Integrating Polish LFG with External Morphology

Katarzyna Krasnowska-Kieraś and Agnieszka Patejuk

Institute of Computer Science
Polish Academy of Sciences
E-mail: `kasia.krasnowska@gmail.com, aep@ipipan.waw.pl`

Abstract

This paper presents a previously undocumented approach to combining an extensive LFG grammar, employed in the construction of an LFG parsebank, with an exhaustive external morphological component. It shows how the Polish morphological analyser Morfeusz is plugged into the XLE grammar architecture as a basis for tokenisation and morphological analysis steps. The proposed solution also takes into account phenomena such as the treatment of MWEs and abbreviations. Finally, it is demonstated how the tokeniser and analyser components interact with the grammar rules.

1 Introduction: problem and previous solutions

This paper presents a previously undocumented approach to integrating an exhaustive external morphological component, Morfeusz[1] [15], with an extensive XLE/LFG grammar for Polish, POLFIE [5], in an effective and economic way. The proposed method is general and could be adapted for use by other XLE/LFG grammars.

The grammar is employed in the construction of an LFG treebank of Polish [6], more specifically, a "parsebank" containing both constituency structures and functional structures which feature dependency-like information, among other linguistic information. To obtain valence information, POLFIE uses converted entries from Walenty [9] – a state-of-the-art valence dictionary of Polish.

The previous solution, briefly described in [5], used a Python script to create lexicon files containing the entries for the exact forms (rather than lemmata) found in the sentence to be parsed – such entries bypass morphology (`*` morphcode

[†]The authors are greatly indebted to Paul Meurer, who kindly shared his implementation of the C library interface for Georgian, which, in the absence of documentation of this feature of XLE, made it possible to develop the current solution for Polish. The authors are also grateful to John Maxwell III for discussion of tokenisation and morphology issues in XLE. The usual disclaimers apply.

[1]In this paper, unless explicitly stated otherwise, Morfeusz refers to Morfeusz 2 – the recent reimplementation of the original Morfeusz [14].

specification in XLE). The information about segmentation and morphosyntactic interpretation of identified segments could be taken from a variety of sources: the previous version of Morfeusz (the predecessor of Morfeusz 2), where segmentation is sometimes ambiguous, while the morphosyntactic interpretation of segments is often highly ambiguous, or from XML files from Składnica treebank [12, 16] or the National Corpus of Polish (NKJP; [8]), which provide unambiguous information about segmentation and morphosyntactic interpretation. The script runs in two modes: interactive, where it runs XLE, intercepts the sentence to be parsed, creates a dedicated lexicon and passes the sentence to XLE for parsing, or in batch mode, where it creates a lexicon for the provided list of sentences – either one file for all sentences or individual files for particular sentences; the obtained lexicon files can be used subsequently with XLE for parsing.

Although such a solution is satisfactory for the specific and restricted purposes of creating subsequent versions of the Polish parsebank when using the disambiguated information from Składnica or NKJP, it is suboptimal for parsing running, unprocessed text: it is incapable of handling ambiguous segmentation (it uses heuristics to choose one segmentation) and, while it can be used with XLE (as described above), it cannot be used to make the grammar available via XLE-Web – an INESS ([10]; `http://iness.uib.no/`) web-service for parsing using XLE: the Python script for creating the lexicon on the fly cannot be used in XLE-Web (without introducing modifications in INESS) and a lexicon containing all Polish forms is too big to load (not to mention doing so in reasonable time limits).[2]

It was therefore decided to devise a solution following the general architecture assumed in XLE, which requires specifying transducers that will handle tokenisation and morphological analysis of an input sentence. It is a common practice in LFG grammars developed within the XLE framework to build such a transducer using the XFST tool [1]. Since a high-quality, effective tool that is well-adjusted to Polish is available, such a solution would require a lot of redundant work whose outcome is not guaranteed to be of comparable quality. Instead, an alternative solution was chosen: to use a programming interface provided within XLE that makes it possible to implement a wrapper library in C/C++ that passes the output of an external morphological tool on to the grammar.

2 Interfacing Morfeusz

The basis of the morphological component for POLFIE is Morfeusz, a state-of-the-art morphological analyser for Polish. Morfeusz is built on the grammatical description and linguistic data of *Grammatical Dictionary of Polish* (SGJP; [11]). Its default inflectional dictionary (mapping between word forms and morphological interpretations, i.e. ⟨lemma, tag⟩ pairs), derived from SGJP, contains over 4,000,000 word forms belonging to over 250,000 lemmata. Another crucial com-

[2]In preliminary experiments, soon abandoned due to excessive size of the resulting files, a full-form lexicon for 8 740 most frequent lemmata in a corpus was a 13.4 GB file.

ponent of Morfeusz is its set of hand-crafted segmentation rules, which allow, for instance, to account for situations where some elements (mostly clitics) are treated as separate segments even though they are not separated by whitespace characters.

