Processing Persian Text: Tokenization in the Shiraz Project

Karine Megerdoomian and Rémi Zajac

Memoranda in Computer and Cognitive Science
MCCS-00-322

Computing Research Laboratory
New Mexico State University
Las Cruces, New Mexico
April 2000

Abstract

Prior to morphological analysis or syntactic parsing, a text needs to undergo tokenization, in order to determine sentence and word boundaries. This report describes the tokenizer used in the Shiraz Persian-English machine translation project at the Computing Research Laboratory. The Persian writing system and the methods that can be used in recognizing token boundaries in written text are presented. The system uses a low-level language-independent tokenizer, which outputs an unambiguous sequence of basic tokens. Difficulties arise in analysis of Persian text since certain detachable morphemes need to be reattached to the word before morphological analysis takes place. In addition, words are often concatenated in written form. These pre-processing tasks are accomplished by a post-tokenizer that contains language-specific information.

Contents

1. Introduction 1

2. Persian Letters 3

3. Word Boundaries 6
3.1 Punctuation 6
3.2 Space 7
3.3 Word and Morpheme Boundary 7
3.3.1 Character Forms 7
3.3.2 Algorithm for Word and Morpheme Boundary Detection 8
3.3.3 Algorithm for Unknown Words 9

4. Disambiguating Sentence Boundaries 12
4.1 Acronyms 12
4.1.1 Descriptive Analysis 12
4.1.2 Ambiguous Constructions 14
4.1.3 Summary 15
4.2 Abbreviations 15
4.2.1 Descriptive Analsysis 15
4.2.2 Ambiguous Constructions 16
4.2.3 Summary 16
4.3 Abbreviations, Acronyms and Sentence Interaction 17

Conclusion 19

Appendix 20

1

Introduction

Prior to morphological analysis, the text needs to undergo tokenization, in order to determine sentence and word boundaries. This report provides a description of the tokenizer used in the Shiraz project[1] at the Computing Research Laboratory. The system uses a low-level language-independent tokenizer, the output of which is an unambiguous sequence of basic tokens. The low-level tokenizer is then followed by a post-tokenizer, which contains language-specific information and operates on the output of the low-level tokenizer. The post-tokenizer is mainly used to reattach inflectional elements that have been separated by the low-level tokenizer.

Persian uses the Arabic alphabet (with four additional characters), written from right to left, but the characters are converted into Unicode for all processing within the system. The language-independent tokenizer uses generic character properties to classify strings into basic token types:

- Word: sequences of letters;
- Number: sequences of digits;
- Separator: a Unicode separator, which includes punctuation characters, dashes and spaces;
- Alphanumeric: a sequence of mixed letters, digits and symbols;
- Symbols: characters such as %, #, etc.;
- Control: characters such as line breaks, tabulations and control characters such as joiners.

The low-level tokenizer does not segment the text into textual units, such as headers, paragraphs and sentences, and it does not distinguish between various punctuation characters. Hence, the *stop*, which usually marks a sentence boundary, and the *comma*, are treated identically; the distinction between various punctuation markers has to be incorporated within the system at a later stage. Furthermore, Persian written text separates compounds and certain inflectional morphemes by using a zero-width-joiner, a control character. Since a control character is treated as a distinct token by the low-level tokenizer, the two parts of a Persian compound or inflected word are often separated. The separated parts of an inflected word need to be reattached before morphological analysis applies. Compounds are analyzed at a later stage in the analysis.

This report describes the Persian writing system and the tools that can be used for recognition of token boundaries in written text. It also explains the approach utilized by the post-tokenizer for reattaching separated morphemes. A common problem in analyzing Persian text resides in the fact

1. http://crl.nmsu.edu/Research/Projects/shiraz

that words are often concatenated. This problem is currently resolved within the pre-processor component. The process used by the pre-processor in separating concatenated words is alos discussed. The final section is devoted to the punctuation mark for ‘stop’ or ‘period’, since it has several roles in marking boundaries. Persian acronyms and abbreviations are described as well, although it should be noted that they are not recognized in the current version of the Shiraz project. Throughout this report, the term tokenizer will be used to refer to the module consisting of the low-level tokenizer plus the post-tokenizer.

2

Persian Letters

Persian uses the Arabic alphabet with four additional letters that do not exist in standard Arabic: *pe, che, zhe* and *gâf.* A complete list of the Persian alphabet is presented in Table 1. In the Shiraz system, the characters are converted into Unicode, which is used for all system-internal processing. Persian distinguishes between the initial, medial and final forms of the letters, which indicate the position of the characters within a word. The recognition of the final form characters is crucial for morphological analysis. In Unicode, the final form characters in Persian are represented as a zero-width-joiner (a control character) which can be used during processing of morphological tokens. Determining word boundaries using the character forms is discussed in detail in Section 3.3. Note that Table 1 lists the Persian characters in isolated form only; the initial and medial forms of the characters are not displayed.

