru o elaborare a unui model Entitate-Relaţie.

3.1.3 Relaţia de consecinţă logică dintre două texte

Principalele raţionamente care se efectuează relativ la un text sau un grup de texte se bazează pe demonstrarea sau verificarea relaţiei de consecinţă logică sau implicaţia dintre două texte. Relaţia de implicaţie logică dintre două texte, *T* textul şi *H* ipoteza, reprezintă un fenomen fundamental al limbajului natural. Ea se notează cu $T \rightarrow H$ şi înseamnă ceea ce se înţelege din *H* poate fi dedus din *T*.

Chiar dacă problema nu e nouă, majoritatea abordărilor automate sunt rezultatul direct al competiţiilor Pascal Textual Entailment Challenges [***RTE]. Cu toate acestea,

doar puţini autori au folosit caracterul direcţional al acestei relaţii: dacă are loc $T \rightarrow H$ este puţin probabil să aibă loc şi $H \rightarrow T$. Dintre metodele propuse la competiţia RTE-1, metoda cu cele mai bune rezultate a fost cea direcţională a lui Glickman [Gli05] 58,5%, iar metoda direcţională a lui Kŏuylekov [Kŏu06] a obţinut 56%.

Prima metodă de evaluare a relaţiei de consecinţă logică propusă în [Tăt07b] utilizează similaritatea textelor. În [Cor05] se defineşte următoarea măsură a similarităţii dintre două texte, similaritatea dintre textele T_i şi T_j în raport cu T_i:

$$sim(T_i, T_j)_{T_i} = \frac{\sum_{pos}(\sum_{w_k \in WS_{pos}^{T_i}}(\max Sim(w_k) \times idf_{w_k}))}{\sum_{pos}\sum_{w_k \in WS_{pos}^{T_i}} idf_{w_k}} \qquad (3.1.3.1)$$

Mulţimea cuvintelor din textul T_i care au o anumită parte de vorbire *pos* se notează cu $WS_{pos}^{T_i}$. Pentru un cuvânt w_k cu o anumită parte de vorbire *pos* în T_i, cea mai mare similaritate a cuvintelor cu aceeaşi parte de vorbire *pos* din celălalt text T_j, se notează cu $maxSim(w_k)$. Deoarece ipoteza H conţine mai puţină informaţie faţă de textul T, pentru o implicaţie între cele două texte va avea loc următoarea relaţie [Tăt09a]:

$$sim(T,H)_T < sim(T,H)_H \qquad (3.1.3.2)$$

Acestă formulă este aplicată textelor dezambiguate cu algoritmul CHAD prezentat în subcapitolul anterior. În formula (3.1.3.1) s-au folosit doar substantivele şi verbele ca şi părţi de vorbire *pos*, iar similaritatea dintre două cuvinte s-a considerat a fi 1 dacă cele două cuvinte sunt identice sau sunt sinonime şi 0 altfel. Totodată s-au folosit câteva reguli de identificare a falselor implicaţii, notate cu *COND*.

Pentru a se uşura scrierea algoritmului de verificare a relaţiei de consecinţă logică bazat pe similaritatea textelor, vom nota textul T cu T_1, iar ipoteza H cu T_2 şi vom nota cu: NP_1 şi NP_2 – numele proprii din T_1 respectiv T_2, I_c – cuvintele comune care nu sunt nume proprii din T_1 şi T_2, $SYN(T_1)_{T_2}$ – cuvinte din T_1 care nu sunt nume proprii, nu apar în T_2, dar sunt substantive sau verbe sinonime cu unele cuvinte din T_2, analog $SYN(T_2)_{T_1}$, $C_1 = |SYN(T_1)_{T_2}|$, $C_2 = |SYN(T_2)_{T_1}|$, $W_{T_1} = NP_1 \cup I_c$, $W_{T_2} = NP_2 \cup I_c$.

Din (3.1.3.1) şi (3.1.3.2) se obţine următoarea condiţie: $C_1 \leq C_2$, adică $|SYN(T_1)_{T_2}| \leq |SYN(T_2)_{T_1}|$. Relaţia nu este strictă datorită definiţiei mulţimilor $SYN(T_1)_{T_2}$ şi $SYN(T_2)_{T_1}$.

Funcţia *Consecinţă_logică (T_1,T_2)* ***este***
  date *T_1, T_2 {două texte}*
    dacă $W_{T_1} \subseteq W_{T_2}$
      atunci dacă $T_2 = NP_2 \cup I_c$
        atunci dacă *@ se verifică toate regulile din COND*
          atunci *Consecinţă_logică* ← *False {nu are loc $T_1 \rightarrow T_2$}*
          altfel *Consecinţă_logică* ← *True { cazul I: $T_1 \rightarrow T_2$ }*
        altfel dacă $C_1 \leq C_2$
          atunci *Consecinţă_logică* ← *True { cazul II: $T_1 \rightarrow T_2$ }*
          altfel *Consecinţă_logică* ← *False {nu are loc $T_1 \rightarrow T_2$}*
      altfel *Consecinţă_logică* ← *False {nu are loc $T_1 \rightarrow T_2$}*
  rezultate *True {are loc $T_1 \rightarrow T_2$}, respectiv False {nu are loc $T_1 \rightarrow T_2$}*
sf *Consecinţă_logică*

Figura 4. Funcţia de verificare a relaţiei de consecinţă logică pe baza similarităţilor

Un caz important este acela în care T_2 conţine doar nume proprii care apar şi în T_1, de aceea algoritmul de verificare a relaţiei de consecinţă logică pe baza similarităţilor va începe cu verificarea acestei condiţii, vezi figura 4.

Aplicaţia construită pornind de la acest algoritm necesită ca intrare texte în care s-au identificat părţile de vorbire, inclusiv numele proprii. Algoritmul a fost evaluat folosind baza de 800 de perechi de texte din competiţia RTE-1 [***RTE].

Precizia globală a algoritmului propus este de 52,625%, iar cea medie 56,60647%.

A doua metodă de evaluare a relaţiei de consecinţă logică se bazează pe măsura cosinus [Tăt09a]. S-au definit trei măsuri cosinus pentru a evalua distanţa dintre vectorii de cuvinte $T = t_1,t_2,...,t_m$ şi $H = h_1,h_2,...,h_n$. Prima măsură cosinus, $\cos_T(T,H)$ măsoară distanţa dintre vectorul m-dimensional $\vec{T} = (1,1,...,1)$ şi vectorul m-dimensional $\vec{H}$ care reprezintă proiecţia lui H pe T, adică $\vec{H}_i = 1$, dacă t_i este cuvânt din H şi $\vec{H}_i = 0$ altfel. A doua măsură cosinus, $\cos_H(T,H)$ măsoară distanţa dintre vectorul n-dimensional $\vec{H} = (1,1,...,1)$ şi vectorul n-dimensional $\vec{T}$ care reprezintă proiecţia lui T pe H, adică $\vec{T}_i = 1$, dacă h_i este cuvânt din T şi $\vec{T}_i = 0$ altfel. Pentru ultima măsură cosinus, $\cos_{H\cup T}(T,H)$, primul vector se obţine înlocuind cuvintele din $H\cup T$ conţinute de T cu 1 şi restul cu 0, iar al doilea prin înlocuirea cuvintelor din $H\cup T$ conţinute de H cu 1 şi cele din $T\backslash H$ cu 0.

Dacă notăm cu c numărul de cuvinte comune celor doi vectori, cele trei măsuri sunt: $\cos_T(T,H) = \sqrt{\frac{c}{m}}$, $\cos_H(T,H) = \sqrt{\frac{c}{n}}$ şi $\cos_{H\cup T}(T,H) = \sqrt{\frac{4c^2}{(n+c)(m+c)}}$.

Pentru a se îndeplini condiţia: *T implică H dacă şi numai dacă H conţine mai puţină informaţie decât T* [Mon01], similarităţile dintre T şi H calculate în raport cu T respectiv $H\cup T$ trebuie să fie foarte apropiate. În mod analog similarităţile dintre T şi H calculate în raport cu H respectiv $H\cup T$ trebuie să fie foarte apropiate. De asemenea toate cele trei similarităţi (cosinusuri) trebuie să fie mai mari decât o anumită limită. Deci trebuie respectate următoarele condiţii: $|\cos_{H\cup T}(T,H) - \cos_T(T,H)| \le \tau_1$, $|\cos_H(T,H) - \cos_{H\cup T}(T,H)| \le \tau_2$ şi $\max\{\cos_T(T,H), \cos_H(T,H), \cos_{H\cup T}(T,H)\} \ge \tau_3$. Valorile identificate pentru cele trei limite sunt: $\tau_1 = 0{,}095$, $\tau_2 = 0{,}15$ şi $\tau_3 = 0{,}7$.

În urma evaluării rezultatelor obţinute s-a observat că precizia pentru perechile adevărate este 68,92230576%, iar pentru cele false este de 46,36591479%. Precizia globală este de 57,62%.

A treia metodă originală se bazează pe o variantă modificată a distanţei Levenshtein [Tăt09a]. Notăm cu $LD(w_1,w_2)$ distanţa Levenshtein modificată dintre cuvintele w_1 şi w_2. Această distanţă se defineşte ca numărul minim de transformări (ştergeri, inserări şi substituţii) astfel încât cuvântul w_1 *se transformă în cuvântul* w_2. Notăm cu T_{word} „cuvântul” obţinut din propoziţia T, unde spaţiul se consideră a fi o literă nouă şi toate cuvintele lui T sunt juxtapuse. În mod analog se obţine „cuvântul” H_{word}. Se consideră că $LD(T_{word},H_{word})$ reprezintă caracteristica informativă a lui H în raport cu T. Această distanţă Levenshtein nu e o distanţă reală, deoarece $LD(w_1,w_2) \ne LD(w_2,w_1)$.

Deoarece *T implică H dacă şi numai dacă H conţine mai puţină informaţie decât T*, atunci ar trebui să se verifice următoarea relaţie: $LD(T_{word},H_{word}) < LD(H_{word},T_{word})$. Aplicând acest criteriu celor 800 de perechi din baza de texte RTE-1, s-a obţinut precizia globală de 53,19%.

Costurile *transformării cuvântului* w_1 *în cuvântul* w_2 sunt următoarele: transformarea literei mari în mici sau invers 1, costul de inserare 3, costul ştergerii 3, costul substituţiei 5, costul interschimbării 2.

Rezultatele obţinute se încadrează între rezultatele obţinute la ultimele competiţii RTE, metoda a doua fiind cea mai bună. Pe lângă faptul că a decide dacă un text implică un alt text este de folos şi în această formă dezvoltatorilor, acestă relaţie a fost folosită cu succes şi pentru a segmenta şi sumariza un text, care de asemenea uşurează sarcina de

analiză şi prelucrare a specificaţiilor scrise în limbaj natural, după cum se va putea observa din secţiunile următoare.

3.1.4 O simplă ontologie bazată pe relaţia de consecinţă logică dintre două texte

În [Mih08b] s-a prezentat un mod de a crea o ontologie informală bazată pe relaţia direcţională de consecinţă logică dintre texte. S-a folosit metoda cu cel mai bun scor de identificare a implicaţiilor dintre texte, metoda cosinusurilor, prezentată în subcapitolul anterior.

Pornindu-se de la un text scris în limbaj natural, se verifică toate relaţiile de consecinţă logică existente între propoziţiile textului. Astfel se obţine un graf direcţional între propoziţiile textului. Graful poate reprezenta o ontologie informală simplă a textului respectiv în care entităţile sunt reprezentate de propoziţii, iar relaţiile dintre entităţi de relaţia de consecinţă logică.

Aceeaşi relaţie de consecinţă logică poate fi de asemenea folosită pentru a relaţiona texte întregi, obţinându-se un graf al relaţiilor dintre texte. Dacă respectivele texte reprezintă definiţiile entităţilor, atunci graful relaţiilor dintre entităţi poate fi graful relaţiilor de consecinţă logică.

3.2. Noi abordări ale segmentării unui text

În cazul implementării unui sistem informatic de mari dimensiuni volumul cerinţelor va fi de asemenea mare. Astfel prelucrarea sa va necesita munca în paralel a mai multor persoane sau echipe. Bineînţeles se impune împărţirea specificaţiilor după funcţionalităţi, ceea ce corespunde cu operaţia de segmentare specifică limbajului natural.

3.2.1 Segmentare liniară

În [Tăt08e, Tăt09c] s-a propus o metodă de segmentare liniară a unui text folosindu-se relaţia de consecinţă logică prezentată în subcapitolul anterior. Cel mai simplu algoritm de segmentare utilizând implicaţiile se numeşte „PureEntailment” (PE), unde un nou segment se consideră că începe în momentul în care ultimul segment nu mai implică noua propoziţie.

O a doua abordare a segmentării, Logical TextTiling (LTT) [Tăt08e, Tăt09c], este asemănătoare cu algoritmul de detectare a structurii subtemei TextTiling (TT) [Hea93], dar LTT detectează structura logică a textului. Principalele diferenţe sunt:

- În LTT unităţile de bază sunt propoziţiile, iar în TT sunt pseudopropoziţiile, toate având aceeaşi dimensiune predefinită;
- În TT fiecare limită dintre pseudopropoziţii se scorează cu similaritatea pseudopropoziţiilor de la *i*-*k* la *i* faţă de pseudopropoziţiile de la *i*+1 la *i*+*k*+1. În LTT se scorează propoziţiile, fiecare propoziţie s_i primind ca şi scor $scor(s_i)$ numărul de propoziţii din întregul text implicate de s_i. O propoziţie importantă va avea un scor mare, ceea ce este în concordanţă cu criteriul implicaţiei textelor: *Un text T implică un alt text H dacă şi numai dacă conţine mai multă informaţie decât H* [Tăt07b].

În LTT o limită dintre două segmente, determinată de schimbarea subiectului discursului este cauzată de schimbarea configuraţiei importanţei logice a propoziţiilor (vezi figura 5). Astfel există o bază fundamentală de descompunere a discursului într-o mulţime de segmente mai mici, fiecare cu o configuraţie diferită a importanţei propoziţiilor componente.

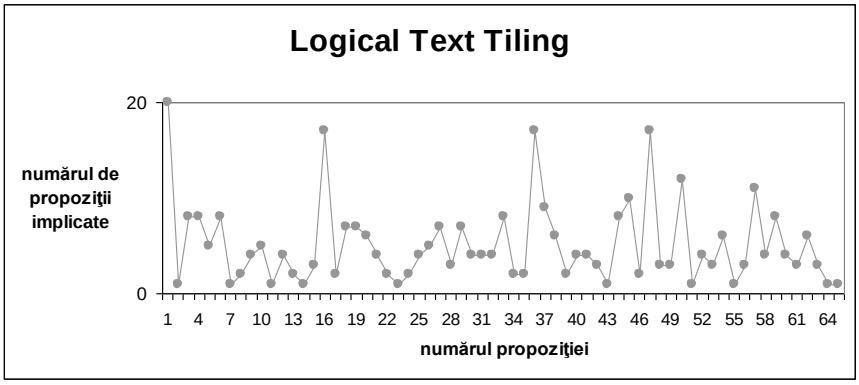

Figura 5. Structura logică a textului

Spre deosebire de alte metode de segmentare, LTT este o metodă liniară:

Algoritmul *LTT(S, n, SC, SEG)* ***este***
  date *S {un text, s_i – propoziţiile textului, i←1,n}, n {numărul de propoziţii din text},*
    SC {sci = scor(si) scorurile propoziţiilor din text}
  j ← 1, p ← 1, iniţializează(Seg_1, s_1) {iniţial segmentul Seg_1 conţine doar propoziţia s_1}
  dir ← „urcă”
  Câttimp *p<n* ***execută***
    Dacă *$sc_{p+1} > sc_p$*
      atunci Dacă *dir = „urcă”*
        atunci *adaugă(Seg_j, s_{p+1}) {adaugă segmentului Seg_j propoziţia s_{p+1}}*
        altfel *j ← j+1*
          iniţializează(Seg_j, s_{p+1})
          dir ← „urcă”
        sfd
    altfel *adaugă(Seg_j, s_{p+1})*
        Dacă *scor(s_{p+1}) < scor(s_p)*
          atunci *dir ← „coboară”*
        sfd
    sfd
    p ← p + 1
  sfc
  rezultate *SEG={Seg_1,...,Seg_j}*
sf *LTT*

Având ca punct de pornire măsurarea importanţei unei propoziţii în funcţie de numărul de alte propoziţii cu care aceasta se află în relaţie (sunt implicate de ea), s-au propus două noi variante de determinare a scorului propoziţiilor:

$$ArcInt_k = \sum_{i=1}^{k} \sum_{j=k, i\neq j}^{n} implică_{i,j} + \sum_{i=k}^{n} \sum_{j=i, i\neq j}^{k} implică_{i,j}$$

$$ArcReal_k = \sum_{i=1}^{k} \sum_{j=k, i\neq j}^{n} \frac{implică_{i,j}}{|i-j|} + \sum_{i=k}^{n} \sum_{j=i, i\neq j}^{k} \frac{implică_{i,j}}{|i-j|}$$

unde

$$implică_{i,j} = \begin{cases} 1, & dacă\ s_i \rightarrow s_j \\ 0, & altfel \end{cases}.$$

O altă abordare a problemei segmentării este cea a aplicării unui algoritm de programare dinamică. Această metodă realizează mai întâi o sumarizare (vezi subcapitolul 3.3.1). Limitele dintre segmente sunt acele propoziţii care au un scor minimal şi sunt situate între două propoziţii din sumar (vezi figura 6). Astfel numărul de segmente obţinute este corelat cu lungimea *x* a sumarului.

Pentru a evalua metodele de segmentare propuse s-a utilizat textul literar, „The Koan” [Ric91], text care a fost segmentat, sumarizat şi i s-au rezolvat anaforele în mod manual, pe baza criteriilor lingvistice de către un specialist [Tăt08e].

Conform cu [Hea93], compararea cu segmentarea manuală constă în a evalua:

- câte dintre limitele găsite de metodă sunt limite manuale exacte (sau sunt foarte apropiate de ele) (precizia);
- câte dintre limitele adevărate au fost identificate (recall-ul).

Totodată s-a folosit şi măsura $F = \frac{2 * Precizia * Recall}{Precizia + Recall}$.