Segmentation rules also increase the coverage of the analyser beyond dictionary-defined words by providing a limited derivational component. As an example, the words *europoseł* ‘member of the European Parliament’ or *hiperaktywny* ‘hyperactive’ are not explicitly accounted for in the inflectional dictionary of Morfeusz. Instead, the dictionary contains prefixes EURO- and HIPER-, and the segmentation rules admit composing those prefixes with any noun or adjective. In conjunction with the presence of noun POSEŁ ‘MP’ and adjective AKTYWNY ‘active’ in the dictionary, the whole mechanism makes it possible to correctly analyse *europoseł* and *hiperaktywny*.

Apart from its vast coverage and reliable linguistic data, Morfeusz introduced another very advantageous feature. In its previous version, the analyser was strictly bound to one dictionary and the segmentation rules were hard-coded into its implementation. The reimplementation, however, has a more flexible architecture, making it possible to provide one’s own dictionary and/or segmentation rules instead, either obtained by a modification of the default ones, or constructed from scratch. Morfeusz can be used either as a stand-alone program or, more conveniently from a programmer’s point of view, its core library can be called directly from C/C++ or Python code. This section presents a morphological component for the Polish LFG grammar that uses all those features of Morfeusz, while keeping in line with the grammar architecture of XLE framework.

SGJP serves as a basis for the tagset of NKJP [7], adopted in turn by POLFIE. Although similar, the tagset diverges from SGJP in several respects. This is where the aforementioned flexibility of Morfeusz proves very useful – it makes it possible to introduce some grammar- and tagset-specific modifications to the original, SGJP-based dictionary. These include some systematic changes (such as the reduction of SGJP’s 9 grammatical genders to NKJP’s 5 or a different analysis of some numerals) as well as a few word-specific adjustments. The modified version of the dictionary is kept consistent with updates of the original one. The compiled dictionary’s size is about 7.6 MB.

The XLE grammar architecture assumes two steps of processing an input sentence before it is analysed using the grammar rules. The first step is tokenisation: a string of characters is divided into tokens representing particular words. Each token output at this stage is subsequently passed on to the morphological analysis step, where it is associated with a word lemma and morphological tags. The form of the token determines the string that appears in the corresponding LFG c-structure leaf, whereas the morphological information is used in the grammar rules to construct an appropriate analysis. Such a division of tasks between the tokeniser and the analyser is quite the opposite of the architecture of Morfeusz. Due to inflectional features and orthographic rules of Polish, morphological interpretation of some segments depends on the segmentation itself – it is therefore natural and convenient to tightly couple the two steps. As a result, Morfeusz processes an input text in a single run, yielding a so-called morphological analysis graph, representing the (possibly ambiguous) segmentation of the text together with the possible morphological interpretations of particular segments. Figure 1 shows an analysis graph produced by Morfeusz for the input text *Czym rzuciła?*

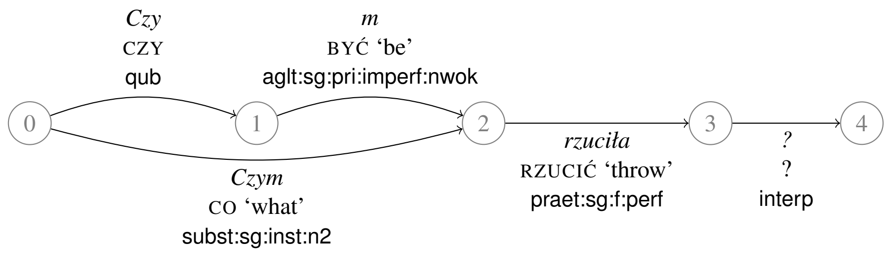

Figure 1: Morfeusz analysis of *Czym rzuciła?*

Depending on the chosen segmentation, the question has two readings:

(1) *Czym* *rzuciła?*
what.SG.INST.N threw.SG.F
'What did she throw?'

(2) *Czy* *m* *rzuciła?*
QPART SG.1 threw.SG.F
'Did I throw (something)?'

