Правительство Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
Национальный исследовательский университет
«Высшая школа экономики»

Факультет гуманитарных наук
Образовательная программа
«Фундаментальная и компьютерная лингвистика»

Картина Элен Геннадьевич

Правиловый морфологический парсер для шугнанского языка: существительные, глаголы и прилагательные

Выпускная квалификационная работа студента 4 курса бакалавриата группы БКЛ211

Академический руководитель образовательной программы
канд. филологических наук, доц.
Ю.А. Ландер
____
«_____» ____________ 2025 г.

Научный руководитель
канд. филологических наук, доц.
Г.А. Мороз
____

Научный консультант
Стажёр-исследователь
М.Г. Мельченко
____

Москва 2025

Abstract

In this work I present a rule-based morphological analysis tool based on Helsinki Finite-State Technology (HFST) for the Shughni language (ISO: sgh; glottocode: shug1248), a language of the Iranian branch of the Indo-European family, a member of 'Pamiri' areal language group. While one existing rule-based parser exists for Shughni (Melenchenko, 2021), it does not utilize finite-state transducer technology. This work proposes the first HFST-based morphological parser implementation for Shughni, offering the advantages of this well-established framework for morphological analysis. The parser is presented in two variations: a morphological parser that breaks each word-form into stem and morphemes and assigns morphological tags to each one of them; a morphological generator that outputs word-forms taking a stem and morphological tags as an input. **TODO: prev sentence is questionable** This is a continuation my previous work, where nouns, pronouns, prepositions and numerals were implemented (Osorgin, 2024). This project covers **TODO: what**

TODO: Review abstract after finishing the work

Contents

1 Introduction

1.1 Shughni

The Shughni language (ISO: sgh; Glottolog: shug1248) is a language of the Iranian branch of the Indo-European family (Plungian, 2022, p. 12). As of June 1997, it was estimated to be spoken by approximately 100,000 people (Edelman & Yusufbekov, 1999, p. 225) in the territories of Tajikistan and Afghanistan. Both countries have a subregion where Shughni is the most widely spoken native language. The Shughni-speaking subregion of Tajikistan is called ‘Shughnon’ and it belongs the to ‘Gorno-Badakhshan Autonomus’ province. In Afghanistan, the Shughni-speaking region is called ‘Shughnan’ and it lies within the territory of ‘Badakhshan’ province (Parker, 2023, p. 2). Shughni belongs to ‘Pamiri’ areal language group, which is spoken along the Panj river in Pamir Mountains area.

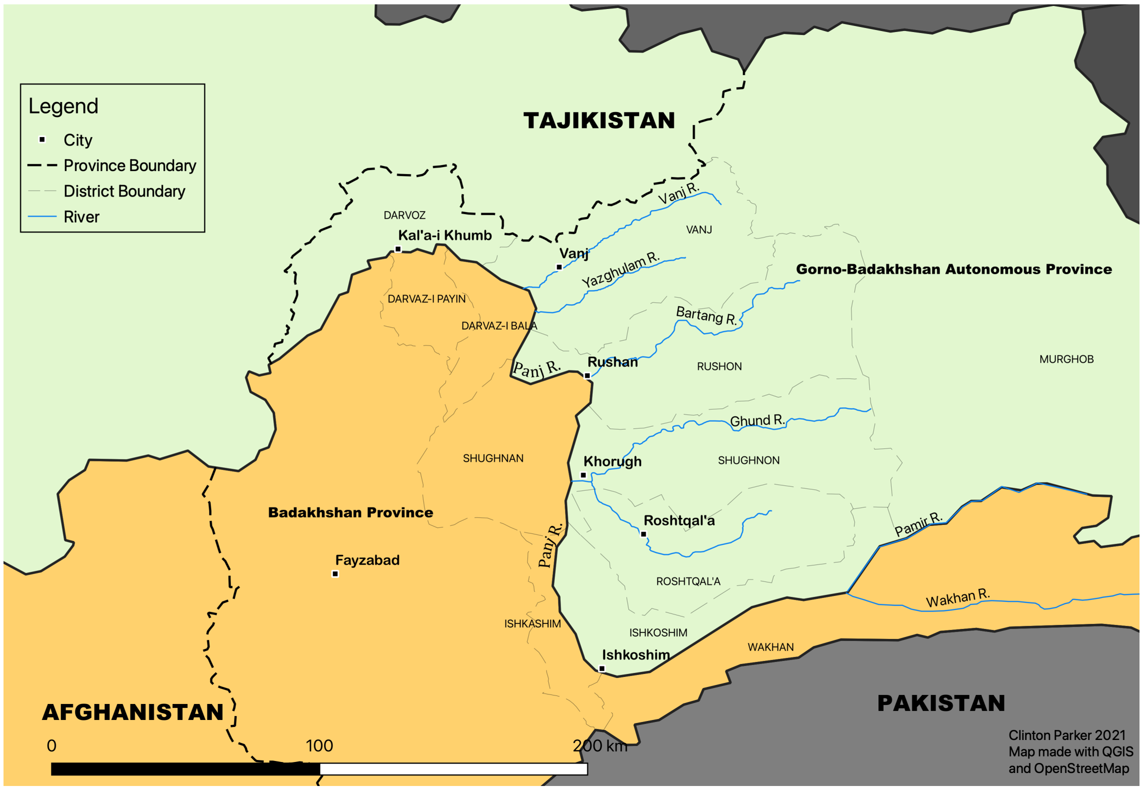

Figure 1: Mountainous Badakhshan Autonomous Province of Tajikistan and Badakhshan Province of Afghanistan, (Parker, 2023, Fig 1.1)

