

Finite State Phonetic Transcription Generator for Bulgarian

Aleksandar Savkov

International Studies in Computational Linguistics

aleksandar@savkov.eu

1 Introduction

Being able to generate phonetic transcriptions automatically is a great advantage in fields like transliteration, computer-aided language learning, corpus linguistics, speech generation and dialectology. Moreover this generation is all the more valuable if it is achieved independently of any lexical resources. In general the rules and exceptions in the phonology of a given language may make such a generation possible, however this is not always that easy and the ratio of exceptions is not always low enough. This paper will trace the steps of building an implementation that generates the phonetic transcriptions of Bulgarian words and expressions given their orthographical representation and stress marker using finite state methods.

In Bulgarian the differences between what the orthography suggests as pronunciation and what the real output is after the phonological rules have been accounted are relatively small. The number of exceptions is not great either; it can be divided in two groups: one in which new phonemes are introduced by foreign words do not follow the logic of the Bulgarian phonology and another in which some particular parts of speech undergo vowel alternation in their stressed syllables. This low number of cases where lexical resources are needed makes generating Bulgarian phonetic transcription a rather smooth and straight-forward process when it comes to linguistic theory. The only thing it is really dependent on is the stress, which cannot be generated or guessed.

Building a transcription generator for a language could be presented as implementing the phonetic properties and phonology rules of the language into a program. Traditionally computational task in the fields of phonology and morphology are implemented using finite state methods. The implementation that is presented in this paper uses the Stuttgart Finite State Transducer (SFST) tools, which are rather unorthodox in this field, but also provide some convenient mechanisms that have been used for the implementation of many of the phonological rules.

The paper will elaborate on Bulgarian phonetics and phonology in Section 2, focusing on orthography, stress and phoneme pronunciation; in Section 3 it also describes the most important parts of SFST and gives a number of simple examples to better understand it; next it elucidates the strategy and some parts of the SFST-PL code in Section 4, while still centering on input preparation and vowel and consonant processing. Finally, the results and possible applications of the implementation at hand are discussed in section 5.

2 Bulgarian phonetics and phonology

In terms of complexity Bulgarian phonology leans more towards the simpler systems. Moreover, the differences between writing and pronouncing are relatively few and very consistent. Phonological rules rely mostly on the sound context and rarely on the word form or part of speech (see Section 2.3

for more details). However, not all phonological rules deal with the way pronunciation is generated out of orthography – some rules disallow or certain sound sequences. Such rules are relevant for a morphological processes and analyses, but are not of any help to transcription generation, thus they have been ignored in this paper.

(Sabev, 2000) presents in a nutshell an English version of the most important rules in Bulgarian phonetics and phonology with regard to pronunciation. His work summarizes the ideas and arguments of (Boyadzhiev & Tilkov, 1997) and (Boyadzhiev, Kutsarov, & Penchev, 1998). Another English title worth mentioning is (Scatton, 1984) whose contents are based on the same sources as the latter two titles and which thus agree with them closely. This paper refers mostly to the summary that Mitko Sabev created, but for a deeper understanding of the processes and issues of Bulgarian phonology we recommend referring to the other titles.

2.1 Orthography

Bulgarian is written in Cyrillic which is very close fitted to its phonetic inventory and thus allows in most cases each phoneme to be represented by one orthographical symbol. There are thirty letters in the alphabet and forty-five phonemes listed in the sound inventory. Three of the letters represent two different phonemes: **я** as in *Янтра* ['jantɾə]¹ YANTRA or *стая* /'stajə/ ROOM when in the beginning of a word or following a vowel (consonant followed by **я** in fact represent palatalized version of the consonant followed by [a] as in *баня* ['banjə]² BATH); **ю** in the beginning of a word or following a vowel as in *юни* ['juɲi] JUNE and *приютя* [prijo'tʲɤ] SHELTER (**ю** palatalizes preceding consonants in the same way as **я** e.g. *коњар* [ko'njar] STABLES MAN); **щ** represents the phonemes [ʃ] and [tʃ] in this order, but also pronounced very close (e.g. *свещ* ['svɛ ʃ] CANDLE). There are two cases of two phonemes being represented with a pair of letters – **дж** as in *джанка* ['dʒankə] PLUM and **дз** as in *дзифт* ['dʒɪft] TAR. The letter **ь**, which occurs only between a consonant and the phoneme **о** (e.g. *шофьор* [ʃo'fjɔɾ] DRIVER), has no sound value, it is only used for marking the palatalization of the preceding consonant. All other letters in the alphabet may be considered to correspond to only one phoneme in the ideal case of no further phonological changes. Nevertheless, many of the rules described in Sections 2.3 and 2.4 are often ignored because of the natural strive of native speakers to equalize the pronunciation and orthographical representation of the language – a process which in its opposite direction leads to misspelling.