The interpretation of the sentence *Czym rzuciła?* depends on whether the word *Czym* is analysed as one or two segments: in (1) it is a form of the interrogative pronoun CO 'what'. By contrast, in (2) the word *Czym* is split into two segments, where the first is a *yes/no* question marker (*Czy*), while the other is an agglutinate form of the verb BYĆ 'be' – it carries information about the person (first) and number (singular) of the subject of the lexical verb (*rzuciła*).

In order to adapt Morfeusz for use with the grammar, two wrapper libraries implementing the interface required by XLE were created. Both libraries make internal calls to Morfeusz, but they process the results differently in order to provide separate tokeniser and analyser functionality as expected by XLE.[3] The tokeniser takes as its input the whole sentence to be parsed, and uses Morfeusz to obtain its segmentation. The analysis graph produced by Morfeusz is translated into a regular expression as specified by XLE's interface, taking into account any ambiguities at the level of segmentation. The tokeniser library also deals with comma haplology as proposed in [4].[4] For example, the regular expression for the segmentation

[3]The current solution does not include any guesser functionality: only words recognised by Morfeusz are given morphosyntactic analyses.

[4]The general idea is to assume, in grammar rules, that all parenthetical clauses are delimited by commas both at the beginning and at the end. In written text this is usually not the case due to orthographic rules, therefore, the tokeniser inserts "optional commas" where they could be expected by the grammar. Following an example for English from [4], the sentence *Find the dog, a poodle.* (compare with *Find the dog, a poodle, now!* where both delimiting commas are present in written text) can thus obtain an alternative tokenisation ‹Find› ‹the› ‹dog› ‹,› ‹a› ‹poodle› ‹,› ‹.›, allowing the parser to treat *a poodle* as a comma-delimited clause. For simplicity, the inserted commas are not shown in the discussed example.

provided by the analysis from Figure 1 would be:

(3) `(‹Czy› ‹+m›|‹Czym›) ‹rzuciła› ‹?›`

In this example, the segmentation ambiguity is reflected by the alternative `(‹Czy› ‹+m›|‹Czym›)`, and the whole expression encodes the two possible segmentations of the sentence *Czym rzuciła?*:

(4) `‹Czy› ‹+m› ‹rzuciła› ‹?›` (5) `‹Czym› ‹rzuciła› ‹?›`

The tokens output by the tokeniser are then, separately, passed by XLE to the analyser library. The analyser, in turn, once again runs Morfeusz, this time on individual tokens. Such architecture introduces an artificial division of the actions normally carried out simultaneously by the Polish morphological analyser. One negative consequence of such a solution is that when Morfeusz is called from the analyser library, it only has access to a single segment, without the context provided by surrounding text that would normally be available to segmentation rules implemented in Morfeusz. In order to prevent information loss between the two stages, some auxiliary modifications were introduced at the tokenisation and analysis level.[5] The result of analysing the individual tokens from the example *Czym rzuciła?* would be the following morphology outputs:

(6)
```
czy +qub                      (‹Czy›)
być +aglt:sg:pri:imperf:nwok  (‹+m›)
co +subst:sg:inst:n2          (‹Czym›)
rzucić +praet:sg:f:perf       (‹rzuciła›)
?  +interp                    (‹?›)
```

In order to illustrate the benefit from incorporating the Polish-specific segmentation mechanism of Morfeusz into the grammar, the LFG analyses for the input text `Czym rzuciła?` obtained using two different variants of POLFIE are discussed below.[6] The two variants produce the same number of analyses, identical f-structures and identical c-structuresas far non-terminal nodes are concerned. However, there is an important difference at the level of c-structure terminals.

The first variant uses the wrapper analyser library with a tokeniser originally implemented in XFST by Ron Kaplan for the English LFG grammar (see [4]). Since this tokeniser cannot divide a word like *Czym* into two tokens (the only tokenisation being `‹Czym› ‹rzuciła› ‹?›`), additional segmentation is performed at the morphological analysis stage. For any token passed to the analyser library that can be interpreted as more than one segment, the ambiguous segmentation is retrieved from Morfeusz and reflected in the morphology outputs:

[5]One such modification is adding '+' to *m* in the discussed tokenisation example – this solution retains the information that the segment is a clitic and therefore makes it possible to block its "stand-alone" interpretation (METR 'metre').

[6]INESS' XLE-Web component was used for visualising and disambiguating the structures.