There are three alphabets for Shughni that were derived from Cyrillic, Arabic and Latin scripts. Geographically the usage of said scripts correspond to the dominant script of each country where Shughni is spoken. In Tajikistan both official languages (Tajik and Russian) use Cyrillic script, so does Shughni on territory of Tajikistan. In Afghanistan Arabic script is used in Shughni, matching official languages (Pashto and Dari).

Latin script was developed and used in Tajikistan in 1930s (Edelman & Yusufbekov, 1999, p. 226)

(Edelman & Dodykhudoeva, 2009, p. 788), but according to Edelman and Yusufbekov (1999) was not widely adapted. Later around 1980s a Cyrillic script gained popularity in Tajikistan, having some poetic literature and school materials based on Tajik's alphabet, which is Cyrillic (Edelman & Yusufbekov, 1999). Today, Latin script is mostly used by researchers in scientific works.

The morphological parser developed in this work is based on materials that focus on Shughni spoken in Tajikistan. All the base lexicon is Cyrillic and comes from dictionaries that cover Shughni in 'Gorno-Badakhshan Autonomus Province'. Latin script is supported with the help of transliteration.

1.2 Morphology modeling

Today there are two general approaches to the task of morphology modeling. The deep learning (DL) approach and the rule-based approach.

The DL approach today typically makes use of training transformer models like `BERT` (Devlin et al., 2019) on vast amounts of marked-up data. This task becomes challenging, considering that Shughni is a low-resource language, meaning it lacks digital textual data. Although, DL approach was not utilized in this work, some existing DL approaches for low-resource languages are covered in section 2.1.

With the rule-based approach, morphological model is being built by writing grammar rules using some formalism language and by listing base lexicon. In this work, rule-based approach was utilized, as it does not depend on the amount of available marked-up data as the DL approach does. It requires lexicons and morphological grammar descriptions, which exist for Shughni and which are discussed in Section 3.

2 Existing methods

2.1 Machine learning methods

There are a variety of LLM (Large language model) architectures that were applied to the task of language modeling. One significant example is `LSTM` (Long short-term memory) model, that was introduced by Hochreiter and Schmidhuber (1997). `LSTM` is a variation of `RNN` (Recurrent neural network), and it was widely applied to language modeling, including morphology modeling. Another more recent significant example is the transformer architecture presented by Vaswani et al. (2017), off which two years later `BERT` model was based (Devlin et al., 2019).

One of the biggest downsides of ML methods is that its quality depends on training data quantity, which makes it challenging to apply to low-resourse languages such as Shughni. However, with introduction of LLMs this problem was shown to be solvable, for example, as shown by developers of `UDify` model (Kondratyuk & Straka, 2019), which is a `BERT`-based model. In their work authors show, that their model pretrained on a large corpus of 104 languages can be fine-tuned on very little amounts of other languages' data and still show decent results. For an example, they report that for

Belarusian, `UDify` model achieved $UFeats = 89.36\%$ (accuracy of tagging Universal Features) after training on only 261 sentences from 'Belarusian HSE' Universal Dependencies treebank (Kondratyuk & Straka, 2019, Table 7).

However, working with LLM models is a highly resource-demanding task. The authors of `UDify` state, that the fine-tuning of their model for a new language would require at least 16 Gigabytes of RAM and at least 12 Gigabytes of GPU video memory, and the training process would take at least 20 days depending on the GPU model. While a deep learning approach would be interesting to explore, such computational resources are not available for this project. The neural approach is not the main target of this work and is implemented.

2.2 Rule-based methods

2.2.1 Finite-state transducers

The Rule-based approach historically is usually applied with the help of Finite-state transducers (FST), which is a variation of Finite-state machine, a mathematical abstract computational model. Following the terminology of Turing machines (Turing, 1937), a FST has two tapes: the input tape and the output tape. At any point it can read a next symbol from the input tape and then write a symbol to the output tape. Once a symbol was read from the input tape, it can not be read again, as the input tape shifts one symbol forward.

