BABEŞ-BOLYAI UNIVERSITY, CLUJ-NAPOCA

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

FORMAL METHODS IN DEVELOPING CORRECT PROGRAMS

Abstract of the Ph.D. Thesis

Scientific Advisor: **PROF. DR. MILITON FRENŢIU**

Ph. D. student: **MIHIŞ ANDREEA-DIANA**

CLUJ-NAPOCA

2011

ACKNOWLEDGEMENTS

Thanks once again to all who were with me in developing this thesis, especially Mrs. Prof. Dr. Doina Tătar, Mr. Prof. Dr. Militon Frenţiu, Mr. Prof. Dr. Ştefan Măruşter and Mr. Prof. Eng. Ioan Salomie, Mihăilă Family, Mihiş Family, co-workers, colleagues and department chair.

This work was supported partially from: Grant type UBB TP no. 2/2006, Grant CNCSIS TD 400/2007, Conto 91-037/19.09.2007 CNMP Project, 42-117/2008 I-Globe CNMP Project and ID_2412/2009 Project Idei.

Table of Contents of the Abstract of Thesis:

Table of Contents of the Ph. D. Thesis:

I. Introduction

The main purpose of software developers is to fulfil the customer's requirements, regardless of their type: functionality, time or money. Generally it is desirable that the made products are of high quality, but the main requirement is related to the software correctness. By software correctness is understood the capacity of a software product to fulfil its requirements. In some areas, the existence of a software which behaves exactly as expected is essential. Unfortunately, customer requirements are often written in natural language, ambiguous and difficult to process.

A solution is to remove the requirements ambiguity, which is possible due to the formal methods. Formal methods can be seen as a formal mean to describe the problem or to model the system [Gab06]. Practically, any usage of mathematics in problem of software engineering is a formal method. Formal methods have a precise mathematical basis, from which are derived modelling and analysis methods.

By formal methods is understood the usage of the mathematical notations, techniques and methods in the specification, design and software implementation [Bur95, Cla96, Tho95], "Formal methods are the mathematics of computer science" [Hol96].

There are many types of formalizations. The easiest is a utilization of mathematics (notations, methods) in spots in the specification, at the design, implementation or testing or at the product maintenance.

The next formalization level is represented by the specification formal languages, as is the Z language [Abr80], Vienna definition method (VDM) [Jon86] or Gist [Bal85]. Formal specifications have a predefined form, use mathematical notations and is possible the consistence checking. The value of the formal specification languages is well recognized, because even if their elaboration is time consuming, the grown cost of the specification phase, early phase of the software product life cycle, this cost is later recovered because the cost of the design, development and maintenance are significantly decreased [Sch05].

In the same time correct programs can be obtained by the stepwise refinement method [Dij75, Gri85, Mor90]. The refinement method consists in the step by step application of a set of refinement rules. The process starts from an abstract program and stops when the code is obtained. The code obtained in this way will fulfil its requirements, following testing being unnecessary.

In the same time, there are formal methods used in the automated proving and model consistency checking [Hei96, Rus95]. This is the highest level of formal methods.

Formal methods were successfully used in specification, validation, automatic proof and automatic test generation.

As it can be noticed, in order to assure the fact that the client and the developer have understood the same thing from the requirement specification document, it must be developed using a formal language. Unfortunately, this is not always possible. If however is desired the conversion of the natural language requirements written by the client into formal specifications, developers can use the aid of different tools capable of natural language processing.

Thesis structure. The present thesis is structured in six chapters (an introduction, a background chapter, three chapters containing original contributions and a concluding one), has a bibliography including a number of 219 references and five annexes.

Chapter 1 introduces the context, motivations and goals of the thesis, summarizes the contributions brought within it and provides an outline to its contents.

Chapter 2 shortly presents background information in respect with software engineering, formal methods, but also the existing link between the natural language processing methods and formal methods. This chapter also contains a short presentation of Z language, and of stepwise refinement method, the lonely method capable to guarantee that the resulting programs fulfil their specification.

Chapter 3 describes the contribution in the field of natural language processing. It is divided in three subchapters. In the first subchapter are presented the contributions in the disambiguation and text entailment relationship. The second contains the contributions in the field of textual segmentation and in the last those of summary extraction from a text.

Chapter 4 realizes a transition from natural language to formal specifications, presenting different aspects of ontology usage with this purpose. Here are addressed the problem of ontology matching versus a natural language text, of a ontology extraction from a natural language text using the syntactic sentence analysis tree. Also is analysed the role of disambiguation of ontology elements and of the similarity usage in order to enhance the quality of an ontology. And at the end an ontology is used to process requirements in order to obtain structures as close as possible to formal specifications.

Chapter 5 presents different contributions in the field of formal methods. Among them are numbered a tool that assists the Z specification elaboration, Z specification usage and transformation into abstract programs. Another tool which assist the refinement process of abstract programs into code, and a tool which simplifies the condition expression, and also some considerations regarding formal specification reuse and multiformalism.

Chapter 6 concludes the studies made during the thesis.

Keywords: specifications, formal methods, correctness, natural language processing, ontology.

The original contributions are detailed in the chapters 2, 3, 4 and 5. A study they include:

- A debate regarding natural language processing methods applicability in requirements analysis [Mih08c] (subchapter 2.4).
- A new disambiguation algorithm [Tăt07a] (subchapter 3.1.1).
- A new disambiguation and part of speech recognition algorithm [Mih07] (subchapter 3.1.2).
- Three directional methods for textual entailment [Tăt07b, Tăt09a] (subchapter 3.1.3).
- A method for an ontology construction using the textual entailment relationship [Mih08b] (subchapter 3.1.4).
- Three logical text segmentation methods [Tăt08e, Tăt08b] (subchapter 3.2.1).
- Two new version of logical segmentation method [Mih08d] (subchapter 3.2.2).
- A genetic segmentation method with a predefined number of segments [Mih08a] (subchapter 3.2.3).
- A method of top-down based on lexical chains [Tăt08b, Tăt08c, Tăt08d, Tăt09b] (subchapter 3.2.4).
- A summarization post segmentation method [Tăt08e, Tăt08b] (subchapter 3.3.1).
- Three methods for variable length summary extraction [Tăt08d] (subchapter 3.3.2).
- A lexical chain summarization method [Tăt08b, Tăt08c, Tăt08d, Tăt09b] (subchapter 3.3.3).

- Some metrics for ontology matching versus a natural language text [Mih10b] (subchapter 4.1).
- Two methods for a sentence syntactic analysis tree processing and ontology triples extraction [Mih10c] (subchapter 4.2).
- An algorithm for misplaced elements identification in an ontology [Mih10d] (subchapter 4.3).
- A debate regarding the necessity of disambiguation usage in ontology evaluation [Mih10e] (subchapter 4.4).
- A method of formal specification extraction from a requirement natural language text [Mih10f] (subchapter 4.5).
- A study regarding the continuity of software quality assurance activity [Şer05] (subchapter 5.1).
- A debate on the necessity of multiformal approach of complex product specification [Cio04] (subchapter 5.2).
- An analysis of formal specification reuse [Mih05] (subchapter 5.3).
- A tool which assists the Z schemas refinement [Mih10a] (subchapter 5.4).
- A tool which assists the refinement process of abstract programs into code [Mih06a] (subchapter 5.5).
- A tool which automatically simplifies conditional expressions [Mih06b, Lup08, Lup09] (subchapter 5.6).

II. Formal programs in program development

2.1. Software life-cycle

The software development process goes through a “life-cycle”, characterized by three important moments, named phases: definition, development and usage.

2.2. Requirement specification

Requirement specification is the last phase of requirement analysis process. The result of the requirement analysis is the **Requirement specification document**, which represent the contract between the client and the developer.

2.2.1 Informal requirements

In many development projects, the specification document consists in page after page written in natural language, which has the ambiguity fault.

2.2.2 Semiformal specification techniques

There are many types of semi-formal specification techniques: **action-oriented graphical techniques** [DeM78, Gan79, You79], **data-oriented graphical techniques** [Che76] and **other semi-formal specification techniques** (PLS/PLA [Tei77], SADT [Ros85], SREM [Alf85]).

2.2.3 Formal specification techniques

Among the formal specification techniques are: **Finite state machine** [Kam87], **Petri Nets** [Pet62] and **Z language** [Abr80]. A Z specification is composed from four parts: the set of input data, data and constant types, state definitions, initial state and operations. **Other formal techniques** are: Anna [Luc85], Gist [Bal85], VDM [Jon86] and CSP [Hoa85].

2.3. Stepwise refinement

The first person who has proposed a method which assures the program correctness was Floyd [Flo67]. In the same time, it was considered more important to write correct programs by construction. In this way an important role was played by Dijkstra [Dij75], Gries [Gri81], Dromey [Dro89] and Morgan [Mor90]. Dijkstra’s idea, to formally derived programs from specifications was continued later by Gries [Gri81], which considers that is more important to develop correct programs then to prove later their correctness [Fre06]. In order to obtain correct programs, the specifications must be transformed in code using a set of well-defined rules.

2.4. Natural language processing methods used in requirements analysis

At a new product conception, clients and developers meet in order to specify the new product requirements. In order not to omit any requirements, are used different **requirement elicitation techniques**. In this way, a large volume of natural language requirements is obtained. Their process can benefit from the usage of some **natural language processing mechanism**, as it can be seen in Figure1:

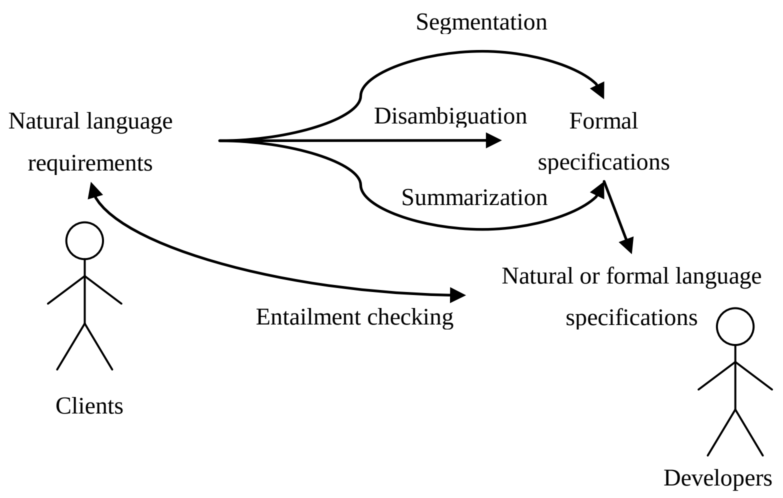


Figure 1. The usage of natural language processing techniques in requirements analysis

2.5. Ontology

An ontology is „A specification of a conceptualization” [Gru93]. Today, the backbone of Semantic Internet is OWL (Web Ontology Language) and RDF (Resource Description Framework) [Pol09]. They represent the best ontology modelling languages. The most general mean of definition of an ontology is by triple usage [Bra85]. A triple is composed from an *object*, a *predicate* which represent a directional relationship through an *object*, which usually is a concept characteristic. For uniformity, all the ontologies items are represented by an URI (Uniform Resource Identifiers) [Ber98].

III. Contributions in the field of natural language processing

3.1. Contributions in the automatic disambiguation of a text

In order to quickly disambiguate the natural language specifications written by the clients, is needed a tool capable to automatically identify the correct sense of words. As is expected, when the purpose is to identify the correct sense of a word the most trustful source is the dictionary.

3.1.1 Word sense identification

One of the best known dictionary-based methods is that of Lesk [Les86]. The algorithm of Lesk was successfully developed in [Ban03] by using WordNet dictionary for English [***WNt]. The algorithm developed by Banerjee and Pedersen improves the original Lesk's method by augmenting the definitions with non-gloss information: synonyms, examples and glosses of related words. Also, the authors introduced a novel overlap measure between glosses which favour multi-word matching.

The original algorithm CHAD [Tăt07a] is similar with the one presented in [Ban03], only that the CHAD disambiguates all the words from the text, by one scroll and not only a target word. The algorithm uses repeatedly a group of three words, among which the first two were already disambiguated, and the third will be disambiguated. In this way the algorithm is similar to a chain, of which mesh are represented by groups of three words. The first three words being disambiguated in the same.

The algorithm uses three ways of computing the score of a sense triplet $s_{w_1}^{i}, s_{w_2}^{j}$, $s_{w_3}^{k}$ of the three words w_1, w_2, w_3: Dice [Dic45], overlap and Jaccard [Jac901]. These three measures evaluate differently the number of existing common words among the corresponding senses $s_{w_1}^{i}, s_{w_2}^{j}$, $s_{w_3}^{k}$ of those three words w_1, w_2, respectively w_3, definitions denoted with $D_{w_1}, D_{w_2}, D_{w_3}$:

$$\text{Dice: score } (s_{w_1}^{i}, s_{w_2}^{j}, s_{w_3}^{k}) = 3*\frac{|D_{w_1} \cap D_{w_2} \cap D_{w_3}|}{|D_{w_1}|+|D_{w_2}|+|D_{w_3}|}$$

$$\text{overlap: score } (s_{w_1}^{i}, s_{w_2}^{j}, s_{w_3}^{k}) = \frac{|D_{w_1} \cap D_{w_2} \cap D_{w_3}|}{\min(|D_{w_1}|,|D_{w_2}|,|D_{w_3}|)}$$

$$\text{Jaccard: score } (s_{w_1}^{i}, s_{w_2}^{j}, s_{w_3}^{k}) = \frac{|D_{w_1} \cap D_{w_2} \cap D_{w_3}|}{|D_{w_1} \cup D_{w_2} \cup D_{w_3}|}$$

Including the disambiguation algorithm of a words triplet, the CHAD algorithm from Figure 2 is obtained.

Since often the intersection of the definition of the three words is 0, there are words with the score 0 for all their senses. For these words is attributed the first sense from WordNet, because in WordNet the word senses are ordered by their frequency.

