Regular expressions for interpreting and cross-referencing Hadith texts

Marco Boella[1]

0. Introduction

Computational linguistics is by definition an interdisciplinary science. One can approach it mainly in two ways: the linguistics-driven approach aims to verify theories on language by using ad hoc computational instruments, while the computational-driven approach tends to apply and adapt existing computational models to specific linguistic contexts. These complementary conceptions are reflected in the relationship between the development of knowledge-based and empirical methods[2]. This contribution aims to test if a computational knowledge-based typical framework, namely the syntax of regular expressions, might be exploited to draft specific strategies depending on a given typology of text. In other words, the needing to deal with a specific text (hadith collections in this case) models and even create original solutions then can enhance and widen the field of regular expressions application.

After an introduction to the syntax of regular expressions, its diffusion as tool in natural language processing and text analysis is covered, with a special focus on Arabic texts. Regular expressions seem to be more powerful than a mere search tool: they could represent – in handling semi-structured texts – a kind of "lens" which read the internal organization of a text and make it explicit. Such an interpretation is applied to a hadith collection, in order to retrieve its textual structure and some key contents. A program named HadExtractor has been therefore designed and implemented: within the framework of regular expression syntax, it automatically extracts from each hadith some relevant explicit and implicit information (*isnād*, *matn*, transmission chain, all transmitter names and typologies of transmission, Koranic quotations) and organizes it in a machine-readable scheme. As for cross-reference, on the basis of the conceived program to find Koranic citation in hadiths, a process flow of a new program extending is field of action to whatever Arabic text is proposed.

1. Roma, Università "La Sapienza".
2. Soudi *et al.*, 2007.

1. Regular expressions and Natural Language Processing

1.1. The syntax of regular expressions

The term "regular expression" (also regex) comes from mathematics and computer science theory, where it reflects a trait of mathematical expressions called regularity. The so-called "regular expressions syntax" was conceived in the 1950s by Kleene as tool of automata theory to describe formal languages. The original regexes were mainly applied to mathematic theoretical environments, but they have been soon implemented by Thompson and others, through nondeterministic finite automation processes, to be used in programming languages and frameworks, such as Perl, .NET(C#), Java, Ruby, Python[3].

A regular expression is a formally structured text string for describing complex search patterns, which are afterwards applied to longer text units in order to look for matches. There are some basic operations that regular expressions are used for:

– searching: involves moving through a string to locate a sub-string that matches a given pattern;

– matching: involves testing a string to see if it fully conforms to a pattern;

– replacing: the text string matching the pattern is substituted with another given string;

– splitting: according to a pattern that identifies some characters or words as separators, a block of text is divided into a list of sub-texts.

The pattern is made up by a sequence of characters, some of which are assigned with reserved functions. Besides a group of characters that keep their mere alphanumeric meaning (e.g. "a" means "a"), there are two more sets of special meaning. One includes symbols that represent alphanumeric categories and subsets, the other is composed of operators that apply specific functions.

Among symbols, `.` means "any alphanumeric character"; `\s` means "any space (single space, tab, end of row, end of paragraph)"; `\d` means "any digit", and so on. Among operators, `*` "find zero or more occurrences", `+` "find one or more occurrences", `?` "find one occurrence", and so on.

All these characters are combined together according to given syntactic rules in order to represent – in a machine-readable form – complex inquiries. As a way of example, the sentence "look for any text chunk included between the word *the* or *a*, or *an* and the word *book*" is translated in the following compact regex:

(1) `(the|a|an)+.+book+.`

Similarly, the sentence "find, just once, one or more space characters followed by one or more digits" is expressed in regex language as follows:

(2) `\s+?(?=\d+)`

which means, by using an interlinear representation:

3. Friedl, 2002.

(3)

Find	just once,	one or more	spaces	followed by	one or more	digits
	?	+	\s	(?=)	+	\d

1.2. Regular expressions and text analysis

Due to the rich syntax and the high degree of "abstract representation" provided, the regexes are amazingly powerful and deeply expressive and their application spreads beyond the original targets, in language processing and text analysis in particular[4].

The regexes were originally conceived mainly for artificial languages treatment – and among them programming languages – in which ambiguity is programmatically avoided and the structure is always clear and explicit. But when regexes deal with natural languages, which are permeated by ambiguity and implicit meanings, it is self-evident that the achievements of results strictly depend on the level of internal organization and equivocalness of the target text.

