RBMT models. Apertium machine translation platform

RBMT models:

- **Morphology and morphotactics**

Morphology is: « the branch of linguistics that studies patterns of word formation within and across languages, and attempts to formulate rules that model the knowledge of the speakers of those languages. »

In machine translation we are interested in building and using computational models of morphology for translation.

We are interested in being able to do the following things:

- *Tokenisation*: Split text into units consisting of words

  Итак, это пример → [Итак] [,] [это] [пример]…

- *Analysis*: Analysing these words, giving for each one a list of possible interpretations / morphological descriptions

  это → это+pron.dem.nt.sg.nom, это+pron.dem.nt.sg.acc,
  этот+det.dem.nt.sg.nom, этот+det.dem.nt.sg.acc

- *Generation*: Given a morphological description of a word generate the corresponding surface form.

  это+det.dem.f.sg.nom → эта

Morphotactics is how we describe how morphemes are put together to form words. This can be further divided into several subprocesses:

- *Inflection*: Inflectional morphemes carry grammatical information, such as number, case, tense, etc., but do not change the word category
- *Derivation*: Derivational morphemes change the basic semantic meaning of a word, and can also change word category.
- *Compounding*: Compounding is a process where two or more words are joined together to form one.
- *Clitics*: A clitic is a syntactically independent word that functions phonologically as an affix of another word.

There is another relevant process for machine translation which is related:

- *Multiword units*: A multiword unit is a set of words which should be treated for the purposes of a given natural language processing task as a single word.

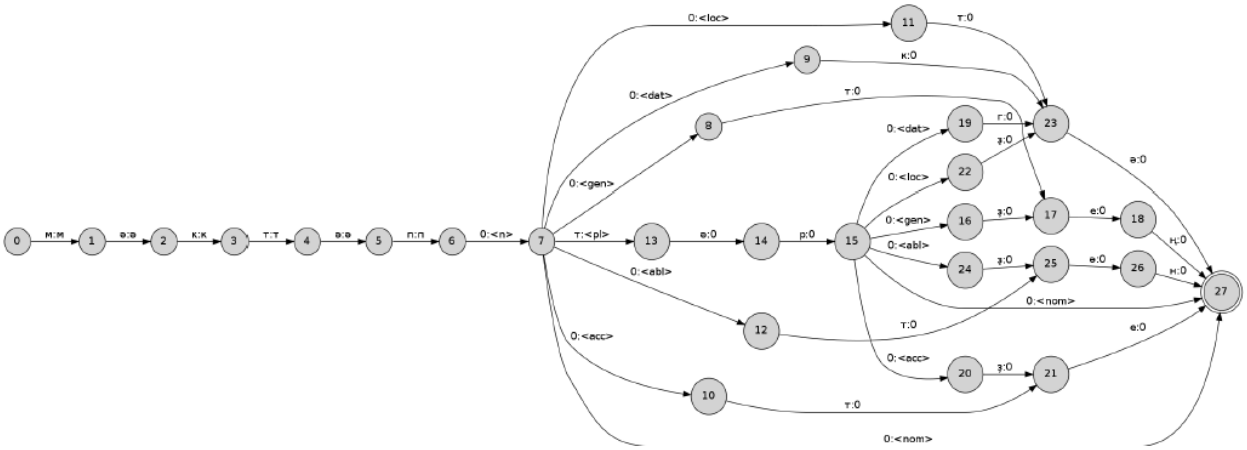
Issues relating computational morphology

- Finite-state transducers
- Lexicon formalisms (lttoolbox and lexc)
- Tags
- Continuation lexica / paradigm
- Flag diacritics

One way of modelling morphology is with a *finite-state transducer*,

- A bit like a flowchart
- Has a set of states (questions) and transitions (decisions)
- Depending on the input, you make different decisions

This is much easier with an example:



So what are the benefits of using finite-state transducers, over for example stemming or other rule-based, or database-based options?