Algoritmul *Segmentare_post_sumarizare(S, n, SC, P, x, SEG, nrseg)* ***este:***
date *S {un text, s_i – propoziţiile textului, i←1,n}, n {numărul de propoziţii din text}, SC {sc_i = scor(s_i) scorurile propoziţiilor din text}, P {p_i – poziţiile din text ale propoziţiilor sumarului, i←1,x}, x {numărul de propoziţii din sumar}*
nrseg ← 1
begpoz ← 1
Pentru *i ← 1, x-1* ***execută***
endpoz ← $arg\,min_{j, j\in\{p_i, p_{i+1}-1\}} scor(s_j)$
@Seg_{nrseg} începe cu begpoz şi se termină pe endpoz
nrseg ← nrseg + 1
begpoz ← endpoz + 1
sfp
rezultate *SEG {Seg_1,...,Seg_{nrseg} – lista segmentelor}, nrseg {numărul segmentelor}*
sf *Segmentare_post_sumarizare*

Figura 6. Algoritmul Segmentare post sumarizare

Câteva dintre rezultatele evaluării comparative a diferitelor metode de segmentare (Logică/Programare dinamică) se află în figura 7. Dintre metodele logice, cele mai bune rezultate le-a obţinut LTT.

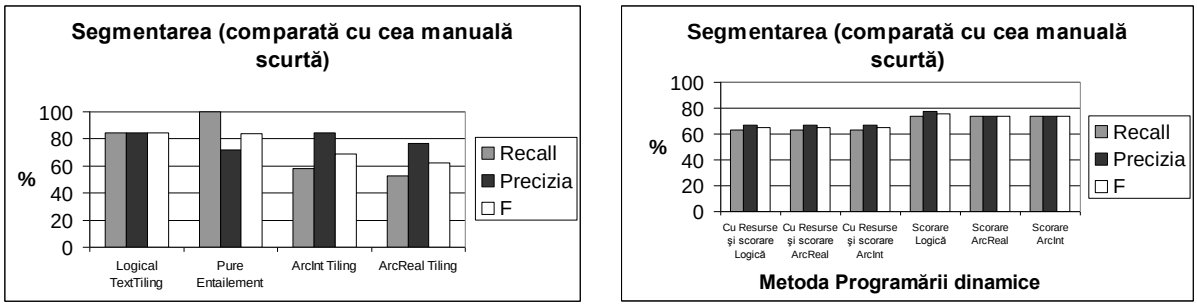

Figura 7. Evaluarea segmentării

3.2.2 Alte variante de segmentare Logical TextTiling

Metoda Logical TextTiling necesită o scorare în prealabil a fiecărei propoziţii, ceea ce conduce la o complexitate de ordinul n^2. Pentru a reduce complexitatea s-a propus un LTT pe vecinătăţi, în care fiecare propoziţie va fi scorată într-o vecinătate de dimensiune predefinită, nu din întregul text şi respectiv un LTT ponderat care face prioritare implicaţiile locale pentru a aplatiza graful care reprezintă structura logică a textului [Mih08d]. Structura logică a textului obţinută în prima variantă generează aceleaşi segmente ca şi LTT-ul pentru o vecinătate de minim 20% din lungimea textului iniţial (pentru textul „The Koan” [Ric91]). Şi structura logică a textului obţinută din a doua variantă generează aceleaşi segmente ca şi LTT-ul. În consecinţă cele două variante de LTT pot înlocui cu succes metoda originală LTT.

3.2.3 Segmentare genetică cu număr predefinit de segmente bazată pe implicaţii

Metoda următoare foloseşte interpretarea logică a textului, adică relaţia de consecinţă logică dintre două propoziţii pentru a realiza acest scop: o coeziune internă ridicată a segmentelor şi o conexiune scăzută între segmente vecine [Mih08a].

Dacă pentru metodele anterioare nu se poate impune numărul de segmente care se vor obţine, metoda propusă în cele ce urmează va împărţi textul într-un număr impus k de segmente. Prin urmare dacă avem textul $T=\{t_1, t_2,..., t_n\}$ care conţine n propoziţii, a-l împărţi în k segmente înseamnă a împărţi mulţimea $\{1, 2, ..., n\}$ în k submulţimi, $\{1, 2, ..., n_1\}$, $\{n_1+1, ..., n_2\}$, …, $\{n_{k-1}+1, n_{k-1}+2, ..., n_k\}$. Această partiţionare a mulţimii iniţiale poate fi reprezentată ca un vector $b = \{n_1, ..., n_k\}$, unde $n_k=n$, iar $n_1, ..., n_k$ reprezintă indicii ultimelor propoziţii din cele k segmente.

Primul pas al metodei de segmentare este construirea matricii implicaţiilor. Textului T i se va asocia o matrice E de dimensiune $n \times n$, care este definită după cum urmează:

$$e_{i,j} = \begin{cases} 1, & daca\ propozitia\ t_i\ implica\ propozitia\ t_j, i, j = \overline{1,n} \\ 0, & altfel \end{cases} \quad (3.2.3.a)$$

o a doua definiţie a aceleiaşi matrici este:

$$e_{i,j} = \begin{cases} \frac{1}{|i-j|+1}, & daca\ propozitia\ t_i\ implica\ propozitia\ t_j, i, j = \overline{1,n} \\ 0, & altfel \end{cases} \quad (3.2.3.b)$$

Implicaţiile sunt calculate utilizând măsura *cos*, metodă prezentată anterior în subcapitolul 3.1.3.

Fiecare segmentare va fi evaluat separat, din puctul de vedere al coeziunii:

$$scor(segment_{i,j}) = \sum_{k=i}^{j}\sum_{l=i}^{j} e_{k,l} \quad (3.2.3.1)$$

normalizarea scorului:

$$scor(segment_{i,j}) = \frac{\sum_{k=i}^{j}\sum_{l=i}^{j} e_{k,l}}{(|i-j|+1)^2} \quad (3.2.3.1')$$

şi varianta în care s-au eliminat auto-implicaţiile:

$$scor(segment_{i,j}) = \frac{\sum_{k=i}^{j}\sum_{l=i}^{j} e_{k,l} - (|i-j|+1)}{(|i-j|+1)^2} \quad (3.2.3.1'')$$

Din punctul de vedere al implicaţiilor, două segmente vecine sunt deconectate unul faţă de celălalt dacă între propoziţiile lor sunt cât mai puţine implicaţii:

$$scor(segment_{i,j}, segment_{j+1,k}) = \sum_{l=i}^{j}\sum_{m=j+1}^{k} e_{l,m} \quad (3.2.3.2)$$

Suma scorurilor locale (3.2.3.2) trebuie minimizată pentru a obţine o segmentare cu o interconectivitate mică, iar suma scorurilor locale (3.2.3.1), (3.2.3.1') respectiv (3.2.3.1") trebuie maximizată, pentru a obţine o segmentare în care segmentele au o coeziune internă ridicată.

Problema determinării unui număr predefinit de segmente poate fi privită ca o problemă combinatorială, iar cei mai potriviţi algoritmi pentru astfel de probleme sunt algoritmii genetici. Algoritmul propus porneşte de la o populaţie de cromozomi generaţi aleator, care reprezintă posibile segmente de dimensiune *k*. Pentru a genera următoarea generaţie, se aplică următorii operatori genetici: „binary tournament selection" pentru selecţia părinţilor şi „one point crossover" şi „flip mutation" [Bac00] pentru încrucişarea şi mutaţia care generează noii cromozomi. Procesul de evaluare este similar cu schema de evaluare a unui algoritm genetic standard. În plus se va folosi o selecţie a celor mai buni indivizi. Soluţia problemei de segmentare este reprezentată în algoritmul genetic numit GATTS. Algoritmul GATTS a folosit următorii parametri: dimensiunea populaţiei de 200, 50 de generaţii, probabilitatea de încrucişare de 0,7 şi cea de mutaţie de 0,1.

Evaluarea s-a realizat pe două dintre textele folosite anterior, un text literar, „The Koan" [Ric91] şi un articol de ziar, Hirst [Mor91] din care s-au eliminat propoziţiile scurte (cu mai puţin de 5 cuvinte inclusiv) celelalte 41 de propoziţii rămase fiind renumerotate. Rezultatele au fost evaluate comparativ cu cele obţinute prin metoda Programării dinamice (DP) prezentată anterior, deoarece ambele metode obţin un număr predefinit de segmente.

Rezultatele medii obţinute sunt comparabile cu cele mai bune rezultate ale metodei programării dinamice, iar cele maxime le depăşesc, unele fiind chiar egale cu 100%, ceea ce indică faptul că s-a identificat chiar segmentarea manuală.

Se observă că pentru Koan cele mai bune rezultate se obţin pentru varianta (3.2.3.a) a matricii *E*, iar dintre definiţiile scorurilor locale (3.2.3.1") şi (3.2.3.2) conduc la cele mai bune rezultate. În ceea ce priveşte matricea (3.2.3.b), combinaţia ei cu scorul (3.2.3.1') pare a fi cea mai potrivită. Pentru textul Hirst, matricea (3.2.3.b) conduce la rezultatele cele mai bune şi din nou se evidenţiază scorurile (3.2.3.1") şi (3.2.3.2).

Ca o direcţie viitoare de cercetare, se va căuta un mod de a corela cele două scoruri locale, (3.2.3.1") şi (3.2.3.2), pentru a îmbunătăţi metoda de segmentare.

3.2.4 O nouă metodă de segmentare top-down bazată pe lanţuri lexicale

O metodă de segmentare care oferă rezultate mai bune decât metoda LTT este cea top-down bazată pe lanţuri lexicale. Un lanţ lexical este o secvenţă de cuvinte astfel încât sensul fiecărui cuvânt din secvenţă poate fi obţinut precis pornind de la sensurile celorlalte cuvinte [Mor91, Bar99, Har97, Sil02]. Toate aceste lanţuri lexicale furnizează o reprezentare a structurii lexicale de coeziune a textului. În aceste abordări lanţurile lexicale au fost obţinute folosind o abordare bottom-up, luându-se fiecare cuvânt candidat al textului şi identificând relaţiile pe care acest cuvânt le are cu vecinii lui, pe baza unui tezaur cum este Rodget-ul [Mor91] sau WordNet-ul [Bar99, Sto04]. Dacă există astfel de relaţii, cuvântul respectiv este inserat cu sensul corespunzător în lanţul curent, iar sensurile celorlalte cuvinte din lanţ sunt modificate dacă este necesar. Dacă cuvântul nu este legat deloc de celelalte cuvinte, atunci se iniţializează un nou lanţ.

Metoda prezentată în continuare [Tăt08b/c/d] funcţionează în ordine inversă: mai întâi se dezambiguează întregul text şi apoi se construiesc lanţurile lexicale de dimensiuni maxime. Pentru dezambiguare s-a folosit algoritmul CHAD prezentat anterior. El identifică printr-o singură parcurgere a textului cuvintele care nu sunt legate ca sens de cuvintele anterioare, iar aceste cuvinte vor primi în mod „forţat" primul sens din WordNet. Dacă scorul fiecărei propoziţii va măsura numărul de cuvinte care au fost „forţate" să ia primul sens din WordNet, se va obţine structura lexicală coezivă a textului. Se notează cu *F* numărul de cuvinte „forţate" să ia primul sens din WordNet dintr-o propoziţie şi atunci în

punctele de minim ale graficului construit cu valorile 1/*F* se vor afla marginile lanţurilor lexicale (vezi figura 8) [Tăt08b/c].

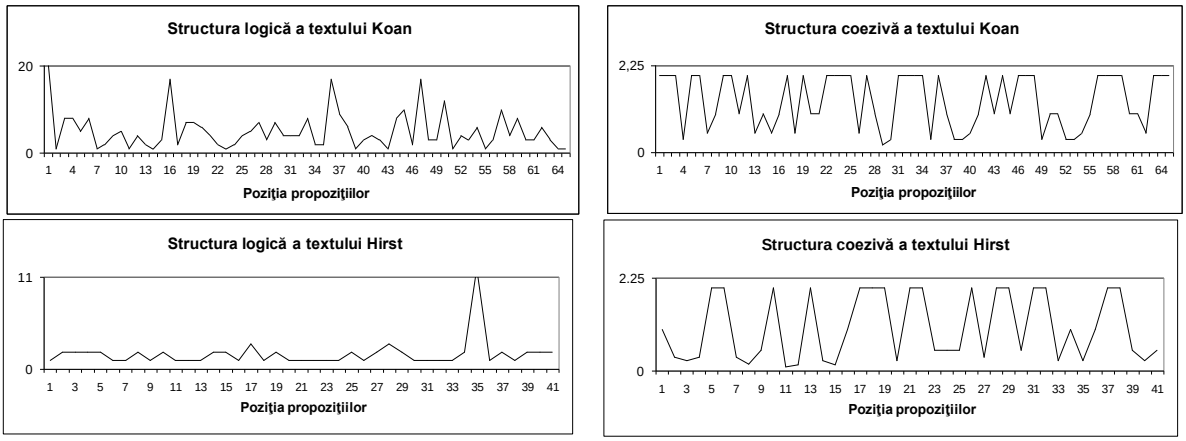

Figura 8. Structura logică şi coezivă a textelor Koan respectiv Hirst

Această idee de identificare liniară a lanţurilor lexicale stă la baza unui algoritm liniar de segmentare CTT (Cohesive TextTiling), similar cu algoritmul LTT prezentat anterior cu singura diferenţă a formulei de scorare a fiecărei propoziţii, $Scor(S_i) = \frac{1}{nuw_i}$, unde nuw_i este numărul de cuvinte „forţate" la primul sens din WordNet din propoziţia S_i. Dacă $nuw_i = 0$, atunci $Score(S_i) = 2$. CTT a fost testat pe textul Hirst [Mor91] şi „The Koan" [Ric91], obţinând rezultate mai bune faţă de LTT.

Metodele de segmentare prezentate mai sus pot fi de un real ajutor echipei de dezvoltatori, în special metodele de segmentare cu un număr predefinit de segmente – numărul de membri ai echipei. Astfel s-ar putea analiza şi rafina mai uşor şi mai rapid specificaţiile clientului, într-un proces paralel.

3.3. Noi abordări ale sumarizării unui text

Sistemele care pot în mod automat să extragă sumarul dintr-un text au devenit din ce în ce mai studiate şi utilizate. Sumarul este o variantă mai scurtă a unui text sau a mai multor texte, cu lungimea de maxim 50% din lungimea textului original, dar care păstrează ideile principale ale textului. În literatură se întâlnesc două feluri de sumare: extras şi abstract [Hov03]. Majoritatea sumarizatoarelor automate realizează un extras. Pentru aceasta în literatură se cunosc mai multe abordări [Mar97]. Propoziţiile sunt selecţionate după criteriile: conţin cele mai frecvente cuvinte, conţin cuvintele din titluri şi subtitluri, se află la începutul sau sfârşitul unor secţiuni, utilizează expresii care accentuează importanţa pe care o au în text, sunt cel mai strâns legate de alte părţi ale textului. Dintre aceste abordări ultima este cel mai dificil de realizat. Conectivitatea poate fi estimată utilizând numărul cuvintelor, sinonimelor sau anaforelor comune [Oră06, Rad02].

3.3.1 Extragerea sumarului dintr-un text segmentat

Există mai multe metode de obţinere a sumarului unui text, dar dacă sumarul este obţinut dintr-un text care a fost anterior împărţit în segmente, calitatea lui ar trebui să crească, deoarece selectarea unor propoziţii care aparţin unor segmente diferite trebuie să îmbogăţească sumarul cu noi informaţii relativ independente. În [Tăt08e, Tăt09c] pe lângă cele câteva metode de segmentare liniară prezentate anterior, se demonstrează faptul că segmentarea îmbunătăţeşte calitatea sumarizării.

Un sumar obţinut simplu ca şi consecinţă a relaţiei de implicaţie dintre două texte este sumarul „PureEntailment” (PE) şi se obţine adăugând în mod secvenţial sumarului fiecare propoziţie care nu este implicată de sumarul curent.

Pornind de la segmentările PE, LTT, ArcInt şi respectiv ArcReal descrise în subcapitolul 3.2.1, se poate construi un sumar alegând cele mai importante propoziţii nu din tot textul, ci din fiecare segment în parte astfel ca în final sumarul să aibă numărul dorit de propoziţii. Numărul de propoziţii care se aleg din fiecare segment şi modul de alegere este dat de algoritmul din figura 9.

Algoritmul *SumPostSeg(S, n, SEG, j, Sum, x)* ***este:***
date *S {un text; $s_1,s_2,...,s_n$ – propoziţiile textului}, n {numărul de propoziţii din text}, SEG {$Seg_1,...,Seg_j$ segmentele în care este împărţit textul}, j {numărul de segmente}, x {mărimea sumarului}*
@ se calculează importanţa fiecărui segment şi se ordonează segmentele după importanţă: $Seg_{i_1},\ldots,Seg_{i_j}$
@ se calculează numărul c_{i_s} *de propoziţii „esenţiale” pentru fiecare segment* $Seg_{i_s}, s = i_1, i_j$
@ se selectează primele k (<j) segmente astfel încât $\sum_{Seg_{i_s}} c_{i_s} = x$
@ se reordonează propoziţiile „esenţiale” din cele k segmente selectate, după ordinea în care apar în text, Seg'$_1$,...,Seg'$_k$
@ propoziţiile din Seg'$_1$,...,Seg'$_k$ sunt propoziţiile sumarului SUM
rezultate *SUM {sumarul de lungime x}*
sf *SumPostSeg*

Figura 9. Algoritmul de obţinere a sumarului dintr-un text segmentat

În această primă abordare a construirii sumarului pornind de la o segmentare, s-a considerat că toate segmentele sunt la fel de importante şi din fiecare segment s-a ales cea mai importantă (cu cel mai mare scor) propoziţie (câte o singură propoziţie „esenţială”).

