

Towards a Set of General Purpose Morphosyntactic Tools for Polish*

Bartosz Broda, Maciej Piasecki, and Adam Radziszewski

Institute of Applied Informatics, Wrocław University of Technology, Poland

Abstract

Morphological processing of Polish is seriously hampered by the poor availability of general-purpose tools. This article presents an attempt to create such a set of tools following the *de facto* standard of the IPIC corpus. Currently, the package contains pieces of software able to perform the following tasks: text tokenisation, morphological analysis with heuristics for unknown words, division into sentences and morphosyntactic disambiguation. The described tools will be made available under the GNU general public licence.

Keywords: Polish, morphological processing, disambiguation, tools

1 Introduction

The complex issue of text processing aimed at analysis of information conveyed by text is typically divided into several steps. One can notice a set of the subsequent steps of processing, which is limited and becoming gradually standardised e.g. the BLARK initiative (Mapelli and Choukri, 2003). The existence of a set of typical language tools for a given language facilitates, or even makes possible, the application of well know solutions to text processing and faster going beyond them. A notable constraint is imposed by the poor availability of general purpose tools. Not only is it caused by the lack of language resources but also by blocking access to existing tools by high cost or very restrictive copyright policy.

As some steps of processing Polish text lack tools and for the others the usage of existing tools is constrained by several limitations (standards, completeness, access etc.), our goal is to build a complete set of universal, publicly accessible language tools for Polish in cooperation with everyone who wants to participate.

In the following sections we are going to describe the typical scheme of text processing up to the syntactic level, present the state of development achieved so far, and discuss the necessary and intended directions of future progress.

2 General Scheme of Processing

A typical scheme of morphosyntactic text processing can be described as consisting of:

*This work was financed by the Ministry of Education and Science the projects No 3 T11E 005 28 and No 3 T11C 018 29.

1. *tokenisation* — the initial text segmentation into the basic, atomic text units (including potential word forms, numbers, symbols, etc.),
2. *morphological analysis* — morphosyntactic description of word form tokens,
3. *morphological guessing* — prediction of morphosyntactic description for potential word forms, which have not been recognised during the morphological analysis,
4. *segmentation* into sentences — division of text into sentences and sentence-like subsequences of tokens (the complete process of text segmentation can encompass many additional granularities of division),
5. *morphosyntactic disambiguation* — selection of the most appropriate description for each word form token, which is ambiguous in relation to morphosyntactic description.

The construction of language tools is mostly based on specific types of language resources. The tools in the steps 1 – 5 require existence of a morphosyntactically annotated and manually disambiguated corpus. This condition is only partially fulfilled in the case of Polish. Fortunately, the IPI PAN Corpus (henceforth, IPIC) (Przepiórkowski, 2004) is morphosyntactically annotated, but only a small part of it (885 669 tokens) has been manually disambiguated (MIPIC); what is more, this manual disambiguation still seems to contain some erroneous annotations. The vast majority of IPIC 2.0 has been automatically disambiguated by the application of TaKIPI tagger discussed later in this paper. Segmentation into tokens and sentences has been also done automatically, except MIPIC.

3 Segmentation into Tokens

Tokenisation in IPIC and TaKIPI 1.0 (Piasecki, 2007) depends exclusively on a simple space-to-space rule followed next by the word form segmentation done by the morphological analyser *Morfeusz* (Woliński, 2006). Dates, numbers and URL addresses, as including punctuation marks, are divided by *Morfeusz* into several segments, while they should arguably be kept together as single tokens. Moreover, they are annotated by the tag **ign** denoting an unrecognised segment, i.e. accidental sequences of characters, digits and any other strings not encoded into *Morfeusz*; all such strings will be identical from the perspective of tools depending on tags.

