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. Меленченко
_____

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. 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; and a morphological generator that outputs word-forms taking a stem and morphological tags as an input. This is a continuation of my previous work, where nouns, pronouns, prepositions and numerals were implemented (Osorgin, 2024). Despite the title, the current work fills the gap by covering all known Part of speech tags for Shughni. The resulting morphological parser covers 87.8% of native corpus' tokens, returning at least one correct glossing variant for 72.6% of covered tokens, with 51.0% of all parser's glossing variants being correct.

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 Okrug. 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. Shughni has a dialect - Bajuwi, which is spoken in Tajikistan, nearby the area of the closely related Rushani language (Karamshoev, 1963, p. 5).

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, including a Bajuwi dialect. 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. The choice of glosses is mainly motivated by the target users of this morphological parser. It will be mainly used by the researchers of ongoing field works in Tajikistan. They also provided me an internal unpublished list of their glosses, which I am giving priority to.

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. Additionally, a native text corpora can significantly ease the development process by providing real wordforms in context.

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` (Bidirectional encoder representations from transformers) 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 process 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 not 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.

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'.

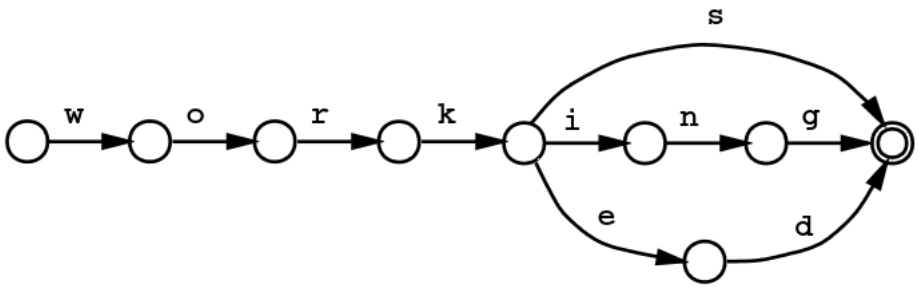

Figure 2: An example of a FST with a single initial state (most left node) and a single final state (most right node) for a language where only three words exist: *works*, *working* and *worked*. The word *worker*, for example will not be considered a valid word by this FSA, since there is no 'r' transition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

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 maximized 100% *Coverage* and 0% *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.

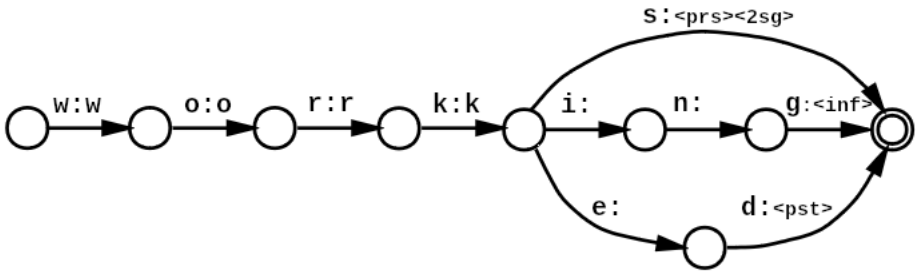

Figure 3: A modified version of Figure 2 which takes as input *works*, *working*, *worked* and outputs *work<prs><2sg>*, *work<inf>*, *work<pst>* respectively.

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.

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 parser for Kyrgyz by Washington et al. (2012), for Andi by Buntyakova (2023), for the languages of the Caucasus (Arakelova & Ignatiev, 2021), 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 existing parser has a flexible script disambiguation, and a rich lexicon, thanks to its tight integration with the project's digital dictionary database.

The main difference from the parser presented in this work is that Melenchenko's parser is not based on FST technology. The FST-based approach results in lightweight (the file size of the biggest FST variation in this work is `8Mb`) and fast (the speed of the slowest FST variation in this work is > 10000 tokens/s) transducers that can be easily combined with other standard tools like Constraint Grammar (Karlsson et al., 1995) that can be used, for example, for contextual disambiguation. The existing `Python`-based parser lacks in speed (the author claims an average speed of 3.8 tokens/s) and outputs results in a `.json` format. It also does not allow wordform generation (a reverse process of morphological parsing), which takes glosses as input and returns wordforms. FST implementation allows inverting any transducer, resulting in a morphological generator. The last difference is in the POS tags: Melenchenko's parser does not tag parts of speech, while the parser presented in this work does.

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 that 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 source	**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 texts are 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 quality metrics 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', POS tag and gloss descriptions are listed in Tables 11, 12 and 13). Glossed strings format is based on Apertium format, 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 key points 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 = stone<n><sg>
- Different morpheme tags are separated with a single right angular bracket '>'.
  stone-PL = stone<n>><pl>
- Multiple stems in a single word are possible.
  stem1<adj>><morph>>stem2<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 segmentation: plain word or segmented word (morphemes separated with ('>') delimeter symbol). Notated in file names as word and segm respectively.
  дарйойен (plain word); дарйо>йен (segmented word)

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

Transducer file name	Input example	Output example
sgh_gen_stem_segm_cyr.hfst	дарйо<n>><pl>	дарйо>йен
sgh_gen_stem_word_cyr.hfst	дарйо<n>><pl>	дарйойен
sgh_gen_rulem_segm_cyr.hfst	река<n>><pl>	дарйо>йен
sgh_gen_rulem_word_cyr.hfst	река<n>><pl>	дарйойен
sgh_gen_stem_segm_lat.hfst	дарйо<n>><pl>	daryo>yen
sgh_gen_stem_word_lat.hfst	дарйо<n>><pl>	daryoyen
sgh_gen_rulem_segm_lat.hfst	река<n>><pl>	daryo>yen
sgh_gen_rulem_word_lat.hfst	река<n>><pl>	daryoyen
sgh_analyze_stem_segm_cyr.hfst	дарйо>йен	дарйо<n>><pl>
sgh_analyze_stem_word_cyr.hfst	дарйойен	дарйо<n>><pl>
sgh_analyze_rulem_segm_cyr.hfst	дарйо>йен	река<n>><pl>
sgh_analyze_rulem_word_cyr.hfst	дарйойен	река<n>><pl>
sgh_analyze_stem_segm_lat.hfst	daryo>yen	дарйо<n>><pl>
sgh_analyze_stem_word_lat.hfst	daryoyen	дарйо<n>><pl>
sgh_analyze_rulem_segm_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_segm_cyr.hfst
$ make sgh_gen_stem_segm_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.

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.

```
$ 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.

```
$ 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 *'хăц'*).

```
$ 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>       хăц-ен          0.000000
река<n>><3pl>       хăцен           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 shown below:

'дарйо<n>><pl>' → generator → 'дарйойен' → cyr2lat → 'daryoyen'
'daryoyen' → lat2cyr → 'дарйойен' → analyzer → 'дарйо<n>><pl>'

The compilation process is the same as it is for Russian lemmas shown in Code block 5. The only difference is that transliterator is applied to the wordform side of a FST, while Russian lemma translator is applied to the glossed side. See Section 4.6 for more details about the transliterator development.

Wordform segmentation

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.

