Krzysztof Dorosz* **, Anna Szczerbińska***

ENHANCING REGULAR EXPRESSIONS FOR POLISH TEXT PROCESSING

The paper presents proposition of regular expressions engine based on the modified Thompson's algorithm dedicated to the Polish language processing. The Polish inflectional dictionary has been used for enhancing regular expressions engine and syntax. Instead of using characters as a basic element of regular expressions patterns (as it takes place in BRE or ERE standards) presented tool gives possibility of using words from a natural language or labels describing words grammar properties in regex syntax.

Keywords: *regular expressions, regex, natural language, the Polish language processing, CLP library*

MECHANIZM ROZSZERZONYCH WYRAŻEŃ REGULARNYCH DO PRZETWARZANIA TEKSTÓW JĘZYKA POLSKIEGO

W artykule zaprezentowano propozycje mechanizmu wyrażeń regularnych w oparciu o zmodyfikowany algorytm Thompsona dostosowany do przetwarzania tekstów w języku polskim. Prezentowane wyrażenia regularne wykorzystują słownik fleksyjny języka polskiego i pozwalają na budowę wzorców, w których elementami podstawowymi są wyrazy języka polskiego lub etykiety gramatyczne, a nie znaki (jak to ma miejsce w klasycznych wyrażeniach regularnych standardu BRE czy ERE).

Sowa kluczowe: *wyrażenia regularne, język naturalny, język polski, biblioteka CLP*

1. Introduction

Regular expressions (also called regex) after years became a very important tool for text processing. They provide concise yet still flexible way for describing patterns of characters. Patterns, written in a formal language, generate a set of strings that match given criteria. In general each regular expression represents a finite state machine (FSM). Evaluating an FSM on a string as an input gives an answer if the string

* Computational Linguistics Department, Jagiellonian University, Krakow, Poland
** PhD Student, Institute of Computer Science, AGH University of Science and Technology, Krakow, Poland, dorosz@agh.edu.pl
*** Msc. student, Institute of Computer Science, AGH University of Science and Technology, Krakow, Poland, szczerbi@student.agh.edu.pl

matches the pattern. There are currently many standards of regular expressions syntax, most popular are the POSIX Basic Regular Expressions (BRE), the POSIX Extended Regular Expressions (ERE) and a whole set of extensions to ERE introduced in Perl regex. Regular expressions can be used for many purposes like: validating a string format, searching patterns in text, replacing patterns with strings in text or even syntax highlighting. A simple regular expressions that validates if a string is a well formated zip code (Polish format) can be written as `/^[0-9][0-9]-[0-9][0-9][0-9]$/` or in a concise way (using the Perl notation) as `/^\d{2}-\d{3}$/` where both mean exactly the same – two digits followed by a dash, followed by three digits. Things get a little bit complicated when it comes to processing a natural language. Texts are build from sentences, which are basically sequences of words, punctuation and some other symbols. The first very basic need of processing natural language texts is to tokenise them in order to work with tokens (that can be further recognised as words) – instead of characters. This is a non trivial issue, because regex engines enable defining patterns only from characters rather than high level entities. This brings a need of defining another characters sets that will be helpful for shortening syntax. Some help can be found in the POSIX character classes standard or in the Perl standard. They introduce following character classes:

- [: *word* :] (POSIX), $\backslash w$ (Perl) – alphanumeric characters plus _ (underscore), `[A-Za-z0-9_]`,
- [: *space* :] (POSIX), $\backslash s$ (Perl) – whitespace characters, `[ \t\r\n\v\f]`.

There are two different ways for text tokenising:

- splitting text using any whitespace characters $/\backslash s/$ as a delimiter – will produce a list of tokens that needs to be cleaned from other elements like punctuation,
- matching any string built from word class characters $/\backslash w+/$ – will produce a list of words, but will omit other elements like numbers, punctuation, etc.

The problem of text tokenising with regular expressions is well described in other papers ([3]) and will not be covered here, because of limitations of this article. A whole bunch of issues connected with national characters, accents, the UTF-8 coding that that have been solved mainly by choosing arbitrary code page also will not be discussed in this article.

After text tokenising one can begin to build more complex regular expressions, that involve dependencies between two tokens. One can find any word after the word *ładny* (Eng. *nice*). Using regular expression `/ładny\s+\w+/` pattern returns few pairs like *ładny dom* (Eng. *nice house*) if evaluated with an enough large text corpus. Results will be limited due to important reasons:

- Polish is highly inflectional – that means that the word *ładny* (Eng. *nice*) can appear in many different forms like *ładna*, *ładnie* – one could extend the given regex with some alternative of forms of the word *ładny*, but this is inconvenient, especially considering wide availability of a digital inflectional language dictionary,

- there is no possibility in this approach to give extra conditions on search criteria: looking for a noun, or looking for a masculine noun, or nominative noun; this can be done only with filtering results with an additional method that will make lookups to an inflectional dictionary evaluating each word for the given criteria,
- given regex will not find a pair of words if something else than whitespace stands between them (e.g. punctuation); this drawback can be easily corrected by extending the $\backslash s$ class with some punctuation marks, but then the following regex would not be sensitive to the end of a sentence,
- it is hard to build a good representation of the end of sentence marker (for a given language) that would be convenient to use in regex. Punctuations like periods, exclamation marks, etc. do not necessarily end a sentence, as they can be a part of abbreviations or quotations. Working with natural language texts is very limited without such marker, because there is large scope of language processing that requires working with sentences rather than whole texts.