The algorithm was tested on ten randomly chosen texts from Brown corpus [***BrC, Nel79] (A01, A02, A11, A12, A13, A14, A15, B13, B20 and C01) text which are POS (Part Of Speech) tagged. The Brown corpus was chosen because the results could be compared with those from SemCor corpus [***Sem], POS and sense tagged. The CHAD algorithm was run first for nouns, then for verbs and the third time for nouns, verbs, adjectives and adverbs.

```
Algorithm CHAD ( W, n, S ) is:
  data W {a text formed by the words $w_1, w_2, ..., w_n$}, n {the number of words from text }
  @Disambiguate_triplet ($w_1, w_2, w_3$, $s^*_{w_1}, s^*_{w_2}, s^*_{w_3}$ )
  For $i \leftarrow 4, n$ do
    $p \leftarrow 1$
    $max \leftarrow scor( s^*_{w_{i-2}}, s^*_{w_{i-1}}, s^1_{w_i} )$
    {the already identified senses for the words $w_{i-2}$ and $w_{i-1}$ are $s^*_{w_{i-2}}, s^*_{w_{i-1}}$ }
    For every sense $s^m_{w_i}$ do
      If $max < score( s^*_{w_{i-2}}, s^*_{w_{i-1}}, s^m_{w_i} )$
        then $p \leftarrow m$
          $max \leftarrow score( s^*_{w_{i-2}}, s^*_{w_{i-1}}, s^m_{w_i} )$
      endif
    endfor
    $s^*_{w_i} \leftarrow s^p_{w_i}$
  endfor
  results S {senses $s^*_{w_i}, i = 1, n$ }
end CHAD
```

Figure 2. CHAD Algorithm

Even if the precision is less than the precision given by the first WordNet sense (the average difference is for nouns: 0.0338, for verbs 0.0340 and for all words 0.0491), these results are comparable to those obtained so far, since at Senseval 2 contest, only 2 out of the 7 teams (with the unsupervised methods) achieved higher precision than the WordNet first sense baseline.

CHAD algorithm is independent on the language in which the text is written, and it can be successfully applied for Romanian language also. It can be applied for enhancing the translation [Tăt07a], and also for POS tagging [Mih07] or for text entailment checking.

3.1.2 Part of speech recognition

When a person finds an unknown word, especially in a language not fully learn, even though grammar skills are not completely developed, he will try to search for the corresponding word in a dictionary, in order to identify the sense corresponding to the context. But some words have several different POS, depending on their context. In the case in which the searched word has this kind of characteristics, in the same time with the sense identification is made a POS identification by the context. This fact is at the base of a new algorithm of simultaneous POS tagging and sense identification derivate from CHAD as it can be seen in Figure 3 [Mih07].

The experiments were made on the same ten texts from the Brown corpus. Because in SemCor, which is used as an evaluation standard exists words with no POS identified (Notag), these words were not considered in the computation of the precision for the method. The minim, maximum and average values are in table 1.

The method is promising despite the fact that the results are below the results obtained using many more resources and grammatical rules, because it can be applied for many languages and in the same time disambiguates the words.

```
Algorithm POS_CHAD ( W, n, S, P ) is
  data W {a text form by the words $w_1, w_2, ..., w_n$}, n {the number of words from the text}
  @Disambiguate_and_identify_PV_triplet ($w_1$, $w_2$, $w_3$, $s^{*}_{1\,{p^{*}_{1}}^{w_1}}$ , $s^{*}_{2\,{p^{*}_{2}}^{w_2}}$ , $s^{*}_{3\,{p^{*}_{3}}^{w_3}}$ )
  For $i \leftarrow 1,n$ do
    $r \leftarrow$ the first POS from the dictionary which corresponds to the word $w_i$
    $k \leftarrow 1$ {the first sense of the first POS of the word $w_i$}
    $max \leftarrow score( s^{*}_{i-2\,{p^{*}_{i-2}}^{w_{i-2}}} , s^{*}_{i-1\,{p^{*}_{i-1}}^{w_{i-1}}} , s^{1}_{i\,r^{w_i}} )$
    For every POS q of the word $w_i$ do
      For every sense $s^{m}_{i\,q^{w_i}}$ of the word $w_i$ with the POS q do
        If $max < score( s^{*}_{i-2\,{p^{*}_{i-2}}^{w_{i-2}}} , s^{*}_{i-1\,{p^{*}_{i-1}}^{w_{i-1}}} , s^{m}_{i\,q^{w_i}} )$
          then $r \leftarrow q$
            $k \leftarrow m$
            $max \leftarrow score( s^{*}_{i-2\,{p^{*}_{i-2}}^{w_{i-2}}} , s^{*}_{i-1\,{p^{*}_{i-1}}^{w_{i-1}}} , s^{m}_{i\,q^{w_i}} )$
        endif
      endfor
    endfor
    $p^{*}_{i} \leftarrow r$
    $s^{*}_{i\,{p^{*}_{i}}^{w_i}} \leftarrow s^{k}_{i\,r^{w_i}}$
  endfor
  results S {senses $s^{*}_{i}, i \leftarrow 1,n$ }, P { POS $p^{*}_{i}, i \leftarrow 1,n$ }
end POS_CHAD
```

Figure 3. POS_CHAD algorithm

Precision	Minimum value	Maximum value	Average value
All word identification	20.16%	64.13%	42.78%
All word disambiguation	7.00%	41.43%	21.78%
Noun	53.04%	82.83%	66.12%
Verbs	67.83%	92.02%	84.45%
Adjectives	29.55%	75.34%	56.39%
Adverbs	46.87%	80.65%	69.41%

Table 1. The minimum, maximum and average values of the precision regarding POS

This algorithm can be successfully used for the automatic POS identification from specifications and in this way the entities and relations are much easily identified for an Entity-Relationship model elaboration.

3.1.3 Text entailment relationship

The main reasoning regarding a text or a group of texts is based on the text entailment relationship. The text entailment relation between two texts: *T* (the text) and *H* (the hypothesis) represents a fundamental phenomenon of natural language. It is denoted by $T \rightarrow H$ and means that the meaning of *H* can be inferred from the meaning of *T*.

Although the problem is not new, most of the automatic approaches have been proposed only recently within the framework of the Pascal Textual Entailment Challenges [***RTE]. Nonetheless, only few authors exploited the directional character of the entailment relation, which means that if $T \rightarrow H$ it is unlikely that the reverse $H \rightarrow T$ also holds. The most notable directional method used in the RTE-1 challenge was that of Glickman [Gli05] 58.5%, and the directional method of Kǒuylekov [Kǒu06] obtained 56%.

The first method which evaluates the textual entailment relationship proposed in [Tăt07b] uses text similarity. In [Cor05] is defined the following similarity measure between two texts, the similarity between the texts T_i and T_j with respect to T_i:

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

The set of words from the text T_i which have a specific part of speech *pos* is denoted by $WS_{pos}^{T_i}$. For a word w_k with a given *pos* in T_i, the highest similarity of the words with the same *pos* in the other text T_j is denoted by $maxSim(w_k)$. Since the hypothesis H is not informative in respect to T, for an entailment between the two texts the following relationship must take place [Tăt09a]:

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

This formula is applied to the texts previously disambiguated with the CHAD algorithm presented in the previous chapter. In the formula (3.1.3.1) only nouns and verbs were used as *pos*, and the similarity between two words was considered to be 1 if those two words are identical or synonyms and 0 otherwise. This identification is completed with a set of heuristics for recognizing false entailment that occurs because of lack of monotony of real texts (*COND*).

The following notations are made: the text T with T_1, the hypothesis H with T_2, NP_1 and NP_2 – Name entities from T_1 respectively T_2, I_c – non-named entities common in T_1 and T_2, $SYN(T_1)_{T_2}$ – words non-Name entities, non common in T_1, which are nouns or verbs and are contained in a synset of T_2, $SYN(T_2)_{T_1}$ $SYN(T_1)_{T_2}$ – words non-Name entities, non common in T_2, which are nouns or verbs and are contained in a synset of T_1, $C_1 = |SYN(T_1)_{T_2}|$, $C_2 = |SYN(T_2)_{T_1}|$, $W_{T_1} = NP_1 \cup I_c$, $W_{T_2} = NP_2 \cup I_c$.

The condition for text entailment obtained from (3.1.3.1) and (3.1.3.2) is: $C_1 \le C_2$, that means $|SYN(T_1)_{T_2}| \le |SYN(T_2)_{T_1}|$. The relation is not strict because of the definition of the sets $SYN(T_1)_{T_2}$ and $SYN(T_2)_{T_1}$.

```
Function Entailment (T1,T2 ) is
  data T1, T2 {two texts}
    if $W_{T_1} \subseteq W_{T_2}$
      then if $T_2 = NP_2 \cup I_c$
        then if @ COND
          then Entailment ← False {not $T_1 \to T_2$}
          else Entailment ← True { case I: $T_1 \to T_2$ }
        else if $C_1 \le C_2$
          then Entailment ← True { case II: $T_1 \to T_2$ }
          else Entailment ← False {not $T_1 \to T_2$}
      else Entailment ← False {not $T_1 \to T_2$}
  results True { $T_1 \to T_2$}, respectively False {not $T_1 \to T_2$}
end Entailment
```

Figure 4. The function that checks the text entailment relationship based on similarity

For our heuristics an important situation is that when T_2 contains only named entities and common with T_1 words, and because of this, the condition $W_{T_1} \subseteq W_{T_2}$ is the first one that is verified in the algorithm (see Figure 4).

The algorithm uses as input POS tagged texts, were also the Name entities were identified. The algorithm was checked on the 800 pairs from the RTE-1 challenge [***RTE].

The global precision is 52,625%, and the average precision 56,60647 %.

The second text entailment check method is based on the cosine measure [Tăt09a]. Three cosine measures were defined in order to evaluate the distance between the word vectors $T = t_1,t_2,...,t_m$ and $H = h_1,h_2,...,h_n$. The first cosine measure, $\cos_T(T,H)$ evaluates the distance between the m-dimensional vector $\vec{T} = (1,1,...,1)$ and the m-dimensional vector $\vec{H}$ which represent the projection of H on T, in other words $\vec{H}_i = 1$, if t_i is a word in the sentence H and $\vec{H}_i = 0$ otherwise. The second cosine measure, $\cos_H(T,H)$ evaluates the distance between the n-dimensional vector $\vec{H} = (1,1,...,1)$ and the n-dimensional vector $\vec{T}$ which represent the projection of T on H, in other words $\vec{T}_i = 1$, if h_i is a word in the sentence T and $\vec{T}_i = 0$ otherwise. For the last cosine measure $\cos_{H\cup T}(T,H)$, the first vector is obtained by replacing the words from $H\cup T$ contained in T with 1 the other with 0, and the second by replacing the words $H\cup T$ contained in H with 1 and those from $T\backslash H$ with 0.

Denoting by c the number of common words of T and H, the three measures are: $\cos_T(T,H) = \sqrt{\frac{c}{m}}$, $\cos_H(T,H) = \sqrt{\frac{c}{n}}$ and $\cos_{H\cup T}(T,H) = \sqrt{\frac{4c^2}{(n+c)(m+c)}}$.

To accomplish the condition: *T entails H if and only if H is not informative in respect to T* [Mon01], the similarities between T and H calculated with respect to T and to $H\cup T$ must be very closed. Analogously, the similarities between T and H calculated in respect to H and to $H\cup T$ must be much closed. Also all these three similarities (cosines) must be larger than an appropriate threshold. So the conditions imposed are: $|\cos_{H\cup T}(T,H) - \cos_T(T,H)| \le \tau_1$, $|\cos_H(T,H) - \cos_{H\cup T}(T,H)| \le \tau_2$ and $\max\{\cos_T(T,H), \cos_H(T,H), \cos_{H\cup T}(T,H)\} \ge \tau_3$. The thresholds found are: $\tau_1 = 0.095$, $\tau_2 = 0.15$ and $\tau_3 = 0.7$.

The accuracy for TRUE pairs is 68.92230576% and for FALSE pairs is 46.36591479%. The overall accuracy is 57.62%.

The third original method is based on a modified Levenshtein distance [Tăt09a]. Let consider that for two words w_1 and w_2 the modified Levenshtein distance as calculated by our algorithm is denoted by $LD(w_1,w_2)$. This is defined as the minimal number of transformations (deletions, insertions and substitutions) such that w_1 *is transformed in* w_2. We denote by T_{word} the "word" obtained from the sentence T by considering the empty space as a new letter, and by concatenating all the words of T. Analogously a "word" H_{word} is obtained. $LD(T_{word},H_{word})$ represents the quantity of information of H with respect to T. This modified Levenshtein distance is not a distance in the usual sense, such that $LD(w_1,w_2) \neq LD(w_2,w_1)$.

As *T entails H if and only if H is not informative with respect to T* the following relation must hold: $LD(T_{word},H_{word}) < LD(H_{word},T_{word})$. By applying these criteria to those RTE-1 800 text pairs, a global precision of 53,19% was obtained.

The costs of *transformations from the word* w_1 *to the word* w_2 are as follows: change case cost 1, insert cost 3, remove cost 3, substitute cost 5 and swap cost 2.

The obtained results are among those of the last RTE competitions, the second method giving the best results. Beside the fact that to be able to decide if a text entails another one is useful to the developers, this relation is successfully used for the segmentation and the summarization of a text, which also makes the natural language requirements analysis process easier, as it can be seen from the following sections.

3.1.4 A simple ontology based on the text entailment relationship

In [Mih08b] is presented a way of constructing an informal ontology based on the directional relationship between texts. The method with the best score for the text entailment relationship, the cosine method was used.

Starting with a natural language text, all the text entailment relationships existing among the sentences of the text are identified. In this way is obtained a directional graph among the texts sentences. The graph can represent a simple informal ontology of the corresponding text, in which the entities are represented by the sentences, and the relationships by the text entailment relationship.

The same text entailment relationship can be also used between entire texts, obtaining in this way a graph with the texts relationships. If the corresponding texts represent the entities definitions, then the entailment relationship graph can be the text entailment relationship graph.

3.2. New approaches of the segmentation

In the case of implementing a large informatics system, the requirements volume will be great. In this case the analysis will involve the work of many persons or teams of persons. So the specifications must be divided by functionalities, which correspond with the segmenting operation, a natural language specific technique.

3.2.1 Linear segmentation

In [Tăt08e, Tăt09c] a method of linear segmentation of a text using the text entailment relationship was proposed. The easiest segmentation algorithm based on the entailment relation is named „PureEntailment” (PE). A new segment begins when the new sentence is not entailed by the last segment.