- The same code, and the same morphological description can be used for both analysis and generation.
- The lookup time is linear (meaning they are very fast).
- Does not require programming knowledge.

In Apertium we have a couple of options when we want to make a new morphological description

lttoolbox:

- The original lexicon format
- Based on XML
- One-level, morphotactics and morphophonology done in the same file
- More restrictive, but more userfriendly, e.g. comes with validation
- Useful features for MT built in (direction restrictions)

HFST lexc:

- Very powerful, but no validation
- Can be one-level or two-level
- Text format
- Transducers can be cyclic

tags are special symbols, which denote grammatical features, a given analysis from a morphological analyser will give a lemma (the dictionary form of the word), and a sequence of tags, for example:

- Number: <sg> = singular, <du> = dual, <pl> = plural …
- Case: <nom> = nominative, <abl> = ablative, <ill> = illative …
- Tense: <pres> = present, <past> = past, <fut> = future …

In Apertium, tags are denoted by the less than < and greater than > characters.

A *continuation lexicon* or *paradigm* is a set of input/output pairs which defines an inflection class of a word

- In an agglutinative language you may have a separate continuation lexica for each of a different set of suffixes, e.g. case, number, possession
- In a fusional language, typically a continuation lexicon will define the inflection pattern for a given word

Sometimes it can be necessary to implement long distance dependencies in your analyser, for example

- To model prefix or circumflex inflection
- To disallow combinations of non-adjacent morphemes

- ...

In lexc, flag diacritics can be used for these purposes

- **Morphophonology**

- "a branch of linguistics which studies the interaction between morphological and phonological or phonetic processes"

- or looking at "the sound changes that take place in morphemes when they combine to form words"

- or, for us, just "which changes occur in morphemes as a result of their combination"

Morphophonology: why it matters

- In many languages, many words and morphemes change form when they cooccur

- MT systems should recognise all forms of any word or morpheme

- ... and we have to tell the computer how it can deal with these processes

What is twol? - A formalism for implementing morphophonological rules.

What do I need to use twol for? - When the form of a word or morpheme changes as a result of phonological rules.

How can I use twol? - Find the correct phonological rules. (They aren't always in the grammars, but the grammars can help). - Write the rules in the twol formalism!

- **Morphological and syntactic disambiguation**

What is morphological ambiguity?

- The ambiguity that comes from a surface form having more than one possible analysis.

Why is it important?

- Sometimes different readings will lead to different translations, and we want to get the appropriate translation.

Constraint Grammar

- Takes input consisting of surface forms and their readings

- Applies hand-written rules to remove inappropriate readings in context

- **Lexical transfer**

Lexical transfer: Translating words and marking them for structural transfer.

Transfer. The transfer module is where the magic happens: the intermediate representation in source language (SL) is converted into an intermediate representation in target language (TL).

The lexical transfer module reads each SL lexical form and delivers the corresponding TL lexical form by looking it up in a bilingual dictionary.

- **Lexical selection**

Lexical selection: Choosing a translation of a polysemous word depending on context.

What are lexical selection rules?

- A source word: The word you want to translate

- A target word: The word you want to translate it to

- A context: Set of pairs of features and positions

- An operation: Either to select a word you want to translate to, or remove a word you definitely don't want to translate to

    - Position: The relative position where the feature should be found (free drugs != drug free)

- Feature: A feature is either a surface form, a lemma, a tag (morphological, syntactic, semantic)

- **Basic structural transfer**

Contrastive analysis is the process of examining two or more languages together to find out what kind of features they share, and how they are distinguished.

We call morphological contrasts those differences between languages which are expressed at the level of a single word. Some examples might be:

- How possession is expressed: suffix, or separate word
- Case inventories
- Gender (or word class) inventories
- Number (in all words, or just in nouns/pronouns)
- "Synthetic" verb tenses
- Person inventory
- Formality
- Morpheme order (e.g. possessive before/after number)

There are several types of transfer arrangements in Apertium language pairs,

- Single-level transfer:
- Multiple-level transfer (without chunking):
- Multiple-level transfer:

Rules detect patterns of words (generally specified as parts of speech):

- DET NOUN. The table
- DET ADJ NOUN. The black table

How the rules work

- words can only be processed by ONE rule: rules can not be overlapped
- patterns are matched following the LRLM principle (left-to-right-longest match)

- **Multi-level structural transfer**

3 transfer submodules: … → chunker → interchunk → postchunk → … .