In order to amend this disadvantage, a tool called *Tokenizer* was introduced as the first tool in the processing chain. Its task is to divide the input text into potential word forms and non-word-form tokens¹, and next to annotate the tokens with a set of predefined tags. Besides grammatical classes, the IPIC tagset includes only tags for punctuation (**interp**), unrecognised segment (**ign**), and two tags for foreign words (**xxx** and **xxs**). For practical applications, it seems reasonable to have more token classes of non-grammatical nature. Such classes would have no grammatical categories as bearing no morphological interpretation; they would rather serve a purpose of distinguishing between unrecognised sequences of

¹Precisely, *Tokenizer* divides input text into *pre-tokens* as potential word forms can be subsequently split into several tokens by *Morfeusz*.

characters and meaningful text units such as numbers. We propose an extension of the IPIC tagset with the following additional token classes:

- **tdate** — a *date* written in the format of numbers separated by ‘.’ or ‘-’; both: day, month, year and vice versa, are accepted;
- **tmail** — an *e-mail address*; the compatibility with standards is described in the **turi** point;
- **tnum** plus the **numtype** ::= **frac** | **integer** attribute — a number as a sequence of digits; **numtype** used for integer / decimal fraction distinction — the decimal symbol must be written in the Polish style, i.e. ‘.’;
- **tsym** — a continuous sequence of letters, digits and non-separating punctuation marks (see below), at least two types of signs, e.g. symbols (*czesio83*), abbreviations² (*m.in.*) etc.;
- **ttime** — *time* — precise description of time written in the format: **hh:mm**, with optional addition of seconds, e.g. *12:13*, *12:37.00*, *00:00:00*, *24.00*); only the structure is taken into account, not exact values,
- **turi** — *address* — (*identifier*) *URI/URL*³ — we did not implement the particular RFC standard, but rather we tried to establish a practical compromise between real-life URLs and character sequences whose occurrences are rather not Internet addresses;
- **twf** — *potential word form* — a token including exclusively letters and the characters: ‘’ (apostrophe) and ‘-’ (hyphen); **twf** tokens are to be removed during the morphological analysis and replaced by the appropriate IPIC tags.

The introduced token classes are encoded on the first position of the IPIC tag, i.e. now 32 grammatical classes plus 5 token classes that can occur on the initial position. The **twf** class will never occur in the annotated text, unless the morphological analysis is not performed.

Our aim was not to affect the future application by an over-detailed extension of the IPIC tagset, e.g. a month name is not treated as a date part and it is still recognised as a Polish word form. Recognition of dates including month names is left open for future language processing applications, e.g. *21 maja 2001* is segmented into three tokens. In the same way, the sign ‘-’ before a number is not treated as a part of the number but as a separated **interp** token.

Tokenizer is built as a stack of layers.

1. *Input Layer* — the input text is divided into blocks: space-to-space, i.e. space-like characters: ‘ ’, ‘\n’, ‘\r’, ‘\t’.
2. *Splitting Layer* — blocks are divided into subsequences (possibly empty) of: *pre-separators*, *post-separators*, *always-separators* and other characters (called *cores*); e.g. the block: (*"praca.pl"*) will be divided into six subsequences: (, ", *praca.pl*, ", and); *always-separators* are characters that should always constitute separate tokens, e.g. *em-dash* and *en-dash* (‘—’, ‘-’).

²Some abbreviations are recognised on the level of *Sentencer*, see Sec. 5, but are not yet analysed morphosyntactically in the present version of the tools.

³Universal Resource Identifier and Universal Resource Locator

3. *Classifying Layer* — classifies core subsequences by applying regular expressions; the subsequent regular expressions from the loaded expression list are matched against a core subsequence; in case a match is found, the given subsequence is annotated as a token of the corresponding type; if any expression was not matched, the subsequence is annotated by `tsym`.
4. *Combining Layer* — some tokens are combined into a more complex one; so far the only tasks to be done in this layer are to attach a dot to a multi-part abbreviation (e.g. *m.in.*, *op.cit.*), and to correct annotation of some abbreviations mistakenly recognised earlier as `turi` — a list of multi-part abbreviations (manually prepared for the segmentation into sentences, see Sec. 5) is used.