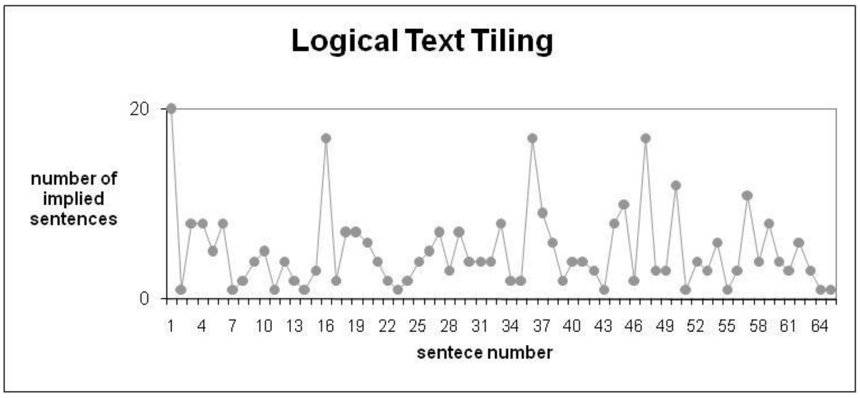


Figure 5. Logical structure of the text

A second segmentation approach, Logical TextTiling (LTT) [Tăt08e, Tăt09c], is similar with the algorithm to detect subtopic structure TextTiling (TT) [Hea93], with the difference that LTT detects the logical structure of the text. The main differences are:

- In LTT the individual units are sentences, and in TT are token-sequences (pseudo sentences) of a predefined size;
- In TT a score is assigned to each token-sequence boundary *i*, calculating how similar the token-sequences *i-k* through *i* are to token-sequence from *i + 1* to *i*+*k*+1. In LLT

the score of a sentence s_i, $score(s_i)$, is the number of sentences of the text which are entailed by s_i. A higher score of a sentence denotes a higher importance. This fact is in accordance with the following text entailment criteria: *A text T entails a text H if and only if the text H is less informative than T* [Tăt07b].

A boundary (a shift in discourse or a gap) in LTT is determined by a change in the configuration of logical importance of sentences (see Figure 5). In such a way we have a structural basis for dividing up the discourse into a series of smaller segments, each with a different configuration of the importance of components.

Unlike other segmentation methods, LTT is a linear method:

```
Algorithm LTT(S, n, SC, SEG) is
  data S {a text, $s_i$ – texts sentences, i←1,n}, n {the number of sentences from the text},
       SC {sci = score(si) the texts sentences scores}
  j ← 1, p ← 1,
  initialize($Seg_1$, $s_1$) {initially the segment $Seg_1$ contains only the sentence $s_1$}
  dir ← „up”
  While p<n do
    If $sc_{p+1}$ > $sc_p$
      then If dir = „up”
        then add($Seg_j$, $s_{p+1}$) {adds to the segment $Seg_j$ the sentence $s_{p+1}$}
        else j ← j+1
          initialize($Seg_j$, $s_{p+1}$)
          dir ← „up”
        endif
    else      add($Seg_j$, $s_{p+1}$)
        If score($s_{p+1}$) < score($s_p$)
          then dir ← „down”
        endif
    endif
    p ← p + 1
  endwh
  results SEG={$Seg_1$,...,$Seg_j$}
end LTT
```

Starting from the way in which is measured the importance of a sentence regarding the number of other sentences with which the first sentence is in a entailment relationship, two new versions to determine the sentences score were proposed:

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

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

where $entail_{i,j} = \begin{cases} 1, & \text{if } s_i \rightarrow s_j \\ 0, & \text{else} \end{cases}$.

Another approach to the segmentation problem is a dynamic programming approach. This method realises first a summarization (see subchapter 3.3.1). The segment boundaries are those sentences with a minimal score and are placed between two summary sentences (see Figure 6). In this way the number of obtained segments is correlated with the length *x* of the summary.

The segmenting methods were tested on the narrative literary text „The Koan” [Ric91], text which was segmented, summarized and anaphora resolved in a manual way, by linguistic criteria by a specialist [Tăt08e].

Regarding the comparison with the human judge, the method is evaluated according to [Hea93]:

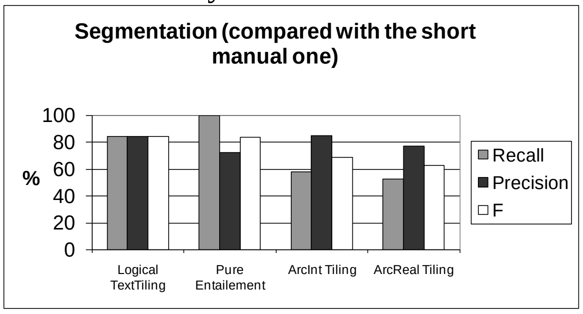
- how many of the same (or very close) boundaries (gaps) with the human judge the method selects out of the total selected (precision);
- how many true boundaries are found out of the total possible (recall)

In the same time, the F measure was used: $F = \frac{2*Precision*Recall}{Precision + Recall}$.

```
Algorithm Segmentation_post_summarization(S, n, SC, P, x, SEG, nrseg) is:
    data S {a text, $s_i$ – the texts sentences, i←1,n}, n {the number of sentences from the
        text}, SC {$sc_i$ = scor($s_i$) the scores for the texts sentences}, P {$p_i$ – the sentences
        from the text belonging to the summary, i←1,x}, x {the number of summary
        sentences}
    nrseg ← 1
    begpoz ← 1
    For i ← 1, x-1 do
        endpoz ← $\arg\min_{j, j \in \{p_i, p_{i+1}-1\}} scor(s_j)$
        @$Seg_{nrseg}$ begins with begpoz and ends with endpoz
        nrseg ← nrseg + 1
        begpoz ← endpoz + 1
    endfor
    results SEG {$Seg_1$,...,$Seg_{nrseg}$ – segments list}, nrseg {number of segments}
end Segmentation_post_summarization
```

Figure 6. The Segmentation post summarization algorithm

A part of the comparative evaluation results of the different segmentation methods (Logical/Dynamic programming) are in Figure 7. From the logical methods the best results were obtained by LTT.



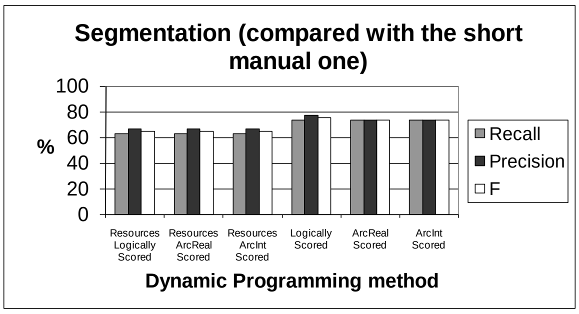


Figure 7. Segmentation evaluation

3.2.2 Various types of Logical TextTiling

The Logical TextTiling algorithm needs as a prerequisite the scoring of every sentence, which leads to a n^2 complexity. In order to reduce the complexity, a neighborhood LTT was proposed, in which every sentence will be scored in a predefined dimension neighborhood, instead on the entire text. Respectively a weighted LTT, which makes priority the local entailments and so the logical structure graph will be flatten [Mih08d]. The neighborhood logical structure generates the same segments as LTT for a neighborhood of minimum 20% from the initial text length (for the text ”The Koan” [Ric91]). And the second type of logical structure generates the same segments. As a consequence, the two new types of LTT can successfully replace the original LTT.

3.2.3 Genetic segmentation with predefined number of segments based on textual entailment relationship

The following method uses the logical interpretation of a text, in other words the text entailment relationship for the purpose of obtaining: an increased cohesion inside a segment and a low connection between the neighboring segments [Mih08a].

If for the previous logical methods it cannot be imposed the number of obtained segments, the following method will split the text in a predefined number k of segments.

So, if there is a text $T=\{t_1, t_2,..., t_n\}$ containing n sentences, to split it into k segments means to split the set $\{1, 2, ..., n\}$ in k subsets, $\{1, 2, ..., n_1\}$, $\{n_1+1, ..., n_2\}$, …, $\{n_{k-1}+1, n_{k-1}+2, ..., n_k\}$. This partitioning of the initial set can be represented as a vector $b = \{n_1, ..., n_k\}$, where $n_k=n$, and $n_1, ..., n_k$ represents the indexes of the last sentences of those k segments.

The first step of the segmentation method is to construct the entailment matrix. To the text T a matrix $n\times n$ E is associated, defined as following:

$$e_{i,j} = \begin{cases} 1, & \text{if } t_i \text{ entails } t_j, i,j=\overline{1,n} \\ 0, & \text{otherwise} \end{cases} \quad (3.2.3.a)$$

a second definition of the same matrix is:

$$e_{i,j} = \begin{cases} \dfrac{1}{|i-j|+1}, & \text{if } t_i \text{ entails } t_j \\ 0, & \text{otherwise} \end{cases}, i,j=\overline{1,n} \quad (3.2.3.b)$$

The entailments are computed with the *cos* measure, method previously presented in subchapter 3.1.3.
Every segment will be evaluated separately, concerning the cohesion:

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

the normalization of the score:

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

and a version without self entailments:

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

Considering entailments, two segments are disconnected if between their sentences are few entailments:

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

The sum of local scores (3.2.3.2) must be minimized in order to obtain a segmentation with a less interconnectivity, and the sum of local scores (3.2.3.1), (3.2.3.1’) respectively (3.2.3.1”) must be maximized, to obtain a segmentation in which the segments have a high intern cohesion.

The problem of determining a predefined number of segments can be viewed as a combinatorial problem and the most appropriate algorithms for such problems are genetic algorithms. The proposed algorithm starts with a chromosome population randomly generated, which represent possible segments of dimension k. To generate the next generation, the following genetic operators are applied: “binary tournament selection” for the parent’s selection, “one point crossover” and “flip mutation” [Bac00] for the crossing and mutation to generate new chromosomes. The evaluation process is similar to an evaluation scheme of a standard genetic algorithm. In addition, a selection of the best individuals will be used. The solution for the segmenting problem is represented by the genetic algorithm named GATTS [Mih08a]. The GATTS algorithm has used the following parameters: population size of 200, 50 generations, the crossing probability of 0.7 and the mutation probability of 0.1.

The evaluation was realized on the text „The Koan” [Ric91], previously used and a newspaper article Hirst [Mor91], from which the short sentences (with 5 or less than 5 words) were removed, the rest 41 sentences being renumbered. The results were compared with those obtained by the Dynamic programming method (DP) previously presented, since both methods generates a predefine number of segments.

The average results obtained are comparable with the best results of the Dynamic programming method, and the maximal ones exceeded them, some being equal to 100%, which indicate the fact that the manual segmentation was obtained.

It can be noticed that for the Koan the best results are obtained with the version (3.2.3.a) of the matrix *E*, and between the local scores definitions, (3.2.3.1”) and (3.2.3.2) give the best results. Regarding the matrix (3.2.3.b), its combination with the score (3.2.3.1’) seems to be the best. For the second text, Hirst, the matrix (3.2.3.b) gives the best results, and again the scores (3.2.3.1”) and (3.2.3.2) stand out.

As a future research direction, a mode to correlate the two local scores, (3.2.3.1”) and (3.2.3.2) will be searched for, in order to obtain an even better segmentation method.

3.2.4 A new top-down segmentation method based on lexical chains

A segmentation method which gives better results than the LTT method is the top-down based on lexical chains method. A lexical chain is a sequence of words such that the meaning of each word from the sequence can be obtained unambiguously from the meaning of the rest of words [Mor91, Bar99, Har97, Sil02]. The map of all lexical chains of a text provides a representation of the lexical cohesive structure of the text. Usually a lexical chain is obtained in a bottom-up fashion, by taking each candidate word of a text, and finding an appropriate relation offered by a thesaurus as Rodget [Mor91] or WordNet [Bar99, Sto04]. If it is found, the word is inserted with the appropriate sense in the current chain, and the senses of the other words in the chain are updated. If no chain is found, then a new chain is initiated.

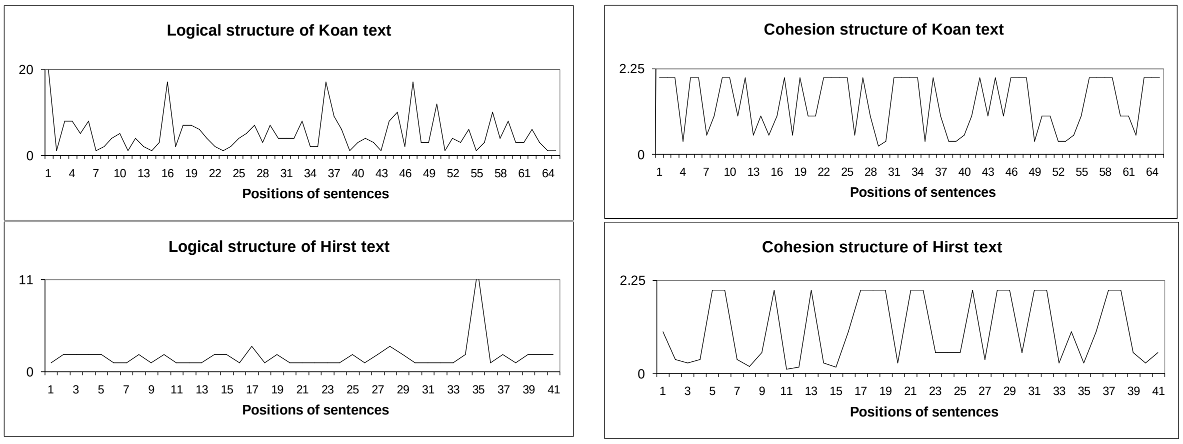


Figure 8. Logical and cohesion structure of Koan, respectively Hirst texts

The following method [Tăt08b/c/d] approaches the construction of lexical chains in a reverse order: first the whole text is disambiguated and then the lexical chains which cover as much as possible the text are constructed. For the disambiguation, the CHAD algorithm previously presented was used. It identifies by only one scroll algorithm shows what words in a sentence are unrelated as senses with the previously words: these are the words which receive a ”forced” first WordNet sense. Scoring each sentence of a text by the number of ”forced” to first WordNet sense words in this sentence, a representation of the lexical cohesive structure of the text is provided. If the number F represents the number of words in a sentence forced to be disambiguated with first WordNet sense, then in the graph representing the function 1/F for all the sentences the valleys (the local minima) of the function will represent the boundaries between lexical chains (see Figure 8) [Tăt08b/c].

This idea for the linear lexical chain identification is at the basis of a linear segmentation algorithm denoted CTT (Cohesive TextTiling), similar with the previously presented LTT algorithm, with the single difference that the score formula for every sentence is changed, $Score(S_i) = \frac{1}{nuw_i}$, were *nuw*$_i$ is the number of words ”forced” to first WordNet sense from the sentence *S*$_i$. If *nuw*$_i$ = 0, then *Score*(*S*$_i$) = 2. CTT was tested on the texts Hirst [Mor91] and „The Koan” [Ric91], and had provided better results than LTT.

The segmentation methods presented above can be very helpful to the team of developers, especially the methods of segmentation with a predefined number of segments - the number of team members. In this way the client requirements can be analysed and refined easier and faster, in a parallel process.