Chunks = group of words that correspond roughly to a phrase. The creation of chunks allows for a more powerful treatment of syntactic/structural divergences.

The chunker is like the transfer module of the single-level transfer, being the main difference:

- The transfer outputs lexical units
- The chunker outputs chunks

The interchunk module:

- detects patterns of chunks in the same way that the chunker detects patterns of words
- applies the necessary transformations to these patterns: agreement, reorderings, syntactic or morphological changes

The postchunk module:

- substitutes numbers in tags by the referenced values
- removes the chunk lemma and tags
- outputs the sequence of lexical units
- can operate only on a single chunk at a time
- can perform some final operations on the chunk content [1]

Apertium machine translation platform

Apertium is a free/open-source rule-based machine translation platform. It is free software and released under the terms of the GNU General Public License. [2]

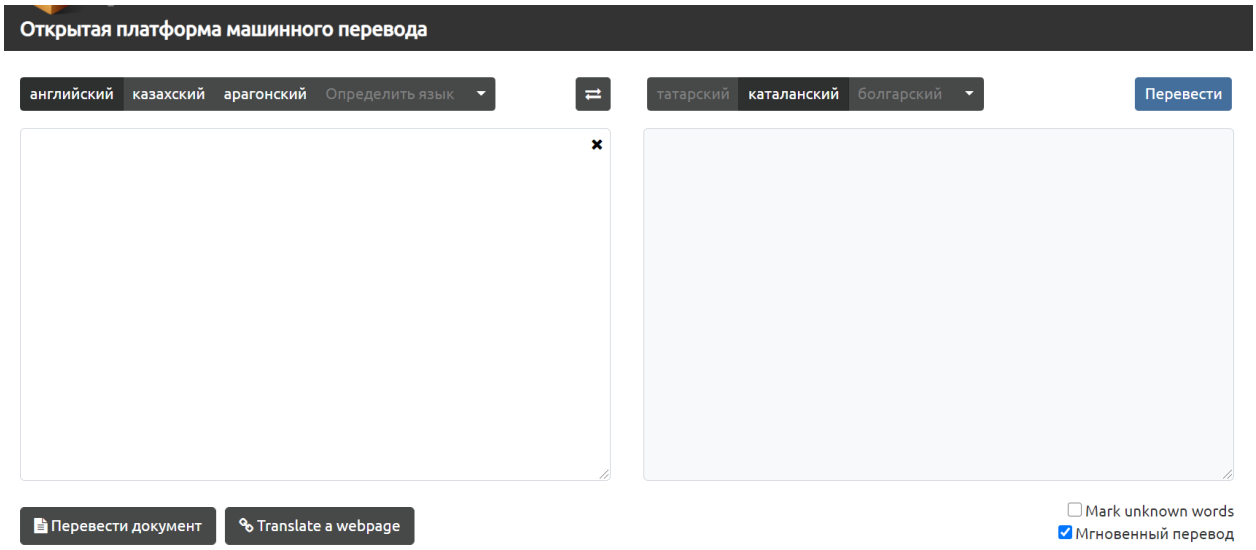


Figure 1. Web page of Apertium machine translation platform

The platform provides

- a language-independent machine translation engine
- tools to manage the linguistic data necessary to build a machine translation system for a given language pair and
- linguistic data for a growing number of language pairs. [3]

Apertium is a shallow-transfer machine translation system, which uses finite state transducers for all of its lexical transformations, and hidden Markov models for part-of-speech tagging or word category disambiguation. Constraint Grammar taggers are also used for some language pairs (e.g. Breton–French).