In order to work with the Persian writing system in an efficient manner, we developed a romanization specifically for the Shiraz project at CRL. It is referred to as the Shiraz romanization. This romanization was designed to be bijective, so that the text could be put back into the original Persian format automatically without losing any information. The attempt was to keep the romanization easily readable for the language acquirers hence the four Persian letters *ze, zal, zat* and *za*, which are pronounced /z/, all have a romanization based on the English letter “z”; the distinction between the characters is provided by different diacritics. The Persian letter *alef* can be pronounced as almost any vowel, depending on the context it appears in, and therefore it has not been romanized as any one vowel in English. In addition, since short vowels are often not written in Persian text, the romanization does not provide any short vowels, thus maintaining the ambiguities available in original text.

For the purposes of this document, however, we use the romanization shown in Table 2 to represent the Persian examples. This romanization does not provide any short vowels, since they are not available in Persian text. This table also provides the reader with a pronunciation guide. Note, however, that the short vowels are not transcribed, hence the pronunciation obtained is not the complete and accurate pronunciation of the Persian word.

Table 1: Persian Alphabet

Glyph	Name
ط	ta
ظ	za
ع	eyn
غ	gheyn
ف	fe
ق	ghaf
ک	kaf
گ	gaf
ل	lam
م	mim
ن	nun
و	vav
ه	he
ی	ye
ئ	ye hamze
ء	hamze
أ	alef hamze
ؤ	waw hamze
ۀ	he ye
ً	Tanvin

Glyph	Name
آ	alef madd
ا	alef
ب	be
پ	pe
ت	te
ث	se
ج	jim
چ	che
ح	He
خ	khe
د	dal
ذ	zal
ر	re
ز	ze
ژ	zhe
س	sin
ش	shin
ص	sat
ض	zat

Note: *alef with madd* and *alef* are vowels. The letters *vav, he* and *ye* can be either a consonant or a vowel given the context. As consonants, they are pronounced ‘v’, ‘h’ and ‘y’ respectively. As vowels, they double as ‘u’ (as in *food*), ‘e’ and ‘i’.

Table 2: Romanization used in the report

Persian Letters	Romanization	Pronunciation
alef with madd	A	father
alef	a	and *or* bed *or* so
be	b	boy
pe	p	pool
te	t	toy
se	s̄	sun
jim	J	Joe
che	ch	church
he	H	horse
khe	x	*similar to* German buch
dal	d	dog
zal	z̄	Zorro
re	r	*similar to* Spanish “r”
ze	z	Zorro
zhe	j	mirage
sin	s	sun
shin	sh	shoe
sat	S	sun
zat	Z	Zorro
ta	T	toy
za	Z̄	Zorro
eyn	e	and *or* bed *or* so *or* uh oh (glottal stop)
gheyn	Q	*similar to* French “r”
fe	f	fun
ghaf	q	*similar to* French “r”
kaf	k	kite
gaf	g	great
lam	l	love
mim	m	Mary
nun	n	nun
vav	v	very *or* food
he	h	horse
ye with hamze	i	you *or* uh oh (glottal stop)
ye	y	you *or* sea
short space (marking a final form character)	~	

3

Word Boundaries

The task of the tokenizer is to detect word boundaries in a written text and to provide a uniform segmentation before the processing of the text takes place. In Persian text, word boundaries can be delimited by space, punctuation, and the forms of the characters indicating its position within a word. Words can also be written concatenated. Similarly, some morphemes may appear in either attached or detached form. This section discusses all the possible word and morpheme combinations in a written Persian text as well as the possible boundary markers. The final segmentations produced by the tokenizer will treat all words, including the subparts of a compound or light verb, as separate tokens. All detachable morphemes, however, will be analyzed as forming a single token unit with the word from which they are separated by a short space character (a zero-width-joiner). If the string is ambiguous, then the tokenizer will produce multiple segmentations.

3.1 Punctuation

Certain punctuation marks denote sentence boundaries. In Persian, the stop, the exclamative and the interrogation mark are unambiguous boundary indicators. The stop also marks a sentence boundary, but it may also appear in the formation of abbreviations or acronyms. Apart from the slash (/), which is used in numbers, and the dash, which could be used to separate compound words, the other punctuation marks unambiguously indicate word boundaries. These include the comma, quotes, brackets and colon.