2.2 Stress

The word stress in Bulgarian is a much discussed matter that unfortunately finds no clear solution. Three facts are defined as important about it in (Pashov & Parvev, 2002):

- it is expiratory, which means that the stressed syllable is pronounced with more emphasis;
- it is free – it may occur on any syllable in terms of order;
- it is mobile, because it may change its position in different word forms.

In addition there are also doublet forms with different stressed syllables (e.g. *казах* ['kazəx] and [kə'zax]), words whose meaning is changed by the stress position (e.g. *пара* [pə'ra] COIN and *па̀ра* [pəɾə] STEAM) and words pronounced without stress – clitics (e.g. *не* [nɛ] NOT in *не_у̀скам* [nɛ'iskəm] NOT WANT-1P.SG.). The general conclusion about Bulgarian stress is that it cannot be identified automatically, leaving lexical reference as the only way of locating its position. However, stress is very important for the phonological processes that take place when pronouncing a word and therefore for transcription generation as well. Hence any automatic method implementing

¹ See Section 2.3 for more information on vowel pronunciation in different contexts.

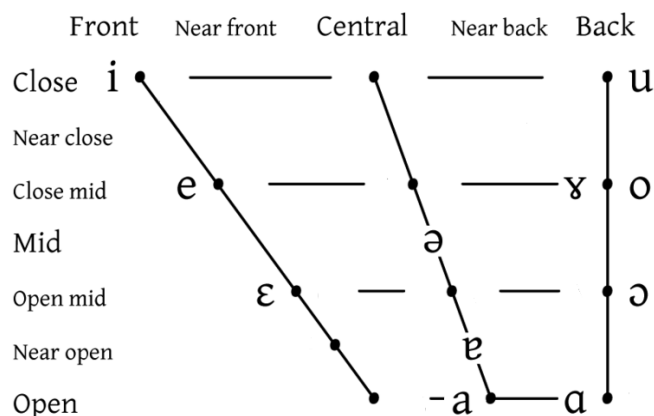
² Here [a] is replaced by [ə] for reasons explained in Section 2.3.

phonological rules has to rely on input with marked stress, which is normally not the usual case in Bulgarian orthography.

2.3 Vowels

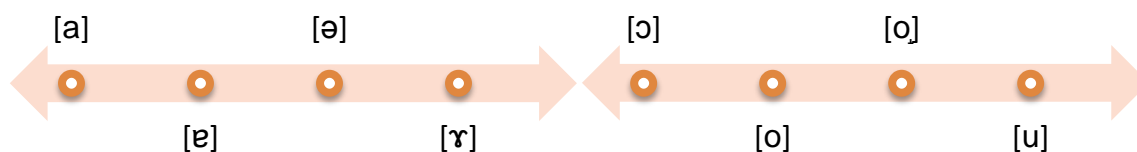
There are six vowels in standard Bulgarian: **а** [a], **ъ** [ɤ], **е** [ɛ], **и** [i], **о** [ɔ] and **у** [u] (see Figure 1). They are divided into three pairs of front, central and back. All of them keep their original pronunciation when bearing the stress, except in two special cases that will be explained below.

Figure 1 : Vowels in Bulgarian



However, as described in (Sabev, 2000), the pairs **а-ъ** and **о-у**, when not stressed, undergo a neutralization³ process (see Figure 2), which alternates vowels in two ways. The first type applies to all unstressed occurrences except the one immediately following the stress and the one syllable preceding it. It neutralizes [a] and [ɤ] to [ɐ] (e.g. *сатен* [sə'tɛn] SATIN and *къди* [kɐ'di] SMOKES) and [ɔ] and [u] to [o] (e.g. *понтон* [pɔntɔn] PONTON and *пуловер* [pɔlɔvɛr] PULLOVER). The second type applies to the other two syllables if present for a given word. It neutralizes the vowels to [ə] and [ɔ] which are closer to [ɤ] and [u] (e.g. *панталонът* [pəntə'lɔnət] trousers-the). These processes are the basic part of the transcription generation that we present in this paper.