Existing machine translation systems available at present are mostly commercial or use proprietary technologies, which makes them very hard to adapt to new usages; furthermore, they use different technologies across language pairs, which makes it very difficult, for instance, to integrate them in a single multilingual content management system.

Apertium uses a language-independent specification, to allow for the ease of contributing to Apertium, more efficient development, and enhancing the project's overall growth.

At present (December 2020), Apertium has released 51 stable language pairs, delivering fast translation with reasonably intelligible results (errors are easily corrected). Being an open-source project, Apertium provides tools for potential developers to build their own language pair and contribute to the project.

History

Apertium originated as one of the machine translation engines in the project OpenTrad, which was funded by the Spanish government, and developed by the Transducens research group at the Universitat d'Alacant. It was originally designed to translate between closely related languages, although it has recently been expanded to treat more divergent language pairs. To create a new machine translation system, one just has to develop linguistic data (dictionaries, rules) in well-specified XML formats.

Language data developed for it (in collaboration with the Universidade de Vigo, the Universitat Politècnica de Catalunya and the Universitat Pompeu Fabra) currently support (in stable version) the Arabic, Aragonese, Asturian, Basque, Belarusian, Breton, Bulgarian, Catalan,

Crimean Tatar, Danish, English, Esperanto, French, Galician, Hindi, Icelandic, Indonesian, Italian, Kazakh, Macedonian, Malaysian, Maltese, Northern Sami, Norwegian (Bokmål and Nynorsk), Occitan, Polish, Portuguese, Romanian, Russian, Sardinian, Serbo-Croatian, Silesian, Slovene, Spanish, Swedish, Tatar, Ukrainian, Urdu, and Welsh languages. A full list is available below. Several companies are also involved in the development of Apertium, including Prompsit Language Engineering, Imaxin Software and Eleka Ingeniaritza Linguistikoa. [4]

Translation methodology

This is an overall, step-by-step view how Apertium works.

The diagram (Figure 2) displays the steps that Apertium takes to translate a source-language text (the text we want to translate) into a target-language text (the translated text).