The low-level tokenizer tags all punctuation and dashes as the token *separator*. The parser used in the Shiraz project does not take into account any tokens that have not been tagged as Word tokens; all tokens that are not Word tokens are taken to be a hard (i.e., sentence) boundary. Hence, another module needs to distinguish between separators that are sentence boundary markers and those that are word boundary markers, otherwise no syntactic parse will be applied across the latter and the sentential analysis will not be obtained. In our system, the word boundary marking separators have been included within the syntactic grammar. For instance, the comma is incorporated in the syntactic rules to determine a phrasal boundary.

3.2 Space

In Persian text, the boundaries for distinct words are usually denoted by a space. No space appears between the two words forming a compound or a light verb construction[2]. This spacing pattern, however, is not very consistent and sometimes different words may appear without a space separating them; these concatenated words will be discussed in the following section. In the low-level tokenizer, a space is tagged as a separator token and is able to separate distinct token words successfully.

3.3 Word and Morpheme Boundary

3.3.1 Character Forms

The Persian writing system distinguishes between the final forms and the initial or medial forms of a character, depending on its position within a word. This is illustrated in Figure 1. An initial form does not mean that the character is in the beginning of a word, it only indicates that the character is not at the end of the word. Characters are in medial form if they have an attaching character both before and after them. A final form character indicates the end of a word and can be used by the tokenizer to determine word boundaries. Hence, two concatenated words can be put into separate tokens if the first word ends in a final form character. These final forms are indicated by a zero-width-joiner following the character in Unicode. For transliteration purposes, we will use the standard *tilda* (~) to denote this control character.

final	*medial*	*initial*	
ب	ـبـ	بـ	**“b”**
گ	ـگـ	گـ	**“g”**
ج	ـجـ	جـ	**“J”**

Figure 1: Sample Persian character forms

There are certain letters (*alef, dal, zal, re, ze, zhe, vav*), however, that have only one form regardless of their position within the word. If such a character ends the first word of a concatenated pair, the tokenizer will not be able to use the character form to determine the word boundary (see Section 3.3.3, “Algorithm for Unknown Words”, for the algorithm used in these situations).

Compound and light verb constructions most often appear without a space separating the two parts. If the first word within the compound ends in a final form character, the two parts will be separated into distinct tokens by the tokenizer, as shown in (1) for a compound string and in (2) for a light verb string. Both compounds and light verbs are recognized as a single lexical unit at a later stage in the processing.

2. A *light verb construction* is a verbal unit consisting of a preverbal element (usually a noun, adjective, or preposition) followed by a light verb (e.g., *krdn* “do”, *dadn* “give”). The latter has partly or completely lost its original meaning within this construction. In that sense, the light verb construction behaves like a compound and needs to be included in the dictionary.

(1) “riys~Jmhvr”-->“riys”“Jmhvr”
Lit.: head republic
‘President‘

(2) “zng~zdnd”-->“zng”“zdnd”
Lit.: bell hit (past/3pl)
‘(they) phoned‘

Certain morphemes always appear attached to the word whereas others could be written either attached or separated by a zero-width-joiner. Rarely, the detached morphemes appear separated from the word by an intervening space. The attached morphemes are analyzed as one token with the word they appear on, but detached morphemes will be treated as a separate token by the low-level tokenizer. The post-tokenizer is then used to join the detached morpheme back on the word in order to form a single token as input to the morphological analyzer. When a morpheme is reattached to the stem, a short space character (~) or zero-width-joiner must separate the morpheme and the word. The reason for the insertion of this character is illustrated in the examples below. Consider the string in (3), which consists of a noun meaning ‘letter’ followed by the Indefinite marker. The low-level tokenizer will separate the stem noun and the indefinite morpheme into two distinct tokens as shown. When the post-tokenizer reattaches the morpheme to the stem, it should insert a zero-width-joiner in order to obtain the original string. If the zero-width-joiner was not inserted between the word and the morpheme, the resulting string will be as given in (4), which is a different word and morpheme combination altogether, as can be seen from the translation provided. Hence, the post-tokenizer algorithm should introduce a zero-width-joiner before reattaching a morpheme in order to maintain the distinction between attached and detached inflection in Persian.[3]

(3) “namh~ay”-->“namh”“ay”
letter-Indef
‘a letter’

(4) “namhay”
name-Plur-Ezafe
‘(the) names of’

3.3.2 Algorithm for Word and Morpheme Boundary Detection

The algorithm used by the post-tokenizer needs to consider all the possible word and morpheme combinations and provide all the segmentation patterns. In the final result, words are separated into distinct tokens. Morphemes that appear in attached form in the text will remain attached to the stem. Detached morphemes will appear separated by a short space (~) or zero-width-joiner.