Algoritmul *DP_Sum(S, n, SC, penalitate, Sum, x)* ***este:***
date *S {un text, s_i – propoziţiile textului, i ← 1,n}, n {numărul de propoziţii din text}, SC {sc_i = scor(s_i) scorurile propoziţiilor din text}, penalitate {constantă}, P {p_i – poziţiile din text a propoziţiilor sumarului, i←1,x}, x {numărul de propoziţii din sumar}*
pentru *i ← 1,n* ***execută***
$\delta_i^1 \leftarrow scor(s_i)$
pentru *k ← 2,x* ***execută***

$$\delta_i^k = \begin{cases} max_{j(i<j)}(scor(s_i)+\delta_j^{k-1}) & \text{, dacă } s_i \text{ şi } s_j \text{ au cuvinte comune} \\ max_{j(i<j)}(penalitate*(scor(S_i)+\delta_j^{k-1})) & \text{, altfel} \end{cases}$$

$$h_i^k = \begin{cases} arg\,max_{j(i<j)}(scor(s_i)+\delta_j^{k-1}) & \text{, dacă } s_i \text{ şi } s_j \text{ au cuvinte comune} \\ arg\,max_{j(i<j)}(penalitate*(scor(s_i)+\delta_j^{k-1})) & \text{, altfel} \end{cases}$$

sfp
sfp
$i \leftarrow arg\,max_j(\delta_j^x)$
iniţializează(Sum) {iniţial sumarul Sum nu conţine nici o propoziţie}
pentru *k ← x,1* ***execută***
adaugă(Sum, s_i) {adaugă sumarului propoziţia s_i}
$i \leftarrow h_i^k$
sfp
rezultate *Sum {sumarul de lungime x}*
sf *DP_Sum*

Figura 10. Algoritmul de obţinere a sumarului dintr-un text segmentat

O altă abordare a problemei sumarizării este cea a aplicării unui algoritm de programare dinamică (vezi figura 10). Cu scopul de a păstra coerenţa sumarului, acest algoritm va selecta un lanţ de propoziţii astfel încât două propoziţii consecutive să aibă cel puţin un cuvânt în comun, ceea ce corespunde cu principiul continuităţii din teoria centrării care stipulează ca două unităţi ale discursului să aibă cel puţin o entitate comună [Oră06].

Se presupune că fiecare propoziţie are asociat un scor al reprezentativităţii ei. Se vor folosi cele trei scoruri logice definite în subcapitolul 3.2.1: suma implicaţiilor, a arcelor întregi respectiv ponderate. Scorul unui sumar este suma scorurilor tuturor propoziţiilor din sumar. Sumarul va fi construit astfel încât acest scor să fie maxim. Pentru metoda Programării dinamice s-a folosit o penalitate de 1/10 şi lista cu 571 de cuvinte „stop" LYRL2004, dezvoltată pentru proiectul SMART [Lew04]. Totodată doar substantivele, verbele, adjectivele şi adverbele au fost folosite drept cuvinte comune şi s-a utilizat Stemmer-ul Porter [***Por] pentru a compara cuvintele.

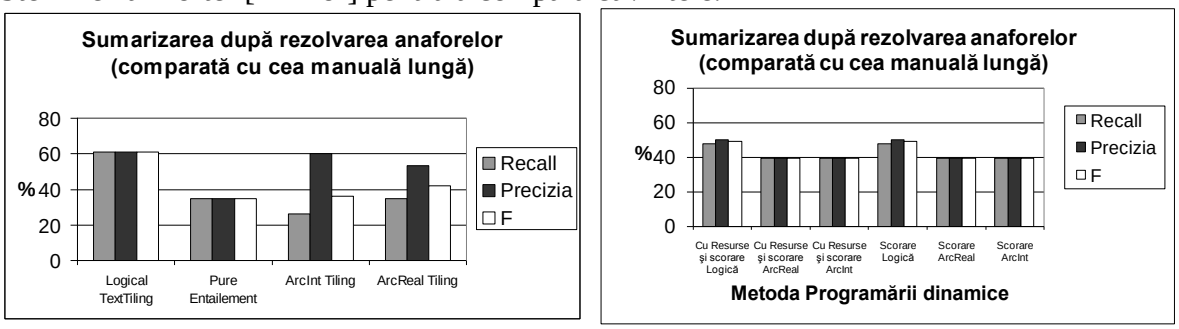

Figura 11. Evaluarea sumarizării

Pentru a evalua metodele de sumarizare propuse s-a utilizat textul literar „The Koan" [Ric91], care pe baza criteriilor lingvistice a fost sumarizat şi i s-au rezolvat anaforele în mod manual [Tăt08e]. Din figura 11 se observă că segmentarea îmbunătăţeşte sumarizarea, iar combinarea metodei programării dinamice cu scorarea logică duce la rezultate bune. Dintre metodele logice, cele mai bune rezultate se obţin pentru LTT.

3.3.2 Sumare de lungime variabilă

Sumarul propus anterior se bazează pe segmentare. Şi tot pornind de la segmentarea LTT (vezi subcapitolul 4.1) se va obţine şi următorul sumar de lungime variabilă. Segmentarea LTT porneşte de la structura logică a textului, care are o formă fixă, cu un număr fix de minime, astfel că şi numărul de segmente obţinute este tot fix. De aceea abordarea anterioară care selecta din fiecare segment propoziţia cu scorul cel mai mare, unde în prealabil segmentele au fost ordonate după importanţa lor, până când s-a obţinut numărul dorit de propoziţii din sumar nu poate fi aplicată tot timpul, cum ar fi în cazul în care numărul de propoziţii din sumar este mai mare decât numărul de segmente. În [Tăt08a] s-au propus trei metode de sumarizare de lungime variabilă care pornesc de la o segmentare existentă, metode care vor fi prezentate în continuare. Testarea s-a realizat pe textul Hirst [Mor91] (din care s-au exclus propoziţiile de cinci sau mai puţine cuvinte), iar rezultatele au fost comparate cu cele obţinute de autori prin lanţuri lexicale.

Un prim pas pentru sumarizare constă în scorarea segmentelor. Sumarul se va obţine prin selectarea unui număr de propoziţii proporţional cu scorul segmentului respectiv. Se vor folosi următoarele notaţii: $Scor(s_i)$ = *numărul de propoziţii implicate de* s_i, $Scor(Seg_j) = \frac{\sum_{s_i \in Seg_j} Scor(s_i)}{|Seg_j|}$, $Scor_{final}(s_i) = Scor(s_i) \times Scor(Seg_j)$, unde $s_i \in Seg_j$,

$Scor(Text) = \sum_{j=1}^{n} Scor(Seg_j)$, *Ponderea unui segment* : $c_j = \frac{Scor(Seg_j)}{Scor(Text)}$; $0 < c_j < 1$,

n = numărul segmentelor obţinute de algoritmul LTT, x = lungimea prestabilită a sumarului, $NSenSeg_i$ = numărul de propoziţii selectate din segmentul Seg_j.

Algoritmul de sumarizare de lungime prestabilită, notat cu AL (Arbitrary Length) se află în figura 12. S-ar putea ca numărul de propoziţii selectate în sumar dintr-un segment, $NSenSeg_j > 1$. Dacă $x < n$, atunci pentru anumite segmente, $NSenSeg_j = 0$.

Algoritmul *AL(Text, SEG, n, Lseg, SC, SelM, Sum, x),* ***este:***
date *Text {un text} SEG {segmentele textului $Seg_1,...,Seg_n$}, n {numărul de segmente}, Lseg {vectorul cu lungimile (în număr de propoziţii) a celor n segmente}, SC {scorurile fiecăror propoziţii $sc_i=Scor_{final}(s_i)$}, SelM {metodă de selecţie}, x {mărimea sumarului}*
@ se calculează ponderea fiecărui segment cj, j=1,n
@ se ordonează segmentele după ponderea cj
{se calculează numărul $NSenSeg_j$, j=1,n}
Câttimp *@ numărul propoziţiilor selectate deja în sumar nu a depăşit numărul prestabilit x* ***execută***
Dacă *$[x \times c_j] \geq 1$ {parte întreagă}*
atunci *$NSenSeg_j = min(lseg_j, [x \times c_j])$*
altfel *$NSenSeg_j = 1$*
sfd
sfc
@ se iniţializează sumarul Sum
@ se selectează conform metodei de selecţie SelM din fiecare segment Seg_j - $NSenSeg_j$ propoziţii şi se adaugă sumarului Sum
@ se reordonează propoziţiile sumarului, în funcţie de ordinea în care sunt în Text
rezultate *Sum {sumarul de lungime x}*
sf *AL*

Figura 12. Algoritmul de obţinere a sumarului dintr-un text segmentat

Metoda de selecţie a propoziţiilor dintr-un segment (*SelM*) este decisivă pentru calitatea sumarului. S-au propus trei metode de selecţie a celor mai importante propoziţii din fiecare segment, fiecare încercând să excludă cât mai puţine informaţii.

Definiţia 1 Se dă un text segmentat $T = \{Seg_1,...,Seg_n\}$. Din fiecare segment $Seg_i, i = \overline{1,n}$, se selectează primele $NSenSeg_i$ propoziţii astfel încât $\sum_{i=1}^{n} NSenSeg_i = x$. Se obţine sumarul $Sum_1 = \{s'_1,...,s'_x\}$.

Definiţia 2 Dându-se o segmentare a textului iniţial $T = \{Seg_1,...,Seg_n\}$, sumarul se obţine în felul următor: $Sum_2 = \{s'_1,...,s'_x\}$, unde $\forall i = \overline{1,n}$, $Sum_2 \cap Seg_i = SelPropSeg_i$, $|SelPropSeg_i| = NSenSeg_i$, $\sum_{i=1}^{n} NSenSeg_i = x$ şi $\forall s_j \in SelPropSeg_i$ şi $\forall s_k \in Seg_i \setminus SelPropSeg_i$, $scor(s_j) \geq scor(s_k)$.

Definiţia 3 Dându-se o segmentare a textului iniţial $T = \{Seg_1,...,Seg_n\}$, sumarul se obţine în felul următor: $Sum_3 = \{s'_1,...,s'_x\}$, unde $s'_1 = s_1$, Seg_0 conţine doar propoziţia s_1 şi $\forall i = \overline{1,n}$, $Sum_3 \cap Seg_i = SelPropSeg_i$, $|SelPropSeg_i| = NSenSeg_i$, $\sum_{i=1}^{n} NSenSeg_i = x$ şi $\forall s_j \in SelPropSeg_i$ şi $\forall s_k \in Seg_i \setminus SelPropSeg_i$, $sim(s_j,Seg_{i-1}) \leq sim(s_k,Seg_{i-1})$, $sim(s_j,Seg_{i-1})$ reprezintă similaritatea dintre s_j şi ultima propoziţie selectată în Seg_{i-1}. Similaritatea a două propoziţii, s, s', fiind calculată cu metrica $cos(s,s')$.

Similaritatea sumarelor de diferite lungimi obţinute cu AL şi cele trei definiţii Sum_1, Sum_2, Sum_3, respectiv metoda programării dinamice, calculată relativ la textul iniţial este prezentată în figura 13.

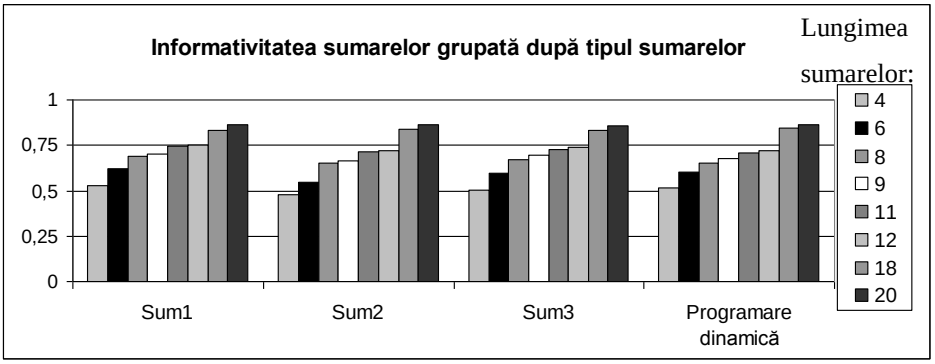

Figura 13. Caracteristica informativă a sumarelor de diferite lungimi, obţinute cu AL şi cele trei definiţii Sum_1, Sum_2, Sum_3, respectiv metoda programării dinamice

Se observă că cele mai bune rezultate au fost obţinute pentru aplicarea algoritmului AL cu definiţia Sum_1. Totodată se observă influenţa pozitivă a segmentării asupra sumarizării. Pentru lungimile 9 şi 12, în 5 din 6 cazuri metoda sumarizării precedată de segmentare are rezultate mai bune faţă de metoda sumarizării directe.

3.3.3 Sumarizare bazată pe lanţuri lexicale

Sumarizarea foloseşte acelaşi algoritm AL prezentat în subcapitolul anterior, doar că algoritmul AL iniţial se consideră a fi prima variantă *Var1*, unde scorul unui segment era calculat după formula $Scor(Seg_j) = \frac{\sum_{S_i \in Seg_j} Scor(S_i)}{|Seg_j|}$. A doua variantă a aceluiaşi algoritm, *Var2* diferă doar prin formula de calcul a scorului unui segment: $Scor(Seg_j) = \sum_{S_i \in Seg_j} Scor(S_i)$. Ambele variante au avantajele şi dezavantajele lor, varianta *Var1* poate dezavantaja segmentele de lungimi mari în care doar câteva propoziţii au un scor ridicat, pe când *Var2* acordă o prea mare importanţă segmentelor lungi.

Dispunând de două segmentări obţinute prin LTT (subcapitolul 3.2.1) şi CTT (subcapitolul 3.2.4), de variantele *Var1* şi *Var2* de aplicare a algoritmului de sumarizare cu lungime variabilă AL, cât şi de cele trei definiţii Sum_1, Sum_2 respectiv Sum_3, s-a încercat o combinare a sumarelor rezultate într-un potenţial sumar Ideal IdS, prin alegerea propoziţiilor cele mai frecvente din sumarele obţinute.

Pentru evaluare s-au utilizat textele „The Koan” [Ric91], Hirst [Mor91], Tucker1 şi Tucker2 [Tuc99]. S-a calculat similaritatea fiecărui sumar cu textul din care a provenit (vezi tabelul 2). Se poate ajunge la concluzia că segmentările LTT şi CTT favorizează obţinerea unor sumare de calitate. Dintre cele două metode, CTT conduce la rezultate mai bune, parţial datorită faptului că precizia verificării relaţiei de implicaţie logică este mai mică faţă de cea a dezambiguării cuvintelor (algoritmului CHAD).

Textul Koan							
seg.	tipul sumarului	lungimea sumarului					
		5		6		10	
		Var1	*Var2*	*Var1*	*Var2*	*Var1*	*Var2*
LTT	Sum_1	0,402941	0,357519	0,490186	0,419314	0,597022	0,614695
	Sum_2	0,357519	0,303239	0,427179	0,376552	0,548034	0,583095
	Sum_3	0,427179	**0,548034**	0,508666	0,531751	0,583095	0,587805
CTT	Sum_1	0,449629	0,514595	**0,531751**	**0,587805**	**0,654998**	**0,662474**
	Sum_2	0,449629	0,442326	0,463739	0,463739	0,568535	0,542697
	Sum_3	**0,483779**	0,463739	0,502625	0,508666	0,627334	0,635489
	IdS	0,419314		0,47724		0,631438	
Textul Hirst							
seg.	tipul sumarului	lungimea sumarului					
		5		6		10	
		Var1	*Var2*	*Var1*	*Var2*	*Var1*	*Var2*
LTT	Sum_1	**0,579**	**0,6205**	0,6229	**0,644**	**0,7301**	**0,7126**
	Sum_2	0,5345	0,5313	0,5439	0,5649	0,6978	0,6939
	Sum_3	0,5377	0,5706	0,5845	0,6253	0,6901	0,6997
CTT	Sum_1	0,5281	0,4635	0,5439	0,5345	0,6761	0,6529
	Sum_2	0,5182	0,501	0,5345	0,559	0,672	0,6615
	Sum_3	0,562	0,501	**0,6253**	0,5248	0,7089	0,6761
	IdS	0,4975		0,5561		0,6802	
Textul Tucker1							
seg.	tipul sumarului	lungimea sumarului					
		5		6		10	
		Var1	*Var2*	*Var1*	*Var2*	*Var1*	*Var2*
LTT	Sum_1	0,5102	0,517512	0,564498	0,579741	0,67433	0,67433
	Sum_2	0,5102	0,517512	0,564498	0,579741	0,67433	0,67433
	Sum_3	0,528176	0,548496	0,554993	0,579741	0,65143	0,665371
CTT	Sum_1	**0,570682**	0,538497	0,588551	**0,588551**	0,678714	0,67433
	Sum_2	0,558193	0,528176	0,582704	0,582704	0,6873	**0,711701**
	Sum_3	0,561361	**0,57675**	**0,602715**	0,579741	**0,703782**	0,691504
	IdS	0,5102		0,594292		0,699744	
Textul Tucker2							
seg.	tipul sumarului	lungimea sumarului					
		5		6		10	
		Var1	*Var2*	*Var1*	*Var2*	*Var1*	*Var2*
LTT	Sum_1	0,637022	0,637022	0,664775	0,664775	0,805387	**0,806**
	Sum_2	0,637022	0,637022	0,664775	0,664775	0,805387	0,805387
	Sum_3	0,594385	0,5547	0,671345	0,581695	0,807441	0,708069
CTT	Sum_1	**0,645**	0,610558	0,680946	0,668078	0,740819	0,740819
	Sum_2	0,644194	**0,637022**	0,680946	0,690257	0,755929	0,755929
	Sum_3	0,644194	0,568481	**0,730297**	**0,696311**	**0,82717**	0,713782
	IdS	0,637022		0,664775		0,727607	

Tabelul 2. Similarităţile sumarelor obţinute automat

IV. Utilizarea ontologiilor în prelucrarea cerinţelor

Cu toate că tehnicile de prelucrare a limbajului natural sunt utile în prelucrarea cerinţelor pentru obţinerea specificaţiilor, de preferat formale sau într-o formă apropiată, sunt necesare resurse suplimentare. Cea mai folositoare resursă este ontologia, de vreme ce ontologia permite asocierea unei informaţii semantice unui text de limbaj natural, text care nu poate fi înţeles altfel de calculator.

4.1. Potrivirea unei ontologii la un text

În prezent a crescut cantitatea de informaţie semantică existentă în internet. Totuşi mai există o cantitate enormă de informaţie scrisă în limbaj natural fără nici un fel de suport semantic, pe care „făpturile umane nu le-ar putea organiza pe toate" [Pol09]. Şi cum există deja ontologii de diverse tipuri, obţinute în diverse moduri, s-ar pune mai degrabă problema asocierii unei ontologii existente unui text, decât extragerea ontologiei din text.

Evaluarea ontologiilor

Considerându-se diferitele tipuri de ontologii şi modul în care ele au fost obţinute, se impune un mijloc de garantare a calităţii ontologiilor. În [Gan05b] metricile de asigurare a calităţii ontologiei au fost clasificate în trei tipuri: metrici structurale, măsuri funcţionale şi măsuri de prelucrare a uşurinţei de utilizare. Potrivirea unei ontologii la un text aparţine de categoria măsurilor funcţionale [Gan05b], astfel că următoarele metrici recomandate vor fi precizia şi recall-ul criteriilor de potrivire propuse [Mih10b].