Because of above limitations of regex in application to natural language processing we propose a dedicated engine of regular expressions based on the following assumptions:

- use of markers defining words and grammar labels instead of characters classes,
- availability of classic regex syntax and operators, including greedy and non-greedy operators,
- possibility of querying words types, forms (inflection) and custom strings,
- sentence by sentence text processing, rather than line by line or globally,
- redefinition of the `^` and `$` symbols with the begining and the end of sentence markers.

To achieve these goals many additional tools are needed. Given text has to be tokenised, divided into sentences and POS tagged to be useful for the FSM based engine. Because tokenising and splitting into sentences is well described in literature in the following section we discuss the inflectional Polish language dictionary that has been used. Our regular expression syntax make use of some CLP data representation conventions (e.g. word identification numbers).

2. The CLP library

The CLP library described in [1] is the Polish inflectional dictionary written in C in the Computational Linguistics Group of Institute of Computer Science AGH-UST. The library consists of two layers: inflectional and morphological. Current size of the library amounts to over 150 000 base words, what basically covers all common words in the Polish. The library allows to identify a word in the Polish by its any form, return a forms vector for any stored word, determine a grammatical label which describes the pattern of a word inflection and also contains information determining a part of speech.

The CLP library is mainly organised by following data structures:

- a unique word number (CLP ID) – identify a word with a given inflection (if a word have two different inflection patterns it has also two different CLP ID's in the library),
- a word label – describes some of grammatical properties, what basically determine a word inflection pattern and a part of speach tagging,
- a vector of forms – a list of word inflection forms.

Each word is assigned a CLP ID number within the library. Multiple CLP IDs are returned if the queried string is ambiguous. The first letter of a label has following meaning:

- A – noun,
- B – verb,
- C – adjective,
- D – numeral,
- E – pronoun,
- F – adverb,
- G – uninflected.

Following data is returned for word *zamek* (Eng. *castle*):

```
?> zamek

ID: 286975040
Forma podstawowa: zamek
Formy: zamek, zamku, zamkowi, zamkiem, zamki, zamków, zamkom, zamkami,
zamkach
Etykieta: ACABA
Opis etykiety: rzeczownik / męski nieżyw. / M.Lp.-0 / M.Lm.-i / D.Lp.-u
Wektor odmiany: [1, 4]

ID: 286975056
Forma podstawowa: zamek
Formy: zamek, zamka, zamkowi, zamkiem, zamku, zamki, zamków, zamkom, zamkar
zamkach
Etykieta: ACABBA
Opis etykiety: rzeczownik / męski nieżyw. / M.Lp.-0 / M.Lm.-i / D.Lp.-a /
D.Lm.-ów
Wektor odmiany: [1, 4]
```

As can be seen the word *zamek* is ambiguous in the Polish and has two different patterns of inflection. The patterns can be distinguished by second position (genitive singular) of the inflection vector (*zamku* and *zamka*). A word label is a hierarchical structure describing its grammatical properties.

Both returned words having labels ACABA and ACABBA are nouns (due to the first letter *A*) with some differences in inflection. These differences are described by letters in labels that are not the same. The first label ACABA ends with ABA

and the second one ACABBA ends with ABBA. Using the CLP ID 286975040 one can access to any form that refers to the given word: e.g. *zamek, zamku, zamkowi, zamkiem, zamki, zamków, zamkom, zamkami, zamkach*. The CLP returns also an inflection vector that refers to an arbitrary described in [1] vector of case and number different for every part of speech. Considering the first and the fourth position of the inflection vector of the form *zamek* this form is nominative or accusative case singular number.

3. Architecture

The architecture of the CLP regular expressions engine is illustrated in Figure 1. The engine consists of several modules:

- sentence splitter – splits text into sentences; the principles of its work are described in Section 4.6,
- CLP expressions parser – parses CLP regular expression in the form of a string object into a list of tokens to be used by the automaton builder; the grammar of a CLP regular expression is described in Section 4.2,
- automaton builder – constructs a nondeterministic finite automaton for a CLP regular expression (using an altered version of Thompson's algorithm) and minimizes the automaton by calculating ϵ-closures of its states; no conversion of the nondeterministic finite automaton to a deterministic finite automaton is performed,
- CLP matcher – checks if tokens in the text match CLP regex symbols (such as CLP labels, CLP IDs or period symbols); the matcher uses the PLP[1] wrapper of the CLP library for Python,
- regex matcher – matches the CLP regular expression to the given text, verifying if the constructed automaton accepts given sentences or parts of the sentences.

As an input, the engine takes the CLP regular expression pattern (in the form of a string object) and the file containing the Polish text to be matched with the pattern. As an output it produces a list of matches for the pattern.

4. Implementation

4.1. Overview of the implementation

The CLP regular expression engine is based on the extended definition of nondeterministic finite automaton (NFA) [5]. The modifications introduced to the classic version of the NFAs are:

- non-greedy ϵ-transitions – needed to introduce non-greedy operators (`??`, `+?`, `*?`); a non-greedy transition should be used as few times as possible in order to match a pattern,

[1] PLP – Python Language Processing; a Python wrapper for the CLP library.