Nowadays several strategies for natural language processing usefully involve the regexes in various fields such as concordances, finding word stems and stemming, tokenization and searching tokenized texts, text normalization, segmentation and lemmatization[5]. The fact remains that, to date, regex applications in literature deal mostly with English texts or Latin script based languages , which allow some regex strategies and deny others. So the target language of selected texts processed by regexes, strongly conditions and typifies the power of analysis and the achievable results (e.g. regexes help to identify proper names just in those language scripts that features capital letters and use them at the beginning of a proper name mostly: this is true in English or in French but not in German and not at all in Arabic)[6].

The effectiveness rate of these tasks grows when some conditions are observed, such as explicitness (e.g. the word borders are marked with a retrievable sign, i.e. a white space) and regularity (e.g. all word borders are marked always with a white space only). It is consequently possible to draft regex strategies for any language, but for each text the value of these two parameters set and define the range of effectiveness.

1.3. Treatment of Arabic texts

Concerning the history of Arabic processing computational techniques, literature shows that the top investigated fields deal with morphological analysis and stemming[7]. Different reasons account for this trend, such as the fact that part of the Arabic linguistic tradition itself emphasized the core role of morphology compa-

4. Goyvaens and Levitan, 2009.
5. Bird *et al.*, 2009, p. 97-112; Jackson and Moulinier, 2002.
6. Bird and Klein, 2006.
7. The procedure that reduces a word to its core or root.

red to syntax and semantics, or the defective nature of Arabic scripts, which compels reader to understand the structure of a word before be able to reading it. A serious analysis of these reasons is here skipped, but one of the main outcomes of such a framework is that, till now, stemming and morphological tagging are often regarded as essential prerequisites also for typically non-morphological approaches, as information retrieval, syntactical parsing, text to speech conversion[8].

Employing regexes as first approach to an Arabic text could reasonably alter such theoretical paradigms in which morphology is so weighty. As regexes merely act on the "surface" of a text, they seem able to easily provide to the scholar useful information prior to any other heavy analysis or pretreatments , such as stemming or similar things.

Literature on the use of regexes in Arabic text analysis is rather slight, but there are some interesting works in which regexes are here originally employed as a powerful and faceted instrument to query corpora in Arabic. Through regex, text is investigated at two levels, namely single words and groups of them. In the first level, regexes allow to extract stems from words showing the potential of "light" treatments compared to heavier ones (namely morphological analysers)[9]. Concerning the search of group of words instead, regexes allow collocations, syntagms and word associations to be easily extracted from an Arabic text by using time-saving, semi-automatic queries[10].

In such a framework the regexes are efficiently exploited by running specific or suite-integrated self-supporting applications, whose aim is precisely to allow the user to search a text through a graphical interface. Similarly, many common word-processing applications feature regexes to offer advanced search tools to user.

2. Regular expressions as a reading key for structured texts

Usually to perform regex processes, existing third party applications are employed. This part will focus instead on an attempt to design and build a specific program that internally uses regexes to look for pattern matching, in order to retrieve information and automatically segment and tag an Arabic text. This seems practicable since regex syntax is integrated in several programming languages.

Such a method is an example on how to evolve the status of regex in text linguistics from an instrument that finds something, to a "key" that – by complex segmentation – allows reading and, somehow, understanding of a text. This evidently needs as prerequisite a text featured by a high grade of internal organization and formalization.

8. Abu al-Ḫair, 2008.
9. Kouloughli, 2008.
10. *Ibid*., 2009.

A study on analysis and translation in symbols of semi-formal texts in descriptions of mathematics could help to clarify this point[11]. The authors examine the formalization of symbolisms and formulas used in mathematics manuals, and draft a model that: (i) finds regularities in a text; (ii) employs them as patterns to extract textual and meta-textual information; (iii) sets a list of rules based on these patterns in order to automatically translate extensive verbal expression in math's formulas and vice versa. This study and others[12], confirm that segmentation could be used not only to identify discrete strings, but to try to assign, through a map of regularities and recurrences, a global structure to the text itself as well. This structure might be seen as governed by a sort of contextual "grammar of rules", which also controls connections between content's information and its textual organization.