```
# 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.

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 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.

In the sections below I will describe some important `lexd` decisions, that had to be made. As well as general inflectional and derivational information.

Global patterns

As was mentioned before, by default, `lexd` in this project is implemented with morpheme segmented wordforms. For this purpose, various wordform separators were defined (see Code block 9). These separator patterns are inserted into morphological patterns between morphemes, resulting in wordforms looking like *'prefix->stem>-suffix'* by default, with hyphens and morpheme separators mixed together. This is later resolved with the help of `twol` in the compilation pipeline, as was shown in Section 4.2. The `_morph_hyphen_` pattern is used for suffixes, and the `_hyphen_morph_` pattern is used for prefixes.

```
PATTERN _morph_
[>:>]

PATTERN _hyphen_
[:-]?

PATTERN _morph_hyphen_
_morph_ _hyphen_

PATTERN _hyphen_morph_
_hyphen_ _morph_

LEXICON Adv
<adv>:{й}aθ
```

Code 9: Global patterns and lexicons. Remember, that the default FST is implemented as a generator, meaning that on the left side of ':' is a glossed string, and on the right side is a wordform. So hyphens do not show up in glosses strings and are optional on the wordform side by default.

There is also a global lexicon `Adv`, that contains an adverbalizer suffix '-(й)aθ', that supposedly attaches to nouns and adjectives: *'půnd-aθ'*='road-ADV' (Parker, 2023, p. 139), *'zūr-aθ'*='great-ADV' (Parker, 2023, p. 433).

Clitics

Clitics is Shughni for the most part can attach to any part of speech. For example, verbal person agreement in past tense happens with the help of PSC (Personal clitics), which attaches to the right side of the first syntactic constituent in a clause (Parker, 2023, p. 262). This exceeds the scope of morphology, so in order to take this into account, clitics were made global, in our formal model they attach to everything. An example of clitics lexicon is shown in Code block 10. The exact possible order of all different clitics is also unknown, so the final `GlobClitics` pattern contains a rule that states: 'any zero to three clitics'. The downside is that this significantly increases overgeneration. There is also a prepositional clitic *'ик-'*(EMPH), that is not mentioned by Parker (2023), but is described by Karamshoev (1988–1999) that is implemented as global.

```
PATTERN GlobClitics
(AnyClitic > AnyClitic > AnyClitic)?

PATTERN AnyClitic
PronClitics
...
FutureClitic

PATTERN PronClitics
_morph_hyphen_ PCS

PATTERN FutureClitic
_morph_hyphen_ FUT

...

LEXICON PCS
<1sg>:{Й}ум
...
<3pl>:{Й}ен

LEXICON FUT
<fut>:та
```

Code 10: A fragment of `lexd/clitics.lexd`. '`...`' is not a part of the source code, here it denotes a content skip in order to take less space and stay informative.

Nouns

Noun inflection implementation consists of number and case, number inflection is shown in Code block 11. A plural suffix *'-(ӑ)ен'* applies to every singular noun form. Other plural suffixes only attach to different semantic categories such as in-law family members (Parker, 2023, p. 148). Parker also lists a plural suffix *'-(а)чев'* that applies to times of day and year, creating a meaning 'in the evenings' if combined with *'evening'* stem. But it conflicted with the internal unpublished glosses of the HSE expeditions to Tajikistan, where it was listed as a derivational suffix glossed 'TIME'. I decided to stick to the field glosses, as these researchers are the target users of this morphological parser.

```
PATTERN NounNumberBase
NounBase                                        [<sg>:]
NounBase                          _morph_hyphen_  [<pl>:{Й}ен]
NounBase[pl_in-laws]              _morph_hyphen_  [<pl>:орӵ]
NounBase[pl_cousins]              _morph_hyphen_  [<pl>:ӧн]
NounBase[pl_sisters]              _morph_hyphen_  [<pl>:дзинен]
LexiconNounPlRegular        [<n>:] _morph_hyphen_ [<pl>:{Й}ен]
LexiconNounPlRegular        [<n>:]                [<sg>:]
LexiconNounPlIrregular      [<n><pl>:]
LexiconAdv                  [<n>:]                [<sg>:]?
LexiconAdv                  [<n>:] _morph_hyphen_ [<pl>:{Й}ен]
```

Code 11: A segment of `lexd/noun.lexd` showing noun number inflection pattern.

Irregular forms of plural nouns were also extracted from the digital dictionary (Makarov et al., 2022) and put into `LexiconNounPlIrregular` lexicon. In the dictionary both regular and ir-

regular plural noun forms are present. Regular ones were filtered out algorithmically by checking if they have any regular suffixes and their stems were put into `LexiconNounPlRegular` lexicon, in case any of them are missing from the `NounBase` lexicon. This was done with the help of `scripts/lexicons/db_dumps/filter.py` script.

Adverbs lexicon `LexiconAdv` is listed here, since they do not have their own POS tag. According to the unpublished field works glosses, the state of adverbs is an open question. I have decided to put it with nouns since at least some are inflected by number as nouns (e.g. *'ахӯб'*='yesterday', *'ахӯб-ен'*='yesterday-PL')(Karamshoev, 1963). This decision is increasing overgeneration, but in my opinion, it is more important to cover such basic lexicon.

Nouns also have several derivational suffixes like the diminutive *'-(ӣ)ик'*/*'-(ӣ)ак'*, the suffix of origin *'-чи'* (*'аммӯм'*='sauna', *'аммӯм-чи'*='sauna-ORIG' ≈ 'sauna operator') and others. The final noun main pattern is shown in Code block 12.

```
PATTERNS
NounPrefix NounNumberBase NounDeriv? NounSuffix

PATTERN NounPrefix
GlobCliticsPrep (NounPrepos _hyphen_morph_)?

PATTERN NounSuffix
(_morph_hyphen_ NounAdpos)? (_morph_hyphen_ Adv)? GlobClitics
```

Code 12: A segment of `lexd/noun.lexd` showing a final noun morphological pattern.

Verbs

There are four base verb stems in Shughni: non-past, past, infinitive and perfect ones. They can be regularly derived from one another, but there are also irregular verb stems. The regular pattern for infinitive and past stems is the addition of '-т/-д' to the non-past stem. And for the perfect stem is the addition of '-ч/ч' to the non-past stem (Parker, 2023, p. 257). For the inflection, verbs can inflect in person, number, gender and negation. Everything except for gender is done via affixation, while the gender inflection is marked with stem-internal alterations (Parker, 2023, p. 261). Gender inflection unfortunately was not implemented regularly in this work, it was done with the help of digital dictionary, that contained great amount of gender-inflected verb stems.