Nevoia de a identifica ontologia corespunzătoare unei anumite sarcini a fost discutată în diferite articole. În [Eng05] ontologiile sunt căutate şi evaluate pe baza unui set de cuvinte cheie. În [Tan09] e propus un cadru de selecţie a celei mai adecvate ontologii pentru o anumită aplicaţie de exploatare a textului din domeniul medical. O altă lucrare similară este [Doa03], unde sunt utilizate anumite măsuri de similaritate şi tehnici de învăţare automată pe bază de nume, context şi etichete.

Metrici de evaluare a potrivirii unei ontologii la un text scris în limbaj natural

Există nenumărate ontologii, multe fiind în mod continuu îmbogăţite. Dar din punctul de vedere al prelucrării limbajului natural, revine următoarea întrebare: Care ontologie este cea mai potrivită pentru un anumit text şi care parte din respectiva ontologie? Pentru a putea răspunde la această întrebare, s-a propus în [Mih10b] utilizarea unui set de metrici specifice.

Prima metrică propusă va evalua câte concepte se regăsesc în textul scris în limbaj natural. Deoarece toate tipurile de ontologii se pot reprezenta ca o colecţie de triplete (concept – predicat - obiect), ontologia analizată s-a presupus că se află reprezentată în această formă.

Metrici cantitative

Cea mai simplă tehnică de evaluare a potrivirii unei ontologii la un text e de a căuta conceptele în text. Astfel prima metrică propusă, similară cu o metrică de evaluare a calităţii unei ontologii şi cu o tehnică de aliniere a ontologiilor este *numărul de concepte identificate în text*. Deoarece conceptele pot să apară în text în diferite forme (singular, plural, ...) s-a considerat că s-a găsit conceptul dacă s-a identificat cel puţin un cuvânt care începe cu conceptul căutat.

Câteva metrici derivate pot fi evaluate în mod similar: *numărul de rădăcini găsite în text*, *numărul de frunze găsite în text*, iar lista poate continua. Rădăcinile sunt acele concepte care apar doar în părţile stângi ale tripletelor, iar frunzele sunt conceptele care apar doar în părţile drepte.

Metrici bazate pe relaţia de consecinţă logică dintre două texte

În ultimii ani tehnicile de evaluare a implicaţiei dintre două texte sau a relaţiei de consecinţă logică dintre două texte au evoluat. De aceea s-a propus folosirea acestei tehnici şi în [Mih10b] pentru evaluarea potrivirii unei ontologii cu un text. Şi aceasta mai ales datorită faptului că un triplet este similar cu o propoziţie simplă de limbaj natural (subiect, predicat, complement).

Utilizarea relaţiei de implicaţie dintre două texte în evaluarea potrivirii unei ontologii la un text este realizată prin verificarea cazului în care ontologia, sau părţi din ea implică textul scris în limbaj natural. S-au propus următoarele metrici: *implicaţia* (prin care se verifică dacă ontologia implică textul), respectiv *numărul de implicaţii* (prin care se verifică câte triplete ale ontologiei implică textul). În acelaşi timp se poate verifica câte propoziţii din text sunt implicate de un triplet al ontologiei şi astfel s-a propus o a treia măsură, *numărul de propoziţii implicate* (se numără fiecare implicaţie dintre un triplet şi o propoziţie a textului), respectiv *numărul propoziţiilor distincte implicate* (se numără propoziţiile distincte implicate de cel puţin un triplet). Pentru a verifica relaţia de consecinţă logică s-a ales metoda cosinusurilor (vezi 3.1.3).

S-a evaluat **precizia** (numărul total de potriviri posibile) şi **recall-ul** (câte potriviri au fost găsite din numărul total de elemente testate). Pentru a **evalua** metricile propuse s-a folosit un set de nouă ontologii din [***WnB], ontologii care au toate ca rădăcină termenul „business”, dar unde cuvântul „business” are pentru fiecare ontologie un alt sens. Cele nouă ontologii au fost comparate cu prima parte a unui articol din Wikipedia despre „small business” [***WSM]. Din cele nouă ontologii primele patru şi cea de a şasea par a fi cele mai potrivite cu textul.

Din păcate metricile care au la bază implicaţia nu sunt suficient de precise, în special datorită conceptului „bussines” care e comun celor nouă ontologii. Metricile „cantitative” propuse confirmă presupunerea făcută, conform căreia doar primele patru ontologii şi cea de a şasea se potrivesc cu textul (pentru ontologia 1 s-a obţinut un recall de 10,5%, pentru cea de a doua 8,1%, a patra 20%, iar a şasea 6,5%).

4.2. Învăţarea unei ontologii dintr-un text de limbaj natural pe baza arborelui de analiză sintactică a unei propoziţii

Rezultatul analizei gramaticale a unei propoziţii este reprezentat de obicei prin intermediul unui arbore. Iar între cuvintele unei propoziţii se pot identifica diferite relaţii de dependenţă [Mar08], relaţii care sunt surprinzător de asemănătoare tripletelor RDF (Resource Description Framework – cadrul de descriere al relaţiilor), cel mai simplu mijloc de a reprezenta o ontologie. Pentru analiza gramaticală a unei propoziţii s-a folosit unealta on-line dezvoltată de grupul de procesare a limbajului natural de la Stanford, Natural Language Processing Group [***STO], care prezintă o precizie mai mare de 87% [Kle03].

În procesul de învăţare a ontologiilor s-a recunoscut importanţa analizei sintactice a textului. Dar relaţiile gramaticale au fost utilizate doar ca şi şabloane pentru recunoaştere în [Cim05, May09] sau ca şi constrângeri pentru identificarea relaţiilor dintre conceptele ontologiei în [Kaw04].

Analiza sintactică a unei fraze

Analizorul Stanford [***STS] furnizează arborele de analiză sintactică, dependenţele de tip şi dependenţele condensate (de exemplu: *nsubjpass*(abbreviated-6, title-3)). Pornind de la dependenţe, se poate genera o structură sub formă de graf (vezi figura 14, corespunzătoare propoziţiei „This function title is often abbreviated to the initials 'HR'.” [***WHR]).

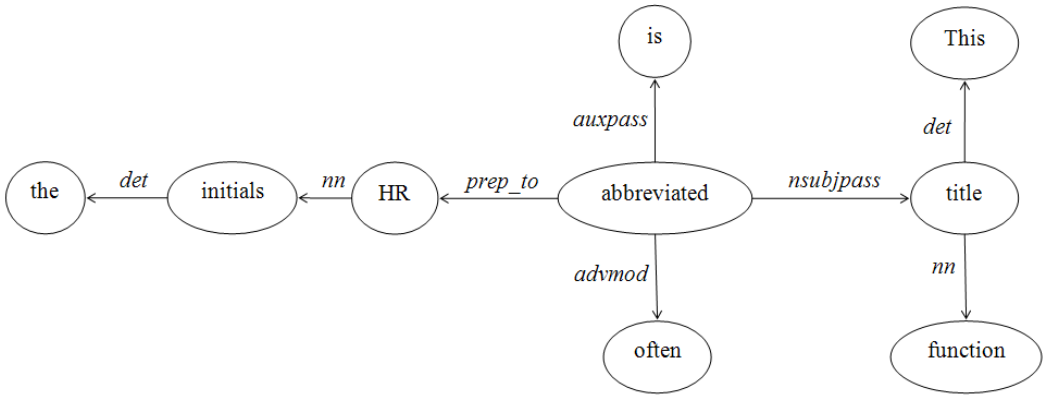

Figura 14. Graful obţinut din dependenţele de tip

Identificarea tripletelor

Prima metodă propusă de extragere a tripletelor este cea de **extragere a tripletelor din graful dependenţelor** (vezi figura 15) [Mih10c].

@elimină toate dependenţele „dep”
Pentru *@fiecare dependenţă* D_1 ***execută***
  Pentru *@fiecare dependenţă* D_2 *diferită de* D_1 ***execută***
    Dacă *@*D_1 *şi* D_2 *au cuvinte comune* ***atunci***
      Dacă *@cuvântul comun este verb iar cuvintele distincte sunt substantive* ***atunci***
        @adaugă tripletul (cuvântul distinct din D_1*, cuvântul comun, cuvântul distinct din* D_2*) la lista tripletelor*
      sfd
    sfd
  sfp
sfp

Figura 15. Extragerea tripletelor din graful de dependenţe

A doua metodă de extragere a tripletelor este cea de **extragere a tripletelor din arborele de analiză sintactică** (vezi figura 16).

Pentru testarea metodelor propuse, s-a folosit acelaşi text şi aceleaşi ontologii ca şi în subcapitolul precedent. S-a evaluat dacă două cuvinte sunt similare sau nu (sunt incluse unul în altul), cuvintele stop [***SPW] şi cuvântul „business” neluându-se în considerare.

Metoda/Ontologia	prima	a doua	a patra	a şasea
I	0/0	2+2(inversele)/1	0/0	0/0
II	2/2	2/3	1/1	2/2

Tabelul 3. Tripletele similare extrase din text/aparţinând de ontologie

Rezultatele **evaluării** se pot observa în tabelul 3. Deoarece cele mai multe triplete extrase din text sunt similare cu triplete din cea de a doua ontologie pentru ambele metode, articolului testat despre o afacere mică i se potriveşte cel mai bine ontologia în care „business” are sensul de afacere (ceea ce se aştepta).

Subalgoritmul *IdentificăTriplet(S)* ***este*** *{S este o propoziţie}*
@citeşte simbolul SB
Dacă *@SB este NN* ***atunci*** *@ conceptul din triplet este cuvântul următor* ***sfd***
Dacă *@SB este NP* ***atunci*** *@analizează constituenţii* ***sfd***
Dacă *@SB este S* ***atunci*** *IdentificăTriplet(@noua propoziţie S) @foloseşte ca şi concept conceptul din noua S* ***sfd***
@citeşte simbolul SB
Dacă *@SB este VB,* ***atunci*** *@predicatul tripletului este următorul cuvânt* ***sfd***
Dacă *@SB este VP* ***atunci*** *@analizează constituenţii* ***sfd***
@citeşte simbolul SB
Dacă *@SB este NN* ***atunci*** *@obiectul tripletului este următorul cuvânt* ***sfd***
Dacă *@SB este NP sau ADJP* ***atunci*** *@analizează constituenţii* ***sfd***
Dacă *@SB este S* ***atunci*** *IdentificăTriplet (@noua propoziţie S) @şi foloseşte ca şi obiect conceptul din noua S* ***sfd***
sf *IdentificăTriplet*
{în „analizează constituenţii", are loc un proces similar: se citeşte următorul simbol, se identifică tipul său până la epuizarea ramurii curente, iar în cazul unei conjuncţii sau a unei disjuncţii de termeni (pentru concepte şi obiecte, sau disjuncţie pentru predicate), atunci fiecare element al conjuncţiei va fi considerat separat, iar în final, în loc să se obţină un triplet, se va obţine o serie de triplete}

Figura 16. Extragerea tripletelor din arborele de analiză sintactică

4.3. Similaritatea utilizată în identificarea nevoii de a refactoriza o ontologie

Ontologiile existente s-au obţinut în diverse forme, de la modul manual, produsul muncii diferiţilor specialişti sau voluntari [Wil06], la modul semi-automat [Mor06] dar şi automat [Dah06]. Mai mult, nevoia de îmbunătăţire a unei ontologii existente a fost discutată în [Suk08, Che06]. Unele sunt dezvoltate într-o perioadă lungă de timp sau au un număr impresionant de elemente. Când într-o lucrare de mari dimensiuni sunt implicate diferite surse, mai mult când între momentul conceperii ontologiei şi momentul final al livrării ei trece mult timp, atunci creşte probabilitatea ca în produsul final (ontologia finală) să existe anumite discrepanţe, să existe anumite elemente plasate greşit.

Diferite măsuri ale similarităţii au fost utilizate pentru evaluarea unei ontologii [Bra05], pentru alinierea ontologiilor [Euz04] sau pentru potrivirea ontologiilor [Euz07]. Dar de obicei măsurarea similarităţii utilizează similaritatea cuvintelor, evaluată prin intermediul măsurilor de evaluare a şirurilor. Omonimele, deşi sunt scrise în mod identic, au sensuri diferite şi sunt recunoscute în mod greşit ca fiind identice. Un alt tip de similaritate utilizat în prelucrarea limbajului natural este similaritatea textelor. Pentru aceasta sunt folosite aproape toate cuvintele care apar în text (mai puţin cuvintele stop), iar similaritatea textului se calculează folosind similaritatea cuvintelor din text.

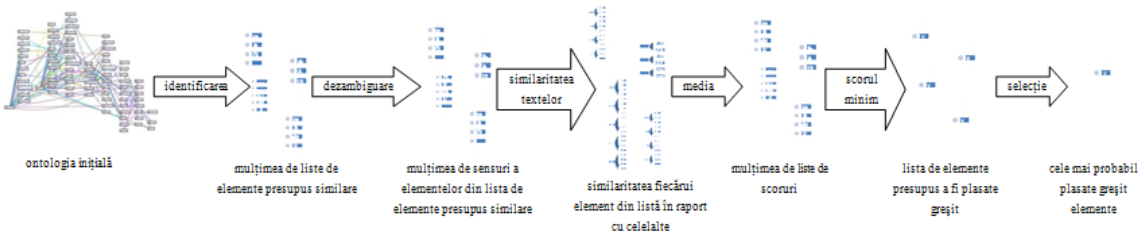

Figura 17. Procesul de selectare a elementelor cel mai probabil plasate greşit

Dacă se consideră cazul în care ontologia este reprezentată de o listă de triplete, atunci se pot extrage subliste de elemente care sunt legate prin acelaşi predicat de acelaşi concept sau predicate care leagă aceleaşi două concepte. Astfel de **elemente** ar trebui să fie **similare**. Dacă se evaluează similaritatea a două câte două elemente, se face o medie a

similarităţilor pentru fiecare element în parte, atunci elementul care obţine cel mai mic scor este elementul cel mai probabil a fi **plasat greşit** (vezi figura 17) [Mih10d].

Algoritmul de identificare a elementelor cel mai probabil plasate greşit

```
Algoritmul Selectarea_Celor_mai_Probabil_Plasate_Greşit_Elemente (O, n) este
  Date O- ontologia dată prin lista ei de triplete
       n- numărul de elemente probabil plasate greşit căutate
  @ extrage lista de elemente (concepte)
  Pentru @fiecare element distinct din ontologie  execută
    @identifică lista de elemente distincte care sunt legate prin acelaşi predicat de elementul curent
    @ identifică lista de predicate care leagă acelaşi element de elementul curent
  sfp
  @păstrează într-o mulţime S doar listele cu cel puţin trei elemente
  @iniţializează o mulţime vidă de liste S'
  Pentru @fiecare element distinct e  din fiecare listă care aparţine de mulţimea S execută
    @ ia toate predicatele distincte şi elementele distincte relaţionate cu e şi se construieşte lista L cu ele
    @adaugă şi elementul e listei L
    sensul_selectat=1
    sim_min=1
    Pentru @ fiecare sens i din dicţionar, sens al numelui elementului curent e execută
      @calculează similaritatea s a textului compus din numele elementelor din L şi sensul i
      Dacă @acesta este primul sens atunci
        sim_min=s
      altfel Dacă sim_min>s atunci
        sim_min=s
        sensul_selectat=i
        sfd
      sfd
    sfp
    @adaugă în S' într-o poziţie analoagă poziţiei curentului element e sensul selectat
  sfp
  i=0
  @iniţializează mulţimea vidă S"
  Pentru @fiecare listă L' din S' execută
    mins=1
    poz_min=1
   i=i+1
    Pentru @fiecare text t din lista L' execută
      nr=0
      sum=0
      Pentru @fiecare text  t' din lista L'
        Dacă e ≠ e' atunci
          @adaugă la sum similaritatea lui t relativă la t'
          nr=nr+1
        sfd
      sfp
      med=sum/nr
    sfp
    Dacă i=1atunci
      mins=med
    altfel Dacă mins>med
      mins=med
      poz_min=i
      sfd
    sfd
    @adaugă la S" perechea formată de elementul de pe poziţia i din lista L a mulţimii  S (unde lista L
    este „părintele" listei de liste L') şi scorul ei, med
  sfp
  @ordonează elementele mulţimii S" în ordinea crescătoare a scorurilor şi se păstrează primele n
```

Rezultate: *primele n elemente cu cele mai mici scoruri*
sf *Selectarea_Celor_mai_Probabil_Plasate_Greşit_Elemente*

Observaţie: în algoritm, prin elemente sau relaţii distincte se înţelege cele care au URI distincte.

Ca exemplu de funcţionare a algoritmului, se presupune că într-o ontologie cuvintele „man", „women", „elderly" şi „driver" reprezintă toate subclase ale clasei „adult", iar în urma dezambiguării s-a identificat primul sens din WordNet [***WNt] pentru toate cele patru cuvinte:

man – man, adult male (an adult person who is male (as opposed to a woman)) "there were two women and six men on the bus"
women – woman, adult female (an adult female person (as opposed to a man)) "the woman kept house while the man hunted"
elderly – aged, elderly (people who are old collectively) "special arrangements were available for the aged"
driver – driver (the operator of a motor vehicle)

În urma calculelor elementul „driver" obţine cel mai mic scor.

4.4. Rolul dezambiguării în evaluarea ontologiilor

Dezambiguarea a fost deja utilizată în domeniul ontologiilor pentru a îmbogăţi o ontologie [Ste02] sau pentru a construi o ontologie [San07]. În cele ce urmează se va discuta rolul pe care îl are dezambiguarea pentru potrivirea ontologiilor, respectiv potrivirea unei ontologii cu un text de limbaj natural.