Figure 2: а-ъ and о-у neutralization gradients



Stressed vowels are pronounced clearly and exactly as they are written (e.g. *мъгла* [mɛg'ɫa] MIST). However, there are exceptions which are derived by another rule. According to that rule, **а** and **я** in a final stressed syllable are pronounced as [ɤ] and [i̯ɤ] in the following cases:

- first person singular and third person plural in the i- and e-conjunction – *кося* [ko's'ɤ] MOW;
- masculine forms with a definite or indefinite article – *брега* [brɛ'gɤ] BANK-THE.

³ In (Scatton, 1984) it is called reduction, but the right term for it is neutralization because pairs of vowels are affected and not in the same way. Scatton also forgets to mention that **е** and **и** are neutralized only in colloquial speech.

There are also exceptions to the first case, where some verbs evade the rule and keep their original pronunciation.

2.4 Consonants

The phonological processes that occur with consonants are more numerous and somewhat more complex than the vowel rules discussed in the last section. There are 23 consonants in Bulgarian, including four affricates (see Table 1: Bulgarian consonants), but there are also a number of allophones (only two shown in Table 1 marked with asterisk) that appear on the surface level only in certain conditions.

	Labial		Coronal			Dorsal		
	bilabial	labiodental	dental	alveolar	palatoalveolar	retroflex	palatal	velar
nasal	m	m*	n					ŋ*
plosive	p b		t d				c ɟ	k g
fricative		f v	s z	ʃ ʒ			ç	x ɣ ⁴
approximant							j	
trill			r					
tap or flap								
lateral fricative								
lateral approximant			l				ʎ	
lateral flap								
affricate			ts dz		tʃ dʒ			

Table 1: Bulgarian consonants (* allophones of some phonemes)

Palatalization

When a consonant sign is followed by one of the letters **й**, **ь**, **ю** or **я** it always means it is palatalized. However, this is not a phonological process but simply an orthographical means of expression. Logically these are twenty-one phonemes, four of which are marked with different letters from the IPA ([c], [ɟ], [ç] and [ʎ]). The rest are marked as palatalized versions of the consonants as in **бял** ['bʲal] WHITE. There is also a case of palatalization that is not noted at all in the orthography. The velar plosive sounds [k] and [g] change to their palatal counterparts [c] and [ɟ] when following a front vowel, e.g. *кум* ['cit] WHALE and *църдан* [ɟɛr'dan] NECKLACE.

Final devoicing and regressive consonant assimilation

In Bulgarian as in other European languages (e.g. Slavic, Germanic, etc.) there is word-final consonant devoicing: **д** is devoiced and pronounced as [t] in *град* ['grat] TOWN. Regressive assimilation with regard to voicing means that the voicing of a consonant sequence depends on the voicing of the last consonant in it. It is important to note that when co-occurring final devoicing always has priority before regressive assimilation. However, these two rules are very close and have very similar outputs, so we suggest that it is possible for them to be rearranged in one rule. The following examples show how regressive assimilation is applied with and without final devoicing:

⁴ Appears only as a result of cross-boundary regressive assimilation.

1. *разписка* [ˈraspiske] RECEIPT
2. *сбивам* [ˈzbivəm] START A FIGHT
3. *крадец* [kreˈdɛt͡s] THIEF, *град* [ˈgrat] TOWN
4. *осветявам* [osvɛˈtʲavəm] ENLIGHTEN
5. *грозд* [ˈgrost] GRAPE CLUSTER

Example 1: Regressive assimilation

The first two words in Example 1 are the typical devoicing and voicing cases of regressive assimilation where the last consonant in a given sequence defines the voicing of the whole sequence. In 3 and 4 it is evident that there are some consonants that do not trigger assimilation. The first case shows that sonorant consonants are not involved in this rule, because neither **к** [k] nor **г** [g] changes their voicing in front of **п** [r]; the second case presents an exception in which a voiced consonant **в** [v] does not voice its preceding consonants. The reason for this exception is etymological: in this rule it kept its sonorant characteristics from Old Bulgarian. The last example shows a sequence of voiced consonants that have been devoiced. This happens because of the final devoicing which occurs before the regressive assimilation triggering devoicing regressive assimilation.

Regressive assimilation also occurs across word boundaries in almost the same way as within words. Typically in Bulgarian many words are pronounced together because of the high number of pro- and enclitics. The only difference from the in-word version of regressive assimilation is that across word boundaries the phoneme **в** [v] has voicing qualities, e.g. *от вода* [otvoˈda] FROM WATER. Another peculiar fact is that the voiced velar fricative [ɣ] occurs only across word boundaries (for example it appears in *изпих го* [izˈpiɣgo] DRANK-1P.SG. IT).

