Cascaded regular grammars and constraints over morphologically annotated data for ambiguity resolution*

Krasimira Ivanova, Dimitar Doikoff
BulTreeBank Project
http://www.BulTreeBank.org
Linguistic Modeling Laboratory, Bulgarian Academy of Sciences
Acad. G. Bonchev St. 25A, 1113 Sofia, Bulgaria
krassy_v@abv.bg, dim_doikoff@abv.bg

Abstract

The work that we present in this paper is part of BulTreeBank Project which aims at the creation of syntactically annotated data for Bulgarian and the tools for their production, management and automatic processing. It provides not only language resources, but develops an infrastructure of research solutions, production scenarios and services. Here we discuss some linguistic problems that can be solved with the help of CLaRK System's tools. We mainly use regular grammars, constrains and removal operations.

1. Introduction

This paper reports one of the steps towards the creation of a linguistically interpreted text archive of Bulgarian. In order to use the text archive in an efficient and linguistically motivated way, we envisage annotation of the texts with the relevant morphosyntactic information for wordforms and appropriate linguistic information for other kinds of expressions in the text, such as numerical expressions, special signs ($, %), foreign words and others. Sometimes these peculiar expressions bear lexemic information similarly to the wordforms, but sometimes they constitute phrases and need a special treatment. Additionally, we developed several partial grammars for processing the rest of the tokens in the texts.

Our work aims at providing better mechanisms for improvement of data processing and minimization of the linguistic work. The solution is implemented within the CLaRK System - an XML-based System for Corpora Development (Simov et. al., 2001).

The linguistic information is encoded in XML documents which are well formed over already developed standards for corpus description (XCES, 2001 and TEI, 2001). For navigation in documents we use XPath language (see XPath, 1999).

The central view on the use of the CLaRK system is that an XML document under processing can be seen as a "blackboard" on which different tools can write some information, reorder it or delete it. We used the following tools of CLaRK:

- Tokenizer tool with a module that supports a hierarchy of token types
- Finite-state engine that supports the writing of cascaded regular grammars
- Constraints tool - especially *Some Children Constraint*

* This work is funded by the Volkswagen Stiftung, Federal Republic of Germany under the Programme "Cooperation with Natural and Engineering Scientists in Central and Eastern Europe" contract I/76 887. The authors are grateful to Kiril Simov, Petya Osenova and the three anonymous reviewers for their comments on the paper.

- Removal operation

This paper discusses several methods for data processing that we called cascade-processing approach. Next section presents abstract view of used features. In the third section we describe the cascade regular grammars. Fourth section focuses on the writing of disambiguation rules. The fifth section presents the usage of temporary mark-up. The last section outlines the conclusion.

2. Abstract view of used features (tools)

In this section we present some of the tools used in our work and incorporated within the CLaRK System

- Linguistic markup
- Tokenizers
- Constraints
- Cascaded regular grammar
- Removal operations

2.1. *Linguistic markup*

We extended the level of data encoding by separating text into tokens - Cyrillin words, punctuation and others (numbers, unrecognized Cyrillin words and non-Cyrillin words). For our purposes we use the following annotation:

- **<w>** - for Cyrillin words
- **<pt>** - punctuation
- **<tok>** - for others - value of attribute **type** refer to the tokenizer category of the recent token.
  Possible values are:
  num - for numbers
  cyr - unrecognized Cyrillin words
  lat - latin words
  alphonum – for mixed tokens like U-571, MIG-29
  symbol - $, % etc

On the word level we have three subelements:

- **<ph>** - phonetics - orthographical representation of the word
- **<aa>** - all analyses - information from the morphological analyzer implemented in the CLaRK system based on (Popov, Simov and Vidinska, 1998) - all possible morphological characteristics associated with the current Cyrillin word.
- **<ta>** - true analysis - the appropriate morphological characteristic of the word in the current context

2.2. *Tokenizer*

XML considers the content of each text element a whole string that is unacceptable for corpus processing. For this reason, it is required for the word-forms, punctuation and other tokens in the text to be distinguished. In order to solve this problem, the CLaRK System supports a user-defined hierarchy of tokenizers. At the very basic level users can define a tokenizer in terms of a set of token types. In this basic tokenizer each type is defined by a set of UNICODE symbols. Above basic level there are tokenizers for which the token types are defined as regular expressions over the tokens of some other tokenizer, the so-called parent tokenizer.

For example we have **Default** tokenizer (basic for CLaRK system) with the following categories:

Category		Expression
LATc	'A'-'Z'	Latin capital letters

LATs	'a'-'z'	Latin small letters
CYRc	'A'-'Я'	Cyrillin capital letters
CYRs	'a'-'я'	Cyrillin small letters
NUMBER	'0'-'9'	Arabic digits
SPACE	' '	
……	There are other tokenizer categories	

We can build a new tokenizer (**Uptok**) based on the **Default** (parent) tokenizer. It will separate words with capital first letter from words with small letters. This is important for named entity recognition and sentence boundary. Here to define new categories we use names of categories defined in the parent tokenizer.