2.1. HADITHS AS STRUCTURED TEXTS

The use of regex as a key to read and explain a text requires that such a text is somehow natively self-structured. Among the corpus of Classical Arabic Literature, several texts and even genres features some internal formalization, such as lexicons, dictionaries, encyclopedias and prosopographies, but the hadith collections seem to be the most representative. The typical hadith structure[13] consists of two parts, the *isnād*, in which the whole chain of transmitter names of the related tradition is reported, and the *matn*, which contains the tradition itself. Since just close sets of words separate both *isnād* from *matn* and the various transmitters inside *isnād* one from another, this explicit organization allows to detect and retrieve information with a relatively small amount of ambiguity.

A detailed description of the internal hadith composition in the light of parameters of regularity and explicitness[14] is discussed elsewhere[15]. Suffice it to say that, according to this description, a hadith alternates strings whose content is typically an information unit (as transmitter names, *matn* itself) with strings whose main function is rather to bound and tag such information units. The example (4) shows the first part of a hadith, in which the "information" strings (marked with INFO#) are bounded and linked each other through the "functional expressions" (marked with FE)[16]:

11. Wolska and Kruijff-Korbayová, 2004.
12. Bird and Klein, 2006.
13. The most important hadith collections of the IX and X centuries are here considered. They are called *muṣannaf*, "classified", as hadiths were organized by subjects and not by early transmitters as in *musnad* works. See also Burton, 1994.
14. See section 1.2.
15. Boella, 2011.
16. Al-Buḫārī [edition 1990], p. 162.

(4)

ḥaddathanī	aḥmadu bnu ʾishkābi	ḥaddathanā	muḥammadu (i)bnu fuḍaylin	ʿan	ʿumārata bni (a)l-qaʿqāʿ	[...]
FE#1	INFO#1	FE#2	INFO#2	FE#3	INFO#3	[...]

The so-called functional expressions seem profitably to cover two main functions, namely: they signal the next presence of an information unit (a transmitter name inside the *isnād*, or the *matn* as a whole); they specify the typology of the next information unit (e.g. private narration, public reading, written authority, etc.) .

The hadith collections can be read – accordingly – as they were "native databases", in which the hadiths are the records. Each hadith organizes its information units (the *isnād* with the transmitter chain and the *matn*) through a set of labeled fields (represented by the functional expressions, which clearly segment the different units and link them each other. This "database" does not take natively the form of a tabular representation, but it is rather linearly expressed in the text itself by means of a sort of internal mark-up language that tags each information unit.

3. READING HADITHS THROUGH "REGEX LENSES": THE HADEXTRACTOR PROGRAM

The hadith collections might be interpreted as a clearly structured set in which data have been originally encoded through a sort of specific internal "programming language". The regex syntax perfectly fits a conversion strategy of the original text in a full database, in which textual and meta-textual information are explicitly stored and nested.

To test this process, a specific program named HadExtractor has been designed. This program belongs to a broader project that aims to provide complementary strategies to approach hadith texts, from segmenting and retrieval to graphical representation of information and arrangement of a fully lemmatized corpus[17].

HadExtractor (HE) aims to: read a full collection and identify single hadiths; segment for each hadith *isnād* from *matn*; extract from each *isnād* all transmitters' names together with relative supplementary information (position, typology and direction of transmission).

HE has been coded with Python, an interpreted programming language that includes a powerful regex module and is featured by several advantages, namely it is object-oriented, its syntax is clear and intelligible even for non-skilled programmers, and the algorithms and logical processes are rendered in code lines in an almost transparent way[18].

17. The project is called SALAH (Segmentation and Linguistic Analysis of ḥadīṯ Arabic Texts); for reference see: Boella *et al.*, 2011.
18. Lutz, 2007; Mertz, 2003; Perkins, 2010.

At the current stage of development, HE is a typical single-run program: once it has been executed on an input text, it produces the related output and then it ends its main scope. The HE process flow is detailed below, with a focus on those operations that involve regex syntax.

3.1. The input file

Among the canonical hadith collections, an on-line edition of *Ṣaḥīḥ al-Buḫārī* was chosen[19], due to its full digitalization and vocalization, which allow various tasks (segmentation, lemmatization, etc.) with a tiny need of pre-treatment. At the present stage of analysis the check on orthography and coherence of the digital version has been delayed as not essential for the program definition. As canonical hadith collections share virtually the same text structure, HE is ready to be tested on other collections, such as of Muslim and Ibn Ḫuzayma.

The chosen version was originally available in simple text files (.txt) encoding Arabic scripts.