Consonant gemination simplification

Geminate phonemes are not typical for Bulgarian and when such consonants are produced by affixation they are simplified to long phonemes like in *уздан* [iˈzːidən] LAID (FOR WALL). It is important to notice that this rule applies after any devoicing rules i.e. regressive assimilation and thus words like *изселвам* [iˈsːɛlvəm] MIGRATE are affected by the gemination simplification, too.

Velarization

The nasal phoneme [n] in Bulgarian is velarized by a following velar plosive ([k] or [g]). It is a process that occurs not only in Bulgarian – it is very common in English, too, for example (*think* [ˈθiŋk]). It appears both in Bulgarian words like *тънко* [ˈtɤŋko] THIN and loan words like *танго* [tɛŋˈgo] TANGO.

The only lateral approximant in Bulgarian, **л** [l], has two allophones: [l] and [ɫ]. After front vowels the lateral approximant remains unchanged (as in *лига* [ˈliɡə] LEAGUE) and everywhere else (including word final position) it is velarized to [ɫ], e.g. *клон* [ˈkɫɔŋ] BRANCH, *кол* [ˈkɫɔ] STAKE and *залп* [ˈzɫɔp] VOLLEY (SIMULTANEOUS DISCHARGE).

N-deletion and neutralization

Nasal sounds [n] and [m] are neutralized to [ŋ] in front of labiodental consonants (which are the only fricatives in Bulgarian). Examples for this transformation are the words *информация* [iŋforˈmat͡sɪɛ] INFORMATION and *инвестиция* [iŋvɛˈstɪt͡sɪɛ] INVESTMENT. When the nasal [n] is preceded by a vowel and followed by a fricative consonant the n-deletion rule is applied. The rule states that [n] is deleted and the preceding vowel is nasalized and often even prolonged. This rule applies to Bulgarian words like *коњски* [ˈkɔːski] HORSE-LIKE and also loan words from languages that have the same or similar

rule, e.g. *брана* [ˈbr̩ɑːŋ] BRANCH (OF BUSINESS) from French. However, n-deletion is not always imported, for example final n-deletion in French is not transferred to Bulgarian, e.g. *Besaŋɔn* [bɛz̩ɑˈsɔ̃], but *Беза̀нсон* [bɛzɛnˈsɔn]. An example of short nasalization is the word *инхалация* [ɪxɛˈtɑf̩ɪɛ] INHALATION.

Since the context of nasal neutralization is a subset of the context of n-deletion only precedence decides which one is applied. So in fact, n-deletion does not occur before labiodental fricatives because it is overruled by nasal neutralization.

Voiced labialized velar approximant

Due to the great number of English loan words in Bulgarian, the language has acquired the phoneme [w] used only for those foreign words that require it, e.g. *уиски* [ˈwiski] WHISKEY and *уикенд* [ˈwɪcɛnt] WEEKEND. Unfortunately, prediction of this phoneme's occurrence is not possible without establishing its foreign character, thus to generate its phonetic transcription one needs some lexically related information.

3 Stuttgart Finite State Transducer (SFST)

To summarize the phonological theory regarding Bulgarian pronunciation we can emphasize two conclusions that will be important for the future implementation. First, most rules are only context and/or stress dependent, which allows any transcription generator to be fairly lexis independent. Second, most of the changes from orthographical to phonological representation are not bidirectional, which blocks one of the strong points of finite state methods – generation in both directions. Nevertheless, we find finite state methods to be the best means to generating phonetic transcription. The FST tools that we used for the implementation are called Stuttgart Finite State Toolkit. This section briefly presents them and lays some basic guidelines for the language SFST-PL.

The Stuttgart Finite State Transducer tools are based on the finite state transducer technology and are aimed at facilitating the development of applications dealing with morphological analysis. Besides the programming language SFST-PL and an FST compiler, there are also tools for printing, comparing and applying finite state transducers, all based on C++ finite state transducer libraries.