3.3. New approaches to text summarization

Systems that can automatically summarize documents become increasingly studied and used. The summary is a shorter text (usually no longer than a half of the source text) which keeps the most informative (salient) parts of the text. In the literature two types of summary are identified: extract and abstract [Hov03]. Most of the automatic summaries are made by extraction. For this, in the literature are known several approaches [Mar97]. Usually the salient parts are determined on the following assumptions: they contain words that are used frequently, they contain words that are used in the title and headings, they are located at the beginning or end of sections, they use key phrases which emphasize the importance in text, they are the most highly connected with the other parts of text. From these approaches, the latter is the most difficult to achieve. Connectivity can be estimated using the number of words, synonyms or common anaphora [Oră06, Rad02].

3.3.1 Summary extraction from a segmented text

There are many methods for summary extraction from a text, but if the text was previously segmented, its quality should increase, because by selecting sentences which belong to different sentences should enrich the summary with relatively independent information. In [Tăt08e, Tăt09c], beside the linear segmenting methods previously presented, is proved the fact that the segmentation improves the summary quality.

A simple summary obtain as a consequence of the text entailment relationship is the „PureEntailment” (PE) summary, and is obtained by sequentially adding to the summary every sentence which is not implied by the current summary.

Starting from the PE, LTT, ArcInt, respectively ArcReal segmentation previously described in subchapter 3.2.1, a summary can be build by choosing the most important sentences not from the entire text, but from each segment in part such that the summary will have the desired number of sentences. The number and the way in which the sentences are chosen from every segment are given by the algorithm from the Figure 9.

```
Algorithm SumPostSeg(S, n, SEG, j, Sum, x) is:
data S {a text; $s_1,s_2,...,s_n$ – texts sentences}, n {the number of texts sentences}, SEG
   {$Seg_1,...,Seg_j$ the segments in which the text is divided}, j {the segments number}, x {the
   summary length}
@ calculate the "salience" of each segment and rank the segments in $Seg_{i_1},\ldots,Seg_{i_j}$
@ is computed the number $c_{i_s}$ of "esential" sentences for every segment $Seg_{i_s}, s = i_1, i_j$
@ the first k (<j) segments are selectes such that $\sum_{Seg_{i_s}} c_{i_s} = x$
@ reorder selected k segments by their occurrence in text, $Seg'_1,...,Seg'_k$
@ sentences from $Seg'_1,...,Seg'_k$ are the sentences of the summary SUM
results SUM {the summary of length x}
end SumPostSeg
```

Figure 9. The algorithm for summary extraction from a segmented text

In this first approach the salience of segments was considered to be equal and from each segment as the most salient sentence, the sentence with the maximal score, was selected.

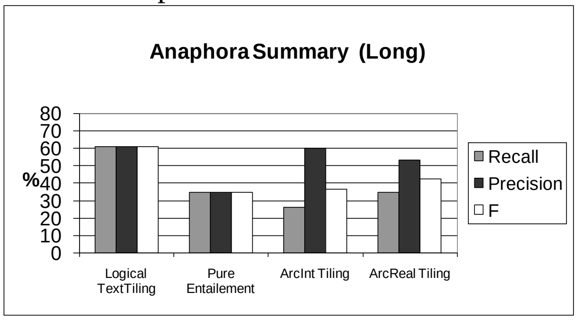
```
Algorithm DP_Sum(S, n, SC, penalty, Sum, x) is:
   data S {a text, $s_i$ – texts sentences, $i \leftarrow 1,n$}, n {the number of the sentences from the text},
        SC {$sc_i$ = score($s_i$) texts sentences scores}, penalty {constant}, P {$p_i$ – positions from the text of
        the summary sentences, $i \leftarrow 1,x$}, x {the number of the sentences from the summary}
   For $i \leftarrow 1,n$ do
      $\delta_i^1 \leftarrow score(s_i)$
      For $k \leftarrow 2,x$ do
         $\delta_i^k = \begin{cases} max_{j(i<j)}(score(s_i) + \delta_j^{k-1}) & ,if\ s_i\ and\ s_j\ have\ common\ words \\ max_{j(i<j)}(penalty*(score(S_i) + \delta_j^{k-1})) & ,otherwise \end{cases}$
         $h_i^k = \begin{cases} argmax_{j(i<j)}(score(s_i) + \delta_j^{k-1}) & ,if\ s_i\ and\ s_j\ have\ common\ words \\ argmax_{j(i<j)}(penalty*(score(s_i) + \delta_j^{k-1})) & ,otherwise \end{cases}$
      endfor
   endfor
   $i \leftarrow arg\,max_j(\delta_j^x)$
   initialize(Sum) {initially th summary Sum is empty}
   For $k \leftarrow x,1$ do
      add(Sum, $s_i$) {adds to the summary the sentence $s_i$}
      $i \leftarrow h_i^k$
   endfor
   results Sum {the summary of length x}
end DP_Sum
```

Figure 10. The algorithm for summary extraction from a segmented text

Another approach to the summarization problem is a dynamic programming algorithm (see Figure 10). In order to meet the coherence of the summary the algorithm selects the chain of sentences with the property that two consecutive sentences have at least one common word, which corresponds to the continuity principle in the centring theory which requires that two consecutive units of discourse have at least one entity in common.

It is assumed that each sentence is assigned a score of its representativity. The three logical scores defined in section 3.2.1 will be used: the sum of entailments, of integer arcs, respectively weighted arcs. A summary score is the sum of all sentences scores from the summary. The summary will be built such that the score is maximized. For the

Dynamic programming method a penalty of 1/10 and the list of 571 ”stop” words LYRL2004 developed for SMART project [Lew04] were used. But only nouns, verbs, adjectives and adverbs were used as common words and the Porter Stemmer [***Por] was used to compare words.



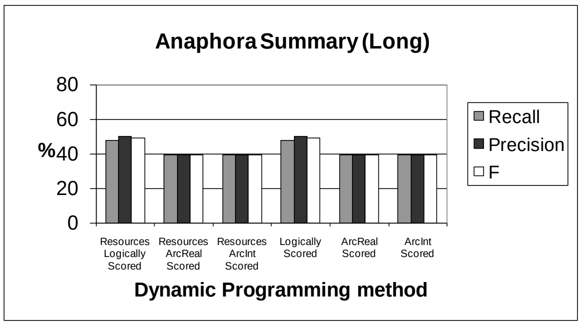


Figure 11. Summarization evaluation

To evaluate the summarization methods the literary ”The Koan” [Ric91] was used, that based on linguistic criteria, and then was summarized and anaphora resolved in a manually way [Tăt08e]. From Figure 11 it can be noticed that the segmentation improves the summarization and the combination of dynamic programming with logical scoring lead to good results. From the logical methods, best results are obtained for the LTT.

3.3.2 Arbitrary length summary

The previous summary is based on segmentation. And also starting from the LTT segmentation (see subchapter 4.1) the following arbitrary length summary is obtained. The LTT segmentation starts from the logical structure of a text, with a fix number of minimum points, so the number of segments is also fixed. Therefore the previous approach which selects from every segment, previously ordered by their importance, the sentence with the greatest score cannot be applied all the time, as in the case in which the number of sentences from the summary is greater than the number of segments. In [Tăt08a] three arbitrary length summarization methods were proposed, all starting from an existing summarization, methods which will be presented in the following. The testing was realized on the Hirst [Mor91] (from which the sentences with five or less words were removed), and the results were compared with those obtained by the authors by lexical chains.

The first step for the summarization is segments scoring. The summary will be achieved by selecting a number of sentences proportionally with the segment score. Is denoted by: *Score*(s_i) = *the number of sentences entailed by* s_i, $Score(Seg_j) = \frac{\sum_{s_i \in Seg_j} Score(s_i)}{|Seg_j|}$, $Score_{final}(s_i) = Score(s_i) \times Score(Seg_j)$, were $s_i \in Seg_j$, $Score(Text) = \sum_{j=1}^{n} Score(Seg_j)$, *Weight of a segment* $: c_j = \frac{Score(Seg_j)}{Score(Text)}$; $0 < c_j < 1$, n = *the number of segments obtained by LTT algorithm*, x = *the desired length of the summary*, $NSenSeg_i$ = *the number of sentences selected from the segment* Seg_j.

The predefined summary algorithm, denoted by AL (Arbitrary Length) is in Figure 12. It is possible that the number of sentences selected from a segment, $NSenSeg_j > 1$. If $x < n$, then for some segments, $NSenSeg_j = 0$.

```
Algorithm AL(Text, SEG, n, Lseg, SC, SelM, Sum, x), is:
    data Text {a text} SEG {text segments $Seg_1,...,Seg_n$}, n {the number of segments},
        Lseg {the vector with the lengths (sentences number) of those n segments}, SC
        {the scores for every sentence $sc_i=Score_{final}(s_i)$}, SelM {selection method}, x
        {summary length}
    @ calculate the weights of segments cj, j=1,n
    @ rank the segments in descending order after cj
    { $NSenSeg_j$, j=1,n is computed}
    While @ the number of already selected sentences is less then x do
        If $[x \times c_j] \geq 1$ {integer part}
            then $NSenSeg_j = min(lseg_j, [x \times c_j])$
            else $NSenSeg_j = 1$
        endif
    endwh
    @ initialize the empty summary Sum
    @ select using SelM from every segment $Seg_j$ - $NSenSeg_j$ sentences and add them to
        the summary Sum
    @ reorder the summary sentences, according to the Text order
    results Sum {the summary with length x}
end AL
```

Figure 12. The algorithm for extracting a summary from a segmented text

The method of extracting sentences from the segments is decisive for quality of the summary. Three selection methods were proposed in order to identify the most important sentences from every segment, every one trying to omit as less information as possible.

Definition 1 Given a segmentation of initial text $T = \{Seg_1,...,Seg_n\}$, for each segment $Seg_i, i=\overline{1,n}$, first $NSenSeg_i$ sentences are selected such that $\sum_{i=1}^{n} NSenSeg_i = x$. The summary $Sum_1 = \{s'_1,...,s'_x\}$ is obtained.

Definition 2 Given a segmentation of initial text $T = \{Seg_1,...,Seg_n\}$, the summary is calculated as: $Sum_2 = \{s'_1,...,s'_x\}$, were $\forall i=\overline{1,n}$, $Sum_2 \cap Seg_i = SelPropSeg_i$, $|SelPropSeg_i| = NSenSeg_i$, $\sum_{i=1}^{n} NSenSeg_i = x$ and $\forall s_j \in SelPropSeg_i$ and $\forall s_k \in Seg_i \setminus SelPropSeg_i$, $score(s_j) \geq score(s_k)$.

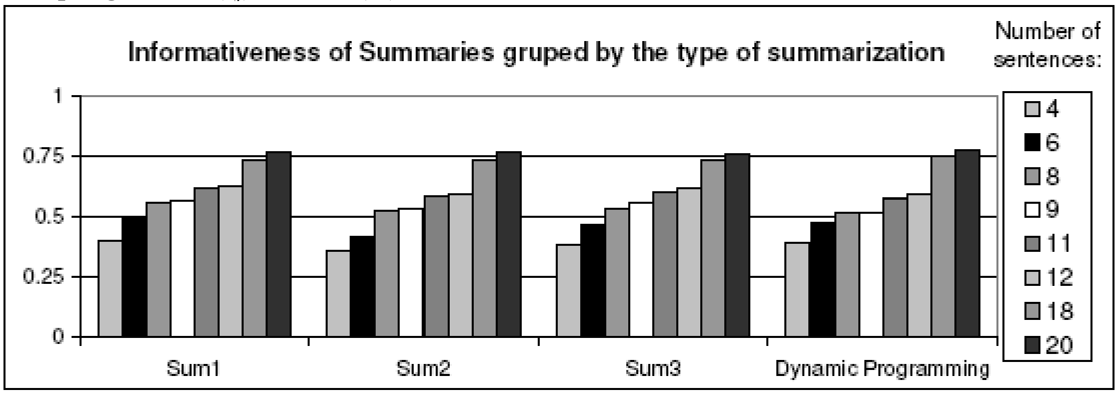


Figure 13. The informativeness of summaries for different lengths, obtained AL and those three definitions Sum_1, Sum_2, Sum_3, respectively the Dynamical programming method

Definition 3 Given a segmentation of initial text $T = \{Seg_1,...,Seg_n\}$, the summary is calculated as: $Sum_3 = \{s'_1,...,s'_x\}$, were $s'_1=s_1$, Seg_0 contains only the sentence s_1 and $\forall i = \overline{1,n}$, $Sum_3 \cap Seg_i = SelPropSeg_i$, $|SelPropSeg_i| = NSenSeg_i$, $\sum_{i=1}^{n} NSenSeg_i = x$ and $\forall s_j \in SelPropSeg_i$ and $\forall s_k \in Seg_i \setminus SelPropSeg_i$, $sim(s_j,Seg_{i-1}) \leq sim(s_k,Seg_{i-1})$, $sim(s_j,Seg_{i-1})$ represents the similarity between s_j and the last sentence selected from Seg_{i-1}. The similarity between two sentences s, s' is calculated by $cos(s,s')$.

The informativeness of the summaries with different lengths obtained by AL combined with those three definitions Sum_1, Sum_2, Sum_3, respectively Dynamical programming method, computed relative to the original text is presented in Figure 13.

It can be noticed that the best results were obtained by the AL algorithm with the definition Sum_1. In the same time, it can be noticed the positive influence of the segmentation on the summarization For the lengths 9 and 12, in 5 of 6 cases the summarization preceded by segmentation method provides better results to direct summarization method.

3.3.3 Lexical chains based summarization

This summarization use the same AL algorithm previously presented, only that the initial AL algorithm is considered to be the first version *Var1*, where the score of a segment is computed using the formula $Score(Seg_j) = \frac{\sum_{S_i \in Seg_j} Score(S_i)}{|Seg_j|}$. The second version of the same algorithm, *Var2* differs only by the score computing formula for a segment: $Score(Seg_j) = \sum_{S_i \in Seg_j} Score(S_i)$. Both have their advantages and disadvantages, version *Var1* may disadvantage long segments with only few large score sentences, whereas *Var2* gives an increased importance to long segments.

Starting from two segmentations, by LTT (subchapter 3.2.1) and CTT (subchapter 3.2.4), the two versions *Var1* and *Var2* of the arbitrary length algorithm AL, and also three summary definitions Sum_1, Sum_2 respectively Sum_3, a combination of them was tries into an potentially ideal summary IdS, by taking the majority occurrences of the sentences in all obtained summaries.

For evaluation, the texts "The Koan" [Ric91], Hirst [Mor91], Tucker1 and Tucker2 [Tuc99] were used. The similarity of every summary to the initial text was computed (Table 2). As a conclusion, the LTT and CTT segmentations favours the obtaining of quality summaries. Among them, CTT leads to better results, partially due to the fact that the precision of text entailment relationship is less than the one of word disambiguation (CHAD algorithm).