Person and number inflection is done via suffixation in present stems and via clitics with past stems. The latter is irrelevant for this work, as it exceeds morphology and was taken into account with global clitics. The present stem `lexd` pattern is presented in Code block 13. All the person- and number-specific lexicons like `LexiconVerbNpst1sg` were exported from the digital dictionary and stripped of any inflection. This was done to take into account any irregular stem alterations that were not present in the main `LexiconVerbNpst` lexicon.

Verbs also have short forms, which lexicon names end with `Sh`. They were not stripped of inflection and were left as they are without segmentation, meaning all glosses are considered to belong to the stem.

Other verb stems mostly do not inflect in person or number, but in other aspects they were implemented generally the same with some tense specific minor inflections. The last important details can be shown with perfect stems (see Code block 14). Verbal resultive participle is formed only with masculine perfect stems (Parker, 2023, p. 370). So perfect lexicon had to be split to take this into

```
PATTERN NPastVerbBase
LexiconVerbNpst|LexiconVerbNpstSh   [<v><prs>:]
LexiconVerbNpst                     [<v>:] _morph_ [<prs>:] PresSuffixes
LexiconVerbNpst1sg                  [<v>:] _morph_ [<prs><1sg>:м]
LexiconVerbNpst2sg                  [<v>:] _morph_ [<prs><2sg>:{й}и]
LexiconVerbNpst2pl                  [<v>:] _morph_ [<prs><2pl>:{й}ет]
LexiconVerbNpst3sg                  [<v>:] _morph_ [<prs><3sg>:{vДТ}]
LexiconVerbNpst3pl                  [<v>:] _morph_ [<prs><3pl>:{й}ен]
LexiconVerbNpst1sgSh                [<v><prs><1sg>:]
LexiconVerbNpst2sgSh                [<v><prs><2sg>:]
LexiconVerbNpst2plSh                [<v><prs><2pl>:]
LexiconVerbNpst3sgSh                [<v><prs><3sg>:]
LexiconVerbNpst3plSh                [<v><prs><3pl>:]
LexiconVerbNpstF                    [<v><prs><f>:]
LexiconVerbNpstF3sg                 [<v><prs><f><3sg>:]
```

Code 13: A fragment of `lexd/verb.lexd` showing a non-past verb inflection pattern. `PresSuffixes` contains person-number present inflection suffixes.

account. Perfect lexicons also show how irregular stems are handled. The digital dictionary contains perfect verb forms inflected for plural number, which have regular and irregular forms mixed up together. They were split into two separate lexicons with the help of `scripts/lexicons/db_dumps/filter.py` script, that tried to backtrack verb inflection and categorized failed to backtrack verbs as irregular. Regular verb stems were again stripped of inflection, so they can be inflected and segmented later by FST, and irregular verb stems were left untouched.

```
### Perfect ###
PATTERN PerfVerbBase
PerfVerbBaseMasc    (_morph_hyphen_ VerbPluQuamPerf)?
PerfVerbBaseOther   (_morph_hyphen_ VerbPluQuamPerf)?

PATTERN PerfVerbBaseMasc
LexiconVerbPerfM            [<v><prf><m>:]
LexiconVerbNpst             [<v>:] _morph_ [<prf>:{vЧЧ}]

PATTERN PerfVerbBaseOther
LexiconVerbPerfF            [<v><prf><f>:]
LexiconVerbPerfPlRegular    [<v>:] _morph_ [<prf><pl>:{vЧЧ}]
LexiconVerbPerfPlIrregular  [<v><prf><pl>:]

### Participle2 ###
PATTERN Participle2Base
PerfVerbBaseMasc _morph_hyphen_ [<ptcp2>:ак]
```

Code 14: A fragment of `lexd/verb.lexd` showing perfect verb inflection.

Pronouns and demonstratives

Demonstratives and personal pronouns' lexicons were listed manually. The implementation is for the most part straightforward, they inflect in two cases: locative '-(а)нд' and dative '-(а)рд'. A second-person plural personal pronoun can also take a '-йет'(HON) suffix, which is used to address someone

honorably.

Common indefinite pronouns were implemented in an elegant way. A list of 20 indefinite pronouns was presented by Parker (2023, Table 6.4). These pronouns were easily converted into a simple pattern, as they basically consisted of combinations of 5 bases and 4 prefixes with some stem irregularities (see Code block 15).

```
LEXICON IndefinitePronRoots # Parker 2023 p.193
чйз[g1,g2]
чай[g1,g2]
царāнг[g1]
цawахт[g1]
рāнг[g2]
wахт[g2]
ҷой[g1,g2]

PATTERN IndefinitePronouns # Parker 2023 p.99
[ар]  _hyphen_ IndefinitePronRoots[g1] _hyphen_ [ца]    # Assertive
[йи]  _hyphen_ IndefinitePronRoots[g1]                  # Elective
[йи]  _hyphen_ IndefinitePronRoots[g1] (_hyphen_ [аθ])? # Negative
[фук] _hyphen_ IndefinitePronRoots[g2] (_hyphen_ [аθ])? # Universal
```

Code 15: A `lexd/pron.lexd` fragment showing an implementation of all 20 common indefinite Shughni pronouns.

Adjectives

Shughni adjectives besides some inflection can be derived from other parts of speech. As shown in Code block 16, adjectives can be derived from noun stems with the help of suffixes like '-(й)ин'(≈ 'made of') (Parker, 2023, p. 169). In addition, adjectives can take noun stems to form compound adjectives like *'рӯшт-куртā'*='red.ADJ-shirt.N', resulting in an adjective meaning ≈'red shirt wearing' (Parker, 2023, p. 173).

```
PATTERN AdjBase
LexiconAdj [<adj>:] (_morph_hyphen_ AdjSuffix|Adv)?
### Derivation from Nouns ###
LexiconNoun [<n>:] _morph_hyphen_ NounAdjectivatorsSuffix
NounAdjectivatorsPrefix _hyphen_morph_ LexiconNoun [<n>:]
### Compound Adjectives ###
LexiconAdj [<adj>:] _morph_hyphen_ LexiconNoun [<n>:]
```

Code 16: A `lexd/adj.lexd` fragment.

Numerals

Shughni employs a numeral system that contains both Shughni and Tajik borrowed lexicon (Parker, 2023, p. 176). Both systems are widely used, so they both were implemented. No derivation or inflection is happening with numerals, except for clitic '=ат'(and) used for complex number formation when attaching to *'δūс'*('ten') in Shughni native numeral system.

Other parts of speech

Everything else is implemented in quite straightforward manner: interjections, conjunctions, prepositions, postpositions and particles contain lexicons with only morphological patterns being global clitics attachment.

4.4 `twol` phonology

4.4.1 Global patterns

Shughni is not rich for morphonological rules. The only global phonological rule is 'й' (Latin: 'y'; IPA: /j/) drop on some morphemes' borders after a consonant. In the rule below '>' symbols stands for morpheme border:

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

A special multichar symbol is introduced to stand in place of 'й' letters that are affected by this phonological rule: '{Й}'. We capitalize it and frame it with curly brackets so it is never confused with plain letters. 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. Examples of lexicon definitions with this special symbol were shown in Section 4.3 (Code blocks 9, 10, 11 and 13).