Ontologiile şi dezambiguarea

Diversitatea mijloacelor prin care s-au obţinut ontologii nu garantează faptul se folosesc aceiaşi URI pentru concepte identice aparţinând unor ontologii diferite. Însă fiecare concept are asociat un nume, care poate fi folosit pentru a decide dacă conceptele sunt identice sau nu, chiar şi în cazul în care au URI diferit. Doar că o simplă măsură a similarităţii cuvintelor nu e utilă în cazul omonimelor. De aceea trebuie folosite şi elementele direct legate de conceptul de dezambiguat din ontologie [Mih10e]:

Algoritmul de dezambiguare

funcţia *DezambiguareRecursivă(WORD, SET, ONTOLOGY)* ***este***
  Date: *WORD, SET, ONTOLOGY*
  Precondiţii: *WORD este cuvântul de dezambiguat*
    SET este mulţimea vecinilor săi din ontologia ONTOLOGY
  SENSE ← 0
  max ← 0
  pentru *@fiecare sens i a cuvântului WORD din dicţionar* ***execută***
    @evaluează scorul „overlap" al mulţimii de cuvinte SET faţă de glosa i din dicţionar
    dacă *@scorul > max* ***atunci***
      SENSE ← i
    sfd
  sfp
  dacă *max = 0* ***atunci***
    dacă *@există vecini in ontologia ONTOLOGY ai elementelor din SET neincluşi în SET* ***atunci***
      @adaugă vecinii la SET
      SENSE ← DezambiguareRecursivă(WORD, SET, ONTOLOGY)
    sfd
  sfd
  Rezultate: *SENSE*
  Postcondiţii: *SENSE este indexul sensului corect al cuvântului WORD, 0 altfel*
sf *DezambiguareRecursivă*

Abordările prezentate accentuează rolul pe care tehnicile de prelucrare ale limbajului natural le joacă în rezolvarea problemei asocierii informaţiilor semantice unui text de limbaj natural, dar şi a îmbunătăţirii calităţii unei ontologii.

4.5. Prelucrarea cerinţelor asistată de ontologii

De vreme ce ontologiile îi permit calculatorului să înţeleagă limbajul natural, ele au o mare aplicabilitate în domeniul extragerii informaţiilor. Acest lucru a fost evidenţiat încă din 2003 în [Mae03], unde autorii au propus o metodă pas cu pas de îmbogăţire a ontologiei existente pentru a o adapta extragerii informaţiei dintr-un nou text.

Într-o lucrare mai recentă [Yil07], este prezentată o metodă de extragere a informaţiei supervizată de o ontologie, fără a se utiliza nici o altă resursă de cunoaştere. Totuşi precizia rezultatului va depinde de calitatea ontologiei folosite. În acelaşi timp metoda identifică elementele neutilizate ale ontologiei şi în acest mod calitatea ontologiei se poate îmbunătăţi, precum şi informaţia extrasă.

Selectarea semi-automată a cerinţelor

Scopul lucrării [Mih10f] este de a se extrage un program abstract dintr-un text scris în limbaj natural care reprezintă cerinţele programului. Cu alte cuvinte de a identifica precondiţia, postcondiţia şi lista de variabile.

Subalgoritmul *PrePostSelection(g, pre_list, post_list)* ***este:***
date*: g – un graf de dependenţe direcţionale*
@Identifică lista de verbe VB: VB_list
@Iniţializează lista listelor de cuvinte, Word_ll şi pune în fiecare listă pe prima poziţie verbul VB
@Iniţializează o listă vidă de cuvinte distincte Var_l
Pentru *@fiecare listă L din Word_ll* ***execută****:*
@Identifică traseele din graf care se termină cu o dependenţă „sub” pentru un VB sau un VBD, iar dacă nu există nici o astfel de dependenţă, traseele care se termină cu o dependenţă „obj” pentru VB, respectiv cele care se termină cu o dependenţă „amod”pentru VBN
@ Adaugă toate cuvintele de pe traseu la L
@ Cuvântul NN care se află în relaţia „sub”, „obj” sau „amod” trebuie adăugat listei de variabile Var_l
@ Adaugă la L toate cuvintele conectate direct sau indirect cu ultimul cuvânt (NN)
sfp
rezultate*: pre_list – lista L din Word_ll ce începe cu VB*
post_list – celelalte liste din Word_ll
Var_l – lista variabilelor
sf *PrePostSelection*

Figura 18. Algoritmul de analiză a frazei

Se pot identifica două tipuri de cerinţe: cerinţe care sunt compuse din mai multe fraze, dar şi mai multe cerinţe care compun o singură propoziţie sau frază. În primul caz se pare că problema identificării precondiţiei şi a postcondiţiei e mai uşor de rezolvat, de vreme ce ea poate fi redusă la problema selectării frazelor. În cel de al doilea caz dintr-o singură propoziţie ar trebui extrase una sau mai multe precondiţii şi una sau mai multe postcondiţii. În cel de al doilea caz, însăşi **fraza** trebuie **analizată**. Se va utiliza din nou Stanford Parser-ul [***STS]. Precondiţiilor le corespunde în limbaj natural o propoziţie la trecut sau prezent, iar postcondiţiilor una la viitor. Lista de variabile este reprezentată de subiectele respectivelor propoziţii. Algoritmul corespunzător se află în figura 18. În cazul în care cerinţele sunt exprimate prin mai multe fraze, înainte de a se aplica algoritmul ar trebui analizate frazele, extrase triplete, identificate sinonimele şi cuvintele subordonate,

unindu-se astfel grafurile de dependenţe (vezi subcapitolul 4.2). Dacă volumul cerinţelor e mare, acestea se vor segmenta mai întâi.

Pentru a se obţine într-adevăr specificaţii Z din cerinţe scrise în limbaj natural, nu este suficientă selectarea listelor de cuvinte corespunzătoare precondiţiei/postcondiţiei. Pentru a se obţine scheme Z este necesară utilizarea unei ontologii, care are noduri-bază tipurile de bază şi operaţiile de bază pentru aceste tipuri. De exemplu în „*Generate the first prime number larger than a given natural number n.”* [***Adr], se poate observa că este utilizat tipul de bază *număr natural* (natural number), iar prim (prime) şi *mai mare decât* (larger than) sunt operaţii definite pe mulţimea numerelor naturale, doar *primul* (first) nu este un predicat care aparţine de mulţimea funcţiilor matematice definite pe mulţimea numerelor naturale.

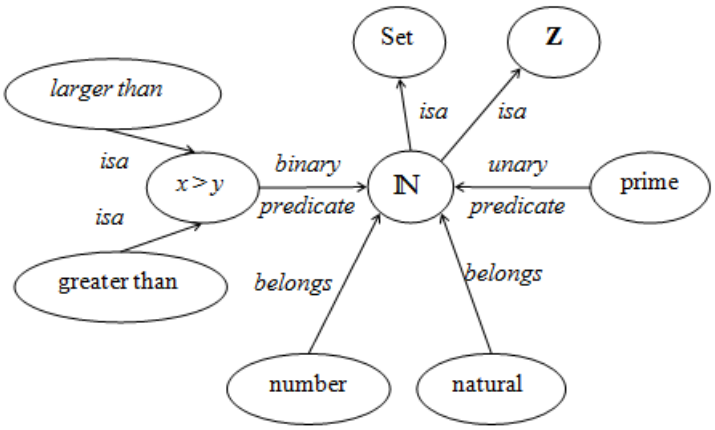

Figura 19. Fragment din ontologia numerelor naturale

În cazul utilizării unei ontologii, prima dată se vor căuta în ontologie cuvintele care reprezintă variabile (vezi Figura 19) şi se vor înlocui cu tipul cel mai apropiat (în exemplu, după ce s-a identificat nodul „natural”, cel mai apropiat tip este ℕ). Apoi se caută şi celelalte cuvinte din listele de cuvinte care exprimă precondiţia şi postcondiţia în sub-ontologiile care au rădăcina tipul identificat. Dacă cuvintele respective sunt identificate în ontologie, atunci predicatul de bază conectat direct la tipul identificat va înlocui cuvântul curent.

În exemplul discutat „*Generate the first prime number larger than a given natural number n.*”, lista de cuvinte identificate care exprimă postcondiţia este *Generate* + *larger* + *number* (the subject) + *the* + *first* + *prime*. *Număr* (Number) poate fi înlocuit de $x \in \mathbb{N}$ şi de asemenea *mai mare* (larger) $x > n$. Calculatorul poate identifica singur aceste tipuri sau predicate de bază şi în acest mod poate asista procesul de transformare a cerinţelor scrise în limbaj natural în specificaţii formale.

Concluzii

Au fost propuse două seturi de metrici pentru evaluarea potrivirii unei ontologii cu un text scris în limbaj natural. Primele se bazează pe evaluarea statistică a numărului de concepte care apar în text, iar cele din urmă pe relaţia de implicaţie logică dintre texte.

Cu toate că din testele efectuate metricile bazate pe implicaţia logică dintre două texte au fost neconcludente, parţial datorită marii diferenţe între dimensiunea ontologiilor analizate şi a textului testat, primul set de metrici a condus spre rezultatele aşteptate.

S-au prezentat două metode de conversie a rezultatului analizei gramaticale ale unui text scris în limbaj natural în triplete ale unei ontologii. Prima metodă propusă furnizează mai puţine triplete dar mai concise, apropiate de calculator, iar cea de a doua

poate furniza triplete mai apropiate de utilizatorul uman. Ele accentuează rolul pe care gramatica îl joacă în procesul de construire a unei ontologii sau de potrivire a unei ontologii la un text.

Totodată s-a prezentat o metodă capabilă să identifice precis cele mai probabil plasate greşit elemente dintr-o ontologie şi să ajute dezvoltatorii să îmbunătăţească calitatea ontologiei.

În final s-a prezentat o metodă semi-automată de extragere a specificaţiilor formale dintr-un text scris în limbaj natural. Ea utilizează Stanford Parser-ul pentru a obţine graful de dependenţe, urmat de un proces de unire a ontologiilor pentru a conecta mini-ontologii, iar la sfârşit pe baza unor principii semantice, sunt extrase propoziţii scrise în limbaj natural care reprezintă precondiţii şi postcondiţii.

V. Aspecte ale utilizării metodelor formale pentru dezvoltarea corectă a programelor

5.1. Asigurarea calităţii softului – o activitate continuă

Este bine cunoscut faptul că din efortul asociat dezvoltării şi utilizării produsului soft, întreţinerea lui reprezintă 60%. Pentru a reduce acest cost, calitatea produsului trebuie garantată încă de la primele faze ale ciclului de dezvoltare. În cazul produselor de mari dimensiuni, întreţinerea lor este imposibilă dacă nu sunt îndeplinite anumite cerinţe de calitate. Măsurarea calităţii softului se face cu ajutorul metricilor soft, existând metrici specifice pentru diferitele faze de dezvoltare a produsului.

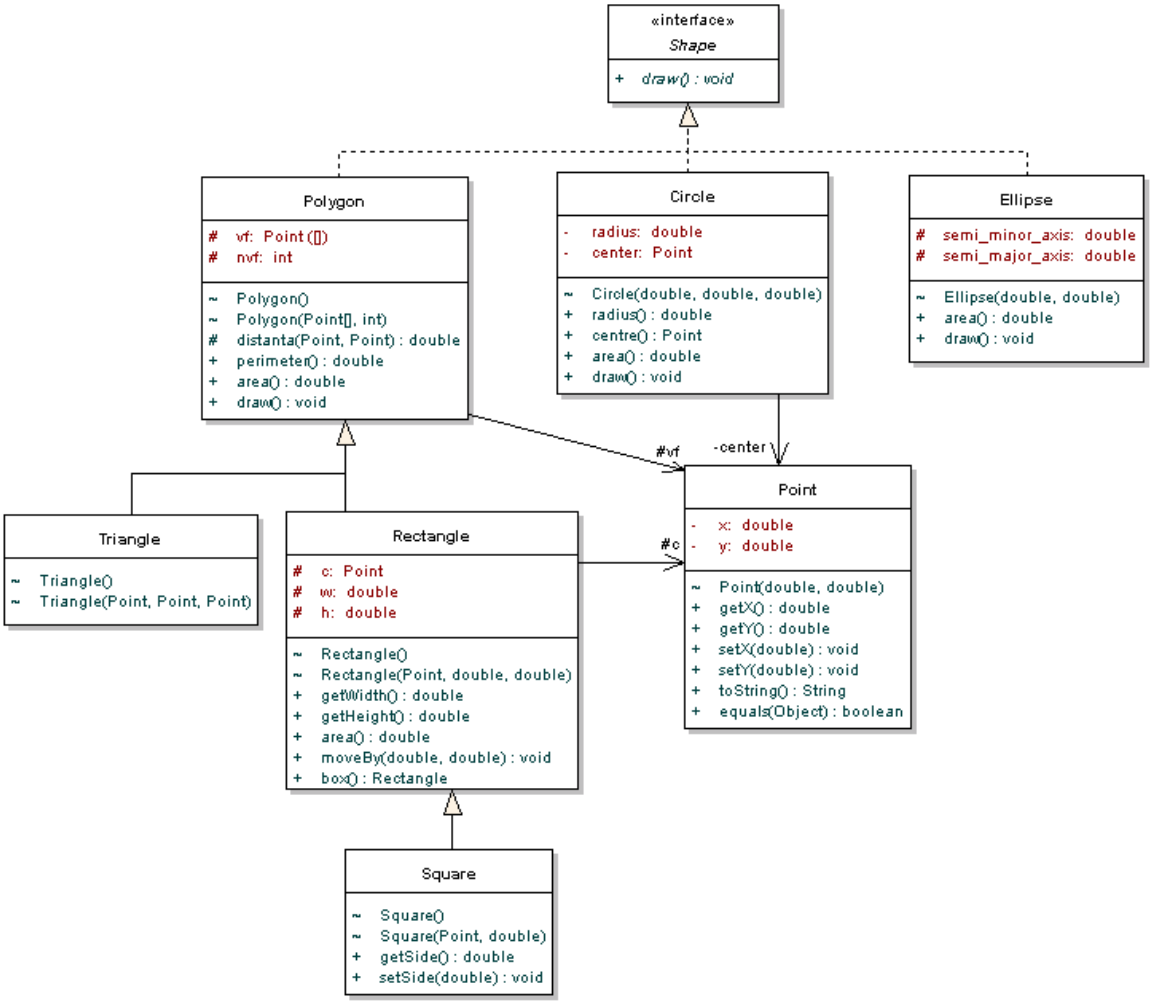

Figura 20. Ierarhia de clase figuri geometrice

O metodă de a asigura calitatea produsului soft este de a-i evalua calitatea în fiecare fază de dezvoltare şi a îmbunătăţi produsul. Pentru a demonstra importanţa evaluării calităţii softului cât mai devreme în procesul de dezvoltare, s-a proiectat o ierarhie de clase [Şer05] (vezi figura 20). În tabelul 4 se află rezultatele evaluării claselor din ierarhie. Se observă că se poate îmbunătăţi această ierarhie de clase, prin derivarea clasei Circle din clasa Ellipse, caz în care complexitatea va scădea prin reducerea

numărului de operaţii şi operatori ai clasei Circle. Totodată valoarea metricii COM ar trebui să fie mai mare, ceea ce înseamnă că ar trebui introduse câteva comentarii suplimentare.

Clasa/Metrica	SIZE (numărul liniilor de cod)	COM (procentul comentariilor)	WMC (greutatea metodelor unei clase)	DIT (adâncimea arborelui de moştenire)	NOC (numărul de „copii”)	LCOM (lipsa coeziunii metodelor)	CBO (cuplare a dintre clase)
Shape	6	33%	0	1	0	-	0
Polygon	37	8%	8	1	2	0	1
Circle	17	17%	5	1	0	0,5	1
Ellipse	34	5%	3	1	0	0	0
Triangle	15	7%	2	2	0	0	1
Rectangle	46	20%	7	2	1	0,46	1
Square	22	9%	4	3	0	0	1
Point	37	8%	9	1	0	0,33	0

Tabelul 4. Rezultatele evaluării metricilor pentru clase

O altă metodă este de a dezvolta soft de calitate superioară cu ajutorul şabloanelor de proiectare sau al metodelor formale. Din păcate nu există un model general valabil pentru a garanta calitatea softului. Pentru fiecare proces de dezvoltare ar trebui specificate cerinţele de calitate de la început, urmărite pe tot parcursul procesului de dezvoltare şi rezolvate discrepanţele în primele faze, când costul de modificare este redus.

5.2. Abordarea multiformală în specificarea sistemelor soft

Sistemele soft de mari dimesiuni prezintă mai multe aspecte. Fiecare aspect necesită aplicarea unui anumit formalism şi verificarea sa formală utilizând unelte specifice. De aceea specificaţiile obţinute prin aplicarea diferitelor metode formale trebuie integrate, rezultând o specificaţie multiformă (integrată sau heterogenă).

Metodele se clasifică în: metode formale orientate pe stare (sau model), orientate pe proprietăţi (axiomatice sau algebrice) şi hibride. Metode formale orientate pe stare sunt: Z [Abr80], VDM [Jon86], B [Joh73], automatele cu stări finite [Gil62], reţelele Petri [Pet62], CCS [Mil80], CSP [Hoa85]. Metodele formale orientate pe proprietăţi sunt: CASL [Ast02], ACT ONE [Ehr83], Anna [Luc85], Larch [Gar93], CLEAR [Bid91], OBJ [Fut85], LOTOS [Eij89]. În general integrarea metodelor formale constă în combinarea a două sau mai multe metode formale de specificare complementare.

În cazul specificării unui sistem complex, se poate aplica o abordare omogenă sau eterogenă. Abordarea omogenă descrie multiplele aspecte ale sistemului soft utilizând un singur formalism, capabil să exprime toate aspectele sistemului. Astfel de specificaţii omogene pot fi realizate sau specificând noi limbaje, sau extinzând unul existent. Abordările eterogene, multiformale, folosesc mai multe formalisme existente pentru a acoperi toate aspectele sistemului specificat. În funcţie de modul de îmbinare sintactică a limbajelor, acest tip de abordare poate fi o integrare puternică a formalismelor, sau o compunere/cuplare a părţilor independente ale specificaţiei, fiecare parte fiind scrisă folosind un alt limbaj.

În cazul combinării sintactice a limbajelor există trei abordări principale. Prima abordare utilizează reprezentări grafice ale comportamentului (reţele Petri, diagrame de stare, sisteme etichetate ale tranziţiilor) şi diferite tipuri de date (specificaţii algebrice, B, Z, VDM). A doua abordare constă în combinarea unei algebre a procesului, cum este CCS sau CSP cu un formalism algebric. Un astfel de multiformalism este LOTOS. A treia abordare încearcă integrarea algebrei procesului (CCS, CSP) cu un formalism orientat pe stări (Z, Object-Z [Smi99], B, VDM). Un astfel de formalism integrat este ZCCS [Gal96].

Premisele unei abordări multiformale sunt: învăţarea metodelor formale, utilizarea metodelor integrate şi a uneltelor necesare. Unul dintre principalele dezavantaje ale metodelor formale, acela de a fi dificil de învăţat datorită notaţiilor, conceptelor şi a metodelor matematice, poate constitui un impediment în cazul unei abordări multiformale. În plus combinând diferite notaţii, pot fi introduse ambiguităţi, incompatibilităţi şi pot să rezulte specificaţii dificil de înţeles. Dar s-a argumentat că prin folosirea mai multor metode, fiecare în cele mai potrivite situaţii, se vor obţine specificaţii mai scurte, clare şi concise faţă de cazul utilizării unui singur formalism [Cio04].