3.1.1. Transliteration

The original text in Arabic scripts has been transliterated prior to being processed in HE, due to most practical rather than theoretical reasons. As Python fully supports Unicode encoding, Arabic script processing is practicable in theory but very hard to manage, in the stage of writing code in particular, where Arabic characters are fully supported but not the switching between right-to-left and left-to-right direction in the same string[20].

The Arabic script has been therefore converted in a specific left-to-right transliteration system, based on Buckwalter's model but modified in order to fit to Python and regular expressions constraints on special reserved characters[21]. In this system each Arabic grapheme matches with just one character in the ASCII set, and besides lower-case letters, upper-case letters and some non-alphabetical characters are employed as well, in order to strictly keep one-to-one relationship and avoid any kind of "vertical" diacritics. For example, the word حَدَّثَنَا, (*ḥaddathanā)* is transliterated as "Had + aCanaA".

The reading of such a transliteration could be tougher but, as the system conserves bijective mapping, it is possible at any stage of text processing to switch from the transliterated version to Arabic script and vice versa, with the aim to produce final output in Arabic scripts again.

3.1.2. Text pretreatment

As regex works on the surface of a string "as is", usually input texts don't require any kind of previous manual treatment, such as tagging or normalization. So our text – once transliterated – was directly processed with HE.

19. Al-Buḫārī [edition 1990].
20. Madhany, 2006.
21. Lancioni, 2011.

3.2. Regex processing

The core of HE is made up by four main regex-based tasks.

3.2.1. Text splitting: from collection to single hadith

The first regex employed is the simplest one, and allows to split the full collection in single hadiths. It is based on the automatic recognition of the point in which a digit starts. The following is an excerpt of full input text, in which every number clearly marks the beginning of a new hadith:

(5)

> [...] 7561 - Had+aCanaA caliy+U Had+aCanaA hiXaAmU Eax_baranaA mac_marU can_ Alz+uh_riy+i H w Had+aCaniy EaH_madu b_nu SaAliHI Had+aCanaA can_basaou Had+aCanaA yuwnusu [...] saEala EunaAsU Aln+abiy+a Sal+aY All+ahu calay_hi wasal+ama can_ Al_kuh+aAni faqaAla ein+ahum_ lay_suwA biXay_-I faqaAluwA [...] Eak_Cara min_ miAJaoi kaV_baoI 7562 - Had+aCanaA Eabuw Aln+uc_maAni Had+aCanaA mah_diy+u b_nu may_muwnI samic_tu muHam+ada b_na siyriyna yuHad+iCu can_ mac_badi b_ni siyriyna can_ Eabiy saciydI Al_xud_riy+i [...] Al_qis_Tu maS_daru Al_muq_siTi wahuwa Al_caAdilu waEam+aA Al_qaAsiTu fahuwa Al_jaAJiru 7563 - Had+aCaniy EaH_madu b_nu eiX_kaAbI Had+aCanaA mu-Ham+adu b_nu fuDay_lI can_ cumaAraoa b_ni Al_qac_qaAci can_ Eabiy zur_caoa can_ Eabiy huray_raoa raDiya All+ahu can_hu qaAla qaAla Aln+abiy+u Sal+aY All+ahu calay_hi wasal+ama kalimataAni HabiybataAni ei-laY Alr+aH_mani xafiyfataAni [...]

Since digits in the text are employed to identify the hadith number only, there is no ambiguity, and a regex has been set to find all digits:

```
p = re.compile(r'\s*?(?=\d+?)')
q = p.split(text)
```

The first line allows to set the pattern of the regex and the second line to apply it. The regex is the bolded code, which means “find just once, one or more space characters followed by one or more digits”. This regex offers a basic example of how to split a text in more sub-texts separated each other by a recurrent non-ambiguous element (digit in this case).