As an example, consider the string “shrab~xvb” (=good wine), which is separated by a short space character indicating a final form letter. For all occurrences of final form letters followed by an initial form character, the low-level tokenizer separates the string into two distinct tokens. The string in this example will be separated into the two distinct tokens “shrab” and “xvb”. If these tokens represent two distinct words, the segmentation is complete. If one of the strings is a morpheme, however, it will need to be reattached before the strings are sent to the morphological analyzer. This is done in the post-tokenizer. The following provides this algorithm, which is applied to the output of the low-level tokenizer.

3. The Ezafe is a morpheme that links the head of a phrase to the following constituents.

Post-Tokenizer Algorithm:

We look at two consecutive strings in each case. Each token is to be checked against the morphemes list[4].

- If both strings are not morphemes, they remain separated. Nothing needs to be done.

(5) “shrab” “xvb”→“shrab”“xvb”
wine good

- If one of the strings is a morpheme which is not ambiguous with a word, reattach the morpheme. Insert a zero-width-joiner between word and morpheme.

(6) “kvtah” “tryn”-->“kvtah~tryn”
short est

- If one of the strings is a morpheme which is ambiguous with a word, two segmentations result: In one case, the strings remain detached; in the other, they get reattached with an intervening zero-width-joiner.

(7) “my~” “rqSm~”-->1. “my” “rqSm”
IMPdancing(1sg)2. “my~rqSm”
‘(I) am dancing.’
[where “my” can also mean “wine”]

3.3.3 Algorithm for Unknown Words

Concatenated words are a very common site in Persian text. The examples in (8) were extracted from our on-line corpus; they represent various instances of concatenated words. In all of these cases, the first word does not end in a final form character, hence the low-level tokenizer will not be able to separate the words into distinct tokens. As shown in these examples, besides actual compounds and light verbs, short prepositions, conjunctions and the object-marking postposition appear next to the following or preceding word without an intervening space. Two distinct words may also appear concatenated. The last example shows a concatenated case that has separated the parts of a light verb. In this example, the object-marking postposition *ra* of the previous word has been concatenated to the preverbal element (*rd*) of the light verb *rd krdnd* (‘they refused’), thus separating the two parts of the light verb.

(8) a. ***compounds***amvrxarJh--> amvr xarJh
affairs foreign
‘foreign affairs’
b. ***light verbs***pyshnhadkrdnd--> pyshnhad krdnd
proposedid(3pl)
‘(they) proposed.’
c. ***preposition***azShyvnystha--> az Shyvnystha
fromZionists
‘from the Zionists’
d. ***conjunction***tablvv-->tablvv
painting and
‘painting and’

4. The Appendix provides a list of the morphemes that can appear detached with an indication of the ones that can be ambiguous with a word.

e. ***postposition***kshvrra-->kshvrra
countryObj
'the country' (object of sentence)

f. ***distinct words***frAyndbykarsazy-->frAyndbykarsazy
processunemployment
'(the) process of unemployment'
nystndayran-->nystndayran
are notIran
'..(they) are not Iran..'

g. ***separated words***rard krdnd-->rard krdnd
Obj refusal did(3pl)
'.. (they) refused ..'

These examples show that it is essential for a Persian tokenizer to include an algorithm to recognize and separate concatenated words.The previous section discussed cases of concatenated words in which the first word ends in a final form character that can be used to detect the word boundary. If the concatenated word ends in a character that does not have a final form version as in the examples in (8), then the low-level tokenizer is unable to recognize the two words as distinct tokens. We suggest to apply an algorithm to these concatenated words that will insert a space following characters without a final form. These characters are *alef(a), dal(d), zal(z̄), re(r), ze(z), zhe(j), vav(v).* Since one or both words may be inflected, the separated tokens need to also undergo morphological analysis. In the current Shiraz system, the algorithm has been implemented (with slight variations) in the pre-processor component of the system, where concatenation problems are resolved before analysis of text begins[5].

Unknown Word Algorithm

- For all occurrences of characters that do not have a final form (i.e., *a d z̄ r z j v*), the tokenizer produces two segmentations: one in which a space is inserted following these characters and one without a space. A space need not be inserted after the final character in the string. We could eliminate some of the unwanted cases by discarding any combination that contains a single letter (except for "v", which is the conjunction 'and'). It is also possible to eliminate certain combinations if the last string is a morpheme which is not ambiguous with a word. For instance, if the string *msafryn* (=travelers) results in the segmentation "msafr" "yn", since "yn" is a plural morpheme and it is not ambiguous with a word, this particular segmentation will be eliminated. As an illustration, consider the example in (9), where all single letter segmentations have been eliminated. In this instance, a space is inserted following the characters *d, v, r.*