(7) `czy +qub być +aglt:sg:pri:imperf:nwok|co +subst:sg:inst:n2`
`rzucić +praet:sg:f:perf`
`? +interp`

Note that though, in this way, all the correct lemmata and morphosyntactic tags are obtained, there is no means of associating them with different tokenisations of the input sentence. All three ⟨lemma, tag⟩ pairs generated for the token ‹`Czym`› (first row of (7)) are mapped by XLE to that same token. The result is that, with the analysis for interpretation (2) chosen, corresponding to the `czy +qub być +aglt:sg:pri:imperf:nwok` option, both preterminal c-structure nodes, QUB[int] and AGLT, are associated with the terminal *Czym* (see Figure 2).

In the second variant, which is the solution offered in this paper, both tokeniser and analyser libraries use Morfeusz. Since the tokeniser has already handled any segmentation ambiguities, the analyser library is configured to treat any token (with an exception explained in Section 3) as "unambiguous": even if the analysis from Morfeusz contains ambiguous segmentation (as in the case of ‹`Czym`›), only the morphological interpretations pertaining to the whole token string are returned.[7] In this way, only the ‹`Czy`› and ‹`+m`› tokens (not ‹`Czym`›) contribute to the `czy +qub` and `być +aglt:sg:pri:imperf:nwok` interpretations respectively. The resulting LFG analysis for interpretation (2) is presented in Figure 3.

Figure 4 shows the analysis for interpretation (1), where *Czym* is one segment. Since this interpretation does not involve any Polish-specific segmentation phenomena, both variants of the grammar yield identical structures.

3 Multi-word expressions

The current version of POLFIE supports a small number of MWEs. The plans for further development of the grammar include enlargement of this stub MWE component using information gathered from resources such as Walenty or SEJF (an MWE dictionary, [3]). It is not obvious that all Polish MWEs should be analysed at the tokenisation and morphology levels, as is often assumed in LFG grammar architectures. Many MWEs are clearly compositional syntactically, thought not semantically. One might therefore want to analyse their syntactic structure, and use the resources mentioned above to mark them as semantically non-compositional for the purposes of future semantic processing. It seems, nevertheless, uncontroversial that some types of MWEs, mostly the fixed, non-inflecting sequences, can and should be handled at the stage of tokenisation and morphological analysis. The currently supported MWEs, mostly belonging to closed grammatical classes (such as conjunctions, complementisers and prepositions), have such characteristics – it was therefore decided to analyse them in the relevant libraries.

[7]The interpretations themselves can nevertheless be ambiguous and they mostly are due to homonymy and ubiquitous syncretism in Polish. For example, the output for the token ‹`pośle`› would be `posłać +fin:sg:ter:perf|poseł +subst:sg:loc:m1|poseł +subst:sg:voc:m1` since the word *pośle* could be a form of either POSŁAĆ 'to send' or POSEŁ 'MP'.

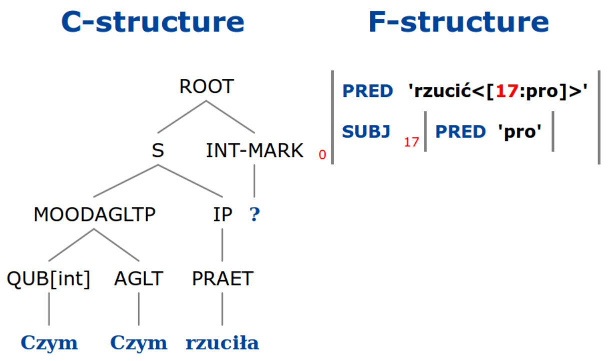

Figure 2: *Czym rzuciła?* – structures for interpretation (2), Ron Kaplan's tokeniser.

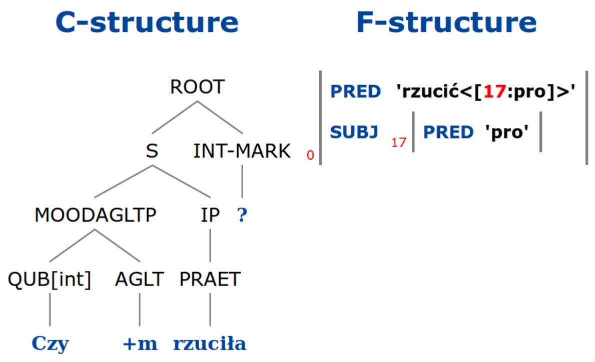

Figure 3: *Czym rzuciła?* – structures for interpretation (2), Morfeusz-based tokeniser.