3.2.2. Reading the structure of a hadith: separating *isnād* from *matn*

At this stage the single hadith are identified, and the result is a list of strings like the following:

(6)

> 7561 - Had+aCanaA caliy+U Had+aCanaA hiXaAmU Eax_baranaA mac_marU can_ Alz+uh_riy+i H w Had+aCaniy EaH_madu b_nu SaAliHI Had+aCanaA can_basaou Had+aCanaA yuwnusu can_ Ab_ni XihaAbI Eax_baraniy yaH_yaY b_nu cur_waoa b_ni Alz+ubay_ri Ean+ahu samica cur_waoa b_na Alz+ubay_ri qaAlat_ caAJiXaou raDiya All+ahu can_humaA saEala EunaAsU Aln+abiy+a Sal+aY All+ahu calay_hi wasal+ama can_ Al_kuh+aAni faqaAla ein+ahum_ lay_suwA [...] fiy EuVuni waliy+ihi kaqar_qaraoi Ald+ajaAjaoi fayax_liTuwna fiyhi Eak_Cara min_ miAJaoi kaV_baoI

The regex here employed is the most complex: the aim is not to find a particular element, as seen in 3.2.1, but rather to model in a pattern the complete structure of the whole text contained in a hadith:

(7)

```
p1 = re.compile(r"""
        ^(?P<num>\d+?)
        [ ]*?-?[ ]*?
        (?P<isn>.*?)
        (?P<sep>
        (Ean\+a(?:hum_|hu|haA|humaA|[ ])+?.{0,50}?qaAla(?:t_|A|[ ])+?)
        |((qaAla(?:t_|A|[ ])+? |yaquwlu) (?!.{0,100}
        (samic(_tu|a)|Had\+aCaniy|Had\+aCanaA|Eax_baranaA|Eax_barahu|can_)
        )))
        (?P<mat>.*?)$
        """,
                re.VERBOSE | re.MULTILINE)
```

Focusing line by line:

(7) ^(?P<num>\d+?)

The sign "^" stands for the beginning of the string, while the regex in brackets catches the hadith number (see 3.2.1 for the description).

(8) [ ]*?-?[ ]*?

(8) recognizes the score after the number. Since in the original text score is often but not always (due to human faults in digitalization) surrounded by blank spaces, this regex shows its flexibility in considering all alternatives, and it means "one or zero spaces, followed by exactly one score, followed by one or zero spaces".

(9) (?P<isn>.*?)

(9) identifies all the text after the previous condition and before the next one as *isnād*, and it means: "find at least zero or more characters and name them as 'isn' ". The size of this string fully depends on the following condition:

(10)

```
(?P<sep>
        (Ean\+a(?:hum_|hu|haA|humaA|[ ])+?.{0,50}?qaAla(?:t_|A|[ ])+?)
        |((qaAla(?:t_|A|[ ])+? |yaquwlu) (?!.{0,100}
        (samic(_tu|a)|Had\+aCaniy|Had\+aCanaA|Eax_baranaA|Eax_barahu|can_)
        )))
```

(10) tries to identify the separation point between *isnād* and *matn*, and name it as "sep". The expression is long and intricate due the fact that the word or group of words (called in 2.1 "functional expression", or FE), to find is not unique but may differ, and it belongs to a list. Moreover many of these terms are ambiguous, so they sometimes recur in the text without any functional meta-textual meaning

but just linguistic one. For example, in separating isnād from *matn* the most recurring FE is definitely qāla(t), 'he/she said'. This word – due to its widespread meaning – might recur everywhere in hadith text without any FE value, for instance inside the *matn*. So regexes must be able to solve two crucial problems: alternation (the FE to find belongs to a list and can alternate) and ambiguity (not all matching expression are actually FEs).

Alternation might be easily scheduled in regex through the sign "|", which has the value of the Boolean operator OR. Each "|" sign in the example above sets an alternative to be searched, and its meaning is "find x or y or z".

Obviously the problem of ambiguity is harder to solve: any operator can act as human reasoning that catches ellipsis and analogies, so specific strategies – greatly context-sensitive – must be conceived. Two complementary positional strategies has been chosen. To avoid redundancy, just the most representative and emblematic is described below:

(10.3) |((qaAla(?:t_|A|[ ])+? |yaquwlu) (?!.{0,100}

(10.4) (samic(_tu|a)|Had\+aCaniy|Had\+aCanaA|Eax_baranaA|Eax_barahu|can_)

The two lines mean "find the word *qāla* (and also consider inflected forms *qālat*, feminine, and *qālā*, dual masculine) or the word *yaqūwlu* only if they are not followed within 100 characters by the word *ḥaddathanī* or *ḥaddathanā* or *'aḫbaranā* or *'aḫbarahu*". Assuming that, 100 characters is the tentative maximum size of a proper name in Arabic, so the regex looks just for the *qāla* which is not followed by a proper name belonging to *isnād*. The last condition is ensured because proper names in *isnād* are followed by a typical FE that marks next transmission.