LATwc	LATc,(LATc|LATs|"-")*	Latin words starting with capital letter
LATws	LATs,("-"|LATs)	Latin words with small letters [1]
CYRwc	CYRc,(CYRc|CYRs|"-")*	Cyrillin words with first capital letter
CYRws	CYRs,("-"|CYRs)*	Cyrillin words with small letters[1]

The syntax will be explained bellow in *regular expression* – section 2.4.1.

2.3. *Constraints (Some children value constraints)*

Here we present the constraints of type "Some Children" which helps the underlying strategy of minimization of the human labor. This kind of constraints deals with the content of some elements. They determine the existence of certain values within the content of these elements. A value can be a token or an XML mark-up and the actual value for an element can be determined by the context. Thus a constraint of this kind works in the following way: first it determines to which elements in the document it is applicable, then for each such element in turn it determines which values are allowed and checks whether in the content of the element some of these values are presented as a token or an XML mark-up. If there is such a value, then the constraint chooses the next element. If there is no such value, then the constraint offers to the user a possibility to choose one of the allowed values for this element and the selected value is added to the content as a first child. Additionally, there is a mechanism for filtering the appropriate values based on the context of the element.

2.4. *Cascaded Regular Grammars*

Cascaded regular grammar (Abney, 1996) is a sequence of regular grammars defined in such a way that the first grammar works over the input word and produces an output word of categories, the second grammar works over the output word of the first grammar, produces a new word of categories and so on. The output of the last grammar constitutes the analyses made by the cascaded regular grammar.

2.4.1. *Regular Expressions*

For the grammar writing within CLaRK system we used regular expressions and we present here some basic principles of the regular expressions and their syntax:

- **Letter -** we call a letter every designate symbol. The symbol is usually understood as a symbol code in the computer.
- **Alphabet -** set of letters.
- **Word -** word is a finite sequence of letters over some alphabet and wildcards.

  Example:
  '"bet" - sequence of letters over Latin alphabet
- **Language** - is a set of words over some alphabet.

[1] We use dash for words like "well-formed"

- **Wildcards** (not letters)
  1 **@ -** matches one symbol.
  2 **#** - matches any number of symbols.

  Example:

  "be@" - all words starting with 'be' and having length 3 symbols. Possible matches are - 'bee', 'bet', 'bed' and so on.
  "be#" - all words starting with 'be' , for instance: 'be', 'bet', 'bed', 'bear', 'better', 'beautiful' and so on.

  Wildcards can be escaped by the symbol **^**

  "be^@" - matches only 'be@' -string

Syntax of Regular Expression - A regular expression over an alphabet is an expression that can be interpreted as the set of words over the alphabet. This set of words is the language that is recognized by the regular expression. Before giving the syntax and semantics of the regular expressions, we will enrich our formal inventory with *letter descriptions.* A letter description is an expression **ld** which denotes a set of letters from a given alphabet. Each letter from a given alphabet is a description of itself. Wildcards are descriptions of a letter from a given alphabet.

1 Basic case - if **ld** is a letter description, then **ld** is a regular expression
2 Concatenation (**,**) - If **r1** and **r2** are regular expressions, then (**r1,r2**) is a regular expression.
3 Union (|) - If **r1** and **r2** are regular expressions, then (**r1**|**r2**) is a regular expression.
4 Kleene Star (*****) - If **r** is regular expressions, then **r*** is a regular expression, which matches zero or more occurrences of **r**.
5 Kleene Plus (+) - If **r** is regular expressions, then **r+** is a regular expression, which matches one or more occurrences of **r**.
6 Question mark (**?**) **-** If **r** is regular expressions, then **r?** is a regular expression, which matches zero or one occurrences of **r**.

Regular expressions are interpretated in the usual way. The difference is in the interpretation of the letter descriptions, which denote the set of words constituted by a single letter from the set of letters described by the letter description.

2.4.2. *Cascade Regular Grammars in the CLaRK System*

In the CLaRK System, as it was mentioned above, there are: cascaded application of regular grammar (Abney, 1996); and extension of the rules with descriptions of the left and the right contexts of a subword recognized by the regular expression of the rule.

Every line in the grammar tool is a rule which consists of four columns: **LC** - a regular expression defining the left context of the words recognisable by the rule, **RE** - a regular expression defining the set of words recognisable by the rule, **RC** a regular expression defining the right context of the word and **RM** - return markup (category for the rule). The return markup is a custom markup that substitutes recognized word. Since in most cases we would also like to save the recognized word we use the variable **\w** for it. The variable **\w** can be repeated as many times as necessary (it can also be omitted).

The regular expressions LC and RC can be empty and then there are no constraints over the left and the right context. We envisage the use of such rules for tasks as sentence boundary recognition where one has to take a look at the word after the full stop in order to determine whether this full stop is marking the end of the sentence or not.