(9) "dvrdnya"-->1."dv" "rd" "nya"
around world2."dvr" "dnya" [correct segmentation]
3."dvrd" "nya"
4."dv" "rdnya"

- Since the concatenated words may be conjugated, the segmented parts need to undergo morphological analysis before being looked up in the dictionary. (10) is a light verb *chaqv zd* ('(he/she) stabbed'; Lit.: knife hit-3sg) with its parts concatenated. The preverbal element *chaqv* (knife) ends in *v,* a character that does not have a final form, and the verbal part *zd*

5. A more efficient method might be to apply this algorithm only to words that have not been recognized after morphological analysis and dictionary look-up (i.e., unknown words). Certain simplifications that have been allowed in the pre-processor component can then be eliminated since the algorithm will not be running on every single word in the text, but only on unknown words.

(hit-3sg) is conjugated. In order to recognize the verb in the dictionary, *zd* has to go through morphological analysis in order to obtain the citation form of the verb: *zdn* (to hit).

(10) “chaqvzd”--> 1. “cha” “qvzd”
2. “chaqv” “zd” [correct segmentation]
3. “cha” “qv” “zd”

4

Disambiguating Sentence Boundaries

The stop is an ambiguous punctuation mark in Persian text since it can represent a period at the end of a sentence or it can be part of an abbreviation or an acronym. This report is a specification for disambiguating the sentence boundary by determining whether a stop encountered in text is a hard boundary marker or part of an abbreviation or acronym. This specification has not been included in the current version of the Shiraz tokenizer, since abbreviations and acronyms are extremely rare in our corpus (3 acronyms and no abbreviations in a 3000-sentence corpus), but it should be considered for a more complete tokenization process.

Although a stop is usually followed by a space in Persian text, this pattern is not very consistent and oftentimes the stop is immediately followed by another character. Acronyms and abbreviations, however, have an easily recognizable structure. Hence, in order to isolate sentences in a Persian text, the tokenizer should determine whether the stop is an element of an acronym or an abbreviation before treating it as a sentence boundary. Note that if the abbreviation or acronym appears at the end of the sentence, the stop can be a sentence boundary as well as an element of the token.

This report defines certain structures that could potentially be recognized as acronyms or abbreviations. When a stop is encountered in a string, the latter could be matched against these structures. The system may also contain sub-dictionaries in order to determine whether a certain string belongs to the abbreviation or acronym token classes. If the match fails, the tokenizer could proceed to segment sentences. In certain cases, the string containing the stop may be ambiguous between two constructions. In such cases, the tokenizer will produce all alternate parses for that structure. Section 4.3 brings together all the possible combinations of acronyms, abbreviations, and stops that denote sentence boundaries, and provides an outline of the tokenization rules needed to resolve the ambiguities produced.

4.1 Acronyms

4.1.1 Descriptive Analysis

The acronym could be identified by its surface structure, based on a conjunction of characters and punctuation. The most general format for forming an acronym consists of one or more characters

(ending in a final form if available) and followed by a stop. In this format, each Roman character of the acronym is transliterated into Persian according to the transliteration pattern in Table 3 on page 14. This is illustrated in the examples below. In the examples in (11), the last character before the stop ends in a final form[6] as marked by the short space character ~. In the acronyms in (12), however, since the Persian characters *r* and *a* do not have a final form, the short space character is not available.

(11) *af*~.by~.Ay~FBI
by~.*by*~.*sy*~BBC

(12) *ar.py*~.*Jy*~RPG
ka.g~.*b*~KGB

There exist, however, variations to this format. Certain magazines and newspapers form the acronyms without a stop as shown in (13).

(13) *by*~*by*~*sy*~BBC

The acronyms that can be represented in this format usually consist of Roman letters that are transliterated into Persian with an ending character that has a final form. In other words, the examples in (11) could be written without a stop separating the transliterated forms since the last character before each stop is a final form character. The examples in (12), however, contain non-final form characters before the stop (e.g. *r* and *a*) and thus could not be written without it.

Certain words that are considered acronyms in English are treated as proper nouns in Persian and are not represented by a letter by letter transliteration; they are written instead as they are pronounced as shown in the following examples.

(14) *sya*CIA
*aydz*AIDS

Note that all acronyms in the language are transliterated forms of foreign acronyms. We have not found any Persian acronyms written in the format described in the examples (11) through (13). Instead, Persian acronyms follow the pattern illustrated in (14) and should be treated as proper names. The examples shown below are instances of such acronyms, where *savak* is the acronym for *sazman amnyt v a'Tla'at kshvr* (Security and Information Agency of the Country), and *ndaJa* stands for *nyrvy dryayy artsh Jmhvry aslamy yran* (Marine Forces of the Military of Islamic Republic of Iran).