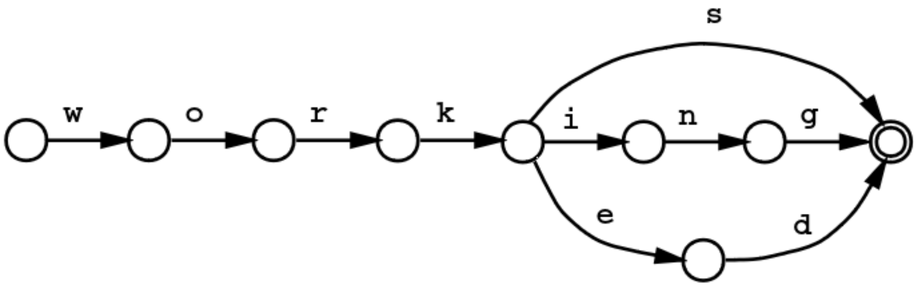

Figure 2: An example of FST with a single initial state (most left node) and a single final state (most right node) for a language where only three words exist: *works*, *working* and *worked*. The word *worker*, for example will not be recognized as a valid word by this FST, since there is no 'd' transition at state *worke*. The only way from *worke* state is via 'd' transition, which corresponds to the *worked* word. (Beesley & Karttunen, 2002)

The inner structure of FST can be illustrated as a directed graph with a set of all *states* (represented by graph's nodes), a set of *transitions* (represented by graph's edges), a set of *initial states* (a subset of all the states, these are states where FST can start reading from the input tape) and a set of *final states* (a subset of all the states, these are the states where FST can stop reading from the input tape). A simplified FST is shown on Figure 2. The letters above the graph's edges denote *transition* rules, for an example *transition* 'w' means 'read *w* from the input tape THEN write *w* to the output tape'.

While working, FST will only make transitions that are possible from the current state. If there are no valid transitions then FST fails to process the input, and the input is considered to be impossible in the current language model. The measure of the amount of language’s grammatical wordforms that successfully pass through the FST from an *initial state* to a *final state* will be called *Coverage* from now and on. The measure of the amount of language’s ungrammatical wordforms that successfully pass through the FST from an *initial state* to a *final state* will be called *Overgeneration* from now and on. The ideal FST model of a language has a maximized 100% *Coverage* and a zero *Overgeneretion*.

The model from the Figure 2 works effectively as a wordform paradigm dictionary, echoing back input wordforms that are grammatical and failing to output the whole ungrammatical wordforms. Now we can slightly adjust the transition rules in our example to make a morphological analysis tool that can bee seen on the Figure 3. The notation of the *transition* ‘w:w’ dictates to read the left symbol from the input tape and write the right symbol to the output tape. If the spot on the right side is left empty, it means ‘write nothing to the output tape’. An important note to remember is that FST can output only one symbol to the output while making a single transition. In this example *‘<inf>’*, *‘<pst>’* and *‘<prs><2sg>’* are ‘multichar’ symbols, meaning they are treated as three individual symbols by a FST, it will be covered in more detail in the Section 4.

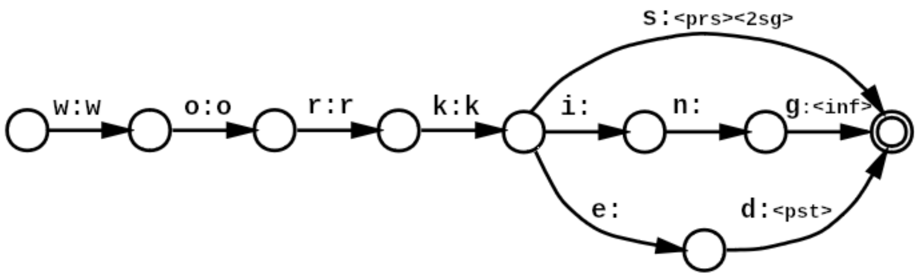

Figure 3: A modified version of Figure 2 which takes as input *works*, *working*, *worked* and outputs *work<prs><2sg>*, *work<inf>*, *work<pst>* respectively.

2.2.2 FST formalisms

By FST formalism I mean a human-readable formal language that can be compiled into a static FST, or from what a FST behavior can be emulated in runtime. A FST formalism usually includes a way to list lexicons and/or list lexicon combination rules and/or list phonological rules.

One of the first major fundamental advances was when (Koskenniemi, 1983) created a model, which introduced a FST formalism named Two-level morphology (TWOL) for describing morphological and morphonological paradigms. Its novice was in the addition of the phonology level of rules, which made it much easier and intuitive to implement cases like *‘eye(-s)’/‘box(-es)’.* This model was capable of word-form recognition and production, but it was not yet compilable into static FSTs, it was working at runtime and was known for being slow. Then Karttunen et al. (1987) at Xerox Research Center developed a Two-level rule Compiler (`twolc`), which compiled TWOL rules into static FSTs. Later a separate compiler for lexicon definitions was introduced named `lexc` (**Lex**icon **C**ompiler)

(Karttunen, 1993), it came with its own formalism language for describing lexicon and morphotactics. The standard approach to modeling a language at that point was using `lexc` to describe lexicon and morphology and `twolc` to describe morphonology, which stayed almost the same to this day.