Finite state transducers, as a special case of finite state automata, have two tapes – for input and output. Every symbol on the input tape is mapped to a symbol on the output tape. Often transducers are built so that when input and output tapes are swapped, they would perform the reversed process. This bidirectional usage is implemented by the FST application tools in the analysis and generation modes. These modes represent the two possible input types that are involved in phonology and morphology analyzing applications. By contrast the implementation described in this paper does not employ bidirectional analysis for a number of reasons. Even though most processes are bidirectional, some of them do not allow deterministic answers when generating graphemes from phonemes. Vowel neutralization, for example, would multiply the number of combinations by two for each unstressed neutralized vowel in a word. Disambiguating in this case is only possible using dictionary check-up, which we do not want to involve in this project.

SFST-PL has a rather idiosyncratic syntax, which might be confusing for someone with an XFST background. Therefore we recommend (Schmid, SFST Manual) as a starting point for working with

SFST, but we also offer short explanations of all operators used in **our** implementation through a small example program.⁵

The most fundamental property of a finite state transducer – the one making it a special case of FSA – is the presence of the output tape and the transformations that take place between it and the input tape. The operator that executes this transformation is called mapping operator and is marked with semicolon in SFST-PL (e.g. *a:b* is a transducer that accepts as input *a* and outputs *b*). For the states where no transformation takes place (i.e. *a:a*) an alternative syntax has been provided – using only the letter/string that has to be matched (i.e. mapped to itself). As a whole the SFST-PL syntax is very similar to Perl and scripting languages similar to it. The operators Kleene star (*) and plus (+), optionality (?), matching dot (.), union (|), set ([...]) and negated set ([^...]) have exactly the same syntax and meaning as in Perl. The only practical difference there is that in SFST the matching dot and the negated set are applied in as if the alphabet defined in this program is semantically the ‘world’ whereas in Perl the ‘world’ is not restricted by the regular expressions. The alphabet is a very peculiar feature of SFST, it is obligatory for transducers that use matching operator or negated set.

```
ALPHABET=[a-zäöüA-Z]
$V$ = [aeyuioäöü] | [auo] e % vowels
$C$ = [^aeyuioäöü] % consonants
($C$ $V$ $C$)+ \! ? \ % accepting only CVC syllables
| | \ % composition operator
({ae}:{ä} | {oe}:{ö} | {ue}:{ü}) ^-> $C$__$C$ % replacing the umlauts
```

Example 2: SFST program

Most of the techniques used in our implementation are grouped in the sample program shown in Example 2. The FST accepts only words built of CVC syllables with the exception of the alternative umlaut representation (*ae*, *oe* and *ue*), which is counted as one vowel and then mapped to their original umlaut letters (*ä*, *ö*, and *ü*). The first two lines following the alphabet definition are variable definitions – one for consonants and one for vowels. Consonants are defined as non-vowel set just the way one would do it in Perl, but the vowels are defined in a little more complicated way because of the umlauts. Two acceptable cases are defined: single vowel including umlauts with diacritics and two vowels forming an umlaut. The first case is implemented as a set of vowels and the second one as a set of the vowels that receive umlaut concatenated with an *e*. The letter solution implements FST concatenation, which is presented as stringing FSTs together optionally separated by white spaces. White spaces are optional, because they are ignored by the compiler except in the case when they are escaped with a back slash. In a similar way newline characters (which normally serve as line ending markers) are escaped to allow multiline expression and better formatting.

The last three lines of the program are in fact one line consisting of two composed FSTs (| | is the sign for composition), which carry out the two tasks that were listed above. The task of accepting only CVC structured words is relatively easy to complete after defining consonant and vowel variables above. This way they only need to be grouped (with parentheses) and assigned a Kleene plus to match at least one syllable. There is also an optional exclamation mark (that has to be escaped too because it is the sign for negation, which we, in fact, never used) following the word. The second FST takes care of replacing the alternative umlaut spelling. Here we used a context matching rule that

⁵ The rest of this Section may be skipped if the reader is familiar with SFST.

enables us to apply the replacing transducer only in the environment between two consonants. Although this is the only kind of replacement rule that is used in our implementation, it needs to be said that it is an upward replacement rule, which means that it takes analysis as input and outputs generation, just as our whole implementation does. There are three more types of replacement rules – downward, leftward and rightward – the first of them is the opposite of the upward rule and the two others are the two cases when left and right context are on different levels (e.g. left on analysis, right on generation). The replacement itself is implemented with the curly braces that serve for replacement of strings with unequal length.