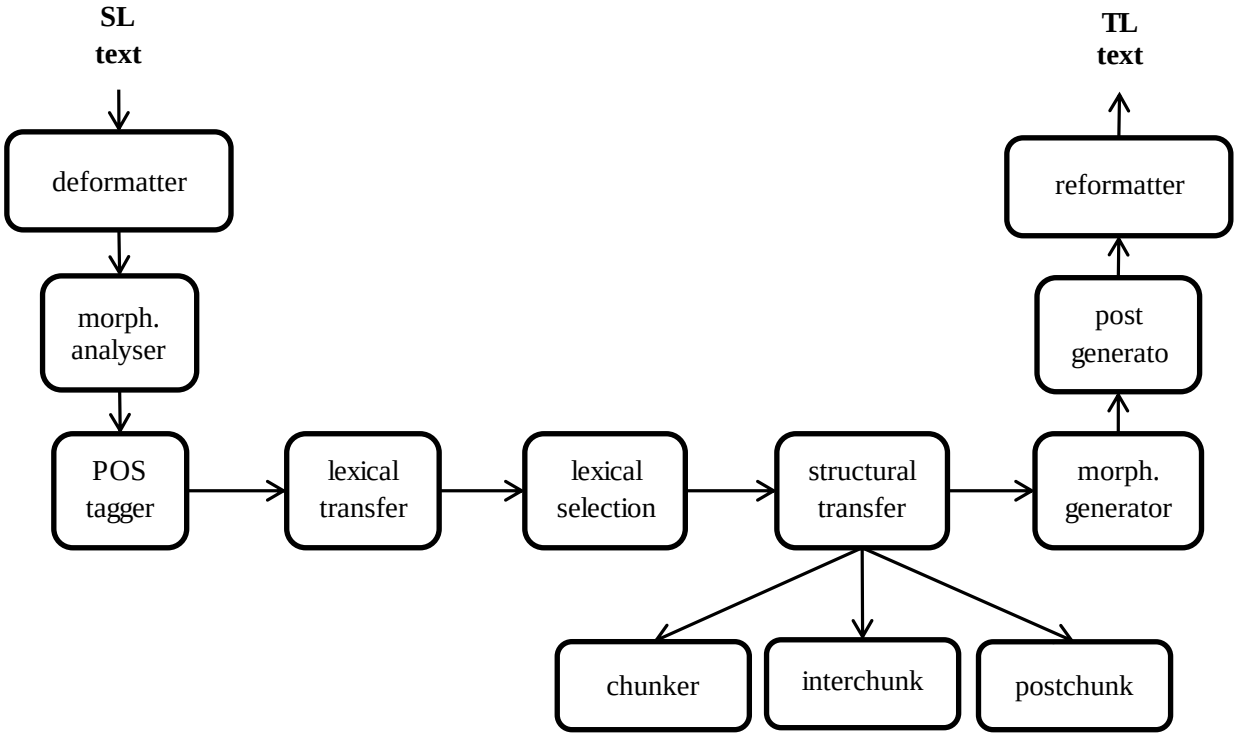


Figure 2. Pipeline of Apertium machine translation system

1. Source language text is passed into Apertium for translation.
2. The ***deformatter*** removes formatting markup (HTML, RTF, etc.) that should be kept in place but not translated.
3. The ***morphological analyser*** segments the text (expanding elisions, marking set phrases, etc.), and look up segments in the language dictionaries, then returning baseform and tags for all matches. In pairs that involve agglutinative morphology, including a number of Turkic languages, a Helsinki Finite-State Transducer (HFST) is used. Otherwise, an Apertium-specific technology, called the lttoolbox, is used.
4. The ***morphological disambiguator*** (the morphological analyser and the morphological disambiguator together form the part of speech tagger) resolves ambiguous segments (i.e., when there is more than one match) by choosing one match. Apertium is working on installing more Constraint Grammar frameworks for its language pairs, allowing the imposition of more fine-grained constraints than would be otherwise possible. Apertium uses the Visual Interactive Syntax Learning Constraint Grammar Parser.
5. ***Lexical transfer*** looks up disambiguated source-language basewords to find their target-language equivalents (i.e., mapping source language to target language). For lexical transfer, Apertium uses an XML-based dictionary format called bidix.

6. ***Lexical selection*** chooses between alternative translations when the source text word has alternative meanings. Apertium uses a specific XML-based technology, apertium-lex-tools, to perform lexical selection.
7. ***Structural transfer*** (i.e., it is an XML format that allows writing complex structural transfer rules) can consist of a one-step transfer or a three-step transfer module. It flags grammatical differences between the source language and target language (e.g. gender or number agreement) by creating a sequence of chunks containing markers for this. It then reorders or modifies chunks in order to produce a grammatical translation in the target-language. This is also done using lttoolbox.
8. The ***morphological generator*** uses the tags to deliver the correct target language surface form. The morphological generator is a morphological transducer, just like the morphological analyser. A morphological transducer both analyses and generates forms.
9. The ***post-generator*** makes any necessary orthographic changes due to the contact of words (e.g. elisions).
10. The ***reformatter*** replaces formatting markup (HTML, RTF, etc.) that was removed by the deformatter in the first step.
11. Apertium delivers the target-language translation. [5]

Structural transfer rules.