One of the latest released tools was `lexd` lexicon compiler (Swanson & Howell, 2021). It is presented as a `lexc` alternative and is claimed to be much faster in the compilation time. It also introduced a tag system, which allows FST developers to specify how diffetent lexicons combine with each other more precisely. **TODO: anything else?**

2.2.3 Helsinki finite-state technology

Helsinki finite-state technology (HFST) is a set of tools for creating and working with languages' morphology models in form of transducers (Lindén et al., 2009). It includes implementations of both `hfst-lexc` and `hfst-twolc` compilers, as well as command line interface commands for mathematical and other miscellaneous operations with transducers like FST combination and format conversion. Also, it comes with a file format `.hfst` designed to store compiled FSTs.

HFST is widely applied when it comes to creating rule-based morphological models. Some of the latest examples of HFST-based morphological tools are: morphological parser for the Tamil language by Sarveswaran et al. (2021), a morphological transducer for Kyrgyz by Washington et al. (2012), a morphological parser for Andi by Buntyakova (2023) and a morphological parser for the Chamalal language by Budilova (2023).

2.3 Existing morphology models for Shughni

At this time only one morphological parser exists for Shughni. It was developed by Melenchenko (2021) and was later included in 'Digital Resources for the Shughni Language' project (Makarov et al., 2022). It is a rule-based parser implemented in Python which shows good coverage and accuracy results. The main difference from the parser presented in this work is that Melenchenko's parser is not based on FST technology.

3 Data

3.1 Grammar descriptions

Several Shughni grammar descriptions were written throughout the years, starting from basic grammar description done by D. L. Ivanov (Salemann, 1895, pp. 274–281). An important mention is a work by Karamshoev (1963), which was the most detailed Shughni grammar description of its time. Latest significant works were 'Shughni language' (Edelman & Yusufbekov, 1999, pp. 225–242), 'Comparative Grammar of Eastern Iranian Languages' (Edelman, 2009) and 'A grammar of the Shughhi language' by Parker (2023), which is the biggest existing grammar, the most detailed and the most recent one.

For this work the main reference for compiling Shughni grammar rules was the work of Parker (2023). It was picked as it is the most recent, the most detailed and the biggest one. Other grammar description works were used too, but only as a secondary reference. The second most used grammar description was a work by Edelman and Yusufbekov (1999).

3.2 Dictionaries

There are two main dictionaries of the Shughni language: one by Zarubin (1960) and one by Karamshoev (1988–1999), both are written using Cyrillic script and include Russian translations. Some early dictionaries are 'Brief grammar and dictionary of Shughni' (Tumanovich, 1906), that is also using Cyrillic and translates to Russian, and 'Shughni dictionary by D. L. Ivanov' (Salemann, 1895), that translates to Russian but uses Arabic script alongside Cyrillic transcriptions for Shughhi word-forms.

An important lexical source for this work was the 'Digital Resources for the Shughni Language' project (Makarov et al., 2022). As a part of their work, authors compiled a digital dictionary for Shughni, where they digitalized both major Shughni dictionaries by Karamshoev (1988–1999) and Zarubin (1960). The digital dictionary is available at their website via a web-interface, but I was given access by the authors to a copy of the underlying database, which simplified the process of exporting lexicons for this project. All the lexicons for FST compilation were taken from their database.

3.3 Text corpora

I was given access to unpublished native texts that were gathered during HSE expeditions to Tajikistan in 2019-2024. It is not a large corpus of texts. It consists of a translation of 'The Gospel of Luke' in Shughni, of a 'Pear Story' text, which is a spoken text that was written down from a retelling of the 'Pear Story' movie during an expedition, and it also includes a group of miscellaneous untitled small texts. Texts size can be seen on Table 1

Text name	**Total tokens**	**Unique tokens**
'The Gospel of Luke'	2978	1001
'Pear Story'	1117	438
Miscellaneous texts	164	106
All texts	4259	1393

Table 1: A list of native Shughni texts and their sizes gathered during HSE expeditions to Tajikistan in 2019-2024

The database provided by Makarov et al. (2022) also contained a lot of different useful data parsed from dictionaries including dictionary entries' usage examples. Such data is not as valuable as native texts, as sometimes it might not come from a native speaker but from a researcher. I would argue that for *Coverage* evaluation it might still be quite useful.

From materials of HSE expeditions to Tajikistan I also acquired manually glossed texts in `.eaf` (ELAN) format. These texts were utilized for *Accuracy* evaluation, which will be discussed in Section 4.9 along with *Coverage*. A full list of text sources can be seen on Table 2.