5.3. Reutilizarea specificaţiilor formale

În momentul în care se produce o nouă variantă a unui soft existent, o parte din cod este reutilizat, dar şi specificaţiile, documentaţiile sau alte produse secundare ale procesului de dezvoltare soft pot fi de asemenea reutilizate. Dintre toate tipurile de specificaţii, specificaţiile formale sunt cele mai utile, dar sunt şi cel mai greu de dezvoltat. Reutilizarea lor ar fi foarte utilă. Un alt caz de reutilizare a specificaţiilor formale este cel de construire a unui produs pentru mai multe platforme.

Totodată specificaţiile formale pot fi folosite pentru a identifica elementele reutilizabile. De exemplu în cazul în care există o librărie de componente şi se doreşte producerea uneia noi, dar se urmăreşte reutilizarea câtor mai multe componente din vechea librărie, atunci identificarea componentelor analoage este cel mai uşor de realizat dacă cele două componente au fost specificate formal cu ajutorul aceluiaşi formalism.

Avantajul folosirii unor specificaţii formale este acela că ele sunt într-o formă matematică şi astfel este uşor de identificat existenţa unei funcţii bijective între două specificaţii, demonstrând similaritatea lor. K. Periyasamy şi J. Chidambaram, [Per96] au definit relaţia de identitate sau analogie a specificaţiilor Z din punctul de vedere al declaraţiei, semnăturii şi proprietăţilor. Forma standard a specificaţiilor formale garantează faptul că aceeaşi specificaţie se va afla tot timpul în aceeaşi formă.

Totodată analogia este mijlocul cel mai simplu de învăţare. Astfel specificaţii formale similare pot fi folosite pentru a uşura procesul de învăţare a respectivei metode formale [Mih05].

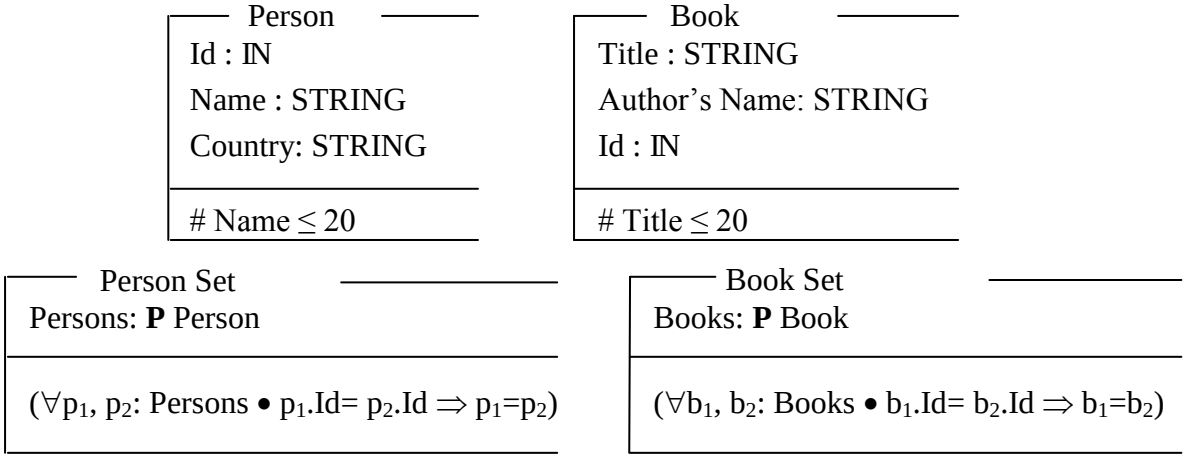

Figura 21. Scheme de date Z

Să presupunem că avem specificaţiile pentru tipurile de date Persoană şi pentru o Mulţime de Persoane, după cum pot fi observate din figura 21. Se doreşte specificarea în Z a entităţii Carte (Book) şi a unei mulţimi de Cărţi (Book Set). În urma studierii cerinţelor legate de Carte, se observă că cele două entităţi sunt similare. Pornind de la modelul

Persoanei, se scriu imediat specificaţiile pentru Carte, după cum se poate observa din figura 21.

Un alt caz de reutilizare a specificaţiilor formale este cazul în care este necesară combinarea a două componente. Pentru a evita introducerea unor erori, se recomandă ca mai întâi să se compună specificaţiile lor. Dacă aceste specificaţii sunt formale, compunerea se va face simplu prin aplicarea unei reguli de compunere, obţinându-se astfel specificaţia componentei rezultat.

Elementele similare vor avea specificaţii similare. Dacă se observă că două elemente sunt similare, se va specifica doar unul, iar apoi în mod analog prin reutilizare se va obţine specificaţia celuilalt element. Similaritatea celor două specificaţii poate constitui un semnal de alarmă pentru client, dacă cele două elemente nu sunt de fapt similare, caz în care cerinţa clientului nu a fost deplin înţeleasă de dezvoltatori.

5.4. O aplicaţie care asistă rafinarea schemelor Z

Având ca scop principal încurajarea folosirii metodelor formale în general şi a limbajului Z în special, s-a realizat o aplicaţie care asistă în mod semi-automat rafinarea schemelor Z, aplicaţie care va fi descrisă în cele ce urmează [Mih10a].

Pentru a putea rafina o schemă, aplicaţia trebuie să permită în primul rând definirea unei scheme Z. În ceea ce priveşte editarea schemelor Z, există deja o astfel de aplicaţie [Gao09]. Aplicaţia proprie permite editarea schemelor Z în mod grafic, utilizatorul putând edita o singură schemă la un moment dat, modificarea realizându-se componentă cu componentă (nume, declaraţii, predicate). Simbolurile speciale pot fi inserate dintr-o listă cu categorii de simboluri [Dil99]. În acelaşi timp pentru a sprijini procesul de învăţare a limbajului Z, în momentul în care un utilizator alege un caracter din lista de simboluri, va apare o scurtă descriere a simbolului împreună cu un exemplu în care respectivul simbol este utilizat.

În momentul adăugării unei noi componente, are loc o analiză sintactică a ei care are ca scop principal identificarea elementelor de bază, conform schemei de notaţie [Mih10a], a listei componentelor identificate deja şi a simbolurilor speciale „?” şi „!”. În cazul în care utilizatorul observă că analizatorul sintactic a identificat în mod greşit un element, el poate interveni pentru a corecta eroarea.

Numele unei scheme noi definite va fi adăugat listei cu nume de scheme. Din această listă schemele pot fi selectate pe rând, câte una sau câte două, în vederea rafinării lor [Woo96].

Dacă se selectează două scheme, acestora li se poate aplica o operaţie binară, cum ar fi conjuncţia, disjuncţia şi compunerea (vezi figura 22). Dacă se selectează o singură schemă, acesteia i se poate aplica o operaţie unară, cum este negarea sau decorarea. Concomitent întrucât majoritatea componentelor unei scheme sunt elemente matematice, acestora li se pot aplica diferite teoreme sau proprietăţi matematice. Toate aceste operaţii se pot efectua în mod asistat, aplicaţia oferind un exemplu de folosire a operaţiei selectate, utilizatorul trebuind să introducă doar elementele noi. Colecţia de operaţii de rafinare a elementelor unei scheme se îmbogăţeşte în mod continuu, prin adăugarea de către utilizator a unor operaţii noi. Prima folosire a unei operaţii noi va fi memorată şi oferită ca exemplu aplicaţiilor viitoare. Operaţiile de rafinare care pot fi aplicate componentei selectate din schema curentă se identifică pe baza relaţiei de similaritate dintre două componente.

Definiţie: Două componente ale unor scheme Z sunt similare dacă sunt compuse din acelaşi număr de elemente de bază, aflate în aceeaşi ordine şi având aceleaşi tipuri.

Cea mai importantă operaţie de rafinare care poate fi realizată în mod asistat este transformarea unei scheme Z într-un program abstract. Pentru a transforma o schemă Z într-un program abstract, trebuie identificate: cadrul, care conţine lista tuturor variabilelor

din schemă, precondiţia şi postcondiţia. Cadrul a fost identificat anterior, prin analiza sintactică a elementelor constituente ale unei scheme Z. Trebuie doar selectate variabilele din lista de elemente. Precondiţia este conjuncţia predicatelor din schemă care conţin variabile al căror nume include simbolul „!”, iar postcondiţia e conjuncţia predicatelor în care numele variabilei se termină cu „?”. Bineînţeles utilizatorul poate interveni pentru corectarea erorilor. Programul abstract obţinut poate fi salvat într-un fişier text, în vederea prelucrării ulterioare.

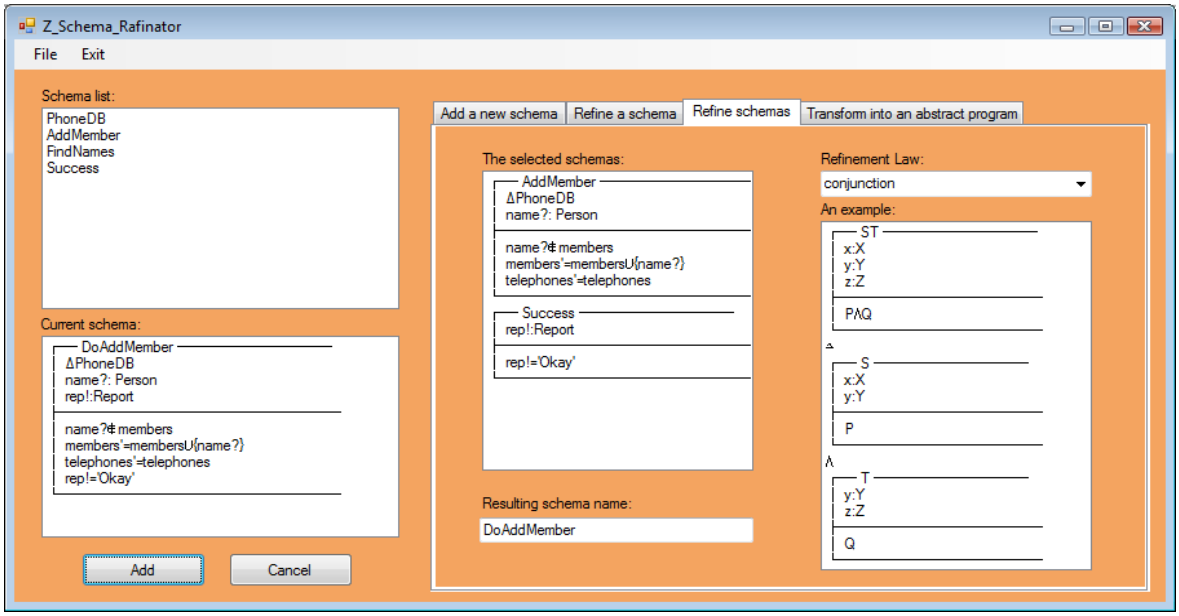

Figura 22. Conjuncţia a două scheme Z

Aplicaţia are în principal scop didactic şi a fost dezvoltată în C#. Programele abstracte obţinute din schemele Z pot fi rafinate în continuare, după cum se poate observa din subcapitolul următor.

5.5. O aplicaţie care asistă procesul de rafinare a codului din specificaţii

În [Mih06a] s-a prezentat un mini instrument CASE, „Asistent rafinare”, care permite rafinarea asistată a codului din programe abstracte, instrument descris în cele ce urmează. Programele abstracte sunt precizate prin lista explicită a variabilelor care primesc valori *w*, predicatul de intrare sau precondiţia, notat cu *pre*, şi predicatul de ieşire sau postcondiţia, notat cu *pos*: „*w:[pre,pos]*”. Rafinarea presupunea trecerea de la o stare la alta prin aplicarea unei reguli de rafinare. Regulile care pot fi aplicate în procesul asistat de rafinare sunt: regula atribuirii, regula alternanţei, regula compunerii secvenţiale şi regula iteraţiei.

Aplicaţia oferă următoarele facilităţi: utilizarea automată a regulilor de mai sus pentru un subprogram abstract, după ce utilizatorul a identificat toate elementele noi necesare procesului de rafinare. Utilizatorii pot modifica însă şi manual liniile de cod. Întregul proces de rafinare este memorat într-un fişier istoric „log.txt”, ceea ce permite reutilizarea procesului de rafinare pentru cazuri similare. Totodată se poate salva forma curentă a programului sau se poate încărca un program salvat anterior.

Această aplicaţie uşurează munca dezvoltatorilor, mai ales în cazul unor programe de dimensiuni medii sau mari, care necesită aplicarea repetată a regulilor rafinării.

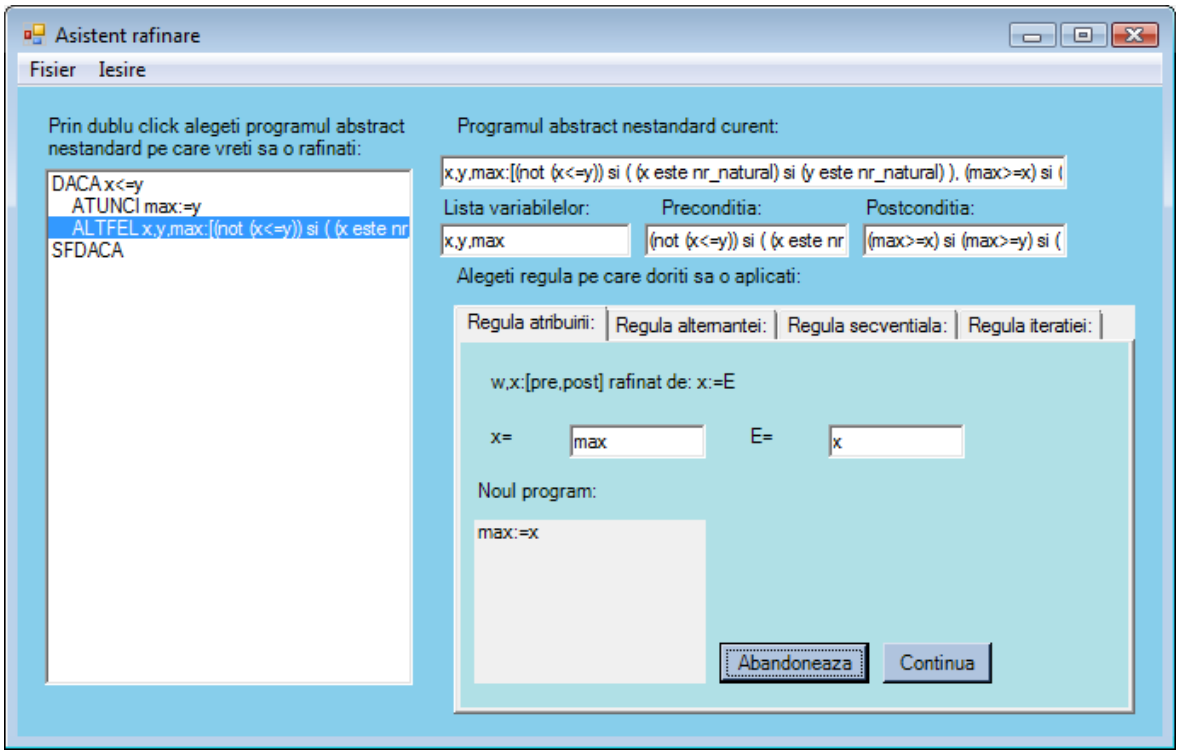

Figura 23. Fereastra de bază a Asistentului rafinării

5.6. Simplificarea codului prin prelucrarea automată a expresiilor condiţionale

Un alt mini instrument CASE este propus în [Mih06b] şi prezentat într-un subcapitol din [Lup08/09] favorizează îmbunătăţirea calităţii codului prin aplicarea simplificării funcţiilor booleene asupra structurilor condiţionale complicate existente în cod. Aceasta deoarece fiecare expresie condiţională simplă are doar două valori de adevăr: adevărat, care poate fi notat cu *1*, respectiv fals, notat cu *0*. Dacă se notează toate expresiile condiţionale simple cu variabile x_i, $i=\overline{1,n}$, unde n reprezintă numărul condiţiilor, expresia condiţională complexă poate fi înlocuită de funcţia booleană $f(x_1, x_2,...,x_n)$. Aşadar problema simplificării unei expresii condiţionale se reduce la simplificarea unei funcţii booleene, proces care a fost automatizat în unealta numită BOOFS (BOOlean Function Simplificator).

Metoda folosită pentru specificare este metoda lui Quine-McClusky combinată cu metoda lui Moisil [Tăt99, Lup08/09]. Această metodă se aplică însă doar formei canonice disjunctive a unei funcţii booleene, formă la care se ajunge după aplicarea unui proces de normalizare.

Pentru a putea aprecia eficient îmbunătăţirile obţinute prin aplicarea metodei de simplificare, unealta BOOFS permite evaluarea automată a câtorva metrici care se bazează pe numărul operatorilor (conectivelor) şi al operanzilor, metrici care sunt definite în continuare: *numărul de operanzi distincţi*, *numărul total de operanzi*, *numărul de operatori distincţi*, *numărul total de operatori*, *suma priorităţilor operatorilor distincţi*, *suma priorităţilor tuturor operatorilor*.

Valoarea acestor metrici este un număr natural, care ar trebui să descrească în cazul simplificării. Pentru metricile 1, 3 şi 5 descreşterea valorii metricii este evidentă, însă pentru metricile 2 şi 4, în unele cazuri valoarea lor creşte. Cea mai bună metrică pare a fi metrica 6. Priorităţile folosite în metricile 5 şi 6 sunt de la 1 la 5, corespunzătoare

operatorilor: ¬, ∧, ∨, →, ↔. Aceste priorităţi sunt folosite ca şi criteriu şi în codul aplicaţiei BOOFS.

De exemplu, pentru expresia condiţională
((*x* < *y*) *and* (*y* <= *z*) *and* (*z* < 5)) *or* ((*x* < *y*) *and* (*y* >= 5) *and* (*z* < 5)),
care s-a simplificat la ((*x* < *y*) *and* (*z* < 5)), valorile celor 6 metrici au scăzut de la 3, 6, 3, 6, 6, 12, la 2, 2, 1, 1, 2, 2.