When we apply a grammar to an element which content is text, we first tokenize the content by a tokenizer and then it is used as input for the grammar. Additionally, each token receives a unique type. For instance the content of following element:

<s> I love my cat Mary</s>

can be segmented (with usage of **Uptok** tokenizer (see above)) as follows:

Word	*Token category*
'I'	**LATwc**
' '	**SPACE**
'love'	**LATws**
' '	**SPACE**
'my'	**LATws**
' '	**SPACE**
'cat'	**LATws**
' '	**SPACE**
'Mary'	**LATwc**

We can refer to the word itself (1) represented as a string (also including wildcards) enclosed in double quotes. On the other hand, we can refer to the token category (2) that is assigned to the word - token categories are represented by **$*name_of_category.***

Examples:

1)

"love"	could be matched only to the token 'love' in the above example
"#y"	could be matched to the tokens 'my' and 'Mary'
"@"	matches 'I' and spaces

2)

$LATwc	is matched to 'I' and 'Mary'
$LAT#	is matched to 'I', 'love', 'my', 'cat', 'Mary'
$#	is matched to any of token categories including spaces

How to consider the content of an element as input word fora grammar if it is a sequence of elements? For instance, if the above sentence is represented as:

```
<s>
  <Pron>I</Pron>
  <V>love</V>
  <Pron>my</Pron>
  <N>cat</N>
  <N>Mary</N>
</s>
```

If we takethe tags, then the input is the following sequence: <Pron><V><Pron> <N><N>. But if the content is sophisticated as the following:

```
<s>
  <w g="Pron">I</w>
  <w g="V">love</w>
  <w g=" Pron">my</w>
  <w g="N">cat</w>
  <w g="N">Mary</w>
</s>
```

then the sequence of tags <w><w>...<w> which is not acceptable as an input.

In order to solve the problem we substitute each element with a sequence of values. This sequence is determined by a XPath expressions (keys) that are evaluated taking the element node as the context node. The sequence defined by a XPath expression is called ***element value*** in the CLaRK System. Thus, each element in the content of the element is replaced by a sequence of text elements. For the above example a possible element value for tag <w> could be defined by: **attribute::g**. This XPath expression returns the value of the attribute **g** for each element with tag <w>. Therefore a grammar working on the content of the above sentence will receive as an input the sequence <"Pron">,<"V">,<"Pron">,<"N">,<"N">. Besides attributes, by using of XPath expressions one can point to arbitrary nodes in the document, so that the element values can be defined in various forms.

When we refer to an element in a regular expression it must be enclosed in **< >**.

2.5 *Removal operation*

Within the CLaRK System there is a possibility to delete elements from the document. Element selection is done by an XPath expression. Selected elements can be removed with or without their content.

3. Grammars for processing of a text archive

In this section, we represent some problems that appeared during the processing of the text archive and have been solved by cascaded regular grammars.

3.1. Abbreviations

First, we extracted all possible abbreviations into a list of abbreviations [written in several ways in Bulgarian language: "tn." (so on), "s/u"(opposite), "g." (year), "g-n" (Mr.), "NATO"] from our corpus. Work on extraction, classification, and linguistic treatment of abbreviations is done by Osenova and Simov (Osenova and Simov 2002). To apply this list over the texts we have two steps (first - grammar applying and second step - removing some auxiliary tags).

3.1.1 Graphical abbreviations

We have written four different grammars for each type of abbreviation.

1 "s/u" (opposite), "km/h" (kilometers per hour) , "v/u" (on) - abbreviations with slash.
2 "g-n" (Mr.), "g-ja" (Mrs.) - abbreviations with dash.
3 "t.g." (this year) - and abbreviations with full stop and dash - "j.-izt." (Southeast).
4 "g."(year) , "str."(page) - abbreviations with only one full stop

To mark-up abbreviations and acronyms we use standard tag **<abbr>,** which is extended with some attributes:

- type - indicates the type of abbreviation. Possible values are "contr" - for contraction or "acronym" for acronyms.
- expan - Possible values are "word"(abbreviation of a word) or "phrase" (abbreviation of phrase).

As subelements of **<abbr>** we have:

<ph> - phonetics - orthographical representation of the abbreviation.
<expan> - contains the expansion of the abbreviation. When the abbreviations have more than one meaning, we can have more than one "expan" elements.
<aa> - all analyses -all possible morphological characteristics associated with all expansions.
<ta> - true analyses - morphosyntactic decision for the relevant expansion.

We have applied the grammars in the same way that we listed above. The right order of grammar processing reflects on the final result. For example if we process grammar for abbreviations with one full stop - "g."

will be matched, and than the grammar for abbreviations with two full stops - "t.g." will not be recognized, because "g." is already marked up.

Example:

1. The text after the tokanization.

We have four separate tokens:

```
<tok>t</tok><pt>.</pt>
<tok>g</tok><pt>.</pt>
```

2. Result after abbreviation grammar applying:

```
<abbr type="contr" expan="phrase">
<ph>
<tok>t</tok><pt>.</pt>
<tok>g</tok><pt>.</pt>
</ph>
<expan>tazi godina</expan><aa>Noun</aa><ta>Noun</ta>
</abbr>
(This year)
```

3. Deletion of unnecessary tags <tok> and <pt> with XPath expression: *//abbr/ph/tok | //abbr/ph/pt*. We remove only the tags, not their text content and the result is:

```
<abbr type="contr" expan="phrase">
<ph>t.g.<ph><expan>tazi godina</expan><aa>Noun</aa><ta></ta>
</abbr>
```