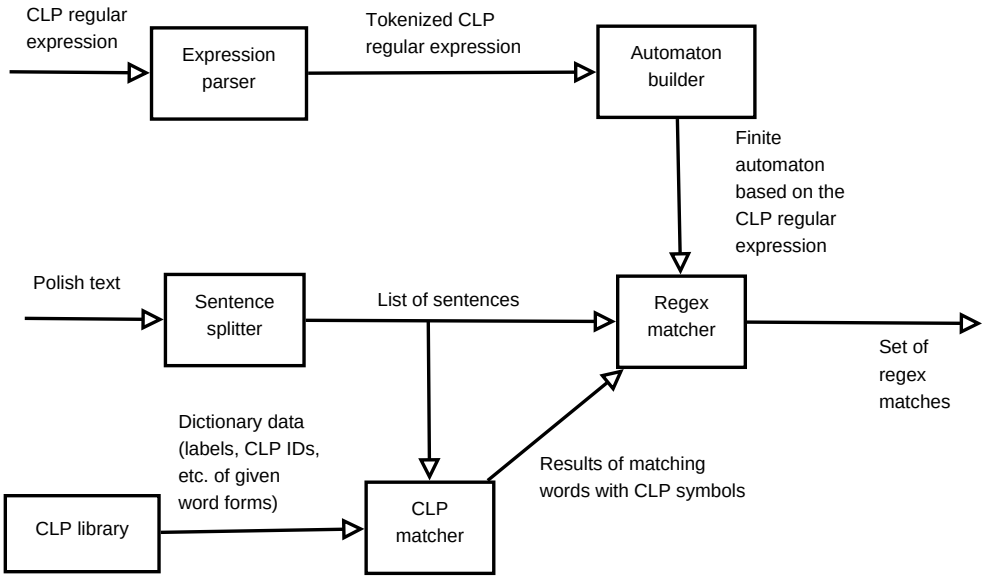

Fig. 1. Architecture of the CLP regular expressions engine

- transitions labeled by ambiguous symbols, such as CLP IDs (which may match more than one form of a word, e.g. `\1058829` matches *pilot, pilota, pilotowi* – Eng. *pilot*) or period symbol (which matches any word); for further explanation of possible ambiguities see Section 4.4.

The engine first constructs a nondeterministic finite automaton according to the modified version of Thompson's algorithm (described in detail in Section 4.3.1). Transitions in the constructed automaton may be labeled with CLP symbols: CLP labels, CLP IDs, words in given forms (incl. numbers), or a period symbol. Some of the ϵ-transitions are marked as non-greedy (it occurs while processing non-greedy operators). After the automaton is built, ϵ-closures are calculated for each state (non-greedy ϵ-transitions not being considered as ordinary ϵ-transitions are not used in this process) and the automaton is minimized according to the algorithm described in Section 4.3.2.

The constructed automaton is subsequently processed by the regex matcher module, which verifies whether the automaton accepts a sequence of tokens. The regex matcher controls the use of non-greedy ϵ-transitions and searches for the longest match available for the pattern.

4.2. CLP regular expressions grammar

The CLP regular expressions grammar is similar to the classic regular expression grammar [6]. The following symbols are available in CLP regex patterns:

- label – a minimum one-character long CLP label prefix, preceded by a backslash; e.g. `\B` stands for a verb, and `\AA` for a noun of a masculine gender (for a detailed description of label meaning see [1]),
- CLP ID – ID of a word in the CLP library, preceded by a backslash; e.g. `\1058829` for the word *pilot* (Eng. *pilot*),
- string – a string built using the `[a-z0-9]` character class; possibly a word in Polish in a given form, e.g. `kotek` (Eng. *kitten*), `pojechała` (Eng. *went*), `2008`, `Janka` (Eng. *Janek's*),
- unary operators: `+, *, ?, {n,m}, ( )` – to be interpreted as in classic regular expressions grammar,
- unary operator: `<n,m,...>` – may only be applied to CLP labels. If the label stands for a noun, pronoun or adjective, the operator specifies the word's grammatical case, gender and number (`n,m,...` being indexes of the acceptable forms); if the label stands for a verb, the operator specifies the tense, gender and number; takes any number of operands (`n,m,...` being indexes of the acceptable forms). Indexes of the chosen forms must remain in accordance with the indexes returned by the `clp_vec()` function from the CLP library,
- binary operator: | – to be interpreted as in classic regular expressions grammar,
- non-greedy operators: `+?, *?, ??` – to be interpreted as in classic regular expressions grammar,
- period (`.`) – symbol standing for a single word or number,
- `$` – the end of a sentence symbol,
- `^` – the beginning of a sentence symbol.

4.3. Automaton construction

4.3.1. Modified Thompson's algorithm

The classic version of Thompson's algorithm is described in [4]. The modified algorithm used for constructing NFAs for CLP regular expressions is based on the following rules:

1. For ϵ-transition construct an automaton as shown in Fig. 2.

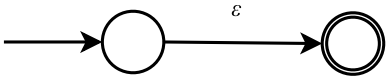

Fig. 2. Automaton constructed for ϵ-transition

2. For a word W (e.g. `pies` – Eng. *dog*, `wyszedł` – Eng. *went out*, `dziewczynkom` – Eng. *girls*, `Kowalskiego` – Eng. *Kowalski's*) construct an automaton as shown in Fig. 3.
3. For I – a CLP ID preceded by a backslash (e.g. `\1058829`) – construct an automaton as shown in Fig. 4.