Regular expressions used in the *Classifying Layer* are stored in a file and can be modified in order to customise the *Tokenizer* work. The output is written in the IPIC XML format with the extended tagset.

4 Morphological Guessing

Morphological analysis is based on *Morfeusz* producing analyses in the IPIC format. However, as the coverage of *Morfeusz* is limited (although large), many potentially correct word forms are not recognised, e.g. *malpi* (*monkey-like, of monkeys*), *konstruktywność* (*constructiveness*), *playboy*. The problem is especially visible when one processes text from a specific domain.

In order to extend automatically the coverage, a morphological guesser called *Odgadywacz* (Piasecki and Radziszewski, 2007) was constructed. *Odgadywacz* uses pseudo-suffixes acquired automatically from IPIC. This approach, described below, is motivated by the observation that in inflectional languages it is basically the word ending that identifies the morphological properties of word forms.

1. A word form list was extracted from IPIC along with their frequencies. Forms belonging to closed grammatical classes, e.g. prepositions, were eliminated.
2. Every remaining form was assigned an analysis by *Morfeusz*. Unknown forms (`ign`) were eliminated.
3. Every remaining form was inverted lexicographically and feed to the tree structure. This way the branches were successively built. The resulting tree contained morphological analyses in those nodes, in which a form ended.
4. The unbranched tree paths were pruned. Whenever a branch was cut, all its tags were passed to the parent node. It corresponds to removing forms' prefixes and leaving pseudo-suffixes only — the main source of generalisation.

The resulting tree of pseudo-suffixes (*a tergo* tree) constitutes the guesser itself. The form being guessed is traversed backwards, letter by letter; the guesser's tree is traversed along. When either the tree path or the form ends, the set of analyses from the tree node reached is returned.

TaKIPI 1.5 combines *Tokenizer*, *Morfeusz* and *Odgadywacz* in one tool: only tokens annotated as `twf` by *Tokenizer* are sent first to *Morfeusz*, and if a `ign`-tag-only sequence is returned, then the token is analysed by *Odgadywacz*; the returned tag is assigned to it. *Odgadywacz* does not introduce new tags, it works

exactly on the IPIC tag set. Still, some non-word letter sequences and even word forms are left unrecognised and the `ign` tag can occur in the output.

5 Segmentation into Sentences

The proper segmentation of text into sentences would facilitate a lot further processing, but the haplology of the dot sign, e.g. (Przepiórkowski, 2004), complicates the problem. For instance, in *ul. Rynek* (*street_{abbr.} main square*) the dot interpretation is ambiguous among: the full stop (as *ul* means also *hive*), the abbreviation mark or both in the same time (a sentence ended with the given abbreviation). In order to disambiguate the dot haplology one needs to look into the morphosyntactic features of tokens in their context. As tokens are still ambiguous in this phase, so one would have to infer it from ambiguous data; but in the next step of morphosyntactic disambiguation it will be done again! Taking into account the properties of the TaKIPI tagger (rule-based, selective usage of context etc.) we decided to split the sentence boundaries identification into two steps:

- initial sentence segmentation — performed before morphosyntactic disambiguation, during which all ambiguous cases are left undecided,
- dot haplology disambiguation done after the morphosyntactic tagging.

The first step is performed by a tool called *Sentencer*⁴. The second step is not yet performed by TaKIPI 1.5 and it is planned for the future extensions. Sentences are marked in the IPIC XML format by the pair: `<chunk type="s">` and `</chunk>`. *Sentencer* closes a sentence in the following cases:

1. A token equal to “!” or “?” followed by a white-space occurred.
2. A token equal to “.” preceded by a white-space occurred.
3. A token T_0 equal to “.” occurred surrounded by tokens T_{-1} and T_1 and:
 - (a) there is no white-space between T_{-1} and T_0 ,
 - (b) T_1 does not start with a lower-case letter,
 - (c) T_{-1} is not a known abbreviation.
4. A token T_0 annotated as `tsym` occurred surrounded by tokens T_{-1} and T_1 and:
 - (a) there is no white-space between T_{-1} and T_0 ,
 - (b) T_1 does not start with a lower-case letter,
 - (c) T_{-1} is not a known multi-part abbreviation.
5. The end of the input.