Text name	**Total tokens**	**Unique tokens**	**Native**	**Glossed**
Dictionary examples	164 225	29 013	Uncertain	No
'The Gospel of Luke'	2 978	1 001	Yes	No
'Pear Story'	1 117	438	Yes	No
Miscellaneous texts	164	106	Yes	No
'The Gospel of Luke'	2 942	635	Yes	Yes
'Pear Story'	228	83	Yes	Yes
'Mama'	267	123	Yes	Yes

Table 2: A list of all available digital textual data

4 Methods

4.1 Input and output format

Every FST converts between two types of strings: wordforms (e.g. '`дарйойен`' ≈ 'rivers') and glossed strings (e.g. '`дарйо<n>><pl>`' = '`дарйо.N-PL`'). Glossed strings format is based on Apertium format **TODO: citation needed?**, which is a standardized format for HFST-based transducers. The choice of this format is motivated by the fact that this is the standard, and I want to keep it consistent with existing tools and practices in the field. Important keys of the format are:

- Grammatical tags are enclosed with angular brackets and are lowercase.

  `stone.V` = `stone<v>`

- Part of speech tags (POS) are obligatory and stand next to the stem or lemma.

  `stone.SG` = `flood<n><sg>`

- Different morpheme tags are separated with a single right angular bracket '>'.

  `stone-PL` = `stone<n>><pl>`

- Multiple stems in a single word is possible.

  `lemma1<adj>><morph>>lemma2<adj>`

The Shughni morphological parser in this work is presented in a variety of input and output formats. A full list of `.hfst` files with their formats of input and output is shown in Table 3. Motivation for this many variations of the morphological parser was support of different formats. Parser comes with four format variables:

- FST directionality: analyzer or generator. This simply shows the direction of a FST, analyzers take wordforms as input and return glossed strings, generators take glossed strings and return wordforms

- Glossed string stem glosses: Shughni stems or Russian lemmas. Notated in file names as `stem` and `rulem` respectively.
  `дарйо<n>><pl>` (Shughni stem); `река<n>><pl>` (Russian lemma)

- Wordform morpheme borders: plain word or morphemes are separated with ('>') delimeter symbol. Notated in file names as `word` and `morph` respectively.
  `дарйойен` (plain word); `дарйо>йен` (morphemes separated)

- Wordform script: Latin or Cyrillic.
  `дарйойен` (Latin); `daryoyen` (Cyrillic)

Transducer file name	**Input example**	**Output example**
sgh_gen_stem_morph_cyr.hfst	дарйо<n>><pl>	дарйо>йен
sgh_gen_stem_word_cyr.hfst	дарйо<n>><pl>	дарйойен
sgh_gen_rulem_morph_cyr.hfst	река<n>><pl>	дарйо>йен
sgh_gen_rulem_word_cyr.hfst	река<n>><pl>	дарйойен
sgh_gen_stem_morph_lat.hfst	дарйо<n>><pl>	daryo>yen
sgh_gen_stem_word_lat.hfst	дарйо<n>><pl>	daryoyen
sgh_gen_rulem_morph_lat.hfst	река<n>><pl>	daryo>yen
sgh_gen_rulem_word_lat.hfst	река<n>><pl>	daryoyen
sgh_analyze_stem_morph_cyr.hfst	дарйо>йен	дарйо<n>><pl>
sgh_analyze_stem_word_cyr.hfst	дарйойен	дарйо<n>><pl>
sgh_analyze_rulem_morph_cyr.hfst	дарйо>йен	река<n>><pl>
sgh_analyze_rulem_word_cyr.hfst	дарйойен	река<n>><pl>
sgh_analyze_stem_morph_lat.hfst	daryo>yen	дарйо<n>><pl>
sgh_analyze_stem_word_lat.hfst	daryoyen	дарйо<n>><pl>
sgh_analyze_rulem_morph_lat.hfst	daryo>yen	река<n>><pl>
sgh_analyze_rulem_word_lat.hfst	daryoyen	река<n>><pl>

Table 3: A full list of available HFST transducers

Four binary variables result in 16 ($= 2^4$) FST variations. Every FST listed in Table 3 can be built with `make` command as shown in Code block 1. Regular `.hfst` binary FSTs are not recommended using in production environments, as they are not optimized. For production use an optimized format called `.hfstol` (HFST **O**ptimized **L**ookup), which can also be compiled automatically for every listed FST using `make`.

```
$ make sgh_gen_stem_morph_cyr.hfst
$ make sgh_gen_stem_morph_cyr.hfstol
```

Code 1: Example of FST compilation with `make`.

4.2 FST compilation pipeline

Analyzers and generators

First, there are two main types of FSTs: generators and analyzers. They differ only in the directionality of a FST. Analyzers take wordforms as input and return glossed strings as output. Generators work in reverse, as shown in Code block 2.

```
$ echo "дарйойен" | hfst-lookup -q sgh_analyze_stem_word_cyr.hfst
дарйойен        дарйо<n>><3pl>  0.000000
дарйойен        дарйо<n>><pl>   0.000000
$ echo "дарйо<n>><pl>" | hfst-lookup -q sgh_gen_stem_word_cyr.hfst
дарйо<n>><pl>   дарйо-йен       0.000000
дарйо<n>><pl>   дарйойен        0.000000
```

Code 2: FST analyzer vs generator output formats.