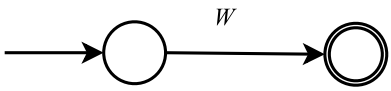

Fig. 3. Automaton constructed for a word W

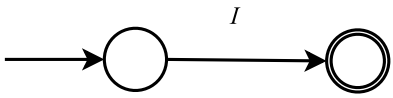

Fig. 4. Automaton constructed for I

4. For L – a CLP label preceded by a backslash and optionally followed by operator `<n,m,...>` (e.g. `\AE` or `\AE<5,6>`; for a detailed description of operator `<n,m,...>` see Section 4.2) – construct an automaton as shown in Fig. 5.

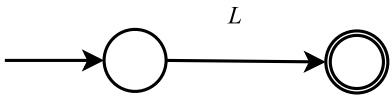

Fig. 5. Automaton constructed for L

5. For S being one of the symbols: `.` (period), `$` (end of sentence) or `^` (beginning of sentence) construct an automaton as shown in Fig. 6.

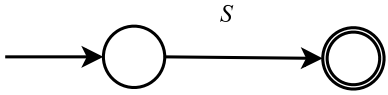

Fig. 6. Automaton constructed for S

6. For $X+$, where X is an automaton representing a CLP regular expression, construct an automaton as shown in Fig. 7.
7. For $X*$ construct an automaton as shown in Fig. 8.
8. For $X?$ construct an automaton as shown in Fig. 9.
9. For $X \mid Y$ construct an automaton as shown in Fig. 10.
10. For XY (concatenation) construct an automaton as shown in Fig. 11.
11. For $X+?$ construct an automaton (ϵ_{NG} standing for non-greedy ϵ) as shown in Fig. 12.
12. For $X*?$ construct an automaton as shown in Fig. 13.
13. For $X??$ construct an automaton as shown in Fig.eps/14.

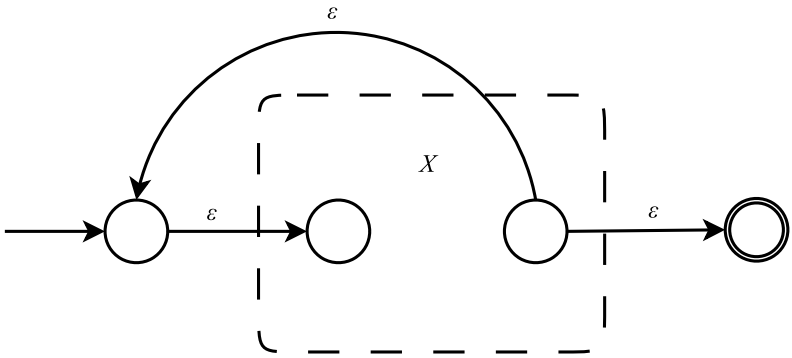

Fig. 7. Automaton constructed for $X+$

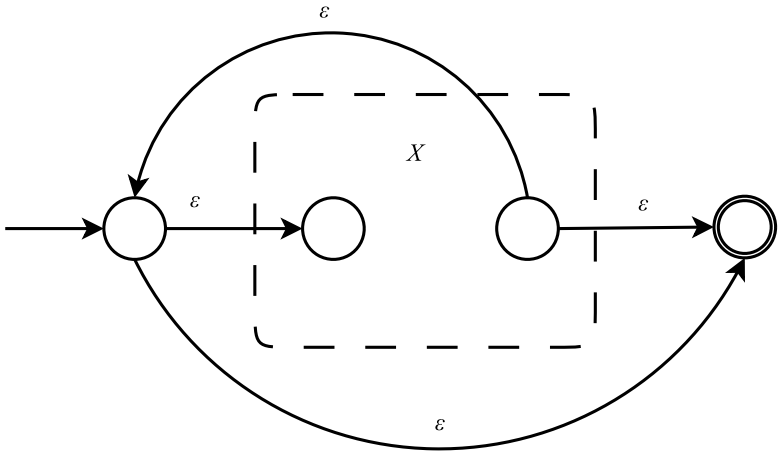

Fig. 8. Automaton constructed for $X*$

The $X\{n,m\}$ expressions are transformed into either concatenations of X (e.g. $X\{4\} \longrightarrow XXXX$), alternatives of concatenations (e.g. $X\{2,4\} \longrightarrow XX \mid XXX \mid XXXX$), or more complex expressions (e.g. $X\{3,\} \longrightarrow XXXX*$).

In cases 11–13 non-greedy ϵ-transitions are used. Such transitions should be used as few times as it is possible for the automaton to accept a regular expression (just as it is in classic regular expressions). Therefore non-greedy ϵ-transitions are treated separately by the NFA minimization algorithm (see Section 4.3.2).

4.3.2. Modified NFA minimization algorithm

The minimization algorithm used in the engine is a modified version of subset construction – a standard method for converting nondeterministic finite automata to deterministic finite automata (described in detail in [4]).