Without `twol` rules these morphemes will remain as they are specified in `lexd`, meaning that feeding 'вирод<n>><dim>'(='brother.N-DIM') to the input of a generator will output literally 'вирод{Й}ик'. After applying a `twol` rule (shown in Code block 17) it generates wordforms 'вироДик' (='brother.N-DIM') and 'туйик' (='you.PERS.2SG-DIM'). The composition process of `twol` rules with the main FST is the same as it was for morpheme border filtration (Code block 7).

```
"й drop after consonant"
%{Й%}:0 <=> Consonant (%>) (%-) (%>) _ ;

"й drop after consonant"
%{Й%}:й <=> Vowel (%>) (%-) (%>) _ ;
```

Code 17: A `twol` rule for 'й' drop depending on the previous morpheme's segment. Symbol '%' in `twol` is used to escape a character. Morpheme separators are enclosed with brackets, which in `twol` syntax is a way to make anything inside brackets optional.

4.4.2 Verb phonology

Some verb stems can regularly be formed from non-past tense stems. This was briefly discussed in 'Verbs' subsection of Section 4.3. Parker (2023, p. 256) describes two context groups for these rules. The first group contains voiced obstruents, vowels and semivowels 'w'(Latin: 'w'; IPA: /v/) and 'й', these contexts are followed by 'д'(Latin: 'd'; IPA: /d/) for past and infinitive stems and by 'ҷ'(Latin: 'ǰ'; IPA: /d͡ʒ/) for perfect stems. The second group contains everything else and is followed by 'т'(Latin: 't'; IPA: /t/) for past and infinitive and by 'ч'(Latin: 'č'; IPA: /tʃ/) for perfect stems. The `twol` formalism rules for verb stem-forming endings are shown in Code block 18.

```
"Past and Inf verb stem endings"
%{vДТ%}:д => [VoicedObstruent | Vowel | Semivowel] (%>) _ ;

"Past and Inf verb stem endings"
%{vДТ%}:т => [VoicelessConsonant | Nasal | Liquid] (%>) _ ;

"Perf verb stem endings"
%{vЧЧ%}:ч => [VoicedObstruent | Vowel | Semivowel] (%>) _ ;

"Perf verb stem endings"
%{vЧЧ%}:ч => [VoicelessConsonant | Nasal | Liquid] (%>) _ ;
```

Code 18: `twol` rules for verb endings phonology.

4.4.3 Pronoun phonology

Pronouns in Shughni can be inflected in locative and dative case with suffixes '-(a)нд' and '-(a)рд' respectively. The optional 'a' letter also depends on the final letter of the previous segment. The contexts are the same here as for the 'й' drop rule, so the `twol` rules are very similar (see Code block 19).

```
"pronouns loc and dat"
%{A%}:a <=> Consonant (%>) (%-) (%>) _ ;

"pronouns loc and dat"
%{A%}:0 <=> Vowel (%>) (%-) (%>) _ ;
```

Code 19: `twol` rules for noun suffixes phonology.

4.5 Stem lexicons processing

Describing morphological rules and morphemes is not sufficient to make a morphological parser. An important part of the development was gathering stem lexicons for various parts of speech. For some POS, like conjunctions or numerals, lexicons were listed by hand. But for others, like verbs, manual lexicon construction is an extremely time-consuming task. Such lexicons were extracted from the `SQLite` database provided by Makarov et al. (2022).

The database in question contains digital versions of Shughni main dictionaries: Karamshoev (1988–1999) and Zarubin (1960). It has a decently organized structure, which allows selecting dictionary entries by parsed metadata like POS or glosses. To transfer stems from the database to a `lexd`-formatted file it follows the following pipeline:

`Database` → `noun.csv` → `form_lexd.py` → `noun.lexd`

An example of real `csv` and resulted `lexd` fragments are shown in Code block 20. I decided to keep meanings from the source as `lexd` comments. Later it makes it easier to debug lexicons manually, this should not influence the compilation process as comments are ignored. The `form_lexd.py` script is located in `scripts/lexicons/` directory. Besides formatting it also

Exported `noun.csv` fragment

```
cyrillic,meaning
аббуст,рукавица
аббустеҷ,материал
абед,обед; время обеда
абӣн,ненавистница
абкӯмпāрт,обком партии
```

Generated `noun.lexd` fragment

```
LEXICON LexiconNoun[pl_all,sg]
аббуст      # рукавица
аббустеҷ    # материал
абед        # обед; время обеда
абӣн        # ненавистница
абкӯмпāрт   # обком партии
```

Code 20: An example of exported noun stems from the database converted to `lexd` source code.

preprocesses the Shughni stems: removes stresses, unifies Cyrillic script and filters out affix entries (source dictionaries have entries for affixes alongside with regular stems). For convenience the script also sorts stems alphabetically.

The `form_lexd.py` script does not read command line arguments. The configuration is 'hard-coded' to read all the `.csv` files from `scripts/db_dumps/` directory and save the generated `.lexd` files to `lexd/lexicons/`.

4.6 Transliteration

The transliteration in this work is implemented for Latin and Cyrillic scripts of the Shughni language. It is developed as a pair of separate FSTs: `translit/lat2cyr.hfst` (Latin to Cyrillic transliteration) and `translit/cyr2lat.hfst` (Cyrillic to Latin transliteration). I found it more convenient to compile transducers with `lexd` (opposed to raw 'strings' format). `Lexd` allows splitting characters and special symbols into separate lexicons and writing comments, which makes the source file much more readable. Also, as a side bonus, `lexd` syntax allows repeating (making it infinitely cyclic) the transducer without the need to repeat it via HFST tools in the `Makefile`.

The transliterator only works for wordforms (e.g. '`daryoyen`'='river.N-PL'). It does not work for glossed strings (e.g. '`daryo<n>><pl>`'). The reason for this is that in case of Latin to Cyrillic transliteration the FST would also transliterate grammatical tags, outputting '`дарйо<н>><пл>`' given '`daryo<n>><pl>`' as input. This could be solved by listing all the possible grammatical tags as multichar symbols in the transliterator's lexicon, but this seems unnecessarily complicated to me for transliteration purposes.

As shown in Code block 21, transliteration for wordforms works for both plain text wordforms and morpheme-separated wordforms (the difference was explained in Section 4.2: *Morpheme borders*). This ensures that the transliteration FST can be successfuly composed with any version of wordform side of FSTs.