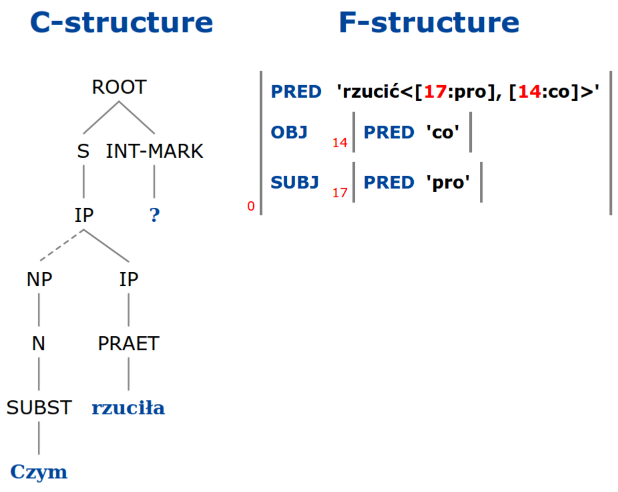

Figure 4: *Czym rzuciła?* – structures for interpretation (1).

Since Morfeusz, by design, does not admit segments longer than those delimited by whitespace characters, it is not possible to analyse MWEs by specifying them in Morfeusz dictionary. A small dictionary of multi-word expressions that is used by the tokeniser and analyser libraries was created for this purpose. For each MWE to be analysed, there is a specification of the lemmata and morphosyntactic tags of its component words as well as the lemma and morphosyntactic tag of the whole expression.[8] Two examples of entries in the dictionary are:

(8) `w +prep:loc:nwok czas +subst:sg:loc:m3 : 'w czasie' +prep:gen`

(9) `a +conj nie +qub : 'a nie' +conj`

The entry in (8) corresponds to W CZASIE 'during, lit. in time (of)', which is treated in POLFIE as a preposition selecting for the genitive case. The left-hand side of this entry specifies a sequence of two tokens having `w +prep:loc:nwok` and `czas +subst:sg:loc:m3` respectively among their Morfeusz analyses. Whenever such a sequence is encountered, it obtains an additional analysis as one token,[9] with a morphological interpretation `'w czasie' +prep:gen`. The entry in (9) corresponds to the conjunction A NIE 'but/and not'.

The general architecture of tokenisation and morphology libraries in XLE and the way how Morfeusz handles whitespaces makes introducing some redundancy into the analysis of MWEs inevitable. First, the tokeniser library tries to match the consecutive segments returned by Morfeusz with the MWE entries. Whenever a match is found, the segments obtain an alternative tokenisation as a single token. For example, the sentence (10) would obtain the tokenisation in (11):

(10) *W czasie podróży dużo czytał.*
in.LOC time.LOC travel.GEN much read
'During the travel, he read a lot.'

(11) `(‹W› ‹czasie›|‹W czasie›) ‹podróży› ‹dużo› ‹czytał› ‹.›`

Second, if the analyser library is given a token containg a space, it tries once again to match its analysis with the sequences of interpretations specified in the MWE dictionary, this time using the second part of a matched entry to provide a suitable lemma and morphosyntactic tag.

4 Abbreviations

Another type of token that is handled in a special way are abbreviations. Morfeusz makes a distinction between punctuated abbreviations (assigned a `brev:pun` tag and required to be followed by a period) and non-punctuated ones (tagged as

[8]The reason for operating on lemmata and tags returned by Morfeusz rather than simply specifying the MWE's text form is that the former solution makes it possible to benefit from the analyser's robust handling of lowercase/uppercase orthographic variations.

[9]The analysis as separate tokens is also kept as it could also be valid.

`brev:npun`). Both types of abbreviations are lemmatised to the lemma of their corresponding full form. For instance, the word *ul* can be interpreted either as a form of UL 'beehive' or, when followed by a period, as an abbreviated form of ULICA 'street'. Polish orthographic rules generally require that abbreviations must be punctuated if their last letter is different from the last letter of the word form they stand for, it therefore follows that a non-punctuated occurrence of *ul* should not be interpreted as abbreviated form of ULICA – it can only be a form of UL. The segmentation rules of Morfeusz take this into account: its analyses of *ul* and *ul.* are shown in Figure 5. Another example of a punctuated abbreviation is *prof.* 'professor'. Since, unlike *ul*, the word *prof* has no other meanings in Polish, its non-punctuated occurrences are marked by Morfeusz as unrecognised by assigning them an `ign` (ignotum) tag. Analyses of *prof* and *prof.* are shown in Figure 6. The non-punctuated abbreviations, such as *wg* (WEDŁUG 'according to') are interpreted as `brev:npun` in any context.