(15) *savak*~ (Savak)
*ndaJa(*Nedaja)

6. See the document "Persian Tokenization", included in this volume, for a description of final form characters in Persian. The characters that lack the final form are *a, d, z, r, z, zh, v.*

Table 3: Roman characters and the corresponding Persian transliterations

Roman character	Persian transliteration
A	*a* (or *A* word-initially)
B	*by~* or *b~*
C	*sy~*
D	*dy~*
E	*ay~*
F	*af~*
G	*g~*
H	*ach~*
I	*ay~*
J	*Jy~*
K	*ky~* or *ka*
L	*al~*
M	*am~*
N	*an~*
O	*a*
P	*py~*
Q	
R	*ar*
S	*as~*
T	*ty~*
U	*yv*
V	*vy~*
W	*dblyv*
X	*ayks~*
Y	*vay~*
Z	*z*

4.1.2 Ambiguous Constructions

Since acronyms have an easily recognizable structure, they are not usually ambiguous with other constructions. The only acronym format that might be ambiguous with words is the one exemplified in (13) which appears without any stops. The construction in this example has the surface form of a compound consisting of three parts and it can easily be recognized in the compound lookup component.

When a potential acronym has been detected, it could then be checked in the dictionary. If the token does not exist in the dictionary, it can be translated following either the transliteration patterns given in Table 3.

The stop ending an acronym is also ambiguous with a full-stop, a sentence boundary marker.

4.1.3 Summary

Format

The format for acronyms could be represented by the following rules:
For the case of acronyms, the characters [Aa-y] do not represent an arbitrary sequence of letters but are the Persian transliterations of foreign letters following the formats given in Table 3.

1. ([Aa-y]+\~?\.)+[Aa-y]+\~?\.?
   One or more combination of one or more letters followed by a stop. The last stop is optional. This rule represents the examples shown in (11) and (12).
2. ([Aa-y]+\~)+[Aa-y]+\~?
   One or more combination of one or more letters, ending in a final-form character, not followed by a stop. This is the format illustrated in example (13).

Ambiguities

Format 1 above is not ambiguous with words, but it may end with a stop, which makes it ambiguous with the end of a sentence.

Format 2, on the other hand, represents acronyms that are ambiguous with words and morphemes but are not potential markers for an end of sentence since they contain no stops. These tokens are treated as compounds, hence they are sent to the low-level tokenizer as is.

4.2 Abbreviations

4.2.1 Descriptive Analsysis

Abbreviations can appear as a single character with or without a stop, as shown in (16). The letter *v* is not usually written without a stop since *v* is a word in Persian, the conjunction “and”. A word is also abbreviated if it consists of one or more characters and the last character before the stop is left non-final as illustrated in the examples in (17), where characters such as *m* or *g* that have final forms, appear without a short space character (~).[7] Note that if the last character is among those which do not have a final form (e.g., *r*), this distinction would not be available as the examples in (18) show.

(16) *S*~for *SfHh*~ (=page)
m~for *mylady*~ (=A.D.)
m~.for*mylady*~ (=A.D.)

(17) *Alm.*for*Almany*~ (=German)
*ang.*for*anglysy*~ (=English)

(18) *fr.*for*fransh*~ (=French)
*ar.*for *armny*~ (=Armenian)

The example below indicates the usual format for abbreviating authors’ names:

7. In order to obtain a non-final form before a stop, a control character (the zero-width-non-joiner, ZWNJ) appears in certain encodings to force the non-final form of the letter.

(19) *J~. m~.*for*Jlal~ mtyny~* (Jalal Matini)

The abbreviation formats, however, are not very consistent. Example (20) illustrates the three possible abbreviation forms used, in various articles in the same magazine[8], for indicating the lunar calendar year *hJry~ qmry~*. As can be seen from these cases, the two characters abbreviating the lexical element can be written with stops following both, or a stop following only the last character, or simply separated by a space without any stops. Note that in all of these instances, the first character appears in non-final form whereas the last character is final.

(20) *h*.q~.
h q~.
h q~

4.2.2 Ambiguous Constructions

Once more, any stop appearing on an abbreviation is ambiguous with a full-stop and can thus mark the end of a sentence.

Any of the single character abbreviation patterns (i.e., the examples in (16), (19) and (20)) are unambiguous with words but if followed by a stop, these tokens might also indicate the end of the sentence. In these cases, the tokenizer should produce both possibilities: the token as an abbreviation only or the token as an abbreviation but also indicating the end of the sentence. Similar ambiguous outputs should be available when the tokenizer encounters a token of the format in (17).