Two separate `lexd` files were used to generate transliterators of `lat2cyr` and `cyr2lat` directionalities. Earlier in the development, only one direction was defined with `lexd`, while the other was compiled by inverting the default direction. This brought some complications, as Cyrillic and Latin scripts are not strictly standardized. Some letters have different variations, for example the /θ/ representation, which can be ther be representated either by 'ϑ'(Unicode '`U+03D1`') or 'θ'(Unicode '`U+03B8`') (lower- and upper-cased variations of a symbol, for human eye it looks the same in some fonts, so it gets confused sometimes). The another example is variations of Latin /ð/, which can be represented by both 'δ'(Unicode '`U+03B4`') and 'ð'(Unicode '`U+00F0`') symbols, but only by 'δ' for Cyrillic. The issue with having a FST definition for only a single direction is that one of the direc-

```
$ cat lat.txt | hfst-lookup -q translit/lat2cyr.hfst
na>čis>en   на>чис>ен   0.000000
na-čis-en   на-чис-ен   0.000000
načisen     начисен     0.000000
$ cat cyr.txt | hfst-lookup -q translit/cyr2lat.hfst
на>чис>ен   na>čis>en   0.000000
на-чис-ен   na-čis-en   0.000000
начисен     načisen     0.000000
```

Code 21: An example of transliteration FST work. The mentioned word 'na>čis>en' can be glossed as *NEG-look.V.PRS=3PL*

tions will always be overgenerated for all the letter variants. A real example is shown in Code block 22. The `lat2cyr` direction standardizes different /ð/ variations, while the inverted `cyr2lat` FST generates all 4 possible combinations of two /ð/ instances.

```
$ lexd translit/lat2cyr.lexd | hfst-txt2fst -o translit/lat2cyr.hfst
$ cat lat.txt | hfst-lookup -q translit/lat2cyr.hfst
ðāðān   ðāðāн   0.000000
ðāðān   ðāðāн   0.000000
$ hfst-invert translit/lat2cyr.hfst -o translit/cyr2lat.hfst
$ echo "ðāðāн" | hfst-lookup -q translit/cyr2lat.hfst
ðāðāн   ðāðān   0.000000
ðāðāн   ðāðān   0.000000
ðāðāн   ðāðān   0.000000
ðāðāн   ðāðān   0.000000
```

Code 22: A transliterator FST pair example, where one of the directions is compiled via the inversion of other.

It was solved by two separate `lexd` definitions for each of the directions. This approach made transliteration more flexible, as it was possible to control each direction separately. A real example of the final transliterator FST version is shown in Code block 23. Transliterator FSTs are later attached to the FST generators' output and analyzers' input. This ensures that both transliteration directions minimize variability, which prevents unnecessary overgeneration on the transliteration stage.

```
$ lexd translit/lat2cyr.lexd | hfst-txt2fst -o translit/lat2cyr.hfst
$ cat lat.txt | hfst-lookup -q translit/lat2cyr.hfst
ðāðān   ðāðāн   0.000000
ðāðān   ðāðāн   0.000000
$ lexd translit/cyr2lat.lexd | hfst-txt2fst -o translit/cyr2lat.hfst
$ echo "ðāðāн" | hfst-lookup -q translit/cyr2lat.hfst
ðāðāн   ðāðān   0.000000
```

Code 23: A transliterator FST pair example, where bith directions are compiled from individual `lexd` definitions.

4.7 Lemma translation

The Shughni stem translator is a separate FST which translates Shughni stems to Russian lemmas. It is supposed to be applied to the glossed side of transducers as shown in Code block 24. Shughni has some cases of compound words which consist of multiple stems, e.g. *ʹдароз-зӣӯʹ* (='long-sleeved') (Parker, 2023, p. 173). This was taken into account, so the translator FST translates any amount of stems in a single word. This FST is blind to morphology rules and will translate anything that contains a set of existing stems and tags, even if the combination is nonsensical. But after composition with a morphology model FST ungrammatical and nonsensical strings are left outside the of resulted FST's paradigm.

```
$ cat glossed.txt | hfst-lookup -q translate/sgh2rulem.hfst
зӯӯ<n>                    рукав<n>                        0.000000
дароз<adj>>зӯӯ<n>         длинный<adj>>рукав<n>           0.000000
дароз<adj>><dim>          длинный<adj>><dim>              0.000000
$ cat nonsense.txt | hfst-lookup -q translate/sgh2rulem.hfst
<adj><v>>зӯӯ<n><pl><sg>   <adj><v>>рукав<n><pl><sg>      0.000000
<dim><pl><sg><1sg><2sg>   <dim><pl><sg><1sg><2sg>        0.000000
$ hfst-compose translate/rulem2sgh.hfst sgh_gen_stem_word_cyr.hfst
  -o sgh_gen_rulem_word_cyr.hfst
$ cat nonsense.txt | hfst-lookup -q sgh_gen_rulem_word_cyr.hfst
<adj><v>>зӯӯ<n><pl><sg>   <adj><v>>зӯӯ<n><pl><sg>+?      inf
<dim><pl><sg><1sg><2sg>   <dim><pl><sg><1sg><2sg>+?      inf
```

Code 24: An example of Shughni stem translator work. '+?' at the end of second column's string and 'inf' in the third column means that FST did not find a valid path for the current word.
'зӯӯ'=*a sleeve*; 'дароз'=*long*
Note: output is edited, contextually insignificant lines are removed to keep Code section small.

The `lexd` source code pattern is shown in Code block 25. `RuLemmasBase` pattern contains lexicons grouped by POS. This ensures that homographs (wordforms with same graphical form but different meanings) with different POS will not share contexts. An example of a homograph in Shughni is *'ду'*, which can stand for a verb (*'hit.V.PRS/IMP'*), a demonstrative (*'D2.M.SG'*) or a conjunction (='*when.CONJ'*). An example of FST with and without POS-grouping is shown in Code block 26. A wordform 'ди', when fed into `analyze_pos_ignored.hfst`, where homographs are not taken into account, returns ungrammatical glosses like *'когда<v>'*('when<v>') and *'этот<v>'*('this_one<v>'), which is unwanted. But when fed into `analyze_pos_fixed.hfst`, where translator has POS tags are glued to stems, the result contains only verbal Russian lemma '*быть<v>*'('hit<v>').

```
PATTERNS
(RuLemmasBase|RuLemmasTags)+

PATTERN RuLemmasBase
RuLemmasAdj [<adj>]
RuLemmasAdv [<adv>]
...
RuLemmasPost [<post>]
RuLemmasV [<v>]
```

Code 25: A fragment of translator FST's `lexd` source code. The fragment contains only pattern rules.

```
$ hfst-compose translate/rulem2sgh_no_pos.hfst sgh_base_stem.hfst |
  hfst-invert -o analyze_pos_ignored.hfst
$ echo "mā>ди" | hfst-lookup -q analyze_pos_ignored.hfst
ди      бить<v><imp>                0.000000
ди      когда<v><imp>               0.000000
ди      этот<v><imp>                0.000000
$ hfst-compose translate/rulem2sgh.hfst sgh_base_stem.hfst |
  hfst-invert -o analyze_pos_fixed.hfst
$ echo "mā>ди" | hfst-lookup -q analyze_pos_fixed.hfst
ди      бить<v><imp>            0.000000
```

Code 26: A comparison of FST analyzers. The first one called `analyze_pos_ignored.hfst` treats all homographs independently of their part of speech, which results in incorrect output. The second one (`analyze_pos_fixed.hfst`) returns only verb Russian lemmas.
Note: output is edited: contextually insignificant lines are removed to keep Code section small.