One problem with abbreviations finishing with full stop is that this full stop could be also end of a sentence. In order to distinguish such full stops, the grammar for the abbreviations adds full stop in the token lists and changes their markup. Attribute **type** with value "abbr" indicates such full stops. For example, if we have in the text abbreviations

```
<w>str</w><pt>.</pt> (page) and
<w>dr</w><pt>.</pt> (other)
```

After running the grammar we will have:

```
<abbr><ph><tok>str</tok><pt>.</pt></ph>
<expan>stranica</expan><aa></aa><ta></ta></abbr><pt type="abbr">.</pt>

<abbr><ph><tok>dr</tok><pt>.</pt></ph>
<expan>drugi</expan><aa></aa><ta></ta></abbr><pt type="abbr">.</pt>
```

The grammar rules for the given examples have empty left and right context

```
RE:     <"str">,<".">
RM:  <abbr><ph>\w</ph><expan>stranica</expan><aa></aa><ta></ta></abbr>
        <pt type="abbr">.</pt>

RE:     <"dr">,<".">
RM:  <abbr><ph>\w</ph><expan>drugi</expan><aa></aa><ta></ta></abbr>
        <pt type="abbr">.</pt>
```

3.1.2. Acronyms

The grammar for acronyms shares the same principle as the grammars for abbreviaions. We have lists with acronyms and their meaning and grammar features. The difference is in attribute of "abbr" tags - we put type = "acronym".

Example:

<abbr type="acronym" expan="phrase"><ph>AOK</ph><expan>Armia za osvobojdenie na Kosovo</expan><aa></aa><ta></ta></abbr>

3.2. Quotation recognition (overlapping hierarchies)

First, we will consider the entirely principles and decisions for quotations. If we analyze all cases of appearance of the quotation, we will see examples like this:

"The new model of "Ford" was presented yesterday."

The grammar for quotation (**quote_gramm**), which marks up the sequence of words between two quotation marks, produces a result like the following:

<q>"The new model of "</q>Ford<q>" was presented yesterday."</q>

Which is not the right segmentation.

For that reason we apply grammar (**quote_entity**) which marks up the quotations with content of one or two words. In that case return mark-up tag <q> - has attribute type="entity". After this operation, we apply the grammar for quotations(**quote_gramm**). The result is:

<q>The new model of ***<q type="entity">****Ford****</q>*** *was presented yesterday.</q>*

Usually features matched by this grammar are name entities as well. There is an idea first grammar for named entities to be processed and then grammar quote_entity over named entities enclosed in quotations. So if we apply quote_gramm on the following example:

<s> Nikola Nikolov se izvini na jurnalistite ot "24 chasa" i "Standart".</s>

(Nikola Nikolov apologized to the journalists from "24 chasa" and "Standart".)

the produced result will not be the correct one

<s> Nikola Nikolov se izvini na jurnalistite ot "24 chasa
<q type="entity"> i</q>Standart".</s>

if we apply the grammar after the grammar for named entity, the produced result will be correct

<s> Nikola Nikolow se izwini na jurnalistite ot
<q type="entity"> <nameE>24 chasa</nameE></q> i
<q type="entity"><nameE> Standart</nameE></q>.
</s>

The overlapping hierarchies are a well known problem for XML based corpus and are one of the discussed problems in XCES. This problem is closely connected with the sentence and quotation boundaries. In this section we show how to implement the XCES Recommendations with the help of the CLaRK System.

The basic level of data model provided by XML is an ordered-labeled tree. And if the document contains the following markup:

* According to the visiting leader, the economy of the country is <q>"better than ever. It is in fact in very good shape."</q> *

a likely segmentation into sentences would be

> *<s>According to the visiting leader, the economy of the country is <q>"better than ever.</s><s>It is in fact in very good shape.</s>"</q> *

However, this is non well-formed XML since the **<s>** and **<q>** tags are not properly nested. There is a basic approach to the problem of overlapping hierarchies offered by XCES:

> *Make one of the hierarchies primary and the other(s) secondary, and break any elements of the secondary hierarcie(s) at those points where they overlap a boundary of the primary hierarchy.*

We decided to use CES recommendations for -<s> -<q> hierarchy.

There still is a question: How to implement this within the CLaRK system. We wrote 3 cascaded grammars, which are processed over result of **quote_gramm**. We process the first one and than we process second and third grammars as many times as possible,but with changed ids. They must be processed in the given order for correct result. Let us consider the following examples:

> *Example 1) taken from XCES recommendations*
> * According to the visiting leader, the economy of the country is*
> *<q>"better than ever. It is in fact in very good shape."</q>*
> * *
>
> *Example 2) taken from XCES recommendations*
> * *
> *<q>"I know precisely what you are feeling. I know all about your contempt, your hatred, your disgust. But don't worry, I am on your side!"</q>*
> *And then the flash of intelligence was gone...*
> * *
>
> *Example 3) translation from Bulgarian*
> * My favorite ice-cream is <q>"Delta"</q>. *

Quote_ref1 grammar

Element value: pt=text() and w=ph/text()
LC:
RE: <"^"">,<"-">?,(<"#"*>|<#>)*,<"."|"?"|"!">
RC: <"-">?,<$CYRwc>
RM: <q id="q1" type="part" next="q2">\w</q>

The regular expression imposes the following constraints over the context:

1 Starts with quotation mark, that can be followed by a dash: <"^"">,<"-">?
2 Contains any number of Cyrillin tokens and punctuation <"#"*> or other elements <#>: (<"#"*>|<#>)*
3 Ends with full stop, question mark or exclamation mark. <"."|"?"|"!">

As right context there is Cyrillin word with first capital letter: <$CYRwc>

This grammar matches everything from the start of the quotation element - first quotation mark - to first punctuation - question mark, full stop and so on, which indicates the sentence boundary (see grammar for sentence boundary below). Recognized sequence can be a sentence (example2) or a part of sentence (example1). Grammar fails if there are not sentence boundaries in the processed quotation (example3). Quote_ref2 and Quote_ref3 will fail as well.