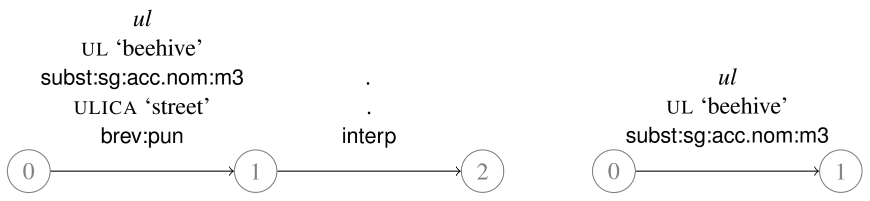

Figure 5: Morfeusz analyses of *ul.* (left) and *ul* (right).

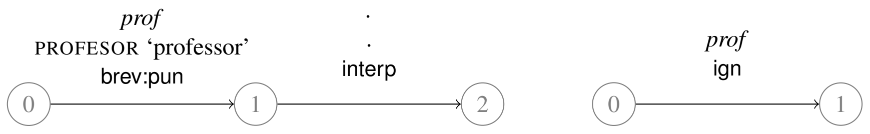

Figure 6: Morfeusz analyses of *prof.* (left) and *prof* (right).

The way Morfeusz analyses abbreviations is incompatible with POLFIE grammar in two respects. First, the grammar requires abbreviations to be assigned not only the lemma, but also morphosyntactic tags of their full forms. For instance, the expected analysis of *wg* is `według +prep:gen`, not `według +brev:npun`. Second, the grammar rules assume that a comma following a punctuated abbreviation is a part of its token, not a separate one. For instance, the expected tokenisation of *ul* is ‹ `ul.` ›, not ‹ `ul` › ‹ `.` ›.[10]

The first issue is addressed by modifying the default dictionary of Morfeusz: the `brev:pun`/`brev:npun` entries are replaced with full morphosyntactic tags of the corresponding forms. For this purpose, a mapping analogous to the one defined in the grammar of *Świgra*, a DCG parser of Polish [13], is used.

[10]Such a way of analysing abbreviations is implemented in Ron Kaplan's tokeniser that was used in the previous version of POLFIE.

The second issue requires more complicated modifications, covering also the problem of period haplology (a period following a punctuated abbreviation can at the same time be a sentence-ending period, see [4]). The proposed solution works as follows: the first step is to recognise fragments of the analysis graph returned by Morfeusz that correspond to punctuated abbreviations and their following periods. In case of a segment that only has an abbreviation interpretation, like *prof*, the period is appended to its token, and an additional, optional period is added (marked with `?`; it would only be consumed by the grammar rules when occurring at the end of a sentence). For instance, the tokenisation for *prof.* would be `‹prof.› ‹.›?`. For segments that also have any non-abbreviation interpretations, the original segmentation from Morfeusz is kept as an alternative tokenisation variant. As an example, the tokenisation for *ul.* would be `‹ul.› ‹.›?|‹ul› ‹.›`.

Second, an auxiliary Morfeusz dictionary is introduced – it is dedicated specifically to punctuated abbreviations. In this dictionary, the entries for those abbreviations contain periods at their ends, and therefore the corresponding tokens passed from the tokeniser library can be analysed as required by the grammar. The reason for introducing this additional dictionary is to avoid interference with the original segmentation mechanisms of Morfeusz that are preserved in the "main" dictionary and make it possible to properly recognise abbreviation-punctuating periods in the tokeniser library. The main and auxiliary dictionaries are combined in the `ANALYZE USEFIRST` section of the morphology configuration.

5 Adapting the grammar

To use morphology in XLE, one must create sublexical rules – the right-hand side contains the stem (`Vsub_BASE`; verbal stem), which introduces constraints appropriate for the given lemma (this includes valence information – arity of the predicate and associated constraints, if any – as well as lexicalised constraints, if applicable), and tags returned by the analyser (`Vpraet_SFX_BASE`; l-participle tag), which introduce morphosyntactic information associated with the given form; the left-hand side corresponds to the resulting category (`PRAET`; l-participle):

(12) `PRAET --> Vsub_BASE   Vpraet_SFX_BASE.`

To use such a rule, lexical entries must be created for stems and tags used in sublexical rules. The example provided in (13) is the lexical entry for RZUCIĆ 'throw' – the first field corresponds to the lemma (`rzucić`), the second one is the category (`Vsub`, verbal), the third is the morphcode signalling that it passes through morphology (`XLE`, unlike `*`) and the last one contains constraints imposed by the particular entry – in this case it is a call to the template `@(rzucić-Walenty)`, which contains valence schemata appropriate for this particular verb:

(13) `rzucić   Vsub   XLE   @(rzucić-Walenty).`

In POLFIE, valence templates are zero-argument templates – this is why the template call such as `@(rzucić-Walenty)` consists of the template name exclusively.