The `lexd` source code is written as a generator, meaning by default, compiled FST takes glossed stem or lemma as input and returns a wordform. To compile any analyzer, a corresponding generator is inverted, as shown in Code block 3.

```
$ hfst-invert sgh_gen_stem_word_cyr.hfst  \
           -o sgh_analyze_stem_word_cyr.hfst
```

Code 3: FST analyzer creation from a FST generator.

Shughni stems and Russian lemmas

The next format only applies to the glossed side of FSTs (to the analyzers' output and to the generators' input). It sets whether a Cyrillic Shughni stem or a Russian translated lemma will be used as a stem's gloss, as shown in Code block 4. Shughni stems can have multiple Russian candidates (*'дарйо'* can be translated as *'река'* or *'море'*). This leads to composed transducer having more output candidates. This works both ways, meaning Russian lemmas can translate as multiple Shughni stems (*‘река’* can be translated as *‘дарйо’* or *‘х̌ац’*).

```
$ echo "дарйо<n>><3pl>" | hfst-lookup -q sgh_gen_stem_word_cyr.hfst
дарйо<n>><3pl>  дарйо-йен     0.000000
дарйо<n>><3pl>  дарйойен      0.000000
$ echo "река<n>><3pl>" | hfst-lookup -q sgh_gen_rulem_word_cyr.hfst
река<n>><3pl>    дарйо-йен     0.000000
река<n>><3pl>    дарйойен      0.000000
река<n>><3pl>    х̌ац-ен        0.000000
река<n>><3pl>    х̌ацен         0.000000
$ echo "дарйойен" | hfst-lookup -q sgh_analyze_rulem_word_cyr.hfst
дарйойен         море<n>><3pl> 0.000000
дарйойен         море<n>><pl>  0.000000
дарйойен         река<n>><3pl> 0.000000
дарйойен         река<n>><pl>  0.000000
```

Code 4: Shughni stem vs Russian lemma versions of FST.

The `lexd` source code contains lexicons with Shughni stems on the glossed side, meaning default compiled FST contains only Shughni stems. The process of creating a FST that works with Russian lemmas on the glossed side is more complicated. It is achieved with the help of a second FST `rulem2sgh.hfst`, its only purpose is translating stems to lemmas. It is attached to the input of a generator FST, creating a pipeline

‘`река<n>><pl>`’ → `rulem2sgh` → ‘`дарйо<n>><pl>`’ → `generator` → ‘`дарйойен`’

It can be done with ‘compose’ transducer operation (see Code block 5), which takes two FSTs, directs first’s output to the second’s input and returns the resulting composed FST. Details of the translator FST’s development are described in Section 4.7.

```
$ hfst-compose rulem2sgh.hfst sgh_gen_stem_word_cyr.hfst
  -o sgh_gen_rulem_word_cyr.hfst
```

Code 5: Shughni stem translator composition.

Latin script support

The source code contains only Cyrillic lexicons. The support of Latin script comes with the help of a separate transliterator FST, as it was the case for Russian lemmas support. Transliteration is only applied to the wordform side of FSTs. The glossed side's stems and lemmas are left Cyrillic. The pipeline for transliteration is below:

'дарйо<n>><pl>' → generator → 'дарйойен' → cyr2lat → 'daryoyen'

The compilation process is the same as it is for Russian lemmas shown in Code block 5. The only difference is that generator comes first and transliterator comes second, as it is applied to the wordform instead of the glossed side. See Section 4.6 for more details about the transliterator development.

Morpheme borders

Theoretical linguists sometimes need wordforms to have morphemes separated by a special symbol that does not appear naturally in a language. In this morphological parser I choose the right angular bracket symbol '>' for this role to keep it consistent with glossed strings morpheme separator. This presented a slight challenge to the development, as regular wordforms often may contain hyphens ('-') between some morphemes. For an example, 'дарйойен' can also be spelled as 'дарйо-йен'. But for the morpheme separated FST wordform must contain only 'дарйо>йен', with no extra optional hyphens.

This was solved with the help of `twol` rules. The base `lexd` FST (`sgh_base_stem.hfst`) contains both hyphens and morpheme separators, which looks like 'дарйо>-йен' and 'дарйо>йен' on the wordform side. Then the filtering is done with help of two FSTs that contain `twol` rules shown in Code block 6. The `sep.hfst` removes all separator symbols ('>') from the wordform, leaving 'дарйо-йен' and 'дарйойен'. And `hyphen.hfst` removes all hyphens from the wordform, leaving 'дарйо>йен' and 'дарйо>йен', which then fold into a single FST path after optimization.

For this case compose-intersect operation is applied, that allows `twol`-compiled FSTs to be composed with regular lexicon `lexd` FSTs. Compilation command is shown in Code block 7.