The usual application of this method consists of three modules: chunker, interchunk, and postchunk. If necessary, this model can be expanded to two or more intertchunk modules. Let's describe what each of the 3 stages of the transfer is:

Chunker stage

At the transfer stage, words are translated using a bilingual dictionary and divided into groups into segments. The file containing the transfer rules file is named .t1x. Here, tags in words can be added, removed, or created in «pointers» that point to tags in nested segments. This file consists of several required sections and may also contain other optional sections. Each section contains at least one element:

```
<?xml version="1.0" encoding="UTF-8"?>
<transfer>
  <section-def-cats>
        ...
  </section-def-cats>
  <section-def-attrs>
        ...
  </section-def-attrs>
  <section-def-vars>
        ...
  </section-def-vars>
  <section-def-macros>
        ...
  </section-def-macros>
  <section-rules>
        ...
  </section-rules>
</transfer>
```

The def-cats section. The def-cats section is required. It allows you to declare the categories of words that we will receive to apply a certain transfer rule. These can be simple words (noun, adjective, verb, etc.) or slightly complicated things like a noun with the <nom> tag (nominative), meaning that it is part of the subject in the sentence. This section contains one or more elements with the following structure:

```
<def-cat n="name_of_what_we_want_to_describe">
  <cat-item tags="its_description"/>
  ... (there can be one or more <cat-item .../> tags)
</def-cat>
```

The def-attrs section. The def-attrs section is required. Allows to collect by name the attributes of functionality for words defined in the sdefs section of the morphological dictionary. For example, we have collected in this section each tag corresponding to:

- the kind of word;
- the number of the word (singular, plural);
- face of the verb;
- verb tense
- .. etc.

This section contains one or more elements with the following structure:

```
<def-attr n="name_of_a_list_of_attributes_with_a_common_rule">
  <attr-item tags="an_attribute_of_the_sdef_section_of_a_dictionary"/>
  ... (we have several tags <attr-item .../> as many as possible values for the attribute)
</def-attr>
```

The def-vars section. The def-vars section is required and must contain at least 1 element with the following syntax <def-var n = "..." />. It lists the global variables used in transfer rules.

The def-macros section. The def-macros section is optional. This section contains one or more elements with the following structure:

```
<def-macro n="name_of_the_macro" npar="number_of_parameters">
  ... (the code of the macro)
</def-macro>
```

The rules section. The rules section is required. It is the longest of the transfer files. It really allows you to define the operations that must be performed to translate a group of words (or sometimes single words).

This section contains one or more elements with the following structure:

```
<rule>
  <pattern>
        <pattern-item n="name_defined_in_def-cat_corresponding_to_the_first_word_to_process"/>
        ... (as many tags <pattern-item ..../> as words we want to process together)
  </pattern>
  <action>
        ... (description of the transfer rule)
  </action>
</rule>
```

Interchank Stage

Interchunk is an intermediate tool of Apertium, level 2 of structural transformation. Interchunk should never be used on its own. This is the second transfer unit of the Apertium structural unit. Interchunk (.t2x file) 2 transfer level after apertium-transfer and before apertium-postchunk. This level takes care of processing operations such as reordering, changing morphosyntactic features of segments from information in adjacent segments, or creating new segments. The interchanka uses the segments that are defined in the transfer (t1x).

Segmentation is based on templates from the source language. It is used in all language pairs.

- First, the words are grouped into chunks.
- Then the chunks are formed lexical forms of reading from left to right, and then the longest matching sequence is built, which matches one of the templates in the rules file.
- On this basis, a "pseudo lemma" is produced with a label containing a type - it can be any kind of phrase, such as «SN» (nominal) or «SV» (verb).

The chunk for an English phrase might look like this: NP (The dog) VP (walked) PP (with boy). "The dog" is a noun phrase, "walked" is a verb, and is broken as a verb, not as a noun phrase. "with boy" is a prepositional phrase, so it's broken into the “PP” chunk.

In interchank rules, segments are reordered, concatenated, and split, and block tags are changed (.t2x file). The structure of the interchank file is as follows:

```
<interchunk>
<section-def-cats> // определение названия чанков
  …
</section-def-cats>
<section-def-attrs>
  …
  </def-attr>
</section-def-attrs>
<section-def-vars> //место, где создаются переменные
  …
</section-def-vars>
<def-var n="possessive"/>
<section-def-macros> //тут создаются макросы
  …
</section-def-macros>
<section-rules> // место начала создания правил
  <rule comment="RULE:  NP">
            …
  </rule>
</section-rules>
</interchunk>
```

Description of rule formats with examples. Consider an example sentence in English with translation into Russian: {I [NP]} {go [VP]} {to university [PP]} {of Kazakhstan [GenP]}

Translation of this sentence into Kazakh: Men [NP] Kazakstannyng [GenP] universitetine [PP] baramyn [VP].