Translation lexicon generation

Translation lexicons (`RuLemmasAdj`, `RuLemmasConj`, etc.) are compiled using a `Python` script `scripts/ru_lemmas/form_lexd.py`. This FST module (`sgh2rulem.hfst`) does not share source code with the main morphology FST, so its lexicons are stored in a separate directory `translate/lexd/`. For convenience every POS lexicon is stored in a separate file under this directory. Compilation process is done by the same principle as for the main FST (see Code block 8).

The biggest challenge of this process was the extraction of Russian lemmas from the database dictionary (provided by Makarov et al. (2022)). Gloss lemmas are preferred to be concise. They usually consist of a single word like (e.g. *swim*) or maximum of 2-3 words (e.g. *swim_deep*) if lemma language can not describe semantics in a single word, and it is important to mention this semantic nuance. Dictionary entries in the database are parsed from real dictionaries, that were written by hand and therefore do not have a strict format. This fact makes it challenging to consistently extract perfect lemmas, some examples are presented in Table 4. In addition, as shown by 'чи' entry, sometimes a real lemma (in this case 'кто') can be hidden in the middle of a text.

Fortunately, most of the lemmas can be extracted automatically without a problem. Russian lemma size minimal statistics are provided in Table 5. Most of the Russian lemmas consist of a couple of words: 76.4% consist of 1 word; 89.1% of ≤ 2 words; 94.6% of ≤ 3 words; 96.8% of ≤ 4 words. Nevertheless, there are still many long lemmas similar to the ones shown in the Tables 4 and 5.

Word	Dictionary entry	Extracted lemma (by script)
цирӯвд	горевать, тосковать, печалиться; скорбеть	горевать
хӣвдow	сбивать палкой орехи, бить по орешнику палкой	сбивать_палкой_орехи
цӯн	подчинительный союз: сколько ни, как ни	сколько_ни
чи	косвенная форма к прямой форме вопросительного местоимения ЧĀЙ кто (см.), употребляемая в различных косвенных позициях	косвенная_форма_к_прямой_форме_ _вопросительного_местоимения_чй_ _кто

Table 4: Examples of extracted Russian lemmas from dictionary entries.

Considering that a single Shughni stem often has multiple Russian lemmas, the probability of getting at least one long lemma rises.

Type	Mean	Median	Max
Length (characters)	10.536	8	105 'указывает_на_косвенное_дополнение_со_ _значением_адресата_действия_при_ряде_ _глаголов_и_глагольных_сочетаний'
Length (words)	1.480	1	16 'косв_форма_мн_ч_указ_мест_дальн_ст_к_ _прямой_форме_w_них_и_т_п'

Table 5: Examples of extracted Russian lemmas from dictionary entries.

4.8 Testing

In context of this work, testing is creating a list of pairs of wordforms and their glossed strings and comparing it to the morphology model's output. These pairs are taken from reliable sources such as field work data or examples from other papers. Testing is useful mostly for the developer for debugging purposes: it lets the developer know if anything stopped working as intended after a change in the source code.

Testing is integrated in the project's `Makefile`. In order to run all existing tests '`make test`' bash command can be used. It runs `scripts/testing/runtests.py` script, which reads pairs of wordforms and glossed strings from all `.csv` files under the `scripts/testing/tests/` directory. An example of a `.csv` with test cases is shown in Code block 27. The script allows testing not only morphology, but any other FSTs too. Currently, this script tests morphology and transliteration. Morphology test cases mainly come from Parker (2023) and unpublished materials of HSE expeditions to Tajikistan. Transliteration tests come from the dictionary database provided by Makarov et al. (2022) project. Every database's dictionary entry has Cyrillic and Latin word versions, which were exported and formatted to match my testing `.csv` format. The only downside of the

transliteration test cases is that most of Latin words most likely were generated automatically from Cyrillic words. Therefore, running transliteration tests ensures that my transliteration FST matches the transliteration tool developed by Makarov et al. (2022). Unfortunately I do not possess other reliable transliteration test data.

A fragment of `scripts/testing/tests/verb.csv`

```
analysis,form,mustpass,hfst,source,page,description
вāр<v><prs>><1pl>,вāр>āм,pass,sgh_gen_stem_morph_cyr,[Parker 2023],258,…
тойд<v><pst><f>,тойд,pass,sgh_gen_stem_morph_cyr,[Parker 2023],113,…
вирӯд<v><pst>,вирӯд,pass,sgh_gen_stem_morph_cyr,[Parker 2023],115,…
находить<v><pst>,virūd,pass,sgh_gen_rulem_morph_lat,[Parker 2023],115,…
```

A fragment of `scripts/testing/tests/translit.csv`

```
input,output,mustpass,hfst,source,page,description
арāн,agān,pass,translit/cyr2lat,https://pamiri.online/,,
арāнт,agānt,pass,translit/cyr2lat,https://pamiri.online/,,
арāнтоw,agāntow,pass,translit/cyr2lat,https://pamiri.online/,,
арāнч,agānč,pass,translit/cyr2lat,https://pamiri.online/,,
```

Code 27: An example of a file with test cases for verbs. First two columns contain reference input and output pair. `hfst` column specifies which FST's format variation must be used for current row. `mustpass` column allows to ignore some cases without the need to remove them from the file. Other columns are ignored by the script, their purpose is to keep track of test case sources.

A **test case** in current context is a single row of a `.csv` file (e.g. Code block 27). For every test case the `scripts/testing/runtests.py` script does the following: (1) feeds the string from the first column to the FST specified in the fourth column, (2) marks the test case as passed if the string from the second column matches **any** string in the FSTs output, (3) logs failed cases. Finally, after all test cases were processed, the script prints general statistics. Script's output example is shown in Code block 28.

```
$ python3 scripts/testing/runtests.py
Loaded 27300 test cases from 1 files
Fail [translit.csv:translit/cyr2lat] анӷ:анӷ (fst output: анӷ+?)
... 44 more Fail lines ...
Fail [translit.csv:translit/cyr2lat] лǫ:lǫ (fst output: лǫ+?)
Loaded 61 test cases from 7 files
Fail [verb.csv:sgh_gen_stem_morph_cyr] вубҷ<v><prs>:вубҷ (fst output: вубҷ
<v><prs>+?)
Transliteration: 27254 (99.83%) passed; 46 failed; 27300 total
Morphology: 60 (98.36%) passed; 1 failed; 61 total
Testing done in 1.415sec
```

Code 28: Example of testing script's output with some failed test cases.

The testing output statistics are not considered as a metric of morphology model's quality. It is only utilized as a development tool: to make sure that FST was compiled as intended.