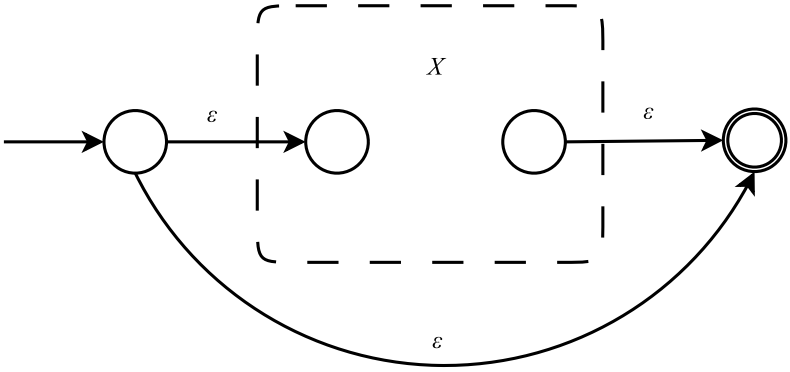

Fig. 9. Automaton constructed for $X?$

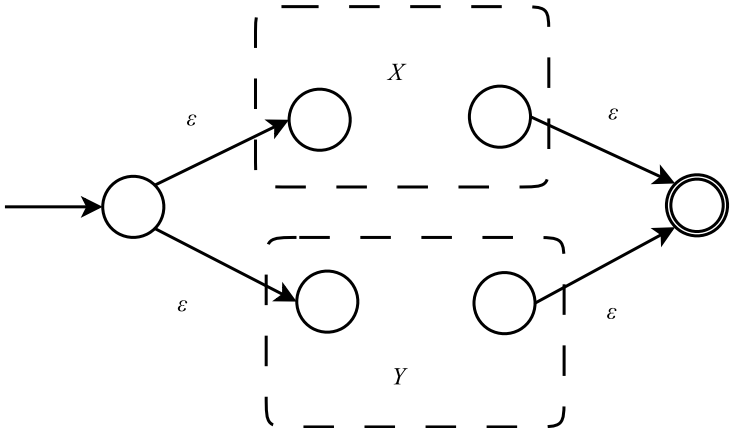

Fig. 10. Automaton constructed for $X \mid Y$

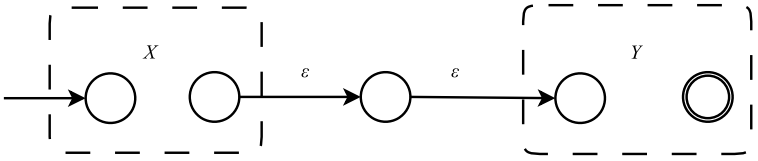

Fig. 11. Automaton constructed for XY

The modifications introduced do not ensure that the obtained automaton is deterministic.

The minimization algorithm is very similar to its origin. The sole modification of the algorithm is introduced during the construction of ϵ-closure sets.

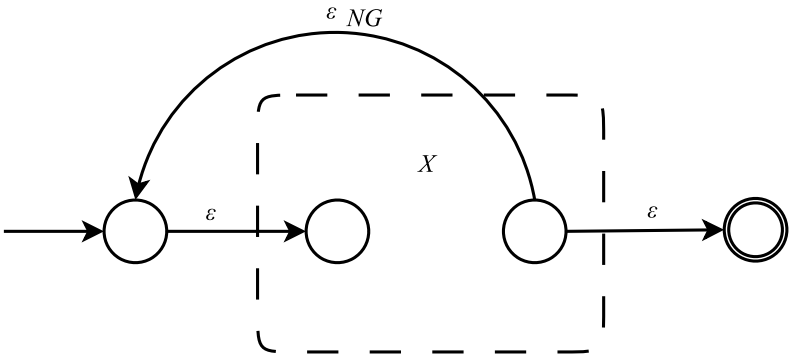

Fig. 12. Automaton constructed for *X+?*

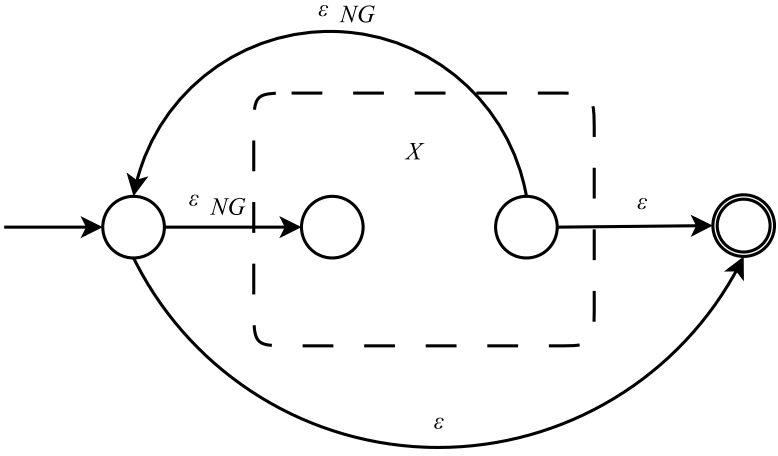

Fig. 13. Automaton constructed for *X*?*

Elements of the sets are calculated just as in classic regular expressions, but in this case non-greedy ϵ-transitions are not treated as ϵ-transitions – instead, during the entire process of minimization, non-greedy ϵ-transition is interpreted as a standard CLP symbol (e.g. a word or a CLP label). Further on the algorithm is the same as its classic version.

4.4. Ambiguities in finite state automata

Transitions in the finite state automata used in the CLP regex engine may be labeled with ambiguous symbols, matching more than one token in the text.