Clearly the version of the program in Example 2 is not optimized: the first sequence of umlaut vowels is matched and then replaced by a special rule. We can improve that by uniting the FST that is embedded in the replacement rule with the vowel variable definition, i.e. making the vowel variable to match and replace umlaut letter combinations at the same time (see Example 3).

```
ALPHABET=[a-zääöüA-Z]
$V$ = [aeyuioääöü] | ({ae}:{ä} | {oe}:{ö} | {ue}:{ü}) % also replacing umlauts
$C$ = [^aeyuioääöü]
($C$ $V$ $C$)+ \!?
```

Example 3: Optimized SFST program

4 Phonetic transcription generator

The two conclusions made at the beginning of Section 3 define the foundations of the phonetic transcription generator and its tasks. In a nutshell the generator has to be able to transform any Bulgarian orthographical word representation into the same word's transcription without using any kind of dictionary. Because of the vital role of word stress in Bulgarian phonological rules, the only extra requirement that such a generator needs is a stress marking in each word. The only exception to the context and stress requirements – stressed vowel change – can be recognized only by using a morphological analyzer for recognizing the forms falling under this exception. Nevertheless, once recognized, the transcription of such forms is not harder to generate than any other word or word form.

Apart from being considered as a multitude of simple rules, phonology also needs to be seen as a whole working system and rule precedence and co-occurrence need to be taken in mind. Fortunately there are only two pairs and one triplet of rules, whose order of occurrence need to be considered. Final devoicing needs to precede regressive assimilation (we already suggested that these two rules might also be seen as one rather than two rules), which needs to precede geminate simplification; nasal neutralization precedes (but in fact overrules) n-deletion. Other than these rules there are no linguistic requirements for the rule application order. However, there are some technical reasons for the non-random order of rules, which will be brought up again along the rule explanations further in this section.

The implementation discussion is divided into three parts depending on the goal of the transducers that were implemented. Apart from implementing all phonological rules in one system, there needs to be a piece of code to ensure the proper form of the input and solve the matching end of string

issue (this will be discussed in Section 4.1). The two vowel-changing rules are explained in Section 4.2 followed by all consonant related rules in Section 4.3.

4.1 Preprocessing

Preparation is an important process which guarantees that the input has been unified and some exceptions have been handled. Its first task is to string all words of the input together to enable the generation of a long cross-boundary transcription. This operation would have been rather trivial if not for the special case of **в** changing its voicing properties across boundaries. A marker for all cases where **в** is voiced at the beginning of a word (as in *врата* [vrə'ta] DOOR and not as in *втора* [ˈftori] SECOND-M) has to replace the **в** in that word with **w**. Since **w** is not used for another phoneme, at least in this implementation, it can be treated as one of the ordinary voiced consonants during the assimilation process.

Since a stress marker is vital for transcription generation and since Bulgarian stress is completely unpredictable, stress markers have to be part of any input word. This, however, poses the question of how to express that marker. The standard implemented in this paper is a single straight quote preceding the stressed vowel. Thus, during the preparation process the implementation rewrites all stressed vowels marked through combining characters or pre-combined stressed characters to the single quote standard that is used for the implementation. For the same unification reasons capital letters are rewritten as lower case with a very simple transducer.

There are two exceptions handled during the preparation process: one letter words and stressed vowel alternation. One letter words in general are much more often vowels than consonants. In Bulgarian there are only two single consonant words (*с* [ˈsɨs] WITH and *в* [ˈvɨf] IN) and they both fall under an exception that practically equalizes them to their longer forms. These two prepositions are also pro-clitics (e.g. *с мен* [ˈsmɛn] WITH ME) and are generally pronounced together with the word following them, but if the joint pronunciation would result in a geminate consonant, the two prepositions are replaced by their longer forms (i.e. *свс сила* [sɨˈs:ilə] WITH FORCE, *ввв вас* [vɨˈv:as] IN YOU). The same pronunciation is used in the cases where the prepositions are pronounced alone. Thus during input preparation they need to be converted into their longer forms, if they are the only input word. The other exception resolved through the preparation process is the already mentioned case of stressed vowel alternation. Since it is a form dependent phenomenon, this implementation cannot deal with recognizing it, but is able to execute the sound alternations if given a signal to do it. This signal is a percent sign in front of words and its execution is implemented using replacement rules like the ones that were demonstrated in Section 3.

Finally, a very important technical issue is resolved during preparation. Unfortunately SFST does not offer any precompiled way or any operator to match the end of string (at least from what is present in its documentation). The way around this inconvenience is very simple – inserting a border marker (a hash sign) at the end of each input string that will be deleted later on when the border dependent rules are carried out. It is important that this happens at the end of the preparation when the input has been already prepared, so other preparation processes do not get encumbered.