In the case of 3 and 4 a dictionary of abbreviations extracted from IPIC 2.0 are consulted before the final decision is made. This dictionary was built the following way. First, two sets of token occurrences (along with their frequencies) have been collected: W — tokens occurring without a dot at the end, and P — followed by a dot. The candidates for abbreviations were acquired according to the following heuristics: if a token T occurred p times in P and w times in W , it is taken as an abbreviation if and only if $w/p \leq \tau$. The value of τ was chosen empirically to 0.1.

⁴Although it would be more appropriate to call it *Pre-sentencer*.

Next, the whole resulting list (1134 candidates) was manually revised. In case of doubts, a candidate was checked against its occurrences in IPIC before the final decision. This work resulted in leaving 314 abbreviations in the dictionary.

6 Language of Morphosyntactic Constraints

Problems with achieving a reasonable accuracy of tagging by the application of probabilistic methods to Polish revealed in (Dębowski, 2004; Kukła, 2007) were a reason to look for different methods of tagging Polish. TaKIPI has a rule-based construction, in which a small set of hand-written tagging rules (Piasecki, 2006) is complemented by a huge set of rules automatically extracted in the form of decision trees (Piasecki and Godlewski, 2006).

Handwritten rules are written in a constraint language called JOSKIPI (Piasecki, 2006). JOSKIPI was also used to define the set of basic morphosyntactic constraints from which automatic rules are generated by the combination of basic constraints. Recently, JOSKIPI engine was successfully used independently from TaKIPI in order to describe occurrence contexts of lexical units during the extraction of semantic relatedness measure (Piasecki et al., 2007).

JOSKIPI is aimed at describing a particular token in IPIC, disambiguated or not, its relations with the context (e.g. agreement), and subsequences of tokens in its surrounding up to sentence boundaries. JOSKIPI still does not build partial syntactic structures as intermediate analysis steps (planned for future extensions), but the applications show how much can be achieved with the present simpler model. JOSKIPI is related to the previous works of Karlsson et al. (1995) and Květoň (2003), and the query language of Poliquarp (Przepiórkowski, 2004).

In JOSKIPI identification of tokens is relative to the central position of application of a given constraint or rule. The position can be described by an exact number or a variable, e.g. `flex[1]`, `flex[-1]`, `flex[$V]`, where `flex` is a *simple operator* returning a grammatical class of the addressed token, `-1` means one token before and `$V` is a variable with some position value stored in.

All simple operators can return a set when applied to a non-disambiguated corpus, e.g. `flex[1]` will return a set of grammatical classes for the token next to the place of the constraint/rule application. Simple operators give access to: grammatical classes (`flex`), values of grammatical categories (`cas`, `gnd`, `nmb`, ... — named by the mnemonics of IPIC), morphological base forms (`base`) and orthographic form (`orth`) — word form. If the addressed token does not exist (e.g. because the offset goes beyond the sentence) or it does not have the given category, the simple operator returns the *empty value*: `none`.

Values returned by the operators (always sets, possibly singletons) can be compared by the *set comparison operators*: `equal`, `inter` (intersection), `in` (inclusion). These operator return a boolean value: `false` or `true`, that can be further used in *complex operators, rules* or can be the final value of a *constraint*. The value `true` is returned if two sets are identical, the intersection is not empty or the inclusion holds, respectively. For example, `inter(flex[1], {subst})` returns `true` if the preceding token have the `subst` category (noun) as one of its categories.

There are two more comparison operators: `agr` and `agrpp`. The first tests

existence of a morphosyntactic agreement for all tokens between the given two positions, for the given list of grammatical categories (or even concrete values), e.g. `agr(-3,-1,{cas,m1,m2,sg},3)`, where the number 3 means that we expect the agreement to hold for three categories: case, gender (only two values are interesting) and number (only singular). The `agr` operator returns true, if there is at least one interpretation for each tested token, such that it satisfies the agreement. The `agrpp` operator tests only two selected tokens.