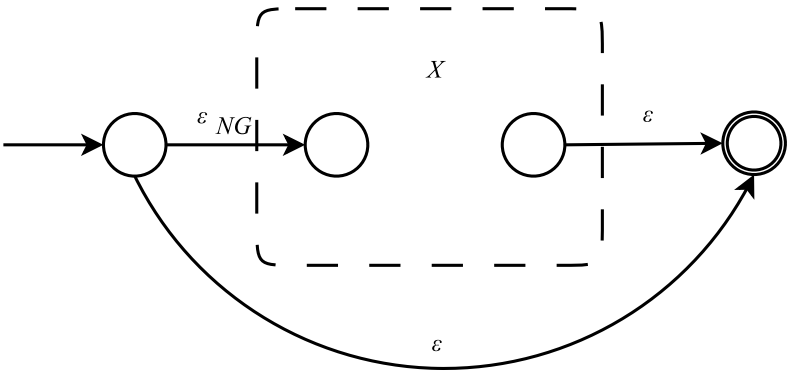

Fig. 14. Automaton constructed for *X??*

There are several kinds of ambiguity in CLP symbols:

- matching multiple words in the same form – the most obvious of the ambiguities; a CLP label only specifies the part of speech and the grammar form of the word searched, and so cannot be expected to match only one word.
  Example: `\A` is a regex with the CLP label for a noun and matches all nouns in the Polish language, e.g. *kot* (Eng. *cat*, singular, nominative, masculine), *domku* (Eng. *little house*, singular, locative, masculine), *lalki* (Eng. *dolls*, plural, nominative, feminine), *dzieciom* (Eng. *children*, plural, dative, neuter)
- matching multiple forms – most parts of speech in Polish are inflecting (e.g. by case, number), therefore a single CLP ID may match more than one word form.
  Example: `\1058829` is a regex with the CLP ID of the word *pilot* (Eng. *pilot*) and matches the forms: *pilot* (base form), *pilota, pilotowi, pilotem, pilocie* (singular forms), *piloci, pilotów, pilotom* and *pilotami* (plural forms).
- matching multiple words with common forms – some words in Polish have different meanings but one or more forms in common. In such cases a string in the CLP regex pattern may match words with different CLP IDs.
  Example: The string `kotek` (Eng. *kitten* – singular, nominative, masculine, or *female cats* – plurar, genitive, feminine) in the CLP regex pattern matches words:
  - *kotek* (Eng. *kitten*) – CLP ID *1031507*, forms: ***kotek**, kotka, kotkowi, kotkiem, kotku, kotki, kotków, kotkom, kotkami, kotkach*,
  - *kotka* (Eng. *female cat*) – CLP ID *1031509*, forms: *kotka, kotki, kotce, kotkę, kotką, kotko, **kotek**, kotkom, kotkami, kotkach.*

4.5. Pattern matching

Matching a regular expressions pattern means verifying whether a sequence of tokens is accepted by the constructed automaton. Beginning in the automaton's initial state, the regex matcher module traverses subsequent states until it cannot make another

transition. If the matcher reaches an accepting state, it remembers the position and length of the matched sequence, and carries on searching for a longer match.

Processing of the automaton by the regex matcher is similar to traversing an automaton while matching a classic regular expression, with one exception: transitions labeled with non-greedy ϵ-transitions have lower priority than all others. The matcher resorts to using non-greedy ϵ-transitions only if no matching sequence of words can be found omitting them.

A single token is matched to a simple CLP expression E (represented by a single transition in the automaton). The matcher's actions depend on the type of E. The following types of E expressions and resulting actions are defined:

- If E represents a CLP ID (e.g. `\12345678`), the matcher retrieves from the CLP library a list of possible CLP IDs for the token. The token matches if the CLP ID provided in E appears on the list.
- If E represents a CLP label (e.g. `\AC`), the matcher retrieves from the CLP library a list of possible labels for the token. The token matches if the label provided in E is a prefix of at least one label on the list.
- If E represents a CLP label followed by operator `<n,m,...>` (e.g. `\AC<1,3>`), the matcher retrieves from the CLP library a list of possible pairs of a label and a corresponding inflectional vector for the token. The token matches if the label provided in E is a prefix of at least one label from the returned pairs and in the same time the inflectional vector from the selected pair contains at least one of the operands of `<n,m,...>`.
- If E is a string built using the `[a-z0-9]` character class (e.g. `kotka`; Eng. *kitten*, dative, singular), the matcher returns the result of a simple case insensitive string comparison between the token and the word provided in E.

4.6. Sentence splitting

The engine splits input text into sentences implementing following rules introduced in [2] and stated below:

1. Unless preceded by a semicolon (*;*), a sentence may begin with either a number or a word beginning with a capital letter. If a sentence is preceded by a semicolon, it may also begin with a word beginning with a small letter.
   Input: Ala ma kota. Ula ma psa; kiedyś Ula miała kanarka. (Eng. *Ala has a cat. Ula has a dog; once Ula had a canary.*)
   Output:
   - Ala ma kota. (Eng. *Ala has a cat.*)
   - Ula ma psa; (Eng. *Ula has a dog;*)
   - kiedyś Ula miała kanarka. (Eng. *once Ula had a canary.*)
2. A sentence may finish with one or more of the punctuation marks: period (.), question mark (*?*), exclamation mark (*!*) or semicolon (*;*).