Koan text							
seg.	summary type	summary length					
		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	

Hirst text							
seg.	summary type	summary length					
		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	

Tucker1 text							
seg.	summary type	summary length					
		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	

Tucker2 text							
seg.	summary type	summary length					
		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	

Table 2. The similarities for the automatic summaries

IV. The ontology usage in requirement analysis

Although natural language processing techniques are useful in processing requirements for obtaining specifications, preferably formal specifications, additional resources are needed. The most useful resource is ontology, since ontology allows the association of semantic information to a natural language text, text that otherwise cannot be understood by the computer.

4.1. Ontology matching versus text

Currently, the amount of semantic information existing in the web has increased. However there is an enormous amount of information written in natural language without any semantic support, that “human beings cannot organize it all” [Pol09]. And since different types of ontologies exists, obtained in various ways, the question of associating an ontology to an existing text prevails in front of an ontology extraction from text.

Ontology evaluation

Considering the different types of learned ontologies and the way in which they were obtained, a quality assurance mean must be enforced. In [Gan05b], the ontology quality assurance metrics are classified in three types: structural measures, functional measures and usability-profiling measures. The ontology matching versus text belongs to the second category, so the following proposed metrics will be precision and recall of the proposed matching criteria [Mih10b].

The need to identify the proper ontology for a given task was discussed in different articles. In [Eng05] the ontologies are searched and evaluated regarding a set of keywords. In [Tan09], is proposed a framework for selecting the appropriate ontology for a particular biomedical text mining application. Another similar paper is [Doa03], which uses some similarities metrics and machine learning techniques, based on the names, context, constrains and labels.

Metrics for the evaluation of ontology matching versus a natural language text

So, there are a lot of ontologies, many of which being continuously enriched. But from the point of view of natural language processing, the following question has to be answered: Which ontology is the best ontology to be used for a particular text, and which part of the ontology. In order to find this answer, the usage of the some ontology evaluation metrics was proposed in [Mih10b].

The first proposed metrics will evaluate how many of the existing concepts can be found in the natural language text. Because all kind of ontologies can be represented in the triple form (concept – predicate – object), the ontology is considered to be in the triple form.

Quantitative metrics

The simplest ontology matching to the text technique is to search the concepts in the text. So, the first proposed metric, similar to an ontology quality metric and to an ontologies matching technique is *the number of concepts found in the text*. Because the concepts can appear in the text both in singular as in plural form, was considered that the concept was found in the text, if is found at least one word which starts with the same letters.

Some derived metrics can be evaluated in a similar manner: *the number of roots found in the text*, *the number of leafs found in the text*, and the list can continue. The roots are those concepts which appear only in the left parts of the triples, and the leafs are the concepts which appear only in the right parts of the triples.

Text entailment base metrics

In the last period of time, the text entailment evaluation techniques have improved. This is the reason for which the use of text entailment in order to evaluate the ontology matching with a text was proposed in [Mih10b]. And the reason is simple, as the most simple and popular way to represent an ontology is by using which is similar with a simple sentence, which is usually composed from a subject, a predicate and an object.

The use of text entailment relation in the ontology versus text matching process is by checking if the ontology, or bits of the ontology entails the natural language text. The following metrics were proposed: *entailment* (the ontology entails the text), respectively *number of entailments* (how many triples from the ontology entail the text). In the same time, it can be checked how many sentences are entailed by an ontology triple, and so the third measure was proposed, *the number of entailed sentences* (every sentence entailed by a different triple is numbered), and *the number of distinct entailed sentences* (only distinct sentences are numbered). For the text entailment check, the cosine method was used (see 3.1.3).

The **precision** (how many matches are recognized from the existing matches) and the **recall** (how many matches are recognized from the total number of tested elements) were evaluated. In order to **evaluate** the proposed metrics, a set of nine ontologies taken from the [***WnB] was used. All the ontologies have as core the “business” term, but with different senses. The nine ontologies were compared with the first part of an article from Wikipedia about “small business” [***WSM]. From the nine ontologies, the first four and the sixth seem to have the greatest chance to match.

Unfortunately, the entailment measures aren’t accurate enough, especially because the “business” concept is common to those nine ontologies. The quantitative metrics proposed confirmed the supposition that only the first four ontologies and the sixth one match the text (for the first ontology the recall 10.5%, for the second 8.1%, for the fourth 20%, and for the sixth 6.5%).

4.2. Ontology learning from text based on the syntactic analysis tree of a Sentence

The result of the grammatical analysis of a sentence is usually represented as a tree. Or between the words of a sentence, dependence relations can be identified [Mar08], relations which are astonishing similar to the RDF triples, the simplest way to represent an ontology. For the grammatical analysis of a sentence, the online tool developed by the Stanford Natural Language Processing Group [***STO], was used. The tool has accuracy greater than 87% [Kle03].

In the process of ontology learning, it was recognized the importance of syntactic analysis of text. But the grammatical relations were used only as recognition patterns in [Cim05, May09], or as constraints for the identification of relations between ontology concepts in [Kaw04].

Syntactic Analysis of a Sentence

The Stanford Parser [***STS] constructs the syntactic analysis tree of a sentence, identifies word dependencies and collapsed (for example: *nsubjpass*(abbreviated-6, title-3)). Starting from the dependencies, a graph can be constructed (see Figure 14, corresponding to the sentence "This function title is often abbreviated to the initials 'HR'." [***WHR]).

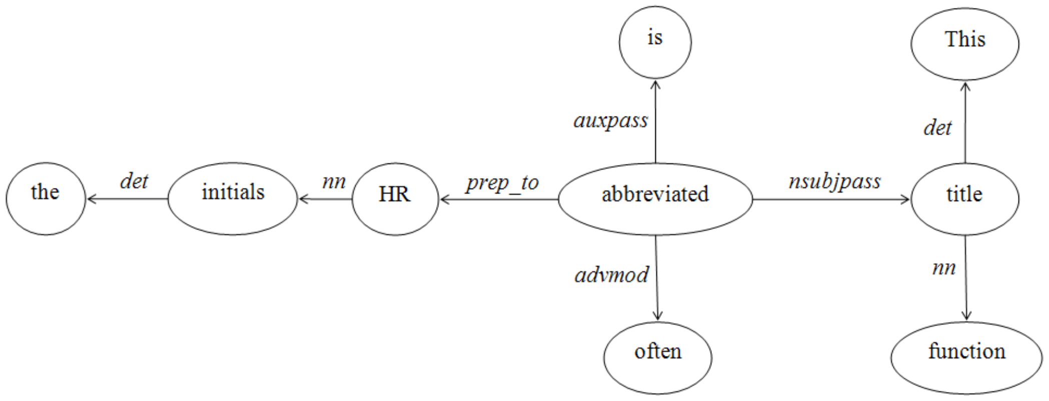


Figure 14. The graph obtained from the type dependencies

The triple identification

The first proposed method of triple extraction is **the triple extraction from the dependencies graph** (see Figure 15) [Mih10c].

```
@remove all the dependencies in which the predicate is "det"
For @every dependency $D_1$ do
  For @every dependency $D_2$ different then $D_1$ do
    If @$D_1$ and $D_2$ have common words then
      If @the common word is a Verb and the distinct words are Nouns then
        @add the triple (distinct word from $D_1$, common word, distinct word from $D_2$) to the triple_list
      endif
    endif
  endfor
endfor
```

Figure 15. Triple extraction from the dependency graph

```
Subalgorithm IdentifyTriple(S @a sentence) is
  @read symbol SB
  if @SB is NN, then @the concept from the triple is the following word endif
  if @SB is NP then @analyze the constituents endif
  if @SB is S  then IdentifyTriple(@new S) @and take the concept as the concept of the new S endif
  @read symbol SB
  if @SB is VB then @the predicate from the triple is the following word endif
  if @SB is VP then @analyze the constituents endif
  @read symbol SB
  if @SB is NN then @the object from the triple is the following word endif
  if @SB is NP or ADJP then @analyze the constituents endif
  if @SB is S then IdentifyTriple(@new S) @and take the object as the concept of the new S endif
end IdentifyTriple
{in the "analyze the constituents", a similar process takes place read a symbol and identify it until current branch of the tree is finished, but if a conjunction or a disjunction is identified (for concepts and objects, or only a disjunction for predicates), then every element of the conjunction will be taken separately, and as a result, not a triple, but a series of triple will be obtained}
```

Figure 16. The triple extraction from the syntactic analysis tree

The second proposed method of triple extraction is **the triple extraction from the syntactic analysis tree** (see Figure 16).

For the methods testing, the same text and the same ontologies as in the previous chapter were used. A simple word similarity was used (one is included in the other), the stop words [***SPW] and the word "business" were not taken into account.

Method/Ontology	first	second	fourth	sixth
I	0/0	2+2(reversed)/1	0/0	0/0
II	2/2	2/3	1/1	2/2

Table 3. The partially match triples inferred from the text/belonging to the ontology

The result of the evaluation can be seen in the Table 3. Because the most triples partially match the triples from the second ontology, for both methods the tested small business article is identified as matching the ontology in which the business is business enterprise (the expected answer).

4.3. Similarity used in the identification of the need to refactoryze an ontology

The existing ontologies are obtained in various forms, from manual way, the work of different specialists or volunteers [Wil06], to semi-automatic [Mor06] and automatic ways [Dah06]. Moreover, the need to enrich an existing ontology was discussed [Suk08, Che06]. Some are developed in a longer period of time. Some have an impressive number of elements. And when in a work of big proportions a lot of sources are involved, furthermore, from the moment of conceiving and the moment of final release, a lot of time passes, then is more than likely that in the final product, in the final ontology exists some discrepancies, some misplaced elements.

Different similarity measures were used in the ontology evaluation [Bra05], ontology alignment [Euz04] or ontology matching [Euz07]. But usually the similarity measurements used are word similarity, evaluated usually by string measurements. Homonyms, although written in the same way, have different meanings and are wrongly recognized as identical. Another kind of similarity used in the natural language processing is text similarity. In the text similarity are considered almost all the words appearing in the text (usually there is a list of stop words which are ignored), and the similarity is computed using their similarity measurements.

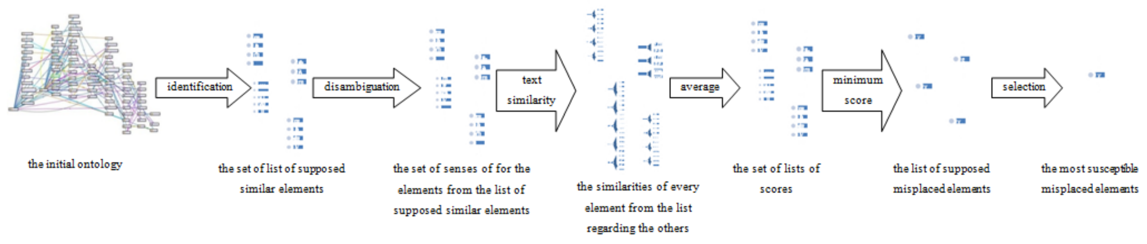

identification
disambiguation
text similarity
average
minimum score
selection
the initial ontology
the set of list of supposed similar elements
the set of senses of for the elements from the list of supposed similar elements
the similarities of every element from the list regarding the others
the set of lists of scores
the list of supposed misplaced elements
the most susceptible misplaced elements

Figure 17. The process of selection of the most susceptible misplaced elements

Considering the case in which the ontology is represented by triples, then is possible to identify sublists of elements which are related by the same predicate to the same concept or predicates bound by the same two concepts. This kind of **elements** must be **similar**. Evaluating the similarity of the two by two elements, an average for every element is computed, the similarities are averaged for each item, then the element with the lowest score is the most likely to be **misplaced** (see Figure 17) [Mih10d].

The algorithm for selecting the most susceptible misplaced elements

```
Algorithm Selecting_the_Most_Susceptible_Misplaced_Elements (O, n) is
  data: O- the ontology given as a set of triples
    n- the number of searched misplaced elements
  @extract the list of distinct elements (concepts) from the ontology
  For @every distinct element from the ontology do
    @identify the list of distinct elements which are bent by the same predicate to the current element
    @identify the list of predicates linking the same element with the current element
  endfor
  @keep in a set S only the lists with at least three elements
  @initialize an empty set of lists S'
  For @every distinct element e from the lists belonging to the set S do
    @take all the distinct predicates and elements related to e, and construct a list L
    @add to the list L the current element e
    selected_sense=1
    min_sim=1
    For @ every sense i  from the dictionary, sense of the name of the current element from L do
      @compute the similarity s of the text composed by the names of the elements from L and the sense
      If @this is the first sense then
        min_sim=s
      else If min_sim>s then
        min_sim=s
        selected_sense=i
        endif
      endif
    endfor
    @add in S' in a analogue position to the position of the current element e the selected_sense
  endfor
  i=0
  @initialize an empty set S"
  For @every list L' from S' do
    mins=1
    poz_min=1
    i=i+1
    For @every text t from the list L'do
      nu=0
      sum=0
      For @every text t' from the list L' do
        If e≠e'then
          @add to sum the similarity of t relative to t'
          nu=nu+1
        endif
      endfor
      avg=sum/nu
    endfor
    If i=1then
      mins=avg
    else If mins>avg then
      mins=avg
      poz_min=i
      endif
    endif
    @add to S" the pair formed by the element from the position i from the list L of set S (where list L is the
        "parent" list of the list L') and its score, avg
  endfor
  @order the elements from the set S" by the scores and keep the first n
  results: the first n elements with the lowers scores
end Selecting_the_Most_Susceptible_Misplaced_Elements
```

Notice: in the algorithm, by distinct elements or relations are understood the elements and the relations which have distinct URIs.

As an **example of operation of the algorithm**, it is assumed that in a ontology the words "man", "women", "elderly" and "driver" are all subclasses of class "adult" and after disambiguation, the first sense from WordNet [*** WNT] was identified for all four words:

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)

After computations, "driver" gets the lowest score.

4.4. The role of disambiguation in ontology evaluation

The disambiguation was already used in the ontology field several times: in order to enrich the ontologies [Ste02], to learn ontologies [San07]. In the following is discussed the role played by the disambiguation ontology alignment respectively for the ontology matching versus a natural language text.