After applying Quote_ref1 we will have:

1) According to the visiting leader, the economy of the country is
<q>
<q id="q1" type="part" next="q2">"better than ever.</q>
It is in fact in very good shape."
</q>
* *

*2) *
<q>
<q id=q1 type=part next=q2>"I know precisely what you are feeling.</q>
I know all about your contempt, your hatred, your disgust. But don't worry, I am on your side!"
</q>And then the flash of intelligence was gone...
* *

Quote_ref2 grammar

Element value: pt=text() w=ph/text()
LC: <q id="q**1**" type="part" next="q**2**">
RE: <"-">?,(<"#"*>|<#>)*,<"."|"?"|"!">
RC: <"-">?,<$CYRwc>
RM: <q id="q**2**" type="part" prev="q**1**" next="q**3**">\w</q>

The regular expression imposes the following constraints over the context:

1 Can starts with a dash <"-">? - if direct speech
2 Contains any number of Cyrillin tokens and punctuation <"#"*> or other elements <#>: (<"#"*>|<#>)*
3 Ends with full stop, question mark or exclamation mark. <"."|"?"|"!">

As right context there is Cyrillin word with first capital letter: <$CYRwc>

This grammar recognizes the first sentence in the quotation tag that is not marked up. If the sentence has as right context (RC) quotation mark grammar fails (example 1), such a sentence will be matched by grammar Quote_ref3. RC should be beginning of a sentence (word with capital first letter, a number or if it is direct speech dash followed by number or word with capital first letter)(example 2).
The result will be:

* *
<q>
<q id=q1 type=part next=q2>"I know precisely what you are feeling.</q>
<q id="q2" type="part" prev="q1" next="q3">I know all about your contempt, your hatred, your disgust.</q>But don't worry, I am on your side!"
</q> And then the flash of intelligence was gone...
* *

Quote_ref3 grammar

Element value:

pt -> text()[not((self::*=".") and (../following-sibling::*[1][self::w[ph/text(4,n,($CYRwc))]]))]
w -> ph/text()

LC: <q id="q**1**" type="part" next="q**2**">

RE: <"-">?,<$CYRwc>,(<"#">|<#>)*,<"^"">
RC: $$
RM: <q id="q**2**" type="part" prev="q**1**" >\w</q>

The regular expression imposes the following constraints over the context:

1 It can start with a dash <"-">? - if direct speech
2 Contains any number of Cyrillic tokens and punctuation <"#"*> or other elements <#>: (<"#"*>|<#>)*
3 Ends with quotation mark <"^"">

This grammar matches everything to the end of the quotation element (end of every element node is marked with **$$)**. It will be successful, only if Quote_ref1 succeeds and Quote_ref2 fails(example 1):

```
<p>According to the visiting leader, the economy of the country is
   <q>
     <q id="q1" type="part" next="q2" >"better than ever.</q>
     <q id="q2" type="part" prev="q1" >It is in fact in very good shape."</q>
   </q>
</p>
```

Quote_ref2 and Quote_ref3 grammars are processed as many times as necessary with changed references (attributes *id next* and *prev*). Let **i** be a variable, then the LC expression and the return mark-up for both grammars will be:

LC: <q id="**i**-1" type="part" next="**i**">
RM: <q id="**i**" type="part" prev="**i-1**" next="**i+1**">\w</q> (Quote_ref2)
RC: <q id="**i**" type="part" prev="**i-1**" >\w</q> (Quote_ref3)
For **i**=2,3...n

When we have finished with all grammar applying we run removal operation. We delete <q> parent tag and all quotation marks. The XPath expression for the removal operation is: //q[child::q]

On next turn over example3 Quote_ref2 fails and Quote_ref3 succeeds:

```
<p>
   <q>
     <q id=q1 type=part next=q2>I know precisely what you are feeling.</q>
     <q id=q2 type=part prev=q1 next=q3>I know all about your contempt, your hatred, your
     disgust.</q>
     <q id=q3 type=part prev=q2>But don't worry, I am on your side!</q>
   </q>And then the flash of intelligence was gone...
</p>
```

Note that: LC and RM for both grammars must have the same attributes, otherwise quotation element will not be devided into sentences correctly.

3.3 Sentence boundary

Here we present a simple grammar for sentence boundary delimitation. The tokenizer in this case is **Uptok -** described in section 2.2. It recognises a sentence as a sequence of tokens – words, numbers and punctuation (we refer to the content of **<w>**, **<tok>** and **<pt>** elements with **Element Values**). As sentence first element we have a capitalized word or tok element, which is a number. The sentence ends with punctuation – full stop, question mark or exclamation mark. The sentence can be followed by another sentence or it can be the

last one in a paragraph – these features are matched by the right context of grammar. Sometimes the sentence can be a direct speech – in this case it starts with a dash.