Clearly the use of tailored strategies partly depends on trial and error method and statistical-based attempts; their effectiveness cannot be universally declared, but they are anyway able to heavily reduce the ambiguities and consequently the segmenting errors.

(11) (?P<mat>.*?)$

Last expression is very simple and it means "find all the text till the end of the string and name it 'mat'". Once the separation between *isnād* and *matn* was found, the *matn* is everything to the end of the hadith.

Summarizing, the long regex in (7) try to interpret a whole hadith and extract from it the following information: the hadith number, the *isnād*, the element that separates *isnād* from *matn*, the *matn*.

3.2.3. Binary categorization of elements: transmitter names and typology of transmission

Once the *isnād* chunk has been identified, it can be read as formed by two sets of text, the transmitter names and the FE that specify the typology of transmission. This binary structure is represented in (12):

(12) FE, Transmitter name, FE, Transmitter name, ... , Last transmitter name

The name set is obviously "open", i.e. it is very hard to determine a priori, but the FE set is "closed" instead, and its elements are known. A list of FEs used in *isnād* can be therefore drafted with the help of hadith criticism literature[22].

Since the structure of *isnād* is strictly binary, if the elements of one set are known, the others can be recovered by exclusion, i.e. all the text that is not a FE and it is bounded by FEs is a transmitter name.

(13) (FE)(.*?)(FE)

(13) represents the model for the regex employed in HadExtractor, and "FE" stands for every element in the list shown in (14):

(14) Had\+aCanaA|Had\+aCaniy|samic_tu|samica|Eax_baran(?:iy|aA)[...]

3.2.4. Identification of non-structural elements: the eulogies

However, a deeper glance to hadith shows that there is a third category, which represents a close set of elements and doesn't influence at all the binary *isnād* structure. This set is formed by eulogies, or *ṣalawāt,* the formulaic salutations that are added to the name of prophet Muḥammad (i.e. *ṣallā 'llāhu 'alayhi wa-sallama*) and to some of his relatives and Companions. Since the list of these eulogies is known and their use is always selected by a given context, a simple regex has been implemented to recognize them as "external parts" of the global hadith structure.

3.3. Output: organising the extracted information in a XML file

Once HadExtractor has run all regex tasks, it produces as output an XML file[23], in which all extracted information is organised, marked and nested. Referring to (5), in which the hadith n. 7561 is presented as is in input file, the same hadith is shown in (15) as output after HE processing, with all information explicitly tagged.

(15)

```
<hadith>
  <source_info>
   <vol>9</vol>
   <num>7561</num>
  </source_info>
  <isn>
    <trasm type="Had+aCanaA">caliy+U</trasm>
```

22. Among the well-known FEs, one can mention *ḥaddathanā*, *ḥaddathanī*, *'aḫbaranā*, *'aḫbaranī*, *'anba'anā*, *sami'a*, *'an.* Actually, the exact semantic value of the FEs and consequently its meaning in the *ḥadīṯ* transmission validation process was and is still a controversial subject in "source criticism" *ḥadīṯ* studies. See Robson, 1978; Günther, 2005; Juynboll, 2007; Melchert, 2001.

23. XML (extensible Markup Language) is a set of rules for encoding documents in machine-readable form. It allows an explicit description of contents and their structure. Data written in XML might be easily employed for further elaborations. See Jones and Drake, 2002.

```
    <trasm type="Had+aCanaA">hiXaAmU</trasm>
    <trasm type=" Eax_baranaA "> mac_marU</trasm>
    [...]
    <trasm type="Ean+ahu samica”> cur_waoa b_na Alz+ubay_ri</trasm>
  </isn>
  <sep> qaAlat_ caAJiXaou raDiya All+ahu can_humaA </sep>
  <mat> saEala EunaAsU Aln+abiy+a Sal+aY All+ahu calay_hi wasal+ama can_
Al_kuh+aAni faqaAla ein+ahum_ [...] waliy+ihi kaqar_qaraoi
Ald+ajaAjaoi fayax_liTuwna fiyhi Eak_Cara min_ miAJaoi kaV_baoI
</mat>
</hadith>
```

The current version of HadExtractor has been run on entire *Saḥīḥ* of *Al-Buḫārī*, and considering the total number of hadith as 7397, the failed segmentations between *isnād* and *matn* (in which the proposed pattern is not found) were 240 (i.e. a performance rate of 96,8 %), and the wrong segmentations (false positives: RE has found information but the text is wrongly segmented) inside the *isnād* are approximately 15 % of the total number of segmentations.