Ontology and disambiguation

The diversity of the means through ontologies were obtained does not guarantees that the same URI is used for the same concept when the concepts belong to different ontologies. But every concept has a name, name which can be used to decide if two concepts are identical or not, even if the URI are different. But in the case of homonyms only a simple word similarity measure isn't enough. Therefore the ontology elements bounded directly to the concept which is disambiguated must be also used [Mih10e]:

The disambiguation algorithm

```
Function RecursiveDisambiguation(WORD,SET,ONTOLOGY) is
  data: WORD,SET,ONTOLOGY
  preconditions: the WORD which must be disambiguated and the SET of its neighbors from the
                 ONTOLOGY
  SENSE ← 0
  max ←0
  for @each sense i of WORD from the dictionary do
    @evaluate the overlap score of the SET and the gloss i from the dictionary
    if @measure > max then
      SENSE ← i
    endif
  endfor
  if max = 0 then
    if @exists new neighbors in ONTOLOGY for the elements from SET then
      @add the neighbors to the SET
      SENSE ← RecursiveDisambiguation(WORD,SET,ONTOLOGY)
    endif
  endif
  results: SENSE
  postconditions: the index of the correct SENSE of the WORD, 0 otherwise
end RecursiveDisambiguation
```

The presented approaches emphasize the role played by the natural language processing techniques in the problem of semantic information association to a natural language text, and also of improving the ontology quality.

4.5. Ontology assisted requirements analysis

Since the ontologies make the computer capable to better understand the natural language, they have a great applicability in the field of information extraction. Since 2003, their applicability for information extraction was emphasized in [Mae03], where the authors propose a step by step method for enriching an existing ontology in order to make it more appropriate for information extraction from a new text source.

In a newer paper [Yil07], an ontology unsupervised method for information extraction is presented, without using any other knowledge sources. However, the precision of the results will depend on the quality of the input ontology. The method also identifies unused elements from the ontology, and in this way the quality of the ontology can be improved, and also the results of the extraction.

The semi-automatic formal specification extraction

The scope of the paper [Mih10f] is to extract an abstract program from a natural language requirements text. In other words to identify the precondition, postcondition and the variable list.

```
Subalgorithm PrePostSelection(g, pre_list, post_list) is
   data: g - a dependencies directional graph
   @Identify the VB words list: VB_list
   @Initialize a list of lists of words, Word_ll and place in every list on the first position the VB
      word
   @Initialize an empty list of distinct words, Var_l
   For @every list L from Word_ll do
      @Identify the graph path which ends with a dependency "sub" for a VB or a VBD, and if
      none exist, the path which ends with a dependency "obj", respective the path
which ends with an "amod" dependency for a VBN
      @Add all the words from the path to L
      @The NN word which is in relation "sub", "obj" or "amod" must be added to the variable list
      Var_l
      @Add all to L all the words connected to the last word in a recursive way (they and all the
      words connected to them)
   endfor
   results: pre_list - the lists L from Word_ll which starts with a VB, post_list - the other lists from
            Word_ll, Var_l - the list of variables
endSub
```

Figure 18. Phrase analysis algorithm

There are two types of requirements: requirements which are expressed by many sentences, and many requirements expressed in a single sentence. In the first case, the identification of the preconditions and postconditions seems to be much easier to be resolved, since it can be reduced to a sentence selection problem. In the second case, a single sentence must be split into one or more preconditions and one or more postconditions. In the second case, the **sentence** itself must be **analyzed**. Again the Stanford Parser [***STS] is used. A precondition is expressed in natural language as a past or present sentence, and a postcondition as a future sentence. The variable list is represented by the subjects of those sentences. The corresponding algorithm is in Figure 18.

In the case in which the requirements text contains many sentences, the proposed algorithm can be applied after the sentences are analyzed, triple extracted, synonym identified and subordinate words identified, binding in this way the dependency graphs (see subchapter 4.2). From every sentence, the dependency graph can be constructed and this graph is a mini-ontology (it contains entities and relations). If the requirements volume is great, they must be segmented first.

For the further refinement from natural language preconditions/postconditions into abstract programs or Z schemas, is necessary the use of an ontology, which for instance for the case of Z schemas, has as base-nodes the base types and the base operations for these types. For instance, for "*Generate the first prime number larger than a* *given natural number n*.” [***Adr], it can be noticed that, the basic type *natural number* will be used, with *prime* and *larger than* as operations, and that only *first* is not a simple predicate involving a natural number.

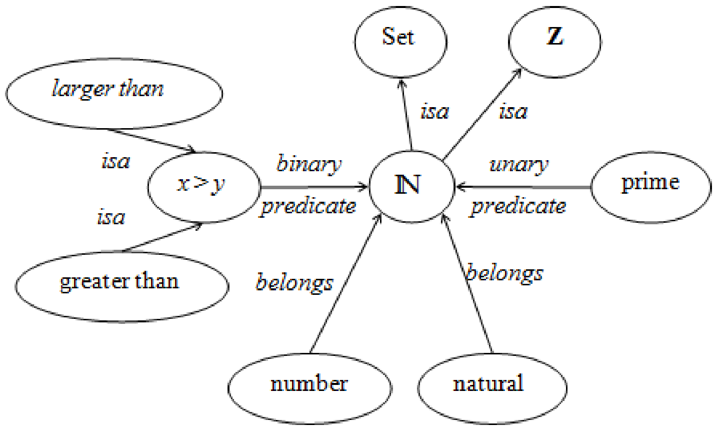


Figure 19. A part of Natural numbers ontology

In this case, first words which represent the variables will be searched in the ontology (see Figure 19), and replaced with the closest base type (after the node "natural" is identified, the closest base type is N). Then, from the natural language preconditions and postconditions all the words are searched between the sub-ontology with the base root the base type identified. If they are found, then the base predicate directly connected to the base type will replace the current word.

In the discussed example, “*Generate the first prime number larger than a given natural number n*.” [***Adr], the identified list of words which express the postcondition is *Generate* + *larger* + *number* (the subject) *the* + *first* + *prime*. *Number* can be replaced by $x \in \mathbb{N}$, and also *larger* $x > n$. The computer can provide these base types/predicates, and in this way to assist the translation process from natural language into formal specifications.

Conclusions

Two sets of metrics were proposed for ontology matching versus text evaluation. The first are based on statistical evaluation of the number of concepts that appear in the text, and the last on the text entailment relationship.

Although the tests for the text entailment metrics have been inconclusive, partly due to the size differences between ontologies and the analyzed text, the first set of metrics has led to expected results.

Two methods of converting the result of a grammatical analysis of a text written in natural language into ontology triples were presented. The first proposed method provides fewer but concise triples, close to the computer and the second provides triples closer to the human user. They emphasize the role that grammar plays in the construction of an ontology or ontology matching the text.

Is also presented a method able to identify precisely the most likely misplaced items in an ontology and help developers to improve the quality of the ontology.

In the end a semi-automatic method for extracting formal specifications from a natural language text is presented. It uses the Stanford Parser to obtain the dependency graph, followed by a process of unification of ontologies to connect mini-ontologies, and in the end based on semantic principles, natural language sentences, which are preconditions and postconditions, are extracted.

V. Aspects of formal methods usage in developing correct programs

5.1. Software quality assurance – a continuous activity

It is well known the fact that from the effort associated with a software products development and deployment, maintenance represents 60%. To reduce this cost, product quality must be guaranteed from the first stages of development. In the case of large scale products, their maintenance is impossible unless they meet certain quality criteria. Measuring software quality is performed using software metrics, with specific metrics for different phases of product development.

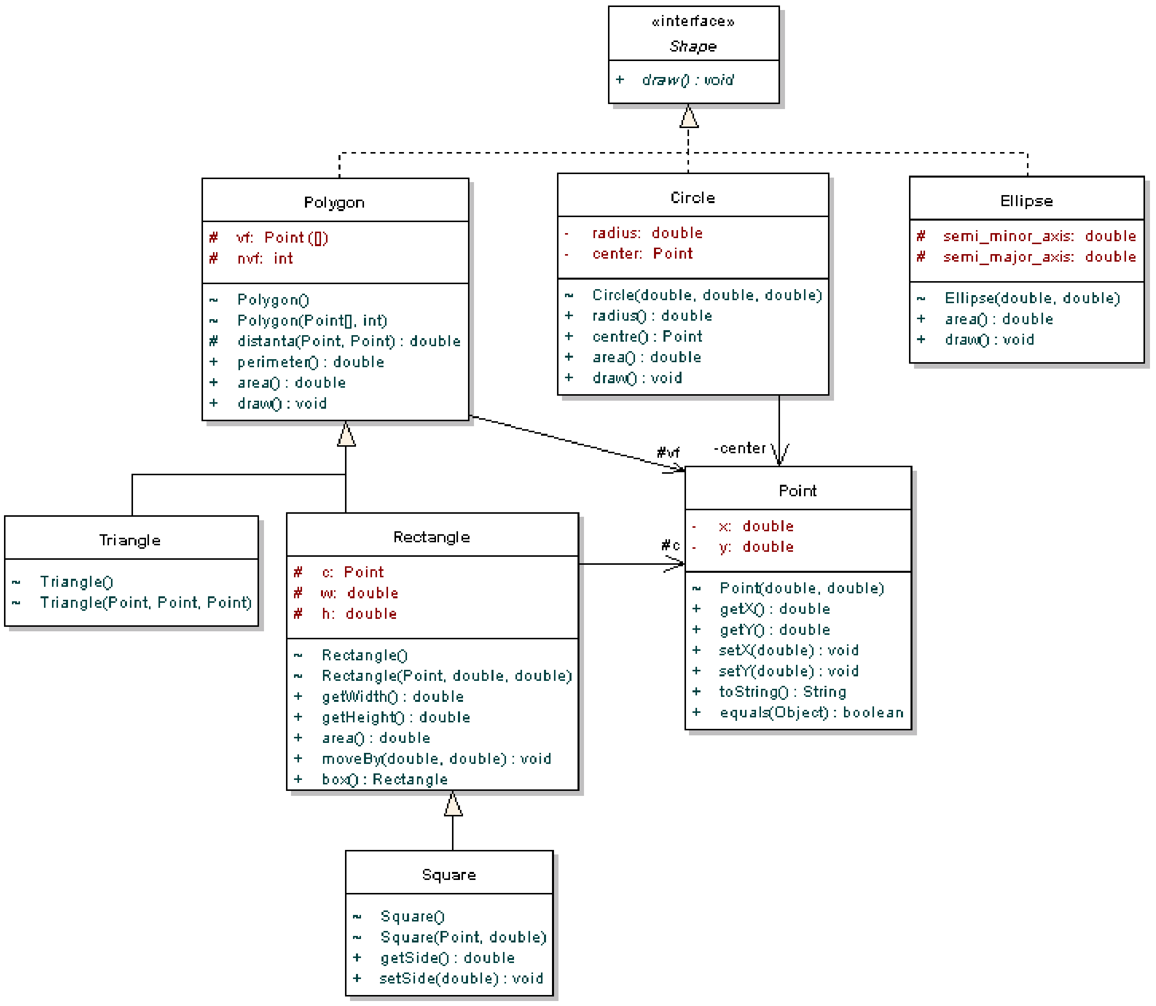


Figure 20. Geometrical shape hierarchy

One way to ensure software product quality is to evaluate the quality in every phase of product development and improve it. To demonstrate the importance of software quality evaluation early in the development process, a class hierarchy was designed in [Şer05] (see Figure 20). In Table 5 are the results of the evaluation of the classes from the hierarchy. It can be noticed that it is possible to improve this class hierarchy by deriving the class Circle from the class Ellipse, case in which the complexity will decrease by reducing the number of operations and operators of class Circle. Also COM metric value should be higher, which means that some additional comments must be added.

Class/Metrics	SIZE (the number of code lines)	COM (comments percents)	WMC (method weight of a class)	DIT (inheritance tree depth)	NOC (the "children" number)	LCOM (lack of methods cohesion)	CBO (class coupling)
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

Table 4. The results for the class metrics evaluation

Another method is to develop high quality software using design templates or formal methods. Unfortunately there is no general model available to ensure software quality. For each development process, quality requirements should be specified from the beginning, pursued throughout the development process and the discrepancies should be resolved in the early stages, when the changing cost is reduced.

5.2. Multiformal approach to specifying software systems

Large scale software systems have several aspects. Each aspect requires a specific formalism application and formal checking using specific tools. Therefore specifications obtained by applying different formal methods must be integrated, resulting a multiformal specification (integrated or heterogeneous).

Formal methods are classified into: state-oriented (or model-oriented), property-oriented (axiomatic or algebraic) and hybrid. State-oriented formal methods are: Z [Abr80], VDM [Jon86], B [Joh73], finite state machines [Gil62], Petri nets [Pet62], CCS [Mil80], CSP [Hoa85]. Property-oriented formal methods are: ADR [Ast02] ACT ONE [Ehr83], Anna [Luc85], Larch [Gar93] CLEAR [Bid91], OBJ [Fut85], LOTOS [Eij89]. Generally, the integration of formal methods is the combination of two or more complementary formal specification methods.

When specifying a complex system, a homogeneous or a heterogeneous approach can be applied. Homogeneous approach describes multiple aspects of software system using a single formalism, able to express all the aspects of the system. Such specifications can be made or by specifying new languages, or by expanding an existing one. The heterogeneous approaches, multiformal ones, use several existing formalisms to cover all the aspects of the specified system. Depending on how the languages are syntactically combined, this approach can be a powerful integration of formalisms, or a composition/coupling of the independent specification parts, each part being written using another language.

For syntactic combination of languages there are three basic approaches. The first approach uses the graphic representation of the behaviour (Petri nets, state diagrams, and labelled transition systems) and different data types (algebraic specifications, B, Z, VDM). The second approach consists in combining process algebras such as CCS or CSP with algebraic formalism. Such a multiformalism is LOTOS. A third approach tries to integrate process algebra (CCS, CSP) with a state-oriented formalism (Z, Object-Z [Smi99], B, VDM). Such an integrated formalism is ZCCS [Gal96].

The prerequisites of a multiformal approach are: learning formal methods, using integrated methods and the necessary tools. One of the main disadvantages of formal methods that are being difficult to learn because of notations, concepts and mathematical methods, can be an impediment for a multiformal approach. In addition, combining different notations can introduce ambiguities, inconsistencies and the resulting specifications may be difficult to understand. But it was argued that by using several methods, each in the best cases, specifications will get shorter, clear and concise towards the use of a single formalism [Cio04].

5.3. Formal specifications reuse

When a new version of software is produced, its code can be reused, but not only its code can be reused, specifications, documentation and other secondary software products can be also reused. From the different types of specifications, formal specifications are the most useful and in the same time the harder to develop. As a consequence, to reuse this type of specification will prove to be very useful. Another case of formal specifications reuse is to build a product for multiple platforms.

Also, formal specifications can be used to identify reusable elements. For example if there is a library of components and the production of a new one is desired, but reusing as much as possible, for the identification of reusable components is better to use formal specifications, since in this case the similar components are spotted immediately.