Element Values

The input is a sequence of elements **<w>, <tok>** and **<pt>** so we refer to the information, that we are interested of, by:

w -> ph/text()
tok -> text()
pt -> text()

RE: <"-">?,<$NUMBER+>?, <$CYRwc>,(<($#|<#>)+>|<#>)*,<"."|"?"|"!">+
RC: $$|<$CYRwc>|<$NUMBER+>|<"-">

It is necessary to determine the restriction by right context

This grammar recognizes the sequence of words and punctations, and marks them up. The new tag <s> (sentence) wraps the whole sequence of nodes.

The grammar for a sentence boundary delimitation marks-up the sequence of some elements that are not sentences. We first discuss some of the problems for which we have solutions, and then the unsolved yet cases are pointed out.

Dates

The sentence boundary delimitation grammar matches wrongly some parts of dates. For example if we have

<tok>05</tok><pt>.</pt><tok>08</tok><pt>.</pt><tok>2002</tok>

It matches 05. and 08. as sentences , but they are not.

<s><tok>05<tok><pt>.</pt><s>
<s><tok>08</tok><pt>.</pt></s>
<tok> 2002<tok>

This problem was easy to solve. We have implemented a grammar that marks-up dates (Simov, Kouylekov and Simov 2002) and it was applied before the grammar for sentences.

Abbreviations

As it was mentioned above in section 3.1.1, we solve one part of the well known problem of sentence boundary delimitation, namely the case when some abbreviations end with a full stop which also marks the end of a sentence. The grammar for sentence boundary delimitation will consider also the full stops marked with attribute value: "abbr" (<pt type="abbr">.</pt>).

For example:

<s>Roden e prez 1928<abbr>g.</abbr><pt type="abbr">.</pt></s><s>Toi e vyzrasten.</s>
(He was born in 1928 year. He is an old man.)

If the abbreviation grammar deletes the full stop of abbreviations then the two sentences will be matched as one.

If the grammar recognizes some of these full stops as the end of sentences, it will replace their markup with the markup for the end of the sentence. After the application of the grammar for sentence delimitation, we run a removal operation to delete all full stops that are part of an abbreviation, but not end of a sentence.

Example:

<s>Kompaniata struvashe 15

<abbr>mln.</abbr><pt type="abbr">.</pt>
<abbr>lv.</abbr><pt type="abbr">.</pt>
</s>

(The price of the company was 15 million leva.)

After removal operation:
// pt[text()="."][following-sibling::*[1][ph/text(1,n,(CYRws))]]

Remove all full stops: pt[text()="."] that are before a Cyrillin word with small letters [following-sibling::*[1][ph/text(1,n,(CYRws))]]

<s>Kompaniata struvashe 15
<abbr>mln.</abbr>
<abbr>lv.</abbr><pt>.</pt>
</s>

Abbreviated names

Very often in the texts first names appear abbreviated - for example J. Smith. Here we have a word with first capital letter and a full stop followed by a word with first capital letter – so that is sentence or end of sentence - <s>J.</s>Smith. To solve this problem we wrote a grammar that matches such names as a part of the grammar (module) for Named-Entity recognition.

Unsolved problem

One unsolved problem is when one or more sentences are embedded in another sentence.

"I love you!" – she said.

Where must be the mark-up? Is it allowed to have sentence in sentence?

<s><q><s>I love you!</s></q> -she said.</s>

What if there are more than one sentences in quotation, if there are paragraphs? In which cases if so?

A possible decision is when there are more than one sentence in quotations – even a paragraph - they to be enclosed in <cit> element with attribute type = "part of sentence". The sentence ends where <cit> element ends.

Another possibility is to allow the sentence to have subsentences (or subparagraphs). If all sentences in the document are with indexes, each subsentence will have reference to sentence where it belongs to. It will be allowed to have structure like <s id ="**i**"> <s type = "subsent" parent = "**i**"></s></s>

All grammar operations described so far are applied in the following order:

- Dates
- Abbreviations (all four types are processed)
- Acronyms
- Abbreviated Names
- Quotations
- Sentence boundary

Removal operations must be applied after each grammar for the right processing of the following grammars.

4. Writing disambiguation rules

As it was mentioned above for text disambiguation we used value constraints, especially Some Children Constraint (a tool of CLaRK System). It is used for a value restriction when the operation *inserting a child in an element* is performed. Constraint in general consists of two parts: A target and a source section.

In target section, the elements to which the constraint will be applied are described (target nodes). First, the elements are selected by their tag name and then further restrictions by an XPath expression are imposed - a context dependence can be expressed (for example the node must not have children or certain sibling).

In the restriction section, the possible values for the content of target nodes (selected by the previous section) are defined. The possible values are XML mark-up or tokens depending on the type of the constraint. Values can be selected by an XPath expression or by typing the options explicitly as an XML markup.

It is important to mention that when the context determines only one possible value for some element, it is added automatically to the content of the target element. In our examples target elements are <ta> elements.