4. Regular expressions and cross-referencing

HadExtractor produces an output that can be employed as source for further researches. Two class of information are available, the one concerns the transmission chain of a hadith (transmitter names, typology of transmission, relationships among transmitters), the other consists of the *matn*.

While the transmission chains data have been employed to try – with the help of graph theory – a graphical representation of relationships among transmitters, the *matn* has been processed to improve and train an existing morphological analyser[24].

As final example of the potentiality of using regex in text treatment, a cross-reference strategy is proposed below.

4.1. Finding Koran in Hadiths

The hadiths contain several quotations of Koranic verses or part of them. Since for the whole Koranic corpus is available a complete morphological and syntactic tagging and not for the hadith corpus, can be interesting to draft strategies to find Koranic quotations in hadith and to cross-reference them with a specific tag. This enables us to exploit pre-existing analysis and integrate them as information for some parts of *matn*, in order to avoid duplications of work (e.g. to reconstruct morphological and syntactic level of texts that has been analysed already elsewhere).

The XML output file of HadExtractor can be therefore integrated with a new category of information, i.e. the presence of Koranic passages and the references to the Koranic source.

24. Boella *et al.*, 2011.

4.1.1. Regexs with variables: a powerful method

To test cross-reference between Koran and hadith, a small program was designed and named CrossQuran. Its aim is to check in hadith text all occurrences of Koranic passages. The program uses as input the *matn* extracted with HadExtractor and a digitalized version of Koran that provides lemmatized text, classed in chapter, verses, words together with morphological and syntactical annotation[25]. Verses has been chosen as discrete pattern to be searched in hadith but, as the source is fully lemmatized, smaller unit may be easily employed (i.e. syntagms or three word groups).

All the regexes previously seen in section 3 were featured by fixed patterns written once for all, since the model to search remains always the same while the target text changes every time (i.e. the same pattern was applied to all different hadiths). Here instead also the pattern varies constantly, because the process requires to find correspondences for each Koranic verse (pattern) in every hadith (target text). A solution is to compile the regex by inserting in its pattern a variable that refers to the items of a list: then for each element of the list, the system automatically creates a new regex to be employed on the target text.

(17)
```
pat_var = r'%s'
for pattern in patterns:
p = re.compile(pat_var % pattern)
```

In (17) the full process is exemplified in Python language. The meaning of these lines could be summarized as "for each item of the list find that item...".

At the present stage of development, CrossQuran returns 132 occurrences, i.e. 132 Koranic verses cited in hadith.

4.2. Automatic cross-reference for Arabic text: a draft

The CrossQuran program was implemented to answer to specific needs and target texts, but the same strategies and processes can be used to draft a program that compares two texts and evaluates which parts of the first text are contained in the other one and vice versa.

This aim requires that the minimum size of string to be searched can be made variable. In the case of referencing Koran in hadith, the string's size was fixed to correspond to the length of verse exactly, which is typically a clause or a sentence included between two periods. Trying to extend the range of application, the new program should have the following process flow:

– to load as input the text A (containing patterns) and the text B (the target);

– to chunk A in discrete units, i.e. tokens delimited by blank spaces, assuming that for Arabic this means isolated words or combination of them, unless to provide a previous lemmatization;

25. Dukes, 2010.

– to set the shortest and longest size of tokens combinations that will form the pattern (e.g. five is here fixed as minimum and the numbers of tokens included between two periods as maximum). If the minimum size is two, collocations and syntagms can be found;

– to set the pattern list, which will include all possible combinations of consecutive tokens, starting from the longest one. So a routine must be set to calculate the size of each two-period string and to collect all possible combinations between the maximum token numbers and 5. Assuming for example that a string contains 13 tokens, the routine will produce 44 pattern items for that string (e.g. A_1+ A_2+A_3+A_4+A_5, A_2+A_3+A_4+A_5+A_6, etc);

– to write a regex that considers the list items as variable patterns to be searched;

– to search the whole B text for matching;

– to return data concerning all occurrences found.