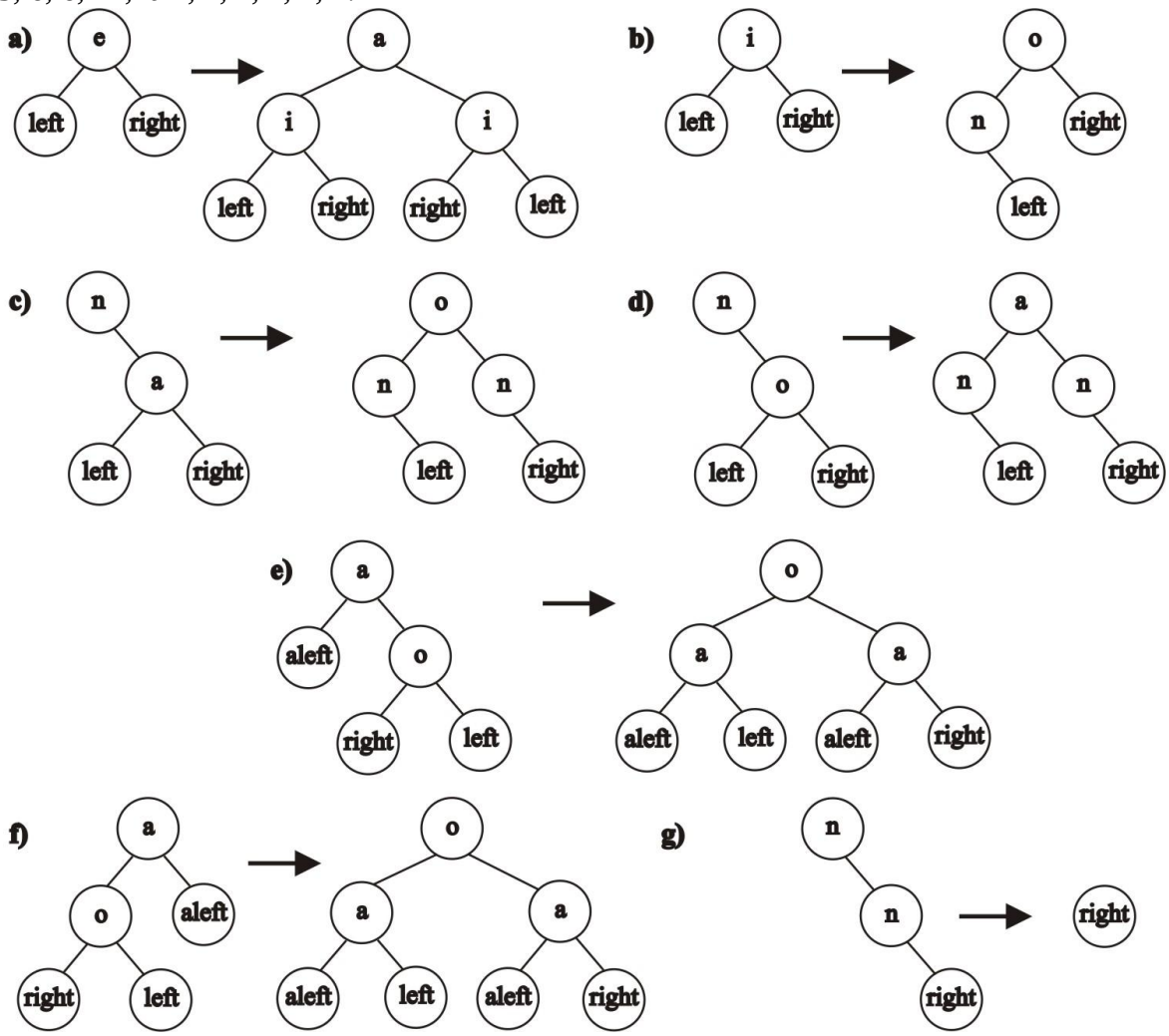

Figura 24. Operaţiile de aducere a funcţiei booleene la forma disjunctivă pe arbore binar: a) asigură eliminarea echivalenţei, b) elimină implicaţia, c) şi d) sunt legile lui DeMorgan, e) şi f) reprezintă distributivitatea (la dreapta respectiv la stânga), iar g) reprezintă legea dublei negaţii.

Aplicaţia BOOFS a fost realizată în Java. Execuţia începe prin deschiderea unui fişier de cod sursă, din care se va putea selecta o expresie condiţională care va fi simplificată. Deoarece aplicaţia nu poate identifica automat expresiile condiţionale simple, ele vor trebui indicate manual. Simultan, întrucât aplicaţia funcţionează indiferent de limbajul în care este scris codul sursă, utilizatorul va indica modul în care apar cele cinci conective logice. Conectivele se memorează prin litere: *n*(¬), *a*(∧), *o*(∨), *i*(→), *e*(↔), iar operanzii prin x_1, x_2, x_3 şi aşa mai departe. Funcţia booleană obţinută trebuie normalizată. Normalizarea s-a efectuat pe arbore, prin aplicarea operaţiilor din figura 24, într-un algoritm back-tracking. Pentru a transforma funcţia din forma infixată în arbore, se va transforma mai întâi în forma poloneză postfixată prin intermediul unei stive. Tot utilizând o stivă se va transforma forma poloneză postfixată în arbore. După aducerea arborelui la

forma sa disjunctivă, se va obţine forma infixată disjunctivă a funcţiei booleene printr-o parcurgere în inordine a arborelui binar.

Se construieşte mulţimea suport, care conţine *n*-uple cu componente egale cu 0 sau 1. Metoda lui Quine-McClusky este folosită pentru obţinerea mulţimii monoamelor maximale. În *n*-uplele corespunzătoare s-a folosit cifra 2 pentru a marca variabilele simplificate. Metoda lui Moisil este folosită pentru a identifica forma simplificată. Metoda lui Moisil se bazează pe transformarea unei forme normale conjunctive într-una disjunctivă, transformare realizată din nou pe arbore.

VI. Concluzii

În prezent nevoia de corectitudine este din ce în ce mai pregnantă. Dacă un produs soft nu îşi respectă specificaţiile, el nu va fi acceptat de client. Existenţa unor specificaţii complete, corecte şi clare este esenţială. Astfel de specificaţii sunt specificaţiile formale.

Din păcate învăţarea şi utilizarea unei metode formale nu este uşoară şi necesită timp. Este de un real ajutor existenţa unei unelte capabile să asiste elaborarea unor astfel de specificaţii. O astfel de unealtă, care permite construcţia specificaţiilor Z, compunerea lor şi transformarea lor în programe abstacte este prezentată în acestă lucrare.

Există mai multe tipuri de specificaţii formale, de exemplu unele sunt orientate pe procese, altele pe date. Sistemele complexe necesită o specificaţie formală care să surprindă toate aspectele aplicaţiei, necesitând o abordare multiformală a specificaţiei. Specificaţiile formale pot fi şi reutilizate cu succes, mai ales în cazul dezvoltării unei noi versiuni a unui program soft a cărui specificaţie formală există deja.

Pornindu-se de la specificaţii formale, se poate genera cod corect prin rafinare pe bază de reguli. Deşi procesul de specificare formală şi rafinare poate părea destul de greu de realizat, costurile totale ale dezvoltării sunt reduse. Existenţa unor unelte care să asiste şi să automatizeze o parte din acest proces este binevenită. O astfel de unealtă este prezentată în această lucrare. O altă unealtă realizată simplifică prin metode logice expresiile condiţionale din cod.

Pe de altă parte s-ar putea ca specificarea formală a cerinţelor clienţilor să nu poată fi realizată de aceştia împreună cu dezvoltatorii. În acest caz dezvoltatorii pot utiliza aplicaţii de prelucrare a limbajului natural pentru a-şi uşura munca, cum ar fi aplicaţii capabile să dezambigueze cuvintele, să împartă textul în segmente sau să realizeze sumarul.

Ontologiile sunt candidate excelente pentru o prelucrare mai complexă a limbajului natural, cum e cea de selectare a cerinţelor pentru obţinerea specificaţiilor. Totuşi, ontologia folosită în acest scop trebuie aleasă cu grijă.

Lucrarea de faţă prezintă diferite aspecte ale utilizării metodelor formale în dezvoltarea corectă a programelor, dar conţine şi o parte importantă de prelucrare a limbajului natural şi ontologii. Însă toate metodele şi aplicaţiile prezentate au ca scop facilitarea utilizării metodelor formale în dezvoltarea corectă a programelor.

VII. Bibliografie

[Abr80] Abrial J.-R., *The Specification Language Z: Syntax and Semantics*, Oxford University Computing Laboratory, Programming Research Group, Oxford, U.K., aprilie 1980.

[Alf85] Alford M., *SREM at the Age of Eight: The Distributed Computer Design System*, IEEE Computer, Vol. 18, Nr. 4, aprilie 1985, pg. 36-46.

[Ast02] Astesiano E., Bidot M., Krieg-Brückner B., Mosses P.D., Sannella D. şi Tarlecki A., *CASL: The Common Algebraic Specification Language*, Theoretical Computer Science, 286(2), 2002, pg. 153-196.

[Bac00] Back T., Fogel D.B. şi Michalewicz Z. (editori), *Evolutionary Computation: Basic Algorithms and Operators*, Vol. 1 şi *Evolutionary Computation: Advanced Algorithms and Operators* Vol. 2, Institute of Physics Publishing, Philadelphia, PA, 2000.

[Bal85] Balzer R., *The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines.*, NATO Science Committee Conference, 1969.

[Ban03] Banerjee S. şi Pedersen T., *Extended Gloss Overlaps as a Measure of Semantic Relatedness*, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexic, 9-15 august 2003, pg. 805-810.

[Bar99] Barzilay R. şi Elhadad M., *Using lexical chains for Text summarization*, editori J. Mani şi M. Maybury, Advances in Automatic Text Summarization, MIT Press, 1999.

[Ber98] Berners-Lee T., Fielding R. T. and Masinter, L., *Uniform resource identifiers (URI): Generic syntax*. RFC 2396, IETF, 1998.

[Bid91] Bidoit M., Kreowski H-J., Lescanne P., Orejas F. şi Sanella D., *Algebraic System Specification and Development*, Lecture Notes in Computer Science, Vol. 501, Springer-Verlag, 1991.

[Bra85] Brachman R. şi Schmolze J., *An Overview of the KL-ONE Knowledge Representation System*, Cognitive Science, vol. 9, nr. 2, 1985, pg. 171-216, http://nlp.shef.ac.uk/kr/papers/klone.ps.

[Bra05] Brank J., Grobelnik M. şi Mladenić D., *A Survey of Ontology Evaluation Techniques*, Proceedings of the Conference on Data Mining and Data Warehouses (SIKDD 2005).

[Bur95] Burgess C.J., *The Role of Formal Methods in Software Engineering Education and Industry*, Proceedings of the 4th Software Quality Conference http://www.cs.bris.ac.uk/Tools/Reports/Abstracts/1995-burgess-3.html.

[Che76] Chen P., *The Entity-Relationship Model – Towards a Unified View of Data*, ACM Transactions on Database Systems, Vol. 1, Nr. 1, martie 1976, pg. 9-36.

[Che06] Chen R., Lee Y. şi Pan R., *Adding New Concepts on the Domain Ontology Based on Semantic Similarity*, International Conference on Business şi Information (BAI 2006), Singapore, 12-14 iulie 2006, http://bai2006.atisr.org/CD/Papers/2006bai6169.pdf.

[Cim05] Cimiano P. şi Voelker J., *Text2Onto - A Framework for Ontology Learning and Data-driven Change Discovery*, Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB), Alicante, Spania, 2005.

[Cio04] Ciobotariu-Boer, V. şi **Mihiş, A.D.**, *The Multiformalism Approach in Software Specifications*, Proceedings of the Symposium „Zilele Academice Clujene", Computer Science Section, Faculty of Mathematics and Computer Science, "Babes-Bolyai" University, Cluj-Napoca, România, Litografia Universităţii "Babeş-Bolyai", Editor: Prof. dr. Militon Frenţiu, 2004, pg. 21-26.

[Cla96] Clarke E.M. şi Wing J.M., *Formal Methods: State of the Art and Future Directions*, ACM Computing Surveys, 1996, http://www.cs.cmu.edu/ CMU-CS-96-178.

[Cor05] Corley C. şi Mihalcea R., *Measuring the semantic similarity of texts*, Proceedings of the ACL Workshop on Empirical Modelling of Semantic Equivalence and Entailment, Ann Arbor, iunie 2005, pg. 13-18.

[Dah06] Dahab M., Hassan H. şi Rafea A., *TextOntoEx: Automatic Ontology Construction from Natural English Text*, International Conference on Artificial Intelligence and Machine Learning (AIML-06), Sharm El Sheikh, Egipt, 13-15 iunie 2006, http://www.icgst.com/con06/aiml06/Final_Articles/P1120615104.pdf.

[DeM78] DeMarco T., *Structured Analysis and System Specification*, Yourdon Press, New York, 1978.

[Dic45] Dice L.R., *Measures of the amount of ecologic association between species*, Ecology, 26(3), 1945, pg. 297–302.

[Dil99] Diller A., *Z: An Introduction to Formal Methods*, a doua ediţie, John Wiley & Sons, aprilie 1999.

[Dij75] Dijkstra E.W., *Guarded commands, nondeterminacy and formal derivation of programs*, Comm. ACM, Vol. 18(1975), Nr. 8, pg. 453-457.

[Doa03] Doan A., Madhavan J., Dhamankar R., Domingos P. şi Halevy A., *Learning to match ontologies on the Semantic Web*, The VLDB Journal – The International Journal on Very Large Data Bases, vol. 12, issue 4, noiembrie 2003, pg. 303-319, http://www.cs.washington.edu/homes/pedrod/papers/vldbj04.pdf.

[Dro89] Dromey G., *Program Derivation. The Development of Programs from Specifications*, Addison Wesley, 1989.

[Ehr83] Ehrig H., Fey W. şi Hansen H., *ACT ONE: An algebraic specification language with two levels of semantics*, Technical Report No. 83-03, Technische Universität Berlin, 1983.

[Eij89] van Eijk P.H.J., Vissers C.A. şi Diaz M., *The formal description technique LOTOS*, Elsevier Science Publisher B.V., 1989.

[Eng05] Engel L., Jaeger M. şi Mühl G., *Search and Evaluation of Ontologies for Semantic Web Services in the Internet*, IADIS International Conference WWW/Internet 2005 Lisabona, Portugalia, 19-22 octombrie 2005, http://www.iadis.net/dl/final_uploads/200507C050.pdf.

[Euz04] Euzenat J. şi Valtchev P., *Similarity-based ontology alignment in OWL-lite*, Proc. 16th European Conference on Artificial Intelligence (ECAI 2004), 2004, pg. 333–337.

[Euz07] Euzenat J. şi Shvaiko, P., *Ontology Matching*, Springer, New York, 2007.

[Flo67] Floyd R.W., *Assigning meanings to programs*, Proc. Symposium in Applied Mathematics, Schwartz J.T. (Ed.), Am.Math.Soc., Vol. 19, 1967, pg. 19-32.

[Fre06] Frenţiu M., Pop H.F., *Fundamentals of Programming*, Presa Universitară Clujeană, Cluj-Napoca, 2006.

[Fut85] Futatsugi F., Goguen J.A., Jouannaud J.P. şi Meseguer J., *Principles of OBJ2*, Annual Symposium on Principles of Programming Languages, Proceedings of 12th

ACM SIGACT-SIGPLANT symposium on Principles of programming languages, New Orleans, Louisiana, Statele Unite ale Americii, 1985, pg. 52–66.

[Gab06] Gabbar H.A. (editor), *Modern Formal Methods and Applications*, Springer, 2006.

[Gal96] Galloway A. şi Stoddart B., *Integrated Formal Methods*, Research Report, Institut de Recherche en Informatique de Nantes, 1996.

[Gan79] Gane C. şi Sarson T., *Structured Analysis: Tools and Techniques*, Prentice Hall, Englewood Cliffs, NJ, 1979.

[Gan05b] Gangemi A., Catenacci C., Ciaramita M. şi Lehmann J., *Ontology Evaluation and Validation. An integrated formal model for the quality diagnostic task*, Technical report, ISTC-CNR, Lab. for Applied Ontology, http://www.loa-cnr.it/Files/OntoEval4OntoDev_Final.pdf.

[Gao09] Gao X., *The Design and Implementation of Z Language Editor*, Algorithms and Architectures for Parallel Processing, vol. 5574/2009, Springer, iulie 2009, pg. 684-692.

[Gar93] Garland S.J., Guttag J.V. şI Horning J.J, *An Overview of Larch*, Lecture Notes in Computer Science, Vol. 693, Functional Programming, Concurrency, Simulation and Automated Reasoning: International Lecture Series 1991-1992, McMaster University, Hamilton, Ontario, Canada, 1993, pg. 329-348.

[Gil62] Gill A., *Introduction to the Theory of Finite-state Machines*, McGraw-Hill, 1962.

[Gli05] Glickman, O., Dagan, I. şi Koppel, M., *Web Based Probabilistic Textual Entailment*, Proceedings of the PASCAL Challenges Workshop on Recognizing Textual Entailment, 2005, pg. 33-36.

[Gri81] Gries D., *The Science of Programming*, Texts and Monographs in Computer Science, 1st ed. 1981. 5th printing, Springer Verlag, Berlin, 1981.

[Gru93] Gruber T., *A Translation Approach to Portable Ontology Specifications*, Knowledge Acquisition, vol. 5, nr. 2, 1993, pg. 199–220.

[Har97] Harabagiu S. şi Moldovan D., *TextNet – a textbased intelligent system*, Natural Language Engineering, 3(2), 1997, pg. 171-190.

[Hea93] Hearst M., *TextTiling: A Quantitative Approach to Discourse Segmentation*, Technical Report 93/24, University of California, Berkeley, 1993.

[Hei96] Heitjmeyer C.L., Jeffords R.D. şi Labaw B.G., *Automated Consistency Checking of Requirements Specifications*, ACM Trans. on Software Engineering and Methodology, 5(1996), nr. 3, pg. 231-261.

[Hoa85] Hoare C.A.R., *Communicating Sequential Processes*, Prentice Hall International, Englewood Cliffs, NJ, 1985.

[Hol96] Holloway C.M., *Why Engineers Should Consider Formal Methods*, NASA Langley Research Center, http://shmesh.larc.nasa.gov/cmh.html, 1996.

[Hov03] Hovy E., *Text summarization*, The Oxford Handbook of Computational Linguistics, editor Mitkov R., Oxford University Press, Chapter 32, 2003.