Let us discuss several disambiguation rules with different degrees of complexity:

"che" and "a" after comma

For instance, one simple rule is for "che" and "a" - they can be particle or conjunction. In position after comma they are always conjunctions.

Before applying the constraint:

```
<pt>,</pt>
<w><ph>che</ph><aa>Conjunction;Particle<aa><ta></ta></w>
```

After applying:

```
<pt>,</pt>
<w><ph>che</ph><aa>Conjunction;Particle<aa><ta>Conjunction</ta></w>
```

We have restricted the cases only for "che" after a comma. Other appearances of the word "che" is still ambiguous.

Target element is given from **Selector** section, restrictions for the target element are described in **Context** section and value for the target element is given in **Value** (restriction section), as it was mentioned, it can be selected by XPath or can be given as an XML mark-up directly:

Selector: ta

Context: not(child::*) and preceding-sibling::ph[text()="che"]

and ../preceding-sibling::*[1][self::pt[text()=","]]

Value: Conjunction

The context description imposes the following constraints over it:

1 The <ta> element is empty: not(child::*)
2 The word is "che": preceding-sibling::ph[text()="che"]
3 The previous token in the sentence is a comma: ../preceding-sibling::*[1] [self::pt[text()=","]]

Agreement-based constraints:

Using XPath language for context describtion allows writing more complecated rules like the following for dealing with agreement in NP.

The agreement phenomenon can be seen in chunks like Noun phrases where we have adjective in first position and it is followed by noun. The adjective agrees with the noun. If the noun is not ambiguous we can easily choose the right grammatical form of the adjective.

Example:

"Bylgarski ezik" (Bulgarian language)
"Bylgarski detza" (Bulgarian children)

<ph>Bylgarski<ph><aa>"Adjective, masculine, singular" or "Adjective, plural"</aa>
<ph>ezik</ph><aa>"Noun, masculine, singular "</aa>

<ph>Bylgarski<ph><aa>"Adj, masculine, singular" or "Adj., plural"</aa>
<ph>detza</ph><aa>"Noun plural"</aa>

We wrote constraints that automatically disambiguate these cases.

Selector: ta
Context: not(child::*) and ../aa[text()=Adjective, masculine, singular ;Adjective, plural]
and ../following-sibling::*[1][self::w/aa[text()="Noun, masculine, singular "]]
Value: Adjective, masculine, singular

The context description imposes the following constraints over it:

1 The <ta> element is empty: not(child::*)
2 The <aa> element of selected word contains two values - Adjective, masculine, singular and Adjective, plural:
../aa[text()=Adjective, masculine, singular ;Adjective, plural]
1 The next token in the sentence is a noun masculine, singular:
../following-sibling::*[1][self::w/aa[text()="Noun, masculine, singular "]]

Restrictions are – not to have children (ta element), its parent – element **w** to be followed by a word element which is Noun, masculine, singular. This is given by XPath expression:

not(child::*) and ../aa[text()=Adjective, masculine, singular ;Adjective, plural]
and ../following-sibling::*[1][self::w/aa[text()="Noun, masculine, singular "]]

In the restriction section we will have XML markup "Adjective, masculine, singular "

During the manual disambiguation of word-forms we have used more than 50 constraints.

In a document 12% are unrecognized words – abbreviations, names and other words that are not in morphology. From the rest 65.95 % are unambiguous and 34.05% are ambiguous. 15.80 % from the ambiguouty can be resolved by constraints and grammars.

5. Using Temporary Mark-up for disambiguation

We already demonstrated the use of grammars, additional mark-up and removal operation in order to solve some problems. Here we demonstrate the use of grammars, constraints and removal operations for automatic disambiguation.

There are some sequences of words that can be automatically disambiguated if they form appropriate linguistic structure. Sometimes such sequences of words cannot be constituents, and their eventual grouping

is not part of the analysis of the sentences, then we mark them only temporarily. In this case we proceed in the following way. First we write a grammar that groups together such sequence of words and marks them up in an appropriate way. Then we run a group of constraints that disambiguates the words within such a sequence of words. After the constraints, we run a removal operation to delete the group markup.

An example of such a case is the so-called 'da'-construction in Bulgarian. It includes the conjunction 'da', a number of clitics and a verb form in present tense. 'Da' is ambiguous between a conjunction and a particle, but within such groups it is only a conjunction. Most of the clitics are ambiguous between personal pronouns and possessive pronouns, but in this context they can be personal pronouns only. The verb can only be in the present tense. Using such a mechanism, we succeeded to reduce the number of ambiguities with more than 10%.

<dacomplex>
<w><ph>da</ph><aa>Conjunction or Particle</aa></w>
<w><ph>mi</ph><aa>Pronoun (personal or possessive) ;Verb</aa></w>
<w><ph>ja</ph><aa>Interjection; Particle or Personal pronoun</aa></w>
<w><ph>dade</ph>Verb – present or past tense</w>
</dacomplex>
(To give it to me)

Linguistic decisions for "da – complex":

We used the similar grammar for "shte"[particle for future tense]. We applied the same solutions for clitics and verb as in "da-complex".