The complex operators include: *logical operators* and *search operators*. Logical operators are: `and`, `or` and `not` (implementing *not or* operation). Each of them can take a list of several comparison operators as its argument, e.g.

```
and( not(equal(cas[-2],{none})),
     agrpp(0,-2,{cas,gnd,nmb},3) )
```

In the above example, the comparison of the case of -2 with `none` is a trick testing whether there is some token with a case category at the position -2.

The subsequences of tokens can be examined by the *search operators*: `llook` — looking for some tokens to the left, `rlook` — to the right, `only` — checks if there are only tokens satisfying some given condition and `atleast` — at least *n* tokens satisfy the given condition. All preserve the same scheme: from, to, iterator — a variable moved across the subsequent positions, condition — any boolean operator. `atleast` adds to this scheme the minimal number of tokens satisfying the condition. The variable is increased or decreased, according to the direction of searching, and the condition is applied to successive values of the variable (the condition should be dependent on the variable). If the condition succeeded desired number of times, i.e. once for (1)`rlook` and for all tested tokens in the case of `only`, then the operator returns `true` and the value of the iterator variable stays set to the last tested position. In the example below, first we look for the first noun (`subst`) to the right (later till the `end` of the sentence), next we test agreement and if between the position 1 and the found noun only adjectives, adverbs, and nouns (i.e. the one found) occur:

```
and( rlook(1,end,$V,inter(flex[$V],{subst})),
     agrpp(0,$V,{cas,gnd,nmb},3),
     only(1,$V,$0,in(flex[$0],{adj,adv,subst})) )
```

There are three simple operators of special access to the token description:

- `suff/pref` — modifiers of `orth` and `base`, limits the result to a suffix or prefix, respectively, of the specified size,
- `catflt` — takes a position, an *interpretation mask*, and a *value mask*, returns values of categories matching the given value mask collected only from the interpretation matching the interpretation mask, e.g. `inter(catflt(1,{adj},{cas}),cas[0])` — tests the case of the adjectival interpretations of the token at the position 1.

Finally, JOSKIPI allows to use operators under specified condition, i.e. an expression of the scheme: `v-operator ? c-operator`, where `c-operator` is of boolean type, runs `v-operator` and returns the value produced by it, only if `c-operator` returns `true`, otherwise `none`.

JOSKIPI has been used so far to define the *rules* of tagging and express *constraints*. Each rule preserves the scheme: *condition of application* plus a boolean constraint identifying interpretations to be *deleted* from the token 0.

Constraints are JOSKIPI operators (simple or complex) and are constituents of automatically extracted disambiguation rules. Besides, they may be employed to describe occurrences of lexical units in corpus, e.g. (Piasecki *et al.*, 2007). Such descriptions are built on the basis of several boolean constraints, where each constraint describes some lexico-syntactic property of the occurrence context, e.g. an occurrence of a particular adjective modifying the lexical unit under description. The lexical unit description is constructed by calculating the frequencies of constraint **true** values across all occurrences of the given lexical unit in the corpus. Comparison of the appropriately transformed descriptions-vectors for two lexical units can be used successfully as the measure of their semantic similarity.

For the needs of construction of huge matrices of description, their further transformation and similarity calculation a package of tools called *SuperMatrix* was built⁵. It works on the IPIC XML format, is integrated with TaKIPI, and applies JOSKIPI engine to use JOSKIPI constraints.

7 Morphosyntactic Tagging

The detailed description of TaKIPI can be found in several papers (e.g. Piasecki and Godlewski (2006); Piasecki (2007)). We will mention some of its features which make it a general-purpose tagger for open-domain Polish texts.

The minimal configuration of TaKIPI is the disambiguation engine plus *Morfusz*. The inclusion of *Tokenizator*, *Odgadywacz* and *Sentencer* is likely to increase the accuracy of tagging on average, yet it is hard to be assessed on the basis of MIPIC. The accuracy of TaKIPI 1.5 without *Odgadywacz* is roughly the same like TaKIPI 1.0, i.e. about 93.44%.