This way, every FST listed in Table 3 can be compiled using a specific combination of bash HFST tools shown in this section. For more details explore the `Makefile` in the repository (Kartina, 2025). **TODO: How to cite properly? Work vs Repo**

4.3 `lexd` rule declaration

The choice of lexicon compiler was made in favor of `lexd` as it provides everything that `lexc` does and in addition has some extra useful functional in form of the tag system, which will be taken advantage of.

The `lexd` source code is stored in the `lexd/` directory. I decided to go with a modular file structure for `lexd` source code, as it helps to keep the source code organized. The `lexd/` directory

```
# hyphen.twol
Rules
"Hyphen removal to avoid '>-'/'->' situations"
%-:0 <=> _ ;

# sep.twol
Rules
"Morpeme separator removal"
%>:0 <=> _ ;
```

Code 6: `twol` rules that filter morpheme border.

```
$ hfst-compose-intersect sgh_base_stem.hfst twol/sep.hfst
  -o sgh_gen_stem_word_cyr.hfst
$ hfst-compose-intersect sgh_base_stem.hfst twol/hyphen.hfst
  -o sgh_gen_stem_morph_cyr.hfst
```

Code 7: Plain text and morpheme separated text FST versions compilation.

contains `.lexd` source code files with morpheme lexicons (suffixes, clitics, prefixes, etc.) and lexicon combination patterns. Stem lexicons are stored separately in the `lexd/lexicons/` directory. For the most part `lexd/lexicons/` directory contains lexicons obtained from database dumps provided by Makarov et al. (2022). The stem lexicon processing is described in Section 4.5.

There is no module import feature in `lexd`. So in order to be able to make a modular `.lexd` source file structure compilable into a single `.hfst` file we can concatenate every `.lexd` module into a single large temporary `.lexd` file and feed it to the compiler. This is achieved with `bash` command shown in Code block 8. The `lexd` compiler outputs FST in AT&T format and `hfst-txt2fst` converts it to a binary `.hfst` file.

```
$ cat lexd/*.lexd lexd/lexicons/*.lexd > sgh.lexd
$ lexd sgh.lexd | hfst-txt2fst -o sgh_base_stem.hfst
```

Code 8: Bash command pipeline compiling multiple `.lexd` files into a single FST.

TODO: Finish when done!!

4.3.1 Nouns

4.3.2 Verbs

4.3.3 Adjectives

4.3.4 Pronouns

4.3.5 Numerals

4.3.6 Anything else(?)

4.4 `twol` phonology

Shughni is not rich for morphonological rules. The only global phonological rule is 'J' drop (**TODO: citation needed; 'drop' is a questionable name for this**) on some morphemes' borders after a consonant:

$$j \rightarrow \emptyset \ / \ [+consonant] \ \_$$

TODO: как показать границу морфемы, нужна ли вообще нотация правила если я не могу на неё сослаться

For this rule a special multichar symbol is introduced in FST lexicons: '`{Й}`' (symbol includes curly brackets). A FST treats multichar symbols as single unique symbols, even though they visually consist of multiple characters. This feature allows us to mark which lexicon entries are affected by this phonological rule and which are not. An example of lexicon definition with this symbol is shown in Code block 9.

```
LEXICON PCS # Personal clitics
<1sg>:{Й}ум
<2sg>:{Й}ат
<2sg>:т
<3sg>:{Й}и
<1pl>:{Й}āм
<1pl>:{Й}ам
<2pl>:{Й}ет
<3pl>:{Й}ен

LEXICON DIM # Diminutive clitic
<dim>:{Й}ик
<dim>:{Й}ак
```

Code 9: A real `lexd` example of '`{Й}`' multichar usage.

But without twol rules these morphemes will remain as they are specified in lexd, meaning that feeding 'brother<n>><dim>' to the input of a generator will literally output 'вирод{Й}ик'. To get rid of this intermediate symbol, we can use twol rule shown in Code block 10. The composition of twol rules with the main FST is the same as it was for morpheme border filtration (Code block 7).

```
"Вставка йота"
%{Й%}:0 <=> Consonant (%>) (%-) (%>) _ ;
```

Code 10: A twol rule for 'j' insertion depending on the previous morpheme's segment. symbol in twol is used to escape a character.

4.5 Stem lexicons processing

4.6 Transliteration

4.7 Russian lemmas TODO: rethink heading

4.8 Testing

4.9 Metrics

5 Results

6 Conclusion

References

Beesley, e. R., & Karttunen, L. (2002). *Finite-state morphology: Xerox tools and techniques*. CSLI, Stanford.