[Jac901] Jaccard P., *Distribution de la flore alpine dans le bassin des Dranses et dans quelques régions voisines*, Bulletin de la Société Vaudoise des Sciences Naturelles 37, 1901, pg. 241-272.

[Joh73] Johnson S.C. & Kernighan B.W., *The Programming Language B*, Technical Report CS TR 8, Bell Labs, ianuarie 1973, Prentice Hall, Englewood Cliffs, NJ, 1986.

[Jon86] Jones C.B., *Systematic Software Development Using VDM*, Prentice Hall, Englewood Cliffs, NJ, 1986.

[Kam87] Kampen G. R., *An Eclectic Approach to Specification*, Proceedings of the Fourth International Workshop on Software Specification and Design, Monterey, CA, aprilie 1987, pg. 178-182.

[Kaw04] Kawtrakul A., Suktarachan M. şi Imsombut A., *Automatic Thai Ontology Construction and Maintenance System*, Workshop on Papillon 2004, Grenoble, Franța, http://www.moac.go.th/knowledgebase/uploadfile/42808973.pdf

[Kle03] Klein D. şi Manning C. D., *Fast Exact Inference with a Factored Model for Natural Language Parsing*, Advances in Neural Information Processing Systems 15 (NIPS 2002), Cambridge, MA: MIT Press, 2003, pg. 3-10.

[Kŏu06] Kŏuylekov M. şi Magnini B., *Tree Edit Distance for Recognizing Textual Entailment: Estimating the Cost of Insertion*, Proceedings of the Second PASCAL Challenges Workshop on Recognizing Textual Entailment, Veneţia, Italia, 2006.

[Les86] Lesk M., *Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone*, Proceedings of the 1986 SIGDOC Conference, Association for Computing Machinery, New York, 1986, pg. 24−26.

[Lew04] Lewis D.D., Yang Y., Rose T. şi Li F., *RCV1: A New Benchmark Collection for Text Categorization Research*, Journal of Machine Learning Research, Vol. 5, 2004, pg. 361-397.

[Luc85] Luckham D.C. şi von Henke F.W., *An Overview of Anna, a Specification Language for Ada*, IEEE Software, Vol. 2, Nr. 2, martie 1985, pg. 9-22.

[Lup08] Lupea M., **Mihiş A.D.**, *Logici clasice şi Circuite logice. Teorie şi exemple*, S.C. Albastră Casa de Editura S.R.L., Cluj-Napoca, 2008, 223 pg.

[Lup09] Lupea M. şi **Mihiş A.D.**, *Logici clasice şi Circuite logice. Teorie şi exemple*, S.C. Albastră Casa de Editura S.R.L., Cluj-Napoca, 2009, 223 pg.

[Mae03] Maedche, A., Neumann, G. şi Staab, S., *Bootstrapping an Ontology-based information extraction system*, in Intelligent exploration of the web, Editori: Piotr S. Szczepaniak, Javier Segovia, Janusz Kacprzyk, Lotfi A. Zadeh, Physica-Verlag GmbH Heidelberg, Germania, 2003, pg. 345 - 359.

[Mar97] Marcu D., *From discourse structure to text summaries*, Proceedings of the ACL/EACL '97 Workshop on Intelligent Scalable Text Summarization, Madrid, Spania, pg. 82-88.

[Mar08] de Marneffe M.-C., şi Manning C. D., *The Stanford typed dependencies representation*, COLING Workshop on Cross-framework and Cross-domain Parser Evaluation, Manchester, United Kingdom, 2008, http://nlp.stanford.edu/pubs/dependencies-coling08.pdf.

[May09] Maynard D., Funk A. and Peters W., *Using Lexico-Syntactic Ontology Design Patterns for ontology creation and population*, WOP 2009 – ISWC Workshop on Ontology Patterns, Washington, 2009.

[Mih08a] Mihăilă A.A, **Mihiş A.D.** şi Mihăilă C.F., *A Genetic Algorithm for Logical Topic Text Segmentation*, International Conference on Digital Information Management, IEEE Computer Society Press, 978-1-4244-2917-2/08, IEEE Xplore, 2008, pg. 500-505.

[Mih05] **Mihiş A.D.**, *Formal Specification Reuse*, Proceedings of Symposium „Colocviul Academic Clujean de Informatică", 2005, pg. 202-206.

[Mih06a] **Mihiş A.D.**, *An Application that Assist StepWise Refinement*, Proceedings of Symposium „Zilele Academice Clujene", 2006, pg. 143-147.

[Mih06b] **Mihiş A.D.**, Chisăliţă-Creţu C., Mihăilă C.A., Şerban C.A., *BOOFS-A Tool That Supports Simplifying Conditional Expressions using Boolean Functions Simplification Methods*, Studii şi Cercetări Ştiinţifice, Seria Matematică, 16 (2006), Supplement, Proceedings of ICMI 45, editori: Mocanu Marcelina & Nimineţ Valer, Bacău, 18-20 Septembrie, 2006, pg. 493-502.

[Mih07] **Mihiş A.D.**, *Chain Algorithm used for Part of Speech Recognition*, Mathematical Reviews, www.ams.org/mathscinet/, Studia Universitatis "Babes-Bolyai", Informatica: KEPT 2007, pg. 89-95.

[Mih08b] **Mihiş A.D.**, *A Simple Ontology based on Text Entailment Directional Relationship*, Proceedings of KM-03 Knowledge Management: Projects, Systems, and Technologies, 23-25 octombrie 2008, Bucureşti, România, Editura ASE Bucureşti, ISBN: 978-606-505-124-9, pg. 531-534.

[Mih08c] **Mihiş A.D.**, *Natural Language Processing Methods Used in Requirement Analysis*, „Zilele Informaticii Economice Clujene“, Mediamira Science Publisher, Cluj-Napoca, editori: prof. dr. Niţchi Ştefan şi alţii, 2008, pg. 251-258.

[Mih08d] **Mihiş, A.D.**, *Various Types of Logical Text Tiling*, Zilele Academice Clujene, Presa Universitară Clujeană, editor: Prof. dr. Frenţiu Militon, 2008, pg. 129-133.

[Mih10a] **Mihiş A.D.**, *A Tool for Refinenent of Z Schemas*, Proceedings of Symposium „Zilele Academice Clujene”, 2010, pg. 64-67.

[Mih10b] **Mihiş A.D.**, *The Evaluation of Ontology Matching versus Text*, Informatica Economică/Economy Informatics, Categ. CNCSIS B+, vol. 14, nr. 4, 2010, pg. 147 – 155.

[Mih10c] **Mihiş A.D.**, *Ontology Learning from Text Based on the Syntactic Analysis Tree of a Sentence*, The 5th International Conference on Virtual Learning (ICVL - 2010), Târgu-Mureş, Editura Universităţii Bucureşti, Editor: Vlada Marian, Albeanu Grigore, Popovici Dorin Mircea, ISSN: 1844-8933, 1842-4708, http://c3.icvl.eu/2010, 2010, pg. 128-134, articol ce a obţinut premiul special „Intel®Education”.

[Mih10d] **Mihiş A.D.**, *Similarity Used In The Identification Of The Need To Refactoryze An Ontology*, Knowledge Management: Projects, Systems and Technologies (KM – conference), "Carol I" National Defence University Publishing House, Editor: Toma Pleşanu, Constanţa Bodea, Luiza Kraft, 2010, pg. 23-28.

[Mih10e] **Mihiş A.D.**, *The Role of Disambiguation in Ontology Evaluation*, The Seventh International Conference on Applied Mathematics (ICAM7), Minisymposium 6-Software Engineering - Principles and Practices, Baia Mare, 1-4 septembrie 2010.

[Mih10f] **Mihiş A.D.**, *Ontology Assisted Formal Specification Extraction from Text*, Studia Universitatis Babeş-Bolyai, Informatica, Vol. 55, No. 4, 2010, Cluj-Napoca, pg. 103-113.

[Mil80] Milner R., *A Calculus of Communicating Systems*, Springer-Verlag, 1980.

[Mon01] Monz C. şi de Rijke M., *Light-Weight Entailment Checking for Computational Semantics*, Proceedings of the third workshop on inference in computational semantics (IcoS-3), editori Blackburn P. şi Kohlhase M., 2001.

[Mor06] Morita T., Fukuta N., Izumi N. şi Yamaguchi T., *DODDLE-OWL: A Domain Ontology Construction Tool with OWL*, Proceedings of the 1st Asian Semantic Web Conference, Lecture Notes in Computer Science, vol. 4185, Beijing, China, 2006, pg. 537-551,

http://iws.seu.edu.cn/resource/Proceedings/ASWC/2006/papers/4185/41850537.pdf.

[Mor91] Morris J., Hirst G., *Lexical Cohesion Computed by Thesaural Relations as an Indicator of the Structure of Text*, Computational Linguistics, Vol. 17, Nr. 1, 1991, pg. 21-48.

[Mor90] Morgan C., *Programming from Specifications*, Programming Research Group, University of Oxford, Prentice Hall International (UK), 1990.

[Nel79] Nelson W.F. şi Kucera H., *Brown corpus manual*, Dept. of Linguistics, Brown University 1979, http://icame.uib.no/brown/bcm.html.

[Oră06] Orăşan C., *Comparative evolution of modular automatic summarization systems using CAST*, Teză de doctorat, University of Wolverhampton, UK, 2006.

[Per96] Periyasamy K., Chidambaram J., *Software Reuse Using Formal Specification of Requirements*, Proceedings of the 1996 conference of the Centre for Advanced Studies on Collaborative Research, IBM Press, Toronto, Canada, 1996 http://www.cs.ubc.ca/local/reading/proceedings/cascon96/pdf/periyasc.pdf

[Pet62] Petri C.A., *Kommunikation mit Automaten*, Ph. D. Dissertation, University of Bonn, Germany, 1962.

[Pol09] Pollock J. T., *Semantic Web for Dummies*, Wiley Publishing, Indianapolis, 2009.

[Rad02] Radev D., Hovy E. şi McKeown K., *Introduction to the Special Issues on Summarization*, Computational Linguistics, Vol. 28, Nr. 8, 2002, pg. 399-408.

[Ric91] Richie D., *The Koan*, Zen Inklings. Some Stories, Fables, Parables and Sermons, New York and Tokyo: Weatherhill,1991, pg. 25-27.

[Ros85] Ross T. D., *Applications and Extensions of SADT*, IEEE Computer, Vol. 18, No. 4, aprilie 1985, pg. 25-34.

[Rus95] Rushby J., *Model Checking and Other Ways of Automating Formal Methods*, http://www.csl.sri.com/Reports/html/SQW95.html.

[San07] Sanchez D. şi Moreno A., *Semantic disambiguation of taxonomies*, Proceeding of the 2007 Conference on Artificial Intelligence Research and Development, Frontiers in Artificial Intelligence and Applications, vol. 163, 2007, pg. 245-254.

[Sil02] Silber H., McCoy K, *Efficiently computed lexical chains, as an intermediate representation for automatic text summarization*, Computational Linguistics, 28(4), 2002, pg. 487-496.

[Sch05] Schach S.R., Vanderbilt University, *Object-Oriented and Classical Software Engineering*, sixth edition, McGraw-Hill, New York, 2005.

[Smi99] Smith G., *The Object-Z Specification Language*, Kluwer Academic Publishers, 1999.

[Ste02] Stevenson M., *Combining Disambiguation Techniques to Enrich an Ontology*, Proceedings of the ECAI 2002 Workshop on Machine Learning and Natural Language Processing for Ontology Engineering, Lyon, Franța, 2002.

[Sto04] Stokes N., *Applications of Cohesion Analysis in the Topic Detection and Tracking Domain*, Teză de doctorat, Faculty of Science, National University of Ireland, Dublin, 2004.

[Suk08] Suktarachan M., Thamvijit D., Rajbhandari S., Noikongka D., Mahasarakram P., Yongyuth P., Kawtrakul A. şi Sini M., *Workbench with Authoring Tools for Collaborative Multilingual Ontological Knowledge Construction and Maintenance*, Proceedings of the Sixth International Language Resources and Evaluation (LREC'08), Marrakech, Maroc, 2008, pg. 2501-2508, http://www.lrec-conf.org/proceedings/lrec2008/pdf/624_paper.pdf.

[Şer05] Şerban, A. C., **Mihiş, A.D.**, *Software Quality Assurance*, Proceedings of the Symposium „Zilele Academice Clujene", 2005, pg. 207-212.

[Tan09] Tan H. şi Lambrix P., *Selecting an Ontology for Biomedical Text*, Mining Human Language Technology Conference, Proceedings of the Workshop on BioNLP, Association for Computational Linguistics, Boulder, Colorado, 4-5 iunie 2009, pg. 55-62, http://www.aclweb.org/anthology/W/W09/W09-1307.pdf.

[Tăt99] Tătar D., *Bazele matematice ale calculatoarelor*, Litografia UBB, 1999.

[Tăt07a] Tătar D., Şerban G., **Mihiş A.D.**, Lupea M., Lupşa D. şi Militon F., *A Chain Dictionary Method for Word Sense Disambiguation and Applications*, Mathematical Reviews, www.ams.org/mathscinet/ , Studia Universitatis "Babeş-Bolyai", Informatica: KEPT 2007, 6-8 iunie 2007, pg. 41-49.

[Tăt07b] Tătar D., Şerban G., **Mihiş A.D.**, Mihalcea R., *Textual Entailment as a Directional Relation*, CALP 2007, INCOMA Ltd., editor: Orăşan Constantin şi alţii, 2007, pg. 53-58.

[Tăt08a] Tătar D., **Mihiş A.D.** şi Lupşa D., *Text Entailment for Logical Segmentation and Summarization*, BDI, Proceedings of the 13th international conference on Natural Language and Information Systems: Applications of Natural Language to Information Systems, Document Processing and Text Mining, http://www.informatik.uni-trier.de/~ley/db/conf/nldb/nldb2008.html, Lecture Notes in Computer Science, Vol. 5039, 2008, pg. 233-244.

[Tăt08b] Tătar D., **Mihiş A.D.**, Şerban G., *Lexical Chains Segmentation in Summarization*, SYNASC, IeAT Technical Report, 08-11, BDI, 2008, Timişoara, pg. 95-101.

[Tăt08c] Tătar D., **Mihiş A.D.**, Şerban G., *Top-down Cohesion Segmentation in Summarization*, BDI, http://www.aclweb.org/anthology/W/W08/W08-2232.bib, Research in Computational Semantics , vol. 1(2008), pg. 389-397.

[Tăt08d] Tătar D., **Mihiş A.D.**, Şerban G., *Top-down Cohesion Segmentation in Summarization*, Step 2008, Veneţia, Italia, 22-24 sept 2008, College Publications, editori: Bos J. şi Delmonte R., 978-1-904987-93-2, BDI, 2008, pg. 145-151.

[Tăt08e] Tătar D., Tămâianu-Morita E.S., **Mihiş A.D.**, şi Lupşa D., *Summarization by Logic Segmentation and Text Entailment*, CICLing 2008, DBLP, http://www.informatik.uni-trier.de/~ley/db/journals/index-r.html, Research in Computing Science, Vol. 33(2008), pg. 15-26.

[Tăt09a] Tătar D., **Mihiş A.D.**, Şerban G., Mihalcea R., *Textual Entailment as a Directional Relation*, Journal of Research and Practice in Information Technology, 41(2009), 1, pg. 53 – 64.

[Tăt09b] Tătar, D., **Mihiş, A. D.**, Şerban, G., *Lexical Chains Segmentation in Summarization*, SYNASC 2008, IEEE Computer Society, Editor: Viorel Negru, Tudor Jebelean, Dana Petcu, Daniela Zaharie, 978-0-7695-3523-4, http://www.computer.org/cps, 2009, pg. 95-101.

[Tăt09c] Tătar D., **Mihiş A.D.**, Lupşa D., Tămâianu-Morita E.S., *Entailment-Based Linear Segmentation in Summarization*, International Journal of Software Engineering and Knowledge Engineering, 19(2009), 8, pg. 1023 – 1038.

[Tho95] Muffy T., *Formal methods and their role in developing safe systems*, www.iee.org.uk/PAB/SCS/wrkshop.htm/, Workshop report, 20 martie 1995.

[Tei77] Teichroew D. şi Hershey E.A. III, *PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems*, IEEE Transactions on Software Engineering, SE-3, ianuarie 1977, pg. 41-48.

[Tuc99] Tucker R. I., *Automatic summarising and the CLASP system*, Teză de doctorat, University of Cambridge, Computer Laboratory, 1999.

[Wil06] Wilkinson M. şi Good B., *Construction and Evaluation of OWL-DL Ontologies*, Microsoft Research Faculty Summit, Redmond, USA, 17-18 iulie 2006, http://research.microsoft.com/en-us/um/redmond/events/fs2006/agenda_mon.aspx.

[Woo96] Woodcock J. şi Davies J., *Using Z, Specification, Refinement and Proof*, Prentice Hall, 1996.

[Yil07] Yildiz, B. and Miksch, S., *ontoX - a method for Ontology-driven information extraction*, Proceedings of the 2007 international conference on Computational science and its applications - Volume Part III, Workshop on CAD/CAM and web based collaboration (CADCAM 07), Springer-Verlag Berlin, Heidelberg 2007, pg. 660-673.

[You79] Yourdon E. şi Constantine L.L., *Structured Design: Fundamentals of a Discipline of Computer Program and Software Design*, Prentice Hall, Englewood Cliffs, NJ, 1979.

[***Adr] http://www.cs.ubbcluj.ro/adriana/Teaching.html, decembrie 2010

[***BrC] http://www.archive.org/details/BrownCorpus

[***Por] http://ir.dcs.gla.ac.uk/resources/linguistic_utils/

[***RTE] http://www.pascal-network.org/Challenges/RTE/

[***Sem] http://www.gabormelli.com/RKB/SemCor_Corpus

[***Sen] http://www.senseval.org/

[***SPW] Probably the most widely used stopword list, http://www.lextek.com/manuals/onix/stopwords1.html

[***STO] The Stanford Natural Language Processing Group, Stanford Parser, http://nlp.stanford.edu:8080/parser/

[***STS] The Stanford Natural Language Processing Group, The Stanford Parser: A statistical parser, http://nlp.stanford.edu/software/lex-parser.shtml

[***WHR] A Wikipedia article about Human Resources, http://en.wikipedia.org/wiki/Human_resources

[***WnB] The definition of the noun business http://www.wordnet-online.com/business.shtml.

[***WNt] http://wordnetweb.princeton.edu/perl/webwn

[***WSM] http://en.wikipedia.org/wiki/Small_business