The input can be is one of the following: plain text, XML/HTML (XML tags are passed unchanged to the output) as well as IPIC XML format. It works on three standards of encoding Polish diacritics: UTF-8, ISO-8859-2 and CP-1250 (Windows/DOS). The output is produced in IPIC XML format; the disambiguating tags are marked. In some rare cases TaKIPI leaves more than one tag for a token: the average ratio of tags per token is 1.03.

The disambiguation engine of TaKIPI consists of hand-written rules and decision trees, arranged in three subsequent levels of processing, disambiguating respectively: grammatical class, number plus gender, and case. The rest of IPIC categories are getting disambiguated in large extent after those three are.

The open architecture allows for relatively easy extension with new classifiers (which may improve the current results) as well as additional processing stages (for instance more sophisticated abbreviation recognition). It is available on the GPL licence in two distributions: a command-line tool suitable for processing of multiple documents and a dynamic library with a simple interface. Both versions can be compiled and run on Linux or Windows platforms (C++ source code).

⁵SuperMatrix package is to be released under GPL licence as well.

8 Further Development

Our goal was to build a complete set of universal, publicly accessible language tools for Polish in cooperation with everyone who wants to participate. The tools presented here are now publicly available under the GPL licence as components of TaKIPI 1.5. Moreover, all of them were integrated into the GATE framework (Cunningham *et al.*, 2002) as *language processing resources*. The tools are aimed at processing free Polish coming from any source with no limitations on the type of text. The universality of the tools is limited only by the assumed IPIC tagset and IPIC formats as the basis. However, IPIC is for now the only existing, large morphosyntactically annotated corpus of Polish, so it defines the standard by its very existence. Moreover, TaKIPI has been co-financed by the Institute of Computer Science PAS and the IAI WUT and developed in co-operation.

TaKIPI 1.5 provides complete morphosyntactic analysis and disambiguation; besides the accuracy, only some cases of dot haplology are left unsolved. The step of shallow parsing — currently lacking for Polish — can be performed by the Spade-based parser (in development) (Przepiórkowski and Buczyński, 2007).

TaKIPI and its components are extensively used in the *SuperMatrix* package (Piasecki and Broda, 2007b) facilitating statistical analysis of lexical units occurrences across very large corpora, construction of huge *coincidence matrices*, the extraction of semantic similarity, and clustering of lexical units on the basis of their meaning. It also utilises a Polish wordnet called *plWordNet (Słowosieć)* (Derwojedowa *et al.*, 2008) as the source of the lexical semantic knowledge for the evaluation of the extracted similarity functions and the clustering lexical units. *plWordNet* and *SuperMatrix* are available under the free research licence.

The *Kolokacje* application Buczyński and Okniński (2006) together with the JOSKIPI engine and the selected components of *SuperMatrix* were utilised in a tool for the extraction of binary, syntactically described collocations of Polish (Broda *et al.*, 2007). *SuperMatrix* and JOSKIPI were also used to construct a mechanism of medical handwriting OCR correction.

References

- B. BRODA, M. DERWOJEDOWA, and M. PIASECKI (2007), Recognition of Structured Collocations in An Inflective Language, in *Proc. of the 2nd International Symposium Advances in Artificial Intelligence and Applications*, pp. 237–246.
- Aleksander BUCZYŃSKI and Tomasz OKNIŃSKI (2006), Program Kolokacje, URL <http://www.mimuw.edu.pl/polszczyzna/kolokacje/>, Program Web page.
- H. CUNNINGHAM, D. MAYNARD, K. BONTCHEVA, and V. TABLAN (2002), GATE: A framework and graphical development environment for robust NLP tools and applications, in *Proceedings of the 40th Anniversary Meeting of ACL*.
- Magdalena DERWOJEDOWA, Maciej PIASECKI, Stanisław SZPAKOWICZ, Magdalena ZAWISŁAWSKA, and Bartosz BRODA (2008), Words, Concepts and Relations in the Construction of Polish WordNet, in *Proceedings of the Global WordNet Conference, Szeged, Hungary 2008*, Global WordNet Association.
- Łukasz DĘBOWSKI (2004), Trigram morphosyntactic tagger for Polish, in M. KŁOPOTEK