If the tokenizer finds a combination of two or more letters ending in one of the characters that does not have a final form as shown in (18), the string could be either an abbreviation or a word. In addition, the stop is ambiguous with a period at the end of the sentence. In such cases, the tokenizer will produce three outputs: the token can be an abbreviation, it could be an abbreviation marking the end of a sentence, or it could be a word marking the end of a sentence.

4.2.3 Summary

Format

The following formats are then available for abbreviations in a Persian text:

1. Single letter: A single letter ending in final form, the stop is optional.
   [Aa-y]\~\.?
   This format is exemplified in (16).
2. Non-final forms: One or more letters ending in a (forced) non-final form, followed by a stop.
   [Aa-y]+[FF]\.
   where [FF] is the set of characters that have a final form.
   This rule represents the formats illustrated in the examples in (17).
3. Non-final characters: One or more letters ending in a character without a final form, followed by a stop.
   [Aa-y]+[NFF]\.

8. *Majalle-ye Iranshenasi* (Iranshenasi - Journal of Iranian Studies)

where [NFF] is the set of characters that don't have a final form.
This rule represents the examples in (18).

4. Initials: Single letters separated with stops and/or spaces, followed by a single final form character. Final stop optional.
[Aa-y]\~?[\.] ?[Aa-y]\~?\.?
The examples in (19) and (20) are represented by this rule.

Ambiguities

Format rules 1, 2, and 4 all represent tokens that are not ambiguous with a word, but which are ambiguous with the end of the sentence if they are followed by a stop. Note that the stop is optional in rules 1 and 4, but it is obligatory in rule 2. The tokenizer will then produce two possible outputs: in the first case, the token is to be treated only as an abbreviation, and in the second case, the token is an abbreviation marking the end of the sentence.

Rule 3 denotes a token which is potentially ambiguous between an abbreviation and a word. Furthermore, since it is followed by a stop, then it is also a potential marker for the end of the sentence. The final output then consists of three possible cases: the token is an abbreviation, the token is an abbreviation marking the end of the sentence, or the token is a word marking a sentence boundary.

4.3 Abbreviations, Acronyms and Sentence Interaction

In this section we bring together the previous discussion by presenting an outline of all the possible cases for abbreviations and acronyms. The section also provides the tokenization rules needed to resolve the ambiguities resulting from the interaction of these tokens with sentence boundaries.

Following the application of the format rules described in the previous sections, the tokenizer recognizes potential acronyms and abbreviations in the text. Each token should be marked as

- ambiguous or not ambiguous with a word.

- ambiguous or not ambiguous with an end of sentence (i.e., determine if the token is followed by a stop or not).

If a token is not ambiguous with a word, then the tokenizer creates the token. If a token is ambiguous with a word, then the tokenizer should create, in addition, a word token. Furthermore, if a token is not ambiguous with the end of the sentence (EOS), then no action should be taken by the tokenizer. However, if the token is followed by a stop, then the tokenizer should also create a new sentence boundary as illustrated in the figure below. In order to do this, the tokenizer needs to copy the open sentence set (e.g., copy sentence 1 in the figure into sentence 2) and close the new

set thus created. A new sentence set is then opened (sentence 3 in the figure).

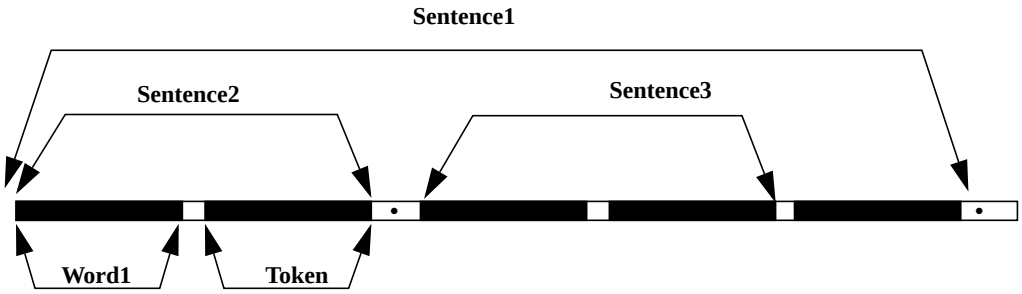

Figure 2: Creating a sentence boundary

The table below shows the ambiguity combinations possible and the action the tokenizer should take in each case. The format rules that were used for the recognition of the token involved in each instance are also indicated in the Input Token column. (The Acronym rule 1 and Abbreviation rules 1 and 4 are present in both of the first cases because the final stop is optional in these instances.).

Table 4: Tokenization Rules for segmenting Acronyms, Abbreviations and Sentence Boundaries.