3. A period following an abbreviation or an ordinal number is not interpreted as the end of a sentence.
4. A single letter followed by a period is considered an abbreviation.
5. A number followed by a period is considered an ordinal number.
   Input: Pani prof. Nowak uczyła w szkole do 1999 r., po czym przeszła na emeryturę. Była laureatką 12. edycji konkursu na najlepszego nauczyciela. (Eng. *Prof. Nowak used to teach in school until 1999, and retired afterwards. She won the 12th best teacher contest.*)
   Output:
   - Pani prof. Nowak uczyła w szkole do 1999 r., po czym przeszła na emeryturę. (Eng. *Prof. Nowak used to teach in school until 1999, and retired afterwards.*)
   - Była laureatką 12. edycji konkursu na najlepszego nauczyciela. (Eng. *She won the 12th best teacher prize.*)

A list of common Polish abbreviations has been used to distinguish abbreviations in examined texts. The presented solution is however not optimal and in few cases may result with an incorrect division of text into sentences.

5. Examples of use

The simplest way to present strength of the tool is to study a few examples made on real text. All examples presented in this section are based on the PAP corpus which contains over 50,000 notes from the Polish news agency and is available from our Computer Linguistics Group. We consider simple examples based on matching the specific words cases.

5.1. A noun with an adjective

The first search finds noun phrases represented in text by a noun with an adjective.

The search defined by the regular expression `(\C\A)|(\A\C)` will recognise every pair of an adjective and a following noun as well as a noun followed by an adjective. Selected output from matching this regex across the PAP corpus is presented below (matched elements are shown in bold):

- *W nocy ze środy na czwartek zmarł po długiej i* ***ciężkiej chorobie*** *minister kultury i* ***dziedzictwa narodowego*** *Andrzej Zakrzewski.*
- *W latach 1991–95 był* ***wysokim rangą*** *urzędnikiem w Kancelarii Prezydenta i jednym z* ***najbliższych współpracowników*** *Lecha Wałęsy.*
- *Podkreślił, że SLD rozpoczął już przygotowania do* ***wyborów parlamentarnych*** *w 2001 r. „Według nas,* ***wybory prezydenckie*** *są już rozstrzygnięte”.*
- ***Wojska rosyjskie*** *poinformowały, że po* ***raz pierwszy*** *użyły półtoratonowych, a nie, jak dotychczas, 500-kilogramowych, bomb podczas nalotów na* ***górskie bazy*** *bojowników.*

- *60 pasażerów natychmiast wystąpiło o* ***azyl polityczny*** *w Wielkiej Brytanii.*
- *„To jest kwestia* ***bezpieczeństwa narodowego****" – powiedział deputowany Mychajło Kosiw, jeden z autorów dokumentu.*
- *Rosjanin Jewgienij Pluszenko został mistrzem Europy w* ***łyżwiarstwie figurowym*** *w konkurencji solistów.*
- *Wyniki sondażu* ***opinii publicznej*** *ze stycznia, zgodnie z którym poparcie dla PSL znacznie przewyższa poparcie dla UW i niemal dorównuje notowaniom AWS, przedstawili dziennikarzom liderzy ludowców.*
- *UW nie poprze AWS-owskiego projektu* ***ustawy uwłaszczeniowej****.*
- *Zdaniem Jerzego Wierchowicza, szefa* ***klubu parlamentarnego*** *UW, może to doprowadzić do* ***kolejnego kryzysu*** *w koalicji przy* ***drugim czytaniu ustawy uwłaszczeniowej****.*
- ***Komitet Ekonomiczny*** *Rady Ministrów uznał za wskazane dokapitalizowanie* ***Rafinerii Gdańskiej*** *SA przed jej prywatyzacją 30 proc. pakietem udziałów Dyrekcji Eksploatacji Cystern.*

5.2. Patterns with the verb *be*

A little different from described previously noun phrases can be considered. The verb *be* (Pol. *być*) is used for pointing sentences where a noun is described by an adjective.

The regular expression defined as `\A \285323840 \C` will find any noun followed by any form of the verb *być* (which has 285323840 CLP ID number), followed by any adjective (matched elements are shown in bold):

- *Zdaniem prezydenta, taka* ***ustawa jest wątpliwa*** *moralnie, budzi roszczenia innych środowisk, a jej* ***skuteczność jest ograniczona****.*
- ***Umowa jest ważna*** *do końca roku.*
- ***Vadim był żonaty*** *pięciokrotnie.*
- *Ponieważ* ***sytuacja jest stabilna****, spodziewamy się, że zarząd giełdy odwiesi od poniedziałku, zawieszenie notowań akcji BIG BG – powiedział.*
- *„Dla nas* ***życie jest święte****" – powiedziała.*
- *Minister edukacji Mirosław Handke podkreślił niemniej, że* ***„studia będą bezpłatne*** *w takim zakresie, w jakim pozwoli wielkość dotacji budżetowej [. . .]".*
- ***Wyrok jest prawomocny****.*
- *Ale to dobrze, że nasi* ***politycy są aktywni****, bo mogą wtedy coś dobrego zrobić.*
- ***Efekty są widoczne****.*
- *Odbudowana z ruin* ***świątynia będzie poświęcona*** *13 sierpnia 2000 r.*
- *Jego* ***stan jest „ciężki****, ale stabilny".*
- *W trzech poprzednich,* ***gra była wyrównana****.*
- *Według SLD* ***debata jest potrzebna****.*

5.3. Patterns with comparison of nouns

Using comparison phrase structure in the given language one can search for relation between two nouns. This relation can be represented as an adjective corresponding to a topic of comparison. The following pattern is proposed as an example for Polish `\A \285323840 \C \285978416 \A` . This regular expression is composed of a noun label followed by any form of the verb *być* (CLP ID 285323840), followed by an adjective label, followed by the preposition *niż* (CLP ID 285978416, this word has to be defined by the ID number because it is ambiguous), followed by any noun.