<shte-complex>
<w><ph>shte</ph><aa>Particle</aa></w>
<w><ph>mi</ph><aa>Pronoun (personal or possessive) or Verb</aa></w>
<w><ph>ja</ph><aa>Interjection; Particle or Personal pronoun</aa></w>
<w><ph>dade</ph>Verb – present or past tense</w>
</shte-complex>
(He/She will give it to me)

We annotated grammatical characteristic with usage of constraints and after this action, we ran a removal operation to delete group markup <dacomplex> and <shte–complex>.

We remove this tags because they do not always mark-up real grammatical constituents. That is why we call them temporal mark-up. In some cases the sequence of elements in da or shte -complex are equal to a "verb complex" and provides a true sentence analysis, but in other cases like:

Toj trjabva da mi e dal knigata.

(He must have given me the book)

The da-complex grammar will mark up only one part of verb complex:

<s>Toj trjabva <dacomplex>da mi e</dacomplex> dal knigata.</s>

The syntactic treatment of these verb complecsses is done by other grammars complecsses is done by other grammars within the project. The result is something like:

<s>Toj <Vm>trjavba<Vm><Vcomplex>da mi e dal</Vcomplex> knigata.</s>

Another example of problematic case is when there is a coordination between two verbs

<s>Kaji i da proveri i sybshti kakvi sa usloviqta.</s>

(Tell her to check and let us know what the conditions are.)

Our da-complex grammar will mark up

<s>Kaji i <da complex>da proveri<da complex> i sybshti kakvi sa usloviqta.</s>

That is only a part of the coordinate phrase *<dacomplex>da proveri i syobshti</dacomplex>*.

This is why we use our grammar only for disambiguation purposes and afterwards we delete the changes made by it. Using temporarily mark-up is very powerful tool.

6. Conclusion

The compilation of a text archive is connected with the appropriate recognition of all meaningful elements, i.e. not only common word tokens, but also named entities (person names, organization names, dates, currency expressions), abbreviations, punctuation. The efficient handling of these preprocessing steps proved out to be of great importance for the consistency of next parsing stages and, at the same time, for data mining. Thus it facilitates earlier direct use of the text archive.

In recent years the existence of syntactically interpreted corpora has become a crucial prerequisite for the development of the linguistic theories and the variety of NLP application tasks. In this paper we presented some tools and an algorithm for resolving ambiguity and minimization of human work.

Algorithm:

- Applying cascade grammars over selected documents – this includes tags or data adding
- Applying constraints (in grammar result) if this operation is necessary – one sufficient action to minimize the ambiguity
- Remove useless data

The examples that we have given demonstrate how the usage of different grammars and constraints improve the manual disambiguation of texts. The Tools of CLaRK system help us to create an XML-based corpus of morphologically annotated texts.

References

Abney, S., 1991. Parsing By Chunks. In: Robert Berwick, Steven Abney and Carol Tenny (eds.), Principle-Based Parsing. Kluwer Academic Publishers, Dordrecht. The Netherlands.

Abney, S., 1996. Partial Parsing via Finite-State Cascades. In: Proceedings of the ESSLLI'96 Robust Parsing Workshop. Prague, Czech Republic.

Corpus Encoding Standard. 2001. XCES: Corpus Encoding Standard for XML. Vassar College, New York, USA. http://www.cs.vassar.edu/XCES

Ide N., Romary, L. (2001). Standards for Language Resources. IRCS Workshop on Linguistic Datebases. Philadelphia. USA.

Ide N., Romary, L. (2001). A Common Framework for Syntactic Annotation. Proceedings of ACL'2001, Toulouse. France.

Mikheev, A. 2000. Tagging sentence boundaries. In Proceedings of the 1st Meeting of the North American Chapter of the Computational Linguistics (NAACL'2000).

Popov, D., Simov, K. and Vidinska, S. A Dictionary of Writing, Pronunciation and Punctuation of Bulgarian Language, Atlantis SD, Sofia, Bulgaria.

Simov Kiril, Peev Zdravko, Kouylekov Milen, Simov Alexander, Dimitrov Marin, Kiryakov Atanas. CLaRK an XMLbased System for Corpora Development. In: Proc. of the Corpus Linguistics 2001 Conference. Lancaster. UK.

Simov K., Osenova P., Slavcheva M., Kolkovska S., Balabanova E., Doikoff D., Ivanova K., Simov A., Kouylekov M. 2002: Building a Linguistically Interpreted Corpus of Bulgarian: the BulTreeBank. In Proceedings from the LREC 2002, Canary Islands. Spane

Simov K., Kouylekov M., Simov A 2002. Cascaded Regular Grammars over XML Documents. In: Proc. of the 2nd Workshop on NLP and XML (NLPXML-2002), Taipei, Taiwan.

Osenova P. and Simov K. 2002. Learning a token classification from a large corpus. (A case study in abbreviations). In: Proc. of the ESSLLI Workshop on Machine Learning Approaches in Computational Linguistics, Trento, Italy.

Text Encoding Initiative. 1997. Guidelines for Electronic Text Encoding and Interchange. Sperberg-McQueen C.M., Burnard L (eds).

XPath. 1999. XML Path Language (XPath) version 1.0. W3C Recommendation. http://www.w3.org/TR/xpath