Budilova, Z. A. (2023). *Создание морфологического парсера для чамалинского языка в системе lexd и twol [Morphological parser of Chamalal in lexd and twol]*, NRU HSE. https://www.hse.ru/edu/vkr/837214661

Buntyakova, V. A. (2023). *Создание морфологического парсера андийского языка в системе lexd и twol [Morphological parser of Andi in lexd and twol*, NRU HSE. https://www.hse.ru/edu/vkr/837214826

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In J. Burstein, C. Doran, & T. Solorio (Eds.), *Proceedings of the 2019 conference of the north American chapter of the association for computational linguistics: Human language technologies, volume 1 (long and short papers)* (pp. 4171–4186). Association for Computational Linguistics. https://doi.org/10.18653/v1/N19-1423

Edelman, D. I. (2009). Сравнительная грамматика восточноиранских языков [Comparative Grammar of Eastern Iranian Languages].

Edelman, D. I., & Yusufbekov, S. (1999). Шугнанский язык [Shughni language]. In *Языки мира: Иранские языки. III: Восточноиранские языки [Languages of the world: Iranian languages. III. Eastern Iranian languages]*. https://iling-ran.ru/web/ru/publications/langworld/volumes/7

Edelman, D. I., & Dodykhudoeva, L. R. (2009). Shughni. In G. Windfuhr (Ed.), *The iranian languages* (pp. 787–824). London & New York: Routledge.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*, 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Karamshoev, D. (1963). *Баджувский диалект шугнанского языка [Badzhuvskij dialect of the Shughni language]*. изд-во АН Тадж. ССР. https://books.google.ru/books?id=8q1GXwAACAAJ

Karamshoev, D. (1988–1999). *Шугнанско-русский словарь [Shughni-Russian dictionary]*. Izd-vo Akademii nauk SSSR.

Kartina, E. (2025). *Afmigoo/bachelor_diploma*. https://github.com/afmigoo/bachelor_diploma

Karttunen, L. (1993). Finite-state lexicon compiler. Technical Report ISTL-NLTT-1993-04-02, Xerox Palo Alto Research Center, Palo Alto, CA.

Karttunen, L., Koskenniemi, K., & Kaplan, R. (1987). A compiler for two-level phonological rules. *tools for morphological analysis*.

Kondratyuk, D., & Straka, M. (2019). 75 languages, 1 model: Parsing universal dependencies universally.

Koskenniemi, K. (1983). Two-level Morphology: A General Computational Model for Word-Form Recognition and Production.

Lindén, K., Silfverberg, M., & Pirinen, T. (2009). HFST Tools for Morphology – An Efficient Open-Source Package for Construction of Morphological Analyzers. In C. Mahlow & M. Piotrowski (Eds.), *State of the Art in Computational Morphology* (pp. 28–47, Vol. 41). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-04131-0_3

Makarov, Y., Melenchenko, M., & Novokshanov, D. (2022). Digital Resources for the Shughni Language. *Proceedings of The Workshop on Resources and Technologies for Indigenous, Endangered and Lesser-Resourced Languages in Eurasia within the 13th Language Resources and Evaluation Conference*, 61–64. https://aclanthology.org/2022.eurali-1.9

Melenchenko, M. G. (2021). *Автоматический морфологический анализ шугнанского языка [Automatic full morphology analysis for Shughni]*, NRU HSE.

Osorgin, I. G. (2024). *Создание морфологического парсера для шугнаского языка в системе lexd и twol [Creating a morphological parser for the Shughni language using lexd and twol systems]*, NRU HSE.

Parker, C. (2023). *A grammar of the Shughni language* [Doctoral dissertation, Department of Linguistics, McGill University].

Plungian, V. (2022). The study of shughni: The past and the future. *RSUH/RGGU Bulletin: "Literary Teory. Linguistics. Cultural Studies", Series. 2022;(5):11-22*.

Salemann, K. (1895). Шугнанский словарь Д.Л. Иванова [Shughni dictionary by D.L. Ivanov]. In *Восточные заметки [Eastern notes]*.

Sarveswaran, K., Dias, G., & Butt, M. (2021). ThamizhiMorph: A morphological parser for the Tamil language. *Machine Translation 35*, 37–70. https://link.springer.com/article/10.1007/s10590-021-09261-5

Swanson, D., & Howell, N. (2021). Lexd: A Finite-State Lexicon Compiler for Non-Suffixational Morphologies.

Tumanovich. (1906). *Краткая грамматика и словарь шугнанского наречия [Brief grammar and dictionary of Shughni]*.

Turing, A. M. (1937). On computable numbers, with an application to the entscheidungsproblem. https://doi.org/10.1112/plms/s2-42.1.230

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Washington, J., Ipasov, M., & Tyers, F. (2012). A finite-state morphological transducer for Kyrgyz. In N. Calzolari, K. Choukri, T. Declerck, M. U. Doğan, B. Maegaard, J. Mariani, A. Moreno, J. Odijk, & S. Piperidis (Eds.), *Proceedings of the eighth international conference on language resources and evaluation (LREC'12)* (pp. 934–940). European Language Resources Association (ELRA). https://aclanthology.org/L12-1642/

Zarubin, I. (1960). *Шугнанские тексты и словарь [Shughni texts and dictionary]*. Izd-vo Akademii nauk SSSR.