Selected results from the PAP corpus are presented below:

- *Dla większości respondentów CBOS* ***wolność jest ważniejsza niż równość****.*
- *Niestety, w bilansie dogrywek* ***podaż była większa niż popyt*** *– powiedział Marek Pokrywka, makler DM BOŚ.*

5.4. Semantically depended adjectives

The next pattern aims at finding pairs of two adjectives joined with one of conjunctions: *and* (Pol. *i*), *or* (Pol. *lub*, *albo*), *as well as* (Pol. *oraz*). Found adjective pairs have various semantic dependencies and are interesting for further evaluation. The regular expression used to prepare the search has form `\C ( i | lub | albo | oraz ) \C`. Selected pairs from the PAP corpus are presented below:

długiej i ciężkiej, Poległym i Pomordowanym, drobnego i średniego, Środkowej i Wschodniej, najszybszego i największego, słaby i umiarkowany, socjalistyczne i socjaldemokratyczne, polityczny i symboliczny, spóźnione i niewystarczające, chorych i upośledzonych, charytatywne i kościelne, ogólnopolskim i regionalnym, prozatorskie i dramatyczne, umiarkowane i duże, trzeciego i czwartego, pierwszym i drugim, moralnego i prawnego, podatkowych i celnych, najstarszych i najdostojniejszych, wieczorowych i zaocznych, fabularne i dokumentalne, państwowych i kościelnych, kompromitujący i żenujący, malarska i graficzna, średnim lub wyższym, nieudolna i głupia, konkretnych i szczegółowych, największych i najlepszych, światowych i regionalnych, zdolny i światły, informacyjnej i telekomunikacyjnej, zawyżone i sprzeczne, niewolniczej i przymusowej, bici i upokarzani, prywatnych i publicznych, bici i torturowani, politycznych i gospodarczych, niekompetentny i nieuczciwy, szybkiego i skutecznego, państwowych oraz jednoosobowych, zwarta i zdyscyplinowana, zagraniczni i krajowi, silny i porywisty, federalne i stanowe, wojskowych i politycznych, jasnych i precyzyjnych, rasowej lub religijnej, wilgotny i ciężki, ciepłym i wilgotnym, chłodnym i wilgotnym, suchych i wilgotnych, sejmowe lub senackie, zniszczonych lub uszkodzonych, rakietowym lub moździerzowym, zniesione lub zlikwidowane, nuklearnej lub biologicznej, żywego lub martwego, przywłaszczone albo rozdysponowane, nierealny albo możliwy, czerwony albo bordowy, naiwnym albo obłudnym, mieszkających albo pracujących, angielskim albo niemieckim, faszystowskimi albo neonazistowskimi, rannego albo zabitego.

6. Conclusion

Presented method for querying natural language texts can be interesting especially for linguists. The CLP regex engine provides new possibilities of making easy search with regex like syntax, yet still extended with grammar properties of words. Good quality of results is also noticeable, because of using sentence splitting (avoiding matching words across sentences boundaries). Presented solution can be easily adapted to any inflectional language on condition that an inflectional dictionary is available for this language. Some modifications to text splitting rules may also be required.

References

[1] W. Lubaszewski et. al.: *Słowniki komputerowe i automatyczna ekstrakcja informacji z tekstu*. Wydawnictwo AGH, pp. 107–126, 2009

[2] E. Branny, M. Gajęcki: *Text Summarizing in Polish*. Computer Science, Annual of AGH University Of Science and Technology, pp. 31–46, 2005

[3] G. Grefenstette, P. Tapanainen: *What is a word, What is a sentence? Problems of Tokenization.*. 3rd Conference on Computational Lexicography and Text Research COMPLEX'94 Budapest, 1994

[4] A. A. R. Sethi, J. D. Ullman: *Compilers: Principles, Techniques, and Tools.*. Addison-Wesley, 1988

[5] J. Hopcroft, J. Ullman: *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979

[6] *Regular Expressions.* The Single UNIX Specification, Version 2, The Open Group, 1997, `http://opengroup.org/onlinepubs/007908775/xbd/re.html`

A. Acronyms used in the article

BRE – Basic Regular Expressions, POSIX standard,

CLP – C Language Processing library, developed in the Institute of Computer Science, AGH University of Science and Technology, Kraków

CLP ID – a word identification number in the CLP library

ERE – Extended Regular Expressions, POSIX standard

FSM – Finite State Machine

PAP – Polska Agencja Prasowa (Eng. *Polish News Agency*)

PLP – Python Language Processing; a Python wrapper for the CLP library, used in the implementation of the CLP regex engine

POS – Part of Speech

POSIX – Portable Operating System Interface for Unix

regex – regular expression

NFA – Nondeterministic Finite Automaton