The most evident weakness of such a system is that it found just exact matches, but in real texts often references and quotations of other texts are not formulated exactly in the same way. The limitations could be overcome however, thanks to the implication of regex syntax, which allows to search after non-contiguous elements and to rank items in order to skip the finding of less important ones.

References

ABU AL-ḪAIR I., 2008, "Arabic information retrieval", *Annual Review of Information Science And Technology*, n° 41, p. 505-533.

AL-BUḴĀRĪ M. b. I. [1990], *Ṣaḥīḥ al-Buḫārī*, Riyaḍ, Dār Ṭawq al-Najāh [digital edition available on: www.almeskkat.net]

BIRD S., KLEIN E., 2006, *Regular Expressions for Natural Language Processing*, Philadelphia, University of Pennsylvania.

BIRD S., KLEIN E., LOPER E., 2009, *Natural Language Processing with Python*, Sebastopol, O'Reilly.

BOELLA M., 2011, "Reading a text, finding a database: an anachronistic interpretation of hadiths in light of information science", in Capezzone L. ed., *Uscire dal tempo. percezioni dell'antico, del moderno, del futuro* (in press).

BOELLA M., ROMANI F. R., AL-RAIES A., SOLIMANDO C., LANCIONI G., 2011, "The SALAH Project: segmentation and linguistic analysis of Ḥadīṯ Arabic texts", in M. V. Salem, K. Shaalan, F. Oroumchian, A. Shakery, H. Khelalfa ed., *Proceedings of the Seventh Asia Information Retrieval Societies Conference*, Heidelberg, Springer.

BURTON J., 1994, *An introduction to the Hadith*, Edinburgh, Edinburgh University Press.

DUKES K., 2010, *Quranic Arabic Corpus (version 0.2) Based on Tanzil Quran Text (Uthmani, version 1.0.2)* [GNU Public License].

FRIEDL J., 2002, *Mastering Regular Expressions*, Sebastopol, O'Reilly.

GOYVAENS J., LEVITAN S., 2009, *Regular Expressions Cookbook*, Sebastopol, O'Reilly.

GÜNTER S., 2005, "Assessing the sources of classical Arabic compilations: the issue of categories and methodologies", *British Journal of Middle Eastern Studies*, n° 32 (1), p. 75-98.

JACKSON P., MOULINIER I., 2002, *Natural Language Processing for Online Applications: Text Retrieval, Extraction & Categorization*, Amsterdam, John Benjamins.

JONES C. A., DRAKE F., 2002, *Python & XML*, Sebastopol, O'Reilly.

JUYNBOLL G. H. A., 2007, *Encyclopedia of Canonical Hadith*, Leiden, Brill.

KOULOUGHLI D., 2008, "Initiation pratique à la constitution et à l'exploitation de corpus électroniques en langue arabe (III^e partie)", *Langues et Littératures du Monde Arabe*, n° 7, p. 75-93.

— 2009, "Initiation pratique à la constitution et à l'exploitation de corpus électroniques en langue arabe (IV^e partie)", *Langues et Littératures du Monde Arabe*, n° 8, p. 117-133.

LANCIONI G., 2011, *An Adaptation of Buckwalter Transcription Model to XML and Regular Expression Syntax*, Technical report, Roma, Roma Tre University, r3a.

LUTZ M., 2007, *Learning Python*, third edition, Sebastopol, O'Reilly.

MADHANY H. N., 2006, *Multilingual Computing with Arabic and Arabic Transliteration*, Chicago, The University of Chicago Press.

MELCHERT C., 2001, "Buḫārī and early hadith criticism", *The Journal of the American Oriental Society*, n° 121 (1), p. 7-19.

MERTZ D., 2003, *Text Processing in Python*, Boston, Addison Wesley.

PERKINS J., 2010, *Python Text Processing With NLTK 2.0.*, Birmingham, Packt Pub.

ROBSON J., 1978, "Ḥadīth", *Enciclopaedia of Islam*, Leiden, Brill, n° III, p. 23-28.

SOUDI A., VAN DEN BOSCH A., NEUMANN G. eds, 2007, *Arabic Computational Morphology. Knowledge-Based and Empirical Methods*, Dordrecht, Springer.

WOLSKA M., KRUIJFF-KORBAYOVÁ I., 2004, "Analysis of mixed natural and symbolic language input in mathematical dialogs", *Proceedings of the 42nd Annual Meeting of the ACL*, Association for Computational Linguistics, Barcelona (Spain).