Each valence template definition rewrites to a disjunction of converted valence schemata from Walenty for the given lemma, where each such disjunct consist of two parts: the specification of the `PRED` attribute, which lists arguments of the relevant predicate (provided below), and a set of constraints related to these arguments (not shown below due to space limits – `[constraints]` is a placeholder).

(14)
```
rzucić-Walenty =
{ (^ PRED)='rzucić<(^ SUBJ)(^ OBJ-TH)>' [constraints]
| (^ PRED)='rzucić<(^ SUBJ)(^ COMP)>' [constraints]
| ... }.
```

Next, lexical entries are created for tags returned by the analyser:

(15)
```
+praet:sg:f:perf   Vpraet_SFX   XLE   @(PRAET sg f perf).
```

The structure of lexical entries of tags is the same as discussed above: the first field in (15) is the full morphosyntactic tag (`+praet:sg:f:perf`), the second is the category (`Vpraet_SFX`), the third is the morphcode marking that it passes through morphology (`XLE`) and the last one introduces constraints (`@(PRAET sg f perf)` is a template call, see the following discussion).

The approach to tags presented here seems to be different from mainstream one in that the analyser returns a single morphosyntactic tag for each interpretation of a particular segment – these tags are created in accordance with the NKJP tagset [7], a positional tagset where the first element of the tag (tag parts are separated by the `:` symbol) is the part of speech (`praet`, l-participle, in (15)) and then values of morphosyntactic categories appropriate for it (if any) follow: in (15) these include `sg` for singular number, `f` for feminine gender and `perf` for perfective aspect.

Using such "glued" tags makes it possible to rewrite them directly to calls to part of speech templates: the entry of `+praet:sg:f:perf` contains the call `@(PRAET sg f perf)`, where particular parameters of the `PRAET` template set appropriate values of relevant attributes (as discussed above). This makes it possible to avoid creating separate lexical entries for values of the same attribute used with different parts of speech – for instance, both verbs and nouns are specified for number, but under the mainstream LFG analysis number in nouns sets the number value of the noun, but in verbs it introduces a constraint on the number of the verb's subject (rather than the verb itself). Under the current solution such differences are accounted for inside the definitions of particular part of speech templates.

The last issue is creating the lexical entries for lemmata – as mentioned above, lexical entries introduce valence constraints and possibly lexicalised constraints. The current grammar makes extensive use of the `-unknown` lexical entry, which serves the purpose of creating lexical entries for lemmata not listed in the lexicon because their behaviour is fully regular – as opposed to items introducing lexicalised constraints, which must be listed explicitly in the lexicon. Below is a fragment of the `-unknown` entry handling adjectives (`Asub`) and adverbs (`ADVsub`):

(16)
```
-unknown   Asub     XLE   @(ZERO-OR-PRED %stem);
           ADVsub   XLE   @(ZERO %stem).
```

The first subentry (Asub) introduces zero-argument (attributive) or one-argument SUBJ (predicative) subcategorisation for adjectives; the second one (ADVsub) assigns zero-argument valence specification for adverbs. Since the subentries of -unknown are matched against the category of the segment, which is in turn determined by the interpretation from the analyser (returned tags), there is no risk of assigning adverbial subcategorisation to a segment which is not an adverb.

As a result, only "special" lemmata are listed in the lexicon – these include, for instance, *n*-words, which introduce constraints looking for sentential negation in the relevant domain (in Polish, *n*-words need to be licensed).

6 Conclusion

This paper offered a method of integrating the Polish LFG grammar with a state-of-the-art morphological analyser Morfeusz. It improves the grammar and the parsebank by making it possible to fully use the Polish-specific mechanisms, such as ambiguous segmentation, implemented in Morfeusz. At the same time, it keeps the grammar compliant with XLE architecture and compatible with XLE-based tools.

Acknowledgements

This work is partially financed by the Polish Ministry of Science and Higher Education within the CLARIN ERIC programme 2015–2016 (http://clarin.eu), and partially by the European Union from resources of the European Social Fund *Project PO KL "Information technologies: Research and their interdisciplinary applications"*.