4.9 Metrics

4.9.1 Quantitative metrics

Coverage metric is defined as relation of the amount tokens that were given any glossed output by FST ($N_{recognized}$) to the total amount of tokens (N_{total}) given to the FST:

$$Coverage = \frac{N_{recognized}}{N_{total}}$$

$N_{recognized}$ does not take into account if the given glosses are correct. It only shows the fraction of words that exist in the FST model's paradigm.

The evaluation is done via a `Python` script `scripts/coverage/eval.py`. It reads Shughni plain text from `stdin`, which must be cleared of all punctuation. Then the script tokenizes it and feeds every token to the FST. Input texts come both in Cyrillic and Latin scripts, so the script detects writing system and calls corresponding FST variation (`sgh_analyze_stem_word_cyr.hfst` or `sgh_analyze_stem_word_lat.hfst`). The script also calculates 5 most frequent unrecognized words and most frequent unrecognized morphemes (it considers any hyphen-separated word fragments as morphemes). An example of its work is shown in the Section **??**.

4.9.2 Qualitative metrics

By the Gold Standard `.eaf` dataset described in Section 3.3 is assumed. The punctuation is filtered out, as well as tokens with no glosses or which glosses fail to be parsed.

There are four qualitative metrics: *Precision*, *Recall*, *F-Score* and *Accuracy(any)*. The first three metrics are evaluated conventionally:

$$Precision = \frac{TP}{TP + FP};\ Recall = \frac{TP}{TP + FN};\ FScore = \frac{2 * Precision * Recall}{Precision + Recall}$$

Where `TP` = '*True positive*', `FP` = '*False positive*' and `FN` = '*False negative*'. The algorithm of these values' evaluation by `scripts/metrics/eval.py` is the following:

1. The script loads the Gold Standard's pairs of wordforms and glossed strings
2. For each wordform$_i$ in the Gold Standard:
   (a) all possible Gold Standard's glossed strings are gathered for the wordform$_i$ ($= G_i$)
   (b) the wordform$_i$ is fed into a FST analyzer and all predicted glossed strings are gathered ($= P_i$)
   (c) the following is calculated:
   $TP_i = |G_i \cap P_i|$ (amount of elements in intersection)
   $FN_i = |G_i \setminus P_i|$ (amount of elements in G_i that are not in P_i)
   $FP_i = |P_i \setminus G_i|$ (amount of elements in P_i that are not in G_i)
3. Total TP, FN and FP are calculated from the sum of respective TP_i, FN_i and FP_i

This algorithm is applied to the Gold Standard two times: one for all the tokens, which contain duplicates, and the second time it is applied to the unique tokens, that contain to duplicates. All metrics are calculated separately for these two token sets.

In other words, TP, FP and FN can be represented as shown in Table 6. Notice, that FP cell is left empty. This is due to the fact that in this work I do not possess negative Gold Standard entries, that represent ungrammatical examples. With this in mind, *Precision* metric can be thought of as 'the fraction of FST's correct answers', and *Recall* can be interpreted as 'the fraction of covered Gold Standard glossed string variants by FST'. *Precision* can also represent the overgeneration, if Gold Standard is guaranteed to contain all grammatical glossed strings of the wordforms. This is not the case in this project, so the *Precision* metric only partially represents the overgeneration degree.

Gold standard / FST output	Present	Not present
Present	TP	FP
Not present	FN	-

Table 6: *TP*, *FP* and *FN* interpretations.

Multiple variations of equality function metric is implemented. An equality function is a *boolean* function that determines if two glossed string variants are considered equal. A full list of measured equality function variants is provided in Table 7.

Equality function variant	**Match conditions**	**Match example**
`Exact`	strings are equal	`amīn<n>><sup>` = `amīn<n>><sup>`
`Stem`	stems are equal	`amīn<n>><sup>` = `<neg>>amīn<v>><1sg>`
`POS`	part of speech tag matches	`amīn<n>><sup>` = `kūdak<n>`
`Tagset`	The set of tags matches independently of order	`amīn<dem><d3><pl>` = `wāδ<d3><pl><dem>`
`Tagset and stem`	`Stem` and `Tagset`	`wāδ<dem><d3><pl>` = `wāδ<d3><pl><dem>`

Table 7: Caption

Accuracy(any)

The *Accuracy(any)* metric is a custom metric that I thought would be helpful to have. It simply represents how many times FST returned **at least one** correct glossed string. Its formula is shown below:

$$Accuracy(any) = \frac{N_{any\_correct}}{N_{recognized}}$$

$N_{recognized}$ is the same as it is in the *Coverage* formula, $N_{any\_correct}$ is simply the amount of tokens that fulfill the condition $TP_i > 0$.

5 Results

The result of the morphological parser development is a set of binary FST files listed in Table 3. They are developed using HFST (Helsinki finite-state technology tools) toolset and come in two variations: `.hfst` files, listed in the table, and `.hfstol` files, an optimized lookup format, which can be compiled as shown in Code block 1. For production use, the latter is strongly recommended.

An unexpected product of this work was the script for qualitative metrics evaluation. It greatly helped during the development process, and it was decided to make an independent user-friendly version of it. The details of its work are not discussed in this work, for more information see the output of '`scripts/metrics/eval.py --help`' command.

The `lexd`, `twol`, `Python` source code with all required `.hfst` and `.hfstol` building dependencies is uploaded to the `GitHub` repository hosting as a public repository under `GNU GPLv3` license. The link is listed in the Appendix: Links Section.

Final morphology parsing metrics are presented in Tables 8, 9 (quantitative) and 10 (qualitative).

Equality function	**Accuracy(any)**	**F-Score**	**Precision**	**Recall**
Exact	0.726	0.390	0.316	0.510
Stem	0.954	0.929	0.879	0.984
POS	0.933	0.802	0.730	0.889
Tagset	0.787	0.437	0.357	0.565
Tagset and stem	0.727	0.396	0.322	0.515

Table 8: Quality metrics evaluated using all tokens in the Gold Standard.

Equality function	**Accuracy(any)**	**F-Score**	**Precision**	**Recall**
Exact	0.684	0.397	0.288	0.637
Stem	0.918	0.853	0.770	0.956
POS	0.885	0.808	0.719	0.923
Tagset	0.730	0.447	0.329	0.698
Tagset and stem	0.686	0.411	0.300	0.648

Table 9: Quality metrics evaluated using only unique tokens in the Gold Standard.

Dataset	Coverage	Coverage absolute
Dictionary examples	0.917	171478/186939
Gold Standard all tokens	0.878	3279/3735
Gold Standard unique tokens	0.804	1039/1293

Table 10: Coverage metric evaluated for different datasets. 'Dictionary examples' dataset was described in Section 3, Table 2.