4.2 Vowel processing

Only two phonological rules that were implemented deal exclusively with vowels. They are both vowel alternation rules and their order of application does matter from a technical point of view.

Technically it is easier to replace all unstressed vowels with the first type (the one that is applied to all but two unstressed syllables) of vowel neutralization and then to apply the other only to the two particular syllables it concerns. Therefore the general neutralization rule is executed first and then the one concerning only two syllables.

The first type of vowel neutralization rule is the first rule that starts mapping the input to an actual transcription, so aside from carrying out its phonological tasks, it also needs to replace all stressed and/or unchanged vowels with their respective IPA symbols. The whole task is executed through a single transducer that resembles the optimized version of the sample SFST-PL umlaut replacement in Example 3. The strings are processed by matching a vowel or a non-vowel, where non-vowels are mapped to themselves and vowels are mapped to their respective stressed, unstressed and alternated variations in a way similar to the umlaut replacement in Example 3.

The second type of vowel neutralization affects phonemes before and after the stress position. To make the implementation simpler the rule is split in two components with their own transducers: pre-stress and post-stress. Both components, however, use the same strategy: construct two patterns – one containing secondary vowel alternation and one not. For the pre-stress pattern the number of syllables preceding stress is defining, so it covers words with more than one syllable preceding the stress and all others (words with at most one syllable preceding the stress) are left to the no vowel alternation pattern. The post-stress alternation pattern accepts words with at least one syllable following the stress and its opposite pattern accepts only words with stress in the final syllable. The syllables are constructed by combining three variables representing the three kinds of phonemes relevant for this process: consonants, stressed vowels and unstressed/neutralized vowels.

4.3 Consonant processing

There are more phonological rules involving consonants than vowels. But even with their greater number they seem to be mostly independent of each other. The order of the rules application is designed almost completely by convenience of the implementation and less by linguistic requirements.

Regressive assimilation

As said before regressive assimilation and final devoicing are very close and thus implementing them together as one rule is a convenient solution. The finite state transducer that realizes these rules is the most elaborate piece of code in the whole transcription generator. In addition to fulfilling its linguistic objectives it also replaces all consonant phonemes with their respective IPA representations. Its highest level consists of the union of five patterns. The first pattern pair is simple – a vowel phoneme pattern (including all vowels) and a neutral phoneme pattern (i.e. consonants that are neither voiced nor devoiced: laterals, trills and approximants). The vowel pattern only matches the IPA vowels because they have already been replaced in the preceding vowel alternation rules and the neutral pattern replaces the phonemes assigned to it with their respective IPA representations. The second pair of patterns is made of the two opposite assimilation processes: voicing and devoicing regressive assimilation. Together they make use of all other patterns defined for the transducer including the ones used on the highest level of the transducer. However, before explaining the functioning of the two assimilation patterns, the task of the fifth pattern has to be outlined. Its goal is to handle the ambivalent behavior of **в** [v]. Its application in the assimilation patterns is as one of the anchor patterns added to ensure the deterministic output of the transducer,

which, as explained in Section 2.4, is dependent on the final consonant in a given sequence. Anchor patterns (vowels, neutrals and **в** [v]) are the final match in the in-word assimilation patterns, which are not a part of the assimilation rules, but only help to technically achieve a deterministic mapping. The bulk of the consonant sequences are matched in mirror-like way using the union of two pairs of sub-patterns for the two assimilation patterns – devoiced and devoicing patterns for devoicing assimilation and voiced and voicing patterns for voicing assimilation. The voiced and devoiced patterns are merely mapping voiced/devoiced consonants to their IPA representations, while the voicing and devoicing patterns map voiced and devoiced consonants to their respective opposite (with regard to voicing) IPA representations. The assimilation triggering patterns are not as trivial as they may seem. On the one hand not all voiced consonants trigger voicing and on the other devoicing is not triggered only by devoiced consonants. These difficulties are the reasons for creating a special voicing triggering pattern (without **в** [v]) and also including the end of string sign as a trigger for devoicing assimilation. Finally, the matching of single consonants is incorporated in the assimilation patterns, but because of its ambivalent properties **в** [v] could not be completely included there and needed to be matched by the fifth high level pattern that we already mentioned.

Other consonant phenomena

The rest of the phonological rules implementations are not very complicated and all use the same replacement rule strategy with minor modifications. Thus, apart from the first example, explanations in more detail are provided only for those which show some small differences from the general case.