- et al.*, ed., *Proceedings of the International IIS:IIPWM'04 Conference held in Zakopane, Poland, 2004*, pp. 409–413, Springer.
- F. KARLSSON, A. VOUTILAINEN, J. HEIKKILÄ, and A. ANTTILA, editors (1995), *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*, Mouton de Gruyter, Berlin and New York.
- Mieczysław A. KŁOPOTEK, Sławomir T. WIERZCHOŃ, and Krzysztof TROJANOWSKI, editors (2006), *Proceedings of the International IIS: IIPWM'06 Conference held in Zakopane, Poland, June, 2006*, Springer, Berlin.
- Piotr KUKLA (2007), *Tager dla języka polskiego oparty na kombinacji metod statystycznych*, MSc thesis, Politechnika Wroclawska.
- Pavel KVĚTOŇ (2003), Language for Grammatical Rules, Report TR-2003-17, ÚFAL/CKL MFF UK, Prague.
- Valérie MAPELLI and Khalid CHOUKRI (2003), Report on a (Minimal) Set of LRs To Be Made Available for as Many Languages as Possible, and Map of the Actual Gaps, Internal report Deliverable 5.1, ENABLER Project, ELDA.
- Maciej PIASECKI (2006), Handmade and Automatic Rules for Polish Tagger, in Sojka *et al.* (2006).
- Maciej PIASECKI (2007), Polish Tagger TaKIPI: Rule Based Construction and Optimisation, *Task Quarterly*, 11(1–2):151–167.
- Maciej PIASECKI and Bartosz BRODA (2007a), Correction of Medical Handwriting OCR Based on Semantic Similarity, in Hujun YIN *et al.*, editors, *Intelligent Data Engineering and Automated Learning, IDEAL 2007, Birmingham, UK, December 16-19, 2007, Proceedings*, LNCS v. 4881, Springer.
- Maciej PIASECKI and Bartosz BRODA (2007b), Semantic Similarity Measure of Polish Nouns Based on Linguistic Features, in W. ABRAMOWICZ, ed., *Business Information Systems 10th Inter. Conf., Poznan, Poland, 2007, Proc.*, Springer.
- Maciej PIASECKI and Grzegorz GODLEWSKI (2006), Effective Architecture of the Polish Tagger, in Sojka *et al.* (2006).
- Maciej PIASECKI and Adam RADZISZEWSKI (2007), Polish Morphological Guesser Based on a Statistical *A Tergo* Index, in *Proc. of the 2nd Inter. Symposium Advances in Artificial Intelligence and Applications (AAIA'07)*, pp. 247–256.
- Maciej PIASECKI, Stanisław SZPAKOWICZ, and Bartosz BRODA (2007), Automatic Selection of Heterogeneous Syntactic Features in Semantic Similarity of Polish Nouns, in *Proc. of the Text, Speech and Dialog 2007 Conference, LNAI*, Springer.
- Adam PRZEPIÓRKOWSKI (2004), *The IPI PAN Corpus: Preliminary version*, Institute of Computer Science PAS.
- Adam PRZEPIÓRKOWSKI and Aleksander BUCZYŃSKI (2007), *Spade: Shallow Parsing and Disambiguation Engine*, in Zygmunt Vetulani ed., *Proc. of the 3rd Language and Technology Conference, Poznań, Poland*.
- Petr SOJKA, Ivan KOPECEK, and Karel PALA, editors (2006), *Proceedings of the Text, Speech and Dialog 2006 Conference, LNAI*, Springer.
- Marcin WOLIŃSKI (2006), Morfeusz — a practical tool for the morphological analysis of Polish., in Kłopotek *et al.* (2006), pp. 511–520.