The advantage of using Formal specifications is that they are in a mathematical form, and so it is easy to identify a bijective function between them, and so to prove that they are similar. K. Periyasamy and J. Cidambaram in [Per96] have defined how two Z specifications are identical or analogous, from declaration, signature and property point of view. The standard form of formal specifications guarantee the fact that the same specification will always have the same form.

Moreover, analogy is one of the most used learning methods. It can be applied in formal methods learning too. It is easy to take an example and produce a new specification similar to the first. Thus similar formal specifications can be used to facilitate learning of those formal methods [Mih05].

Person

Id : $\mathbb{N}$

Name : STRING

Country: STRING

$\# \text{Name} \leq 20$

Book

Title : STRING

Author's Name: STRING

Id : $\mathbb{N}$

$\# \text{Title} \leq 20$

Person Set

Persons: **P** Person

$(\forall p_1, p_2 : \text{Persons} \bullet p_1.\text{Id} = p_2.\text{Id} \Rightarrow p_1 = p_2)$

Book Set

Books: **P** Book

$(\forall b_1, b_2 : \text{Books} \bullet b_1.\text{Id} = b_2.\text{Id} \Rightarrow b_1 = b_2)$

Figure 21. Z data schemas

For instance, let assume there are the specification for a Person and a Person Set, as it can be seen in Figure 21. It is wanted to specify a Book and a Book Set. From the discussions about the Book, it is noticed that those two entities matches. Using Person as a model, immediately specification for Book is written, as shown in Figure 21.

Another case of formal specifications reuse is where it is necessary to combine two components. To avoid possible mistakes, it is better to compose their specifications first. If their specifications are in a formal form, composing them is easier, usually by applying a rule in order to obtain the specification for the resulting component.

Similar elements will have similar specifications.

If it is noticed that two elements are similar, only one will be specified, and then similarly, through specification reuse the other elements specification will be obtained. The similarity of the two specifications can be a warning to the customer, if the two elements are not actually similar, in the case in which the customer's requirement was not fully understood by developers.

5.4. An application that assist Z language usage

Having as main purpose the encouraging of the use of formal methods in general and Z language in particular, an application that assists in a semi-automatic way the refinement of Z schemes was made, application which will be described in the following [Mih10a].

In order to assist the Z schema refinement, the application must be capable of assisting the schema definition and the schema refinement process first. For the first part, the definition of Z schemas lately, and Z language editor was developed [Gao09]. In the new original approach the schemas can be defined only in the graphical manner, one schema at a time, by inserting all the components: name, declarations and predicates. All the special symbols can be found in a categorized list and added from there [Dil99]. In the case of selection of a symbol from a list, its definition and an example of usage will be available before the effective usage of the symbol into the schema definition, in order to help the new users of Z language.

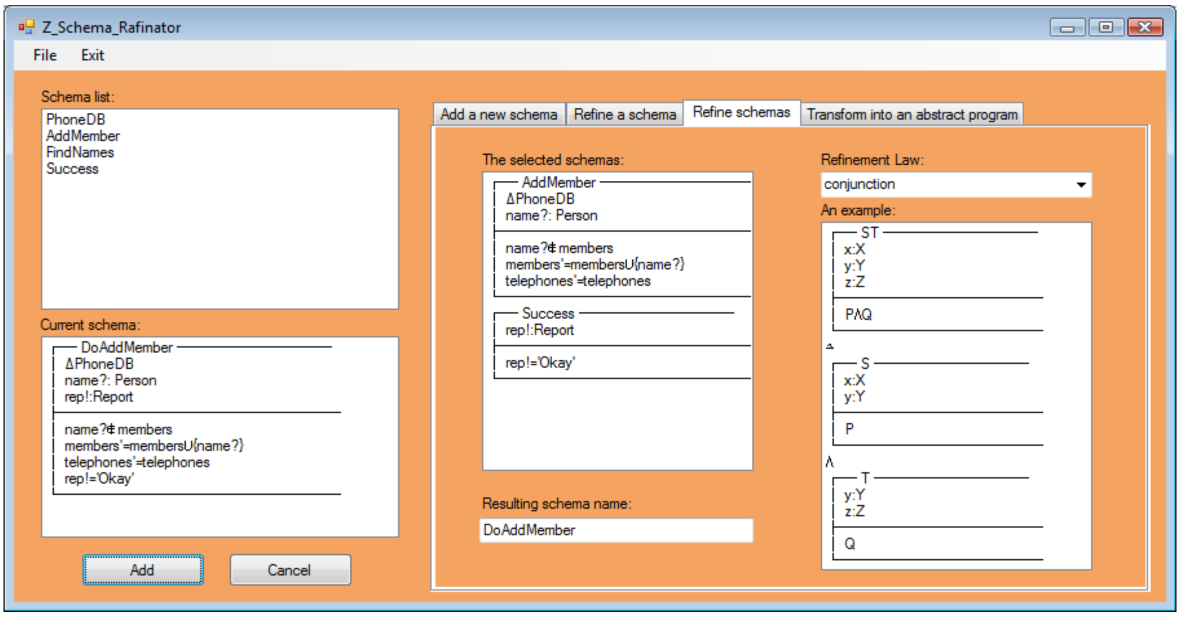
When the schema definition is finished, a syntactic analyze will take place with the purpose of base elements identification, according to the notations conventions [Mih10a], of the previously identified element lists and of the special symbols: “!” and ”?”. In the case in which the user noticed an incorrect identification, he can manually change the type of analyzed elements.

After a schema is defined, it is deposited in a schemas list, from where every schema is available by its name. Only a schema or two schemas can be selected at a time from the list, in order to refine them [Woo96].

If two schemas are selected, a binary operation can be applied, such as conjunction, disjunction and composition (see Figure 22). If one schema is selected, a unary operation can be applied such as denial or decoration. In the same time as the majority of the components are mathematical elements, various mathematical theorems or properties can be applied. All these operations can be carried out in assisted mode, as long as the application provides an example of using the selected operation, where the user must enter only the new elements. The collection of refining operations for the schema elements is continuously enriched adding by the user of new operations. The first use of a new operation will be stored and provided as example for future applications. Refining operations that can be applied to the selected component from the current scheme is identified on the basis of similarity relationship between two components.

Definition: Two components of two Z schemas are similar if they have the same number of base elements, in the same order and with the same types.

The most important refining operation that can be achieved is the transformation of a scheme Z in assisted mode into an abstract program. To transform a Z schema into an abstract program, the following elements must be identified: the frame, the precondition and the postcondition. The framework was previously identified by analyzing syntactic constituent elements of a schema Z. The variables must only be selected from the list of elements. The precondition is the conjunction of schema predicates containing variables whose names include the symbol "!" and the postcondition is the conjunction of predicate where the variable name ends with "?". Of course the user can intervene to correct errors. The obtained abstract program can be saved in a text file, for further processing.



Schema list:

```
PhoneDB
AddMember
FindNames
Success
```

Current schema:

```
┌── DoAddMember ──────────────
│ ΔPhoneDB
│ name?: Person
│ rep!:Report
├─────────────────────────────
│ name?∉ members
│ members'=members∪{name?}
│ telephones'=telephones
│ rep!='Okay'
└─────────────────────────────
```

The selected schemas:

```
┌── AddMember ────────────────
│ ΔPhoneDB
│ name?: Person
├─────────────────────────────
│ name?∉ members
│ members'=members∪{name?}
│ telephones'=telephones
└─────────────────────────────
┌── Success ──────────────────
│ rep!:Report
├─────────────────────────────
│ rep!='Okay'
└─────────────────────────────
```

Resulting schema name: DoAddMember

Refinement Law: conjunction

An example:

```
┌── ST ───────────────
│ x:X
│ y:Y
│ z:Z
├─────────────────────
│ P∧Q
└─────────────────────
≙
┌── S ────────────────
│ x:X
│ y:Y
├─────────────────────
│ P
└─────────────────────
∧
┌── T ────────────────
│ y:Y
│ z:Z
├─────────────────────
│ Q
└─────────────────────
```

Figure 22. The conjunction of two Z schemas

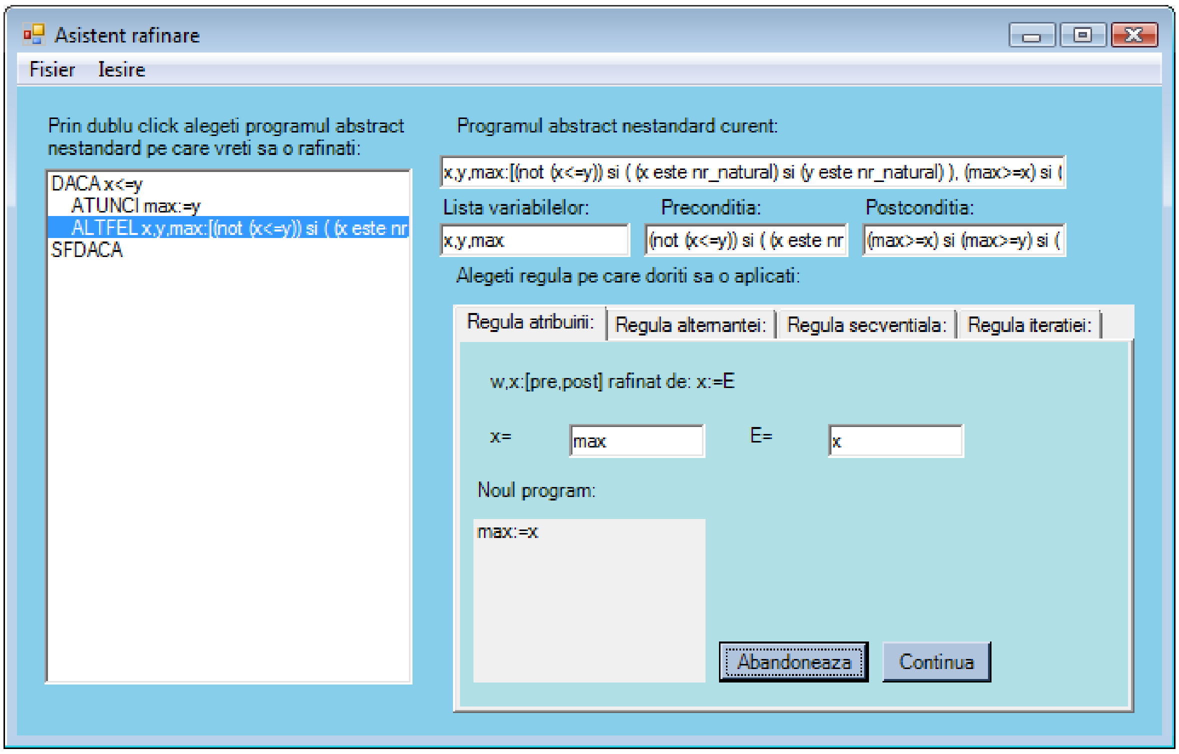
The application is primarily for teaching purposes and was developed in C #. The abstract programs obtained from Z schemas can be refined further, as can be seen from next subchapter.

5.5. An application that assist stepwise refinement

In [Mih06a] a mini CASE tool was presented, "Asistent rafinare" (Refinement Assistant), which allows the assisted refinement of code from abstract programs, tool described in the following. The abstract programs are specified by frame *w*, input predicate or precondition, denoted with *pre*, and output predicate or postcondition, denoted with *pos*: „*w:[pre,pos]*". The refinement assumes a step by step transition through refinement rules. The refinement rules are: attribution rule, alternation rule, sequential composition rule and iteration rule.

The application offers the following features: automatic using of the above rules for an abstract subprogram, after the user has identified all the necessary new elements for the refining process, as can be seen from Figure 23. The users can also change manually the code lines. The entire refining process is stored in a log file "log.txt", which allows the reuse of the refining process for similar cases. Also the current form of the program can be saved, and a previously saved program can be load.

This application makes the developers work easier, especially in the case of medium to large programs that require repeated application of refinement rules.



Prin dublu click alegeti programul abstract nestandard pe care vreti sa o rafinati:

```
DACA x<=y
   ATUNCI max:=y
   ALTFEL x,y,max:[(not (x<=y)) si ( (x este nr
SFDACA
```

Programul abstract nestandard curent: x,y,max:[(not (x<=y)) si ((x este nr_natural) si (y este nr_natural)), (max>=x) si (

Lista variabilelor: x,y,max

Preconditia: (not (x<=y)) si ((x este nr

Postconditia: (max>=x) si (max>=y) si (

Alegeti regula pe care doriti sa o aplicati:

w,x:[pre,post] rafinat de: x:=E

x= max

E= x

Noul program:

```
max:=x
```

Figure 23. The base window for the Refinement Assistant

5.6. Code simplification by automatic processing conditional expressions

Another small CASE tool is proposed in [Mih06b] and presented in a subchapter of [Lup08/09], tool that favour code quality improvement through applying Boolean function simplification to the conditional expressions existent in the code. This is because each simple conditional expression has only two truth values: true, can be denoted by *1* or false, denoted by *0*. If all the simple conditional expressions are denoted by x_i, $i=\overline{1,n}$ variables, where n represents the number of conditions, then the complex conditional expression can be replaced by a Boolean function $f(x_1, x_2,...,x_n)$. So the problem of conditional expression simplification can be reduced to a Boolean function simplification method, a process that was automated in the tool called BOOFS (BOOlean Function Simplifier).

The method used for the simplification was Quine-McClusky method combined with Moisil method [Tăt99, Lup08/09]. This method can be applied only to the disjunctive canonical form of a Boolean function, so a normalization process must be applied.

In order to effectively assess the improvements obtained by the simplification method, the BOOFS tool automatically evaluates some metrics based on the number of operators (connectives) and operands, metrics defined as follows: *count of distinct operands*, count *of all operands*, count *of distinct operators*, count *of all operators*, sum *of priorities of distinct operators*, sum *of priorities of all operators*.

These metrics will take a natural number as value, and as a result of the simplification process their value must decrease. For metrics 1, 3 and 5 this fact is immediately. For metrics 2 and 4, there are cases in which their value will be in fact increased, as it can be seen from the following example. So maybe the best choice is metric 6. The priorities used by the metrics 5 and 6 are from 1 to 5, corresponding to the operators: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$. These priorities were used as criterion in the code from the BOOFS tool.

For instance, for the conditional expression
((*x* < *y*) *and* (*y* <= *z*) *and* (*z* < 5)) *or* ((*x* < *y*) *and* (*y* >= 5) *and* (*z* < 5)),
which was simplified to ((*x* < *y*) *and* (*z* < 5)), the values of those 6 metrics dropped from 3, 6, 3, 6, 6, 12, to 2, 2, 1, 1, 2, 2.