References

[1] Kenneth R. Beesley and Lauri Karttunen. *Finite State Morphology*. CSLI Publications, Nancy, France, 2003.

[2] Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors. *Proceedings of the Ninth International Conference on Language Resources and Evaluation, LREC 2014*, Reykjavík, Iceland, 2014. ELRA.

[3] Monika Czerepowicka and Agata Savary. SEJF – a grammatical lexicon of Polish multi-word expressions. In Zygmunt Vetulani and Joseph Mariani, editors, *Proceedings of the 7th Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics*, pages 224–228, Poznań, Poland, 2015.

[4] Ron Kaplan, John T. Maxwell, Tracy Holloway King, and Richard Crouch. Integrating finite-state technology with deep LFG grammars. In *Proceedings of the Workshop on Combining Shallow and Deep Processing for NLP (ESSLLI)*, 2004.

[5] Agnieszka Patejuk and Adam Przepiórkowski. Towards an LFG parser for Polish: An exercise in parasitic grammar development. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation, LREC 2012*, pages 3849–3852, Istanbul, Turkey, 2012. ELRA.

[6] Agnieszka Patejuk and Adam Przepiórkowski. Synergistic development of grammatical resources: a valence dictionary, an LFG grammar, and an LFG structure bank for Polish. In Verena Henrich, Erhard Hinrichs, Daniël de Kok, Petya Osenova, and Adam Przepiórkowski, editors, *Proceedings of the Thirteenth International Workshop on Treebanks and Linguistic Theories (TLT 13)*, pages 113–126, Tübingen, Germany, 2014. Department of Linguistics (SfS), University of Tübingen.

[7] Adam Przepiórkowski. A comparison of two morphosyntactic tagsets of Polish. In Violetta Koseska-Toszewa, Ludmila Dimitrova, and Roman Roszko, editors, *Representing Semantics in Digital Lexicography: Proceedings of MONDILEX Fourth Open Workshop*, pages 138–144, Warsaw, 2009.

[8] Adam Przepiórkowski, Mirosław Bańko, Rafał L. Górski, and Barbara Lewandowska-Tomaszczyk, editors. *Narodowy Korpus Języka Polskiego [Eng.: National Corpus of Polish]*. Wydawnictwo Naukowe PWN, Warsaw, 2012.

[9] Adam Przepiórkowski, Elżbieta Hajnicz, Agnieszka Patejuk, Marcin Woliński, Filip Skwarski, and Marek Świdziński. Walenty: Towards a comprehensive valence dictionary of Polish. In Calzolari et al. [2], pages 2785–2792.

[10] Victoria Rosén, Koenraad De Smedt, Paul Meurer, and Helge Dyvik. An open infrastructure for advanced treebanking. In Jan Hajič, Koenraad De Smedt, Marko Tadić, and António Branco, editors, *META-RESEARCH Workshop on Advanced Treebanking at LREC2012*, pages 22–29, Istanbul, Turkey, 2012.

[11] Zygmunt Saloni, Marcin Woliński, Robert Wołosz, Włodzimierz Gruszczyński, and Danuta Skowrońska. *Słownik gramatyczny języka polskiego*. Warsaw, 2nd edition, 2012.

[12] Marek Świdziński and Marcin Woliński. Towards a bank of constituent parse trees for Polish. In *Text, Speech and Dialogue: 13th International Conference, TSD 2010, Brno, Czech Republic*, Lecture Notes in Artificial Intelligence, pages 197–204, Berlin, 2010. Springer-Verlag.

[13] Marcin Woliński. *Komputerowa weryfikacja gramatyki Świdzińskiego*. Ph.D. dissertation, Institute of Computer Science, Polish Academy of Sciences, Warsaw, 2004.

[14] Marcin Woliński. Morfeusz — a Practical Tool for the Morphological Analysis of Polish. In Mieczysław Kłopotek, Sławomir T. Wierzchoń, and Krzysztof Trojanowski, editors, *Intelligent information processing and web mining*, pages 503–512. Springer-Verlag, 2006.

[15] Marcin Woliński. Morfeusz reloaded. In Calzolari et al. [2], pages 1106–1111.

[16] Marcin Woliński, Katarzyna Głowińska, and Marek Świdziński. A preliminary version of Składnica—a treebank of Polish. In Zygmunt Vetulani, editor, *Proceedings of the 5th Language & Technology Conference: Human Language Technologies as a Challenge for Computer Science and Linguistics*, pages 299–303, Poznań, Poland, 2011.