The gemination simplification phenomenon seems like a simple enough operation given that one would have the tools available in Perl, for example. However, SFST-PL does not offer any mechanism to match a defined number of the same character or character sequence. Thus, replacing a geminate consonant with a long consonant requires explicitly stating all replacement rules (e.g. Example 4) for all possible (consonant) phonemes and then piping them together in one grouping transducer. It is also important to do this after regressive assimilation is applied, because sometimes it also produces extra geminate phonemes that are orthographically represented with different letters.

```
% Gemination simplification rule
ALPHABET = [a-zɟʃrɐɛɔɣdʒt˘:]
{ʃʃ} : {ʃ} ^-> (.__)
```

Example 4 : Replacing geminate voiced velar fricative

Palatalization of consonants occurs in two different cases – any consonant preceding the approximant **ɟ** or velar consonant preceding a front vowel (i.e. **ɪ** and **ɛ**). They are implemented as two different transducers, which will be described in turn below. Both cases are simple in their nature, but some more attention has to be paid to the technical side of their implementation. The velar consonants are handled with a very simple replacing rule transducer reflecting exactly the phonological rule that imparts the phenomenon. The more general occurrence of palatalization, however, requires some more consideration of its technical side. Some palatalized phonemes are represented with a following **ɟ**, but some others (velar consonants and **ŋ**) use their respective palatal phoneme signs (e.g. **ɟ**, **ʎ** and **ç**). This makes the replacing rule parameters a bit more explicit, but certainly not more complicated with regard to the linguistic theory.

N-deletion, the two cases of velarization and the nasal neutralization are handled perfectly by the simplest replacement rule scenario without any type of technical modifications. Thus they will not be further elaborated on here.

5 Discussion

The transcription generator described in this paper gave perfect results when tested. All implemented phonological rules were tested in parallel and found functional, even when co-existing. Its shortcomings appear, as anticipated in the introduction, only when alternations have lexical or morphosyntactic triggers and conditions. Furthermore, some of these issues were addressed in Section 4.1 where part of speech dependent words could be handled (using the percent sign) if their status has been established beforehand. This means that a considerable part of the exceptions could be handled with the help of a morphological analyzer. Unfortunately there is nothing to be done about transcribing foreign words similarly in an automated way. When putting things into perspective, this project may be expanded in exactly the two directions that we just addressed: preprocessing with a morphological analyzer and foreign word list filtering.

Unfortunately, the transcription generator is absolutely dependent on word stress, which in Bulgarian is completely unpredictable and commonly not marked. This fact sets back some of the most valuable applications of this technology, because in the end it is somewhat lexically dependent given its need of marked stress in its input. Thus a possible application that works out of the box still remains only the area of computer-aided language learning, where word stress is always marked anyway.

In conclusion, it has to be noted that the phonetic transcription generator works excellent as long as it receives stressed input and even though it does not have many real applications at this stage of development, it also creates a good possibility for real-world problem solving when combined with a morphological analyzer.

6 Bibliography

- Boyadzhiev, T., & Tilkov, D. (1997). *Fonetika na balgarskiya knizhoven ezik*. Veliko Tarnovo: Абарар.
- Boyadzhiev, T., Kutsarov, I., & Penchev, Y. (1998). *Savremen en balgarski ezik*. Sofia: "Петър Берон".
- Pashov, P. (2004). *Balgarska gramatika*. Sofia: Хермес.
- Pashov, P., & Parvev, H. (2002). *Pravogovoren i pravopisen rechnik na balgarskiya ezik*. Sofia, Bulgaria: Университетско издателство "Св. Климент Охридски".
- Sabev, M. (2000). *The Sound System of Standard Bulgarian*. Retrieved 06 25, 2009, from University of Reading: http://www.personal.rdg.ac.uk/~llsroach/phon2/b_phon/b_phon.htm
- Scatton, E. (1984). *A Reference Grammar of Modern Bulgarian*. Columbus, Ohio: Slavica Publ.
- Schmid, H. (n.d.). *Developing Computational Morphologies With the SFST Tools*. Stuttgart.

Schmid, H. (n.d.). SFST Manual. Stuttgart.

Ternes, E., & Vladimirova-Buhtz, T. (1999). Bulgarian. In I. P. Association, *Handbook of the International Phonetic Association* (pp. 55-57). Cambridge: Cambridge University Press.