Input token	Tokenizer output
1. Token is not ambiguous with a word/ morpheme Token is not ambiguous with EOS ***Format Rules:*** *Acronym rule 1* *Abbreviation rules 1, 4*	• Create the token as an acronym or an abbreviation • Proceed to the next token
2. Token is not ambiguous with a word/ morpheme Token is ambiguous with EOS ***Format Rules:*** *Acronym rule 1* *Abbreviation rules 1, 2, 4*	• Create the token as an acronym or as an abbreviation • Create sentence boundary • Proceed to the next token. If there is a stop, proceed to the next token following the stop.
3. Token is ambiguous with a word/morpheme Token is not ambiguous with EOS ***Format Rules:*** *Acronym rule 2*	• Create the token as word • Proceed to the next token
4. Token is ambiguous with a word/morpheme Token is ambiguous with EOS ***Format Rules:*** *Abbreviation rule 3*	• Create the token as an abbreviation • Create word token • Create sentence boundary • Proceed to the next token following the stop

5

Conclusion

The Persian tokenizer used in the Shiraz project uses a language-independent low-level tokenizer to separate textual elements into basic tokens. A post-tokenizer containing language-specific information is then applied to the output of the low-level tokenizer in order to reattach any separated detachable morphemes. Hence, the Persian tokenizer can successfully separate most concatenated words into distinct tokens, without losing the inflection that appears on the words. The rest of the concatenated words are taken care of in the pre-processor component which accurately inserts a space between the words.

For a more complete tokenizer, the grammars for recognizing numerical, date and time expressions in newspaper texts should also be included. In addition, specifications for disambiguating sentence boundaries and recognition of acronyms and abbreviations are to be incorporated in the tokenizer.

Appendix

This appendix contains a list of detachable Persian prefixes and suffixes used by the post-tokenizer in reattaching separated morphemes. These affixes may be ambiguous with words. Note that some members of this list consist of the concatenation of several affixes. It is possible to enumerate all such cases in Persian since there are a limited number of affixes and they follow strict morphotactic patterns. Listing the combinations of affixes, as done here, allows the post-tokenizer to recognize the detached suffixes and prefixes without using an algorithm for computing morphotactics.

Persian Affix	**Inflection Information**	**Affix Type**	**Ambiguous with word**
by	derivational *or* subjunctive particle	prefix	no
my	imperfective verbal particle	prefix	no
nmy	negation + imperfective particle	prefix	no
ha	plural	suffix	yes (interjection ‘hey’)
hay	plural + ezafe	suffix	yes (interjection ‘hey’)
ay	indefinite marker	suffix	yes (interjection ‘hey’)
am	copula/auxiliary/pronoun clitic (1sg)	suffix	yes (noun ‘mother’ -arabic)
shan	pronoun clitic (3pl)	suffix	yes (noun ‘dignity’)
tr	comparative	suffix	yes (adjective ‘wet’)
try	comparative + copula (2sg)	suffix	yes (adj. ‘wet’ + copula)
ayst	indefinite + copula (3sg)	suffix	yes (noun/interj. ‘stop’)
ast	auxiliary (3sg)	suffix	yes (copula/3sg ‘is’)
hayy / haiy	plural + indefinite	suffix	no
haym	plural + pronoun clitic (1sg)	suffix	no
hayt	plural + pronoun clitic (2sg)	suffix	no
haysh	plural + pronoun clitic (3sg)	suffix	no
hayman	plural + pronoun clitic (1pl)	suffix	no
haytan	plural + pronoun clitic (2pl)	suffix	no
hayshan	plural + pronoun clitic (3pl)	suffix	no
hast	plural + copula (3sg)	suffix	no
hayym	plural + copula (1pl)	suffix	no
hayyd	plural + copula (2pl)	suffix	no
haynd	plural + copula (3pl)	suffix	no
at	pronoun clitic (2sg)	suffix	no
ash	pronoun clitic (3sg)	suffix	no
man	pronoun clitic (1pl)	suffix	no
tan	pronoun clitic (2pl)	suffix	no
aym	auxiliary (1pl)	suffix	no

Persian Affix	Inflection Information	Affix Type	Ambiguous with word
ayd	auxiliary (2pl)	suffix	no
and	auxiliary (3pl)	suffix	no
tryn	superlative	suffix	no
trynha	superlative + plural	suffix	no
trha	comparative + plural	suffix	no
trynm	superlative + pronoun clitic (1sg)	suffix	no
trynt	superlative + pronoun clitic (2sg)	suffix	no
trynsh	superlative + pronoun clitic (3sg)	suffix	no
trynman	superlative + pronoun clitic (1pl)	suffix	no
tryntan	superlative + pronoun clitic (2pl)	suffix	no
trynshan	superlative + pronoun clitic (3pl)	suffix	no
; (hamze)	ezafe	suffix	no