The order of words in sentences changes when translated into Kazakh:

I [1] go [2] to university [3] of Kazakhstan [4] - Men [1] Kazakstannyng [4] universitetine [3] baramyn [2].

As you can see from the example, the sequence of VP, PP, GenP phrases changes during translation.

Post-chunk Stage

The post-chunk module detects single chunks and performs the specified actions for each of them. Detection is based on a chunk lemma, not in a template (not in tags); this invokes a discovery in this module to make it specific for each "name" chunk. On the other hand, detection and processing in rules is based on the fact that refers to parameters resolved after detection, that is, tags <1>, <2>, etc., are automatically replaced with the value of the parameters before processing starts. Positions (pos attribute) specified in a statement such as <clip> refer to the positions of words within a chunk.

Also, the case policy is automatically applied from the chunk pseudo lemma to the words inside the chunk.

This module has its own specification file, which will have the extension .t3x. Its syntax is based as well on the chunker and the interchunk.

The file format is the same as for the first stage - chunker. Rule file format:

```
<postchunk>
<section-def-cats>
  <def-cat n="sent"> <cat-item name="punt"/>  </def-cat>
```

```
</section-def-cats>
<section-def-attrs>
  <def-attr n="nbr">
        <attr-item tags="sg"/>
        <attr-item tags="pl"/>
        <attr-item tags="sp"/>
        <attr-item tags="ND"/>
  </def-attr>
</section-def-attrs>
<section-def-vars>
  <def-var n="paraula"/>
</section-def-vars>
<section-rules>
<rule comment="CHUNK:">
<pattern> <pattern-item n="sent"/> </pattern>
<action>
  <out>
        <lu> <clip pos="1" part="whole"/> </lu>
  </out>
</action>
</rule>
</section-rules>
</postchunk>
```

Lexical dictionaries:

1.monolingual dictionary for the first language of a language pair: used for morphological analysis and morphological generation (* .dix, * .lexc, * .twol)

2. Monolingual dictionary for the second language of the language pair: used for morphological analysis and morphological generation (* .dix, * .lexc, * .twol)

3. bilingual dictionary: used for lexical conversion (* .dix)

Rule dictionaries:

1. a dictionary of rules for determining parts of speech (* .rlx)
2. a dictionary of rules for lexical choice (* .lrx)
3. a dictionary of rules for the chunker (* .t1x)
4. dictionary of rules for interchank (* .t2x)
5. a dictionary of rules for post-chunk (* .t3x)
6. a dictionary of rules for corrections (* .t4x). [6]

References

1. Helsinki Apertium Workshop/Programme. URL: https://wiki.apertium.org/wiki/Helsinki_Apertium_Workshop/Programme.

2. Open platform for machine translation. URL: https://www.apertium.org/index.rus.html?dir=eng-cat#translation.

3. Apertium. URL: https://wiki.apertium.org/wiki/Apertium.

4. Apertium. URL: https://en.wikipedia.org/wiki/Apertium.

5. M. L. Forcada, B. I. Bonev, S. O. Rojas, J. A. P. Ortiz, G. R. Sanchez, F. S. Martinez, C. Armentano-Oller, M. A. Montava, and F. M. Tyers. Documentation of the Open-Source Shallow-Transfer Machine Translation Platform Apertium. – 2010. – 214 pp.

6. Development of a free / open system of machine translation from Kazakh into English and Russian (and vice versa) based on the Apertium platform: Research report (final) / leader: Tukeyev U.A. – Almaty, 2015. – 134 p. – No. SR 0115RK00778 (in Russian).