The developed morphological parser covers a decent amount of Shughni wordforms: 87.8% of native texts in Gold Standard and 91.7% of tokens from the large corpus of dictionary entries' examples (see Table 10). As for the qualitative performance, which was measured only for the wordforms that were covered by the FST model, the morphological parser gives at least one correct answer for 72.6% tokens in the Gold Standard, 51.0% of all the parser's answers are correct (see Table 8). As also can bee seed from the table, parser shows the best performance when it comes to stemming or POS tagging, returning a correct stem in 98.4% output variants (at least one correct stem for 95.4% of all the tokens).

The low *Precision* scores (excluding POS tagging and stemming) likely reflect significant overgeneration, though this interpretation is partial since the Gold Standard is not guaranteed to contain all possible glossing variants. Thus, *Precision* represents both FST overgeneration and potential gaps in the Gold Standard. Either way, of all parser output variants, only 31.6% are present in the Gold Standard.

It is a strong alternative to the existing parser developed by Melenchenko (2021). It is hard to objectively compare these two tools, as Melenchenko's metrics evaluation methods and data differs from methods and data in this work. The only objective common metric is *Coverage* (it is called '*Recall*' in the existing work, but the evaluation method seems to match my *Coverage* metric's), in which the existing parser covers 90% of all tokens $(886/989)$, when my solution covers 87.8% $(3279/3735)$ of all tokens from the data that is a superset of Melenchenko's data.

6 Conclusion

The resulted morphological parser shows a decent performance, providing a strong alternative for the existing solutions. It can surely be used for usual NLP tasks such as POS-tagging, stemming, morphological analysis, or even wordform inflection and generation, thanks to the FST ability to be inverted.

Qualitative metrics are expected to provide a representative and objective evaluation of the morphology model's performance, there is an important moment to mention. That is, the provided Gold Standard in this work is not guaranteed to contain 100% accurate glosses. I cannot even subjectively evaluate how accurate the Gold Standard is, but while development and debugging I was facing questionable tokens in the Gold Standard from time to time. I made a principal decision not to edit the Gold Standard in any way.

The future possible work including this morphological parser can include an integration with Constrain Grammar (CG) formalism (Karlsson et al., 1995). It will add a possibility to take into account the syntactic aspect of the language and apply disambiguation, which will filter out syntactically invalid gloss variants, given wordform's context.

References

Arakelova, D., & Ignatiev, D. (2021). *Developing morphological analyzers for low-resource languages of the Caucasus.*, NRU HSE.

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.

Karlsson, F., Voutilainen, A., Heikkilae, J., & Anttila, A. (Eds.). (1995). *A language-independent system for parsing unrestricted text*. De Gruyter Mouton. https://doi.org/doi:10.1515/9783110882629

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, Endan-*

gered 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.

Appendix

Links

The complete source code, including all HFST building dependencies were made available under an open-source license at: https://github.com/afmigoo/shughni_morphology

Tables

Table 11: A full list of glosses used in this work. Part 1.

FST tag	Gloss	Meaning
<1pl>	1PL	First-person plural
<1sg>	1SG	First-person singular
<2pl>	2PL	Second-person plural
<2sg>	2SG	Second-person singular
<3pl>	3PL	Third-person plural
<3sg>	3SG	Third-person singular
<add1>, <add2>	ADD	Additive (e.g. '-ga')
<adv>	ADV	Adverbializer suffix
<advs>	ADVS	Adversative conjunction
<agn>	AGN	Verb agent nominalization
<aug>	AUG	Adjective intensifier
<cause>	CAUSE	(≈ 'purpose'/'benefit'/'cause')
<com>	COM	Comitative ('together with', 'in the company of') or instrumental case
<comp>	COMP	Adjective comparative
<comp_at>	COMP_AT	An attenuative comparative of an adjective
<compl>	COMPL	Completive conjunction
<cont1>, <cont2>	CONT	Continuative (usually used with body part nouns)
<coord1>	COORD1	Coordinating conjunction '=xu'
<coord2>	COORD2	Coordinating conjunction '='/'=ad'
<coord3>	COORD3	Tajik coordinating conjunction '=u'
<d1>	D1	Proximal demonstrative
<d2>	D2	Medial demonstrative
<d3>	D3	Distal demonstrative
<dat>	DAT	Dative case
<dim>	DIM	Diminutive noun suffix
<dir>	DIR	Directive 'in direction of'
<disj>	DISJ	Disjunctive conjunction
<emph>	EMPH	Emphasizing
<f>	F	Feminine gender
<f_pl>	F/PL	Feminine gender or plural number
<full>	FULL	Adjective suffix '=full of X'
<fut>	FUT	Future tense, habitual aspect
<hb>	HB	Habilitive noun suffix
<hon>	HON	Honorable personal pronoun suffix
<imp>	IMP	Verb imperative stem
<in>	IN	Noun suffix 'in'/'at' (e.g. 'in a house')
<inf>	INF	Verb infinitive stem
<int>	INT	Intensifier
<lim1>, <lim2>	LIM2	Spacial and temporal limitative, instrumental case

Table 12: A full list of glosses used in this work. Part 2.

FST tag	Gloss	Meaning
<loc>	LOC	Locative
<m>	M	Masculine gender
<neg.q>	NEG.Q	Negative question particle
<neg>	NEG	Negative prefix
<obl>	OBL	Oblique case
<ord>	ORD	Ordinal numeral suffix
<orig>	ORIG	'Origin' noun suffix
<p.loc>	P.LOC	Possessive locative
<pl>	PL	Plural number
<place>	PLACE	'Place' noun suffix
<pqp>	PQP	Plusquamperfect verb suffix
<prf>	PRF	Perfect tense
<proh>	PROH	Prohibitive verb prefix
<prol>	PROL	Prolative (≈ 'along'/'through')
<prs>	PRS	Non-past tense
<pst>	PST	Past tense
<ptcp1>	PTCP1	Adjective resultative participle suffix
<ptcp2>	PTCP2	Verbal resultative participle suffix
<purp>	PURP	Purposive suffix of verb infinitive stem
<q>	Q	Question particle
<redup>	REDUP	Reduplication
<refl>	REFL	Reflexive pronoun
<self>	SELF	Self pronoun
<sg>	SG	Singular number
<subd>	SUBD	Subordinating conjunction
<subst>	SUBST	Substantivizing suffix
<sup>	SUP	Superessive case
<time>	TIME	Noun suffix denoting time (e.g. 'at night')
<with>	WITH	Adjective suffix '=with X'
<with_little>	WITH_LITTLE	Adjective suffix '=with little X'
<without>	WITHOUT	Adjective suffix '=without X'

Table 13: A full list of POS (part of speech) tags used in this work and FST.

FST tag	Gloss	Meaning
<adj>	ADJ	Adjective
<conj>	CONJ	Conjunction
<dem>	D (*is ignored in glosses, as D1, D2 or D3 is always present*)	Demonstrative
<intj>	INTJ	Interjection
<n>	N	Noun
<num>	NUM	Numeral
<pers>	pers	Personal pronoun
<post>	post	Postposition
<prep>	prep	Preposition
<pro>	pro	Pronoun
<prt>	prt	Particle
<v>	v	Verb