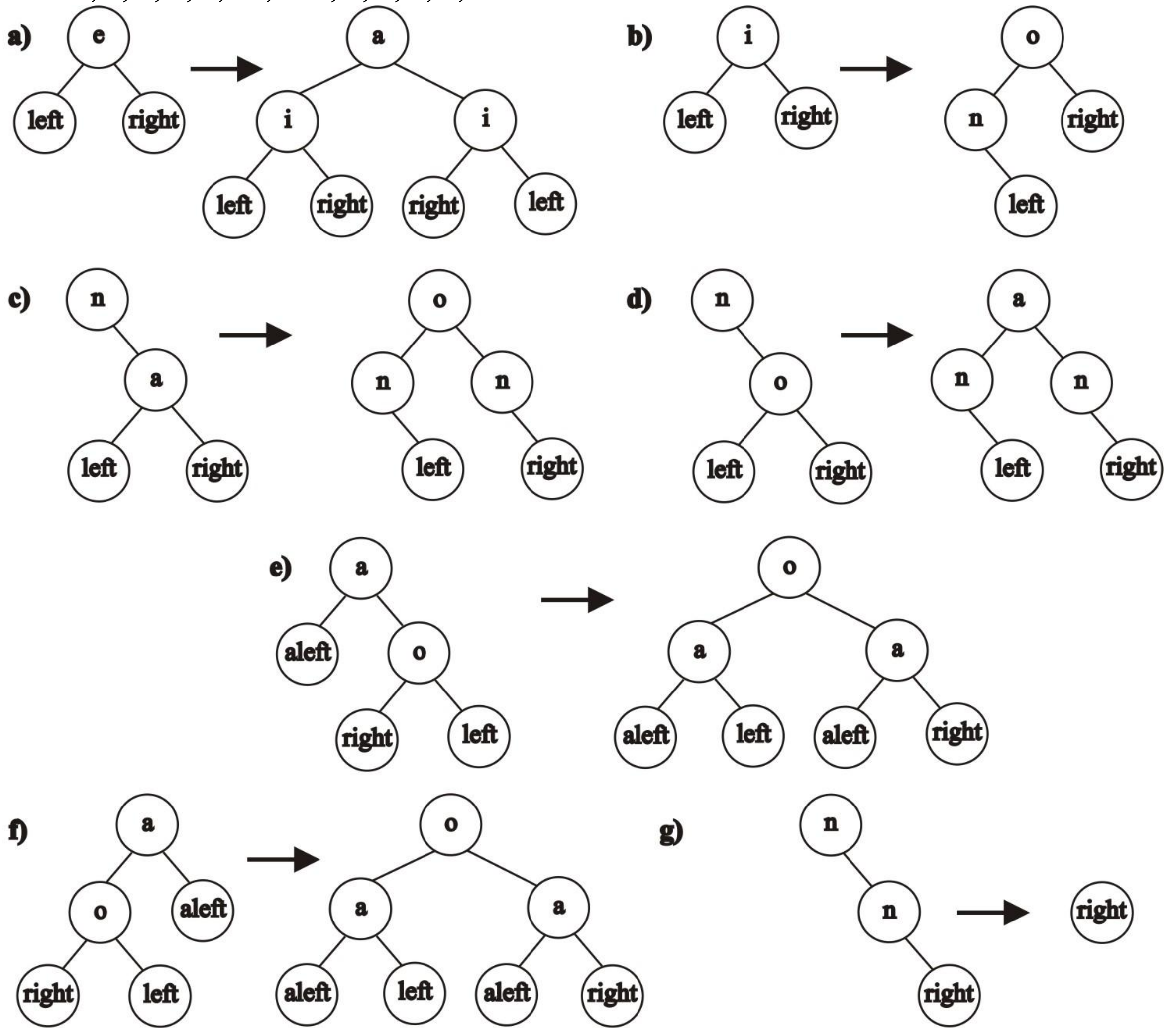


Figure 24. Binary tree operations used in the normalization process: a) equivalence elimination, b) entailment elimination, c) and d) De Morgan’s rules , e) and f) distributive law (to the left respectively to the right), and g) double negation law.

The BOOFS tool was made in Java. The execution begins by opening a source code file, from which a conditional expression to be simplified can be selected. Because the application cannot automatically identify simple conditional expressions, they will be indicated manually. Simultaneously, as the application works regardless of the source code language, the user will indicate how the five logical connectives appear in the code. The connectives are memorized by letter: *n*($\neg$), *a*($\wedge$), *o*($\vee$), *i*($\rightarrow$) and *e*($\leftrightarrow$) and operands by x_1, x_2, x_3 and so on. The obtained Boolean function must first be normalized. The normalization was performed on a binary tree, by applying the operations from the Figure 24, through a back-tracking algorithm. To convert the function from infix form in the binary tree form, the function will be transformed into its postfix Polish form by a stack. Also using a stack will turn the Polish postfix form into the tree. After bringing the tree to the disjunctive form, disjunctive infix form of the Boolean function browse through an inordin binary tree.

The support set is built, by adding *n*-uples with components 0 or 1. Quine-McClusky's method is used to obtain the maximum monoms set. In the corresponding *n*-uples digit 2 was used to mark the simplified variables. The Moisil method is used to identify the simplified form. Moisil's method is based on turning a conjunctive normal form in a disjunctive one, transformation performed again on the binary tree.

VI. Conclusions

Currently the need for correctness is becoming stronger. If a software product does not meet its specifications, it will not be accepted by the client. The existence of complete specifications, accurate and clear is essential. Such specifications are formal specifications.

Unfortunately, learning and using of formal methods is not easy and is time-consuming. It is a real help the existence of a tool able to assist the development of such specifications. Such a tool, which allows construction of Z specifications, their composition and their transformation into abstract programs is presented in this paper.

There are several types of formal specifications, for example some are process-oriented, others data-oriented. Complex systems require a formal specification to capture all aspects of the application, requiring a multiformal approach for the specification. Formal specifications can be reused successfully, especially when developing a new version of a software program whose formal specification already exists.

Starting from the formal specifications, correct code can be generated by refining based on rules. Although formal specification and refinement process may seem difficult to achieve, the entire development costs are reduced. The existence of tools which assist and automate part of this process is welcomed. Such a tool is presented in this paper. Another made tool simplifies by logical means the conditional expressions from code.

On the other hand, the formal specification cannot be realized usually by the developers together with the clients. In this case, developers can use natural language processing tools to facilitate their work, such an application which disambiguates, summarizes or segments a particular text.

Ontologies are excellent candidates for a more complex natural language processing, as is the requirements selection for specifications obtaining. However, the ontology used for this purpose must be chosen carefully.

This paper presents various aspects of using formal methods in developing correct programs, but contains an important part of natural language processing and ontologies. But all the methods and applications presented are intended to facilitate the use of formal methods in developing correct programs.

VII. References

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

[Alf85] Alford M., *SREM at the Age of Eight: The Distributed Computer Design System*, IEEE Computer, Vol. 18, No. 4, April 1985, p. 36-46.

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

[Bac00] Back T., Fogel D.B. and Michalewicz Z. (editors), *Evolutionary Computation: Basic Algorithms and Operators*, Vol. 1 and *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 machine*, NATO Science Committee Conference, 1969.

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

[Bar99] Barzilay R. and Elhadad M., *Using lexical chains for Text summarization*, editors J. Mani and 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. and Sanella D., *Algebraic System Specification and Development*, Lecture Notes in Computer Science, Vol. 501, Springer-Verlag, 1991.

[Bra85] Brachman R. and Schmolze J., *An Overview of the KL-ONE Knowledge Representation System*, Cognitive Science, vol. 9, no. 2, 1985, p. 171-216, http://nlp.shef.ac.uk/kr/papers/klone.ps.

[Bra05] Brank J., Grobelnik M. and 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, No. 1, March 1976, p. 9-36.

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

[Cim05] Cimiano P. and 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, Spain, 2005.

[Cio04] Ciobotariu-Boer, V. and **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, Romania, Lithography "Babes-Bolyai", Editor: Prof. dr. Militon Frenţiu, 2004, p. 21-26.

[Cla96] Clarke E.M. and 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. and Mihalcea R., *Measuring the semantic similarity of texts*, Proceedings of the ACL Workshop on Empirical Modelling of Semantic Equivalence and Entailment, Ann Arbor, June 2005, p. 13-18.

[Dah06] Dahab M., Hassan H. and Rafea A., *TextOntoEx: Automatic Ontology Construction from Natural English Text*, International Conference on Artificial Intelligence and Machine Learning (AIML-06), Sharm El Sheikh, Egypt, June 13-15 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, p. 297–302.

[Dil99] Diller A., *Z: An Introduction to Formal Methods*, second edition, John Wiley & Sons, April 1999.

[Dij75] Dijkstra E.W., *Guarded commands, nondeterminacy and formal derivation of programs*, Comm. ACM, Vol. 18(1975), No. 8, p. 453-457.

[Doa03] Doan A., Madhavan J., Dhamankar R., Domingos P. and 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, November 2003, p. 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. and 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. and Diaz M., *The formal description technique LOTOS*, Elsevier Science Publisher B.V., 1989.

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

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

[Euz07] Euzenat J. and 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, p. 19-32.

[Fre06] Frenţiu M., Pop H.F., *Fundamentals of Programming*, Cluj University Press, Cluj-Napoca, 2006.

[Fut85] Futatsugi F., Goguen J.A., Jouannaud J.P. and 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, United States of America, 1985, p. 52–66.

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

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

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

[Gan05b] Gangemi A., Catenacci C., Ciaramita M. and 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-cno.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, July 2009, p. 684-692.

[Gar93] Garland S.J., Guttag J.V. and 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, p. 329-348.

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

[Gli05] Glickman, O., Dagan, I. and Koppel, M., *Web Based Probabilistic Textual Entailment*, Proceedings of the PASCAL Challenges Workshop on Recognizing Textual Entailment, 2005, p. 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, no. 2, 1993, p. 199–220.

[Har97] Harabagiu S. and Moldovan D., *TextNet – a textbased intelligent system*, Natural Language Engineering, 3(2), 1997, p. 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. and Labaw B.G., *Automated Consistency Checking of Requirements Specifications*, ACM Trans. on Software Engineering and Methodology, 5(1996), no. 3, p. 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, p. 241-272.

[Joh73] Johnson S.C. & Kernighan B.W., *The Programming Language B*, Technical Report CS TR 8, Bell Labs, January 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, April 1987, p. 178-182.

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

[Kle03] Klein D. and 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, p. 3-10.

[Kǒu06] Kǒuylekov M. and 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, Venice, Italy, 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, p. 24−26.

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

[Luc85] Luckham D.C. and von Henke F.W., *An Overview of Anna, a Specification Language for Ada*, IEEE Software, Vol. 2, No. 2, March 1985, p. 9-22.

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

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

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

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

[Mar08] de Marneffe M.-C., and 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.** and 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, p. 500-505.

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

[Mih06a] **Mihiş A.D.**, *An Application that Assist StepWise Refinement*, Proceedings of Symposium „Zilele Academice Clujene", 2006, p. 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 și Cercetări Ştiinţifice, Seria Matematică, 16 (2006), Supplement, Proceedings of ICMI 45, editors: Mocanu Marcelina & Nimineţ Valer, Bacău, September 18-20, 2006, p. 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, p. 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, October 23-25 2008, Bucharest, Romania, ASE Publishing House Bucharest, ISBN: 978-606-505-124-9, p. 531-534.

[Mih08c] **Mihiş A.D.**, *Natural Language Processing Methods Used in Requirement Analysis*, „Zilele Informaticii Economice Clujene", Mediamira Science Publisher, Cluj-Napoca, editors: prof. dr. Niţchi Ştefan & all, 2008, p. 251-258.

[Mih08d] **Mihiş, A.D.**, *Various Types of Logical Text Tiling*, Zilele Academice Clujene, Cluj University Press, editor: Prof. dr. Frenţiu Militon, 2008, p. 129-133.

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

[Mih10b] **Mihiş A.D.**, *The Evaluation of Ontology Matching versus Text*, Informatica Economică/Economy Informatics, Categ. CNCSIS B+, vol. 14, no. 4, 2010, p. 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, Bucharest University Publishing House, Editor: Vlada Marian, Albeanu Grigore, Popovici Dorin Mircea, ISSN: 1844-8933, 1842-4708, http://c3.icvl.eu/2010, 2010, p. 128-134, article received the special award „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, p. 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, September 1-4 2010.

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

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

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

[Mor06] Morita T., Fukuta N., Izumi N. and 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, p. 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, No. 1, 1991, p. 21-48.

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

[Nel79] Nelson W.F. and 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*, Ph.D. Thesis, 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. and McKeown K., *Introduction to the Special Issues on Summarization*, Computational Linguistics, Vol. 28, No. 8, 2002, p. 399-408.

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

[Ros85] Ross T. D., *Applications and Extensions of SADT*, IEEE Computer, Vol. 18, No. 4, April 1985, p. 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. and 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, p. 245-254.

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

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

[Suk08] Suktarachan M., Thamvijit D., Rajbhandari S., Noikongka D., Mahasarakram P., Yongyuth P., Kawtrakul A. and 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, Morocco, 2008, p. 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, p. 207-212.

[Tan09] Tan H. and 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, June 4-5 2009, p. 55-62, http://www.aclweb.org/anthology/W/W09/W09-1307.pdf.

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

[Tăt07a] Tătar D., Şerban G., **Mihiş A.D.**, Lupea M., Lupşa D. and 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, June 6-8 2007, p. 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 & all, 2007, p. 53-58.

[Tăt08a] Tătar D., **Mihiş A.D.** and 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, p. 233-244.

[Tăt08b] Tătar D., **Mihiş A.D.**, Şerban G., *Lexical Chains Segmentation in Summarization*, SYNASC, IeAT Technical Report, 08-11, BDI, 2008, Timişoara, p. 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), p. 389-397.

[Tăt08d] Tătar D., **Mihiş A.D.**, Şerban G., *Top-down Cohesion Segmentation in Summarization*, Step 2008, Venice, Italy, September 22-24 2008, College Publications, editors: Bos J. and Delmonte R., 978-1-904987-93-2, BDI, 2008, p. 145-151.

[Tăt08e] Tătar D., Tămâianu-Morita E.S., **Mihiş A.D.**, and 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), p. 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, p. 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, p. 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, p. 1023 – 1038.

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

[Tei77] Teichroew D. and 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, January 1977, p. 41-48.

[Tuc99] Tucker R. I., *Automatic summarising and the CLASP system*, Ph.D. Thesis, University of Cambridge, Computer Laboratory, 1999.

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

[Woo96] Woodcock J. and 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, p. 660-673.

[You79] Yourdon E. and 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, December 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