A Preliminary Formalism for Simultaneous Rule-Based Tagging and Partial Parsing*

Adam Przepiórkowski

Abstract

This paper presents a formalism for simultaneous rule-based morphosyntactic tagging and partial parsing. A prototype implementation of the formalism exists, while a more efficient implementation is being developed with the aim of annotating the IPI PAN Corpus of Polish.

1 Introduction

After well over a decade of the almost absolute rule of the statistical paradigm in NLP, we seem to be witnessing a renewed interest in rule-based approaches to such common problems as morphosyntactic tagging (Neumann et al., 2000, Neumann and Piskorski, 2002, Hinrichs and Trushkina, 2002, Oliva and Petkevič, 2002, Rudolf, 2004) and partial syntactic parsing (Grover and Tobin, 2006), and in combining statistical and rule-based approaches (Hajič et al., 2001, Piasecki, 2006). The aim of this paper is to present a formalism for rule-based tagging and partial parsing, which is to be used for the processing of Polish, and possibly other languages.

Usually tagging and partial parsing are done separately, with the input to the parser assumed to be morphosyntactically fully disambiguated text. Some approaches (Karlsson et al., 1995, Schiehlen, 2002, Müller, 2006) interweave tagging and parsing, with Karlsson et al. (1995) actually using the same formalism for both tasks. In the latter case, the application of a uniform formalism was possible because all words in this dependency-based approach come with all possible syntactic tags, so partial parsing boils down to rejecting wrong hypotheses, just as in the case of morphosyntactic tagging.

Conversely, rules used in rule-based tagging often implicitly identify syntactic constructs, but do not mark such constructs in texts. A typical such rule may say that when an unambiguous dative-taking preposition is followed by a number of possibly dative adjectives and a noun ambiguous between dative and some other case, then the noun should be disambiguated to dative. Obviously, such a rule actually identifies a PP and some of its structure.

Since both tasks, morphosyntactic tagging and partial constituency parsing, involve similar linguistic knowledge, we propose a formalism for simultaneous tagging and parsing. The input

* Published in: *Data Structures for Linguistic Resources and Applications*, Georg Rehm, Andreas Witt, Lothar Lemnitzer (eds.), Tübingen: Gunter Narr Verlag. 2007. pp. 81–90.

to the rules of this formalism is a tokenised and morphosyntactically annotated XML (XCES, Ide et al., 2000) text. The output contains two new levels of constructions: syntactic words and syntactic groups. In the remainder of this paper we assume the following terminology:

segment (or basic word) is the smallest interpreted unit, i. e., a sequence of characters together with their morphosyntactic interpretations (lemma, grammatical class, grammatical categories); in the XML format assumed here (both input and output) segments are marked as `<tok>` elements;

syntactic word is a non-empty sequence of segments and/or syntactic words; it is marked in the XML output as `<syntok>`;

token is a segment or a syntactic word; it follows that syntactic words may be defined as any non-empty sequences of tokens;

syntactic group (in short: group) is a non-empty sequence of tokens and/or syntactic groups; in the XML output it is marked as `<group>`;

syntactic entity is a token or a syntactic group; it follows that syntactic groups may be defined as any non-empty sequences of syntactic entities.

Syntactic words are named entities, analytical forms, or any other sequences of tokens or syntactic words which, from the syntactic point of view, behave as single words. Just as basic words, they may have a number of morphosyntactic interpretations. On the other hand, syntactic groups are sequences of (basic or syntactic) words and smaller groups; each group is identified by its syntactic head and semantic head, which are (basic or syntactic) words; a rationale for such a description of syntactic groups is given in Przepiórkowski (2006).

2 The Formalism

2.1 The Basic Format

Each rule consists of up to six parts marked as `Left`, `Match`, `Right`, `Cond`(ition), `Synt`(actic action) and `Morph`(osyntactic action), as in the following example:

```
Left:
Match: [pos~~prep] [base~"co|kto"]
Right:
Cond:  agree(case,1,2)
Synt:  group(PG,1,2)
Morph: unify(case,1,2)
```

The application of this rule should consist in:

- finding a sequence of two tokens such that:
  - the first token is an unambiguous preposition (`[pos~~prep]`),
  - the second token is a form of the lexeme *co* 'what' or *kto* 'who' (`[base~"co|kto"]`),
  - there exist interpretations of these two tokens with the same value of case (`agree (case,1,2)`);

- marking thus identified sequence as a syntactic group (`group`) of type `PG` (prepositional group), whose syntactic head is the first token (`1`) and whose semantic head is the second token (`2`; cf. `group(PG,1,2)`);
- rejecting all interpretations of the two tokens which do not agree in case (cf. `unify(case,1,2)`).

Any of the six parts of a rule may be empty; in such a case the whole part may be omitted, as in the following rule which is fully equivalent to the rule above.

```
Match: [pos~~prep] [base~"co|kto"]
Cond:  agree(case,1,2)
Synt:  group(PG,1,2)
Morph: unify(case,1,2)
```

The order of the parts is not meaningful. The character `#` starts a comment, so another equivalent formulation of the same rule is given as follows:

```
Synt:  group(PG,1,2)              # Simple prepositional group: rule 193
Morph: unify(case,1,2)
Cond:  agree(case,1,2)
Match: [pos~~prep] [base~"co|kto"] # 'na co', 'z kim'
```

2.2 'Left', 'Match' and 'Right'

The contents of parts `Left`, `Match` and `Right` have the same syntax and semantics. Each of them may contain a sequence of the following specifications:

- token specification, e. g., `[pos~~prep]` or `[base~"co|kto"]`; these specifications adhere to segment specifications of the Poliqarp corpus search engine (Janus and Przepiórkowski, 2006) as specified in Przepiórkowski (2004); a specification like `[pos~~subst]` says that *all* morphosyntactic interpretations of a given token are nominal (substantive), while `[pos~subst]` means that there *exists* a nominal interpretation of a given token;
- group specification, extending the Poliqarp query language as proposed in Przepiórkowski (2006), e. g., `[semh=[pos~~subst]]` specifies a syntactic group whose semantic head is an unambiguous noun (or, more precisely, the semantic head is a possibly ambiguous token whose all interpretations are nominal);
- one of the following specifications:
  - `ns`: no space,
  - `sb`: sentence beginning,
  - `se`: sentence end,
- an alternative of such sequences in parentheses, e. g.:
  - `([pos~~subst] | [synh=[pos~~subst]])` or
  - `([pos~~prep] [pos~subst] ns [pos~interp] se | [synh=[pos~~prep]])`.

Additionally, each such specification may be modified with one of the three standard regular expression quantifiers: `?`, `*` and `+`.

2.3 'Cond'

The Cond part contains a sequence of conditions that must be satisfied by the tokens matched by the specifications in Left, Match and Right, in order for actions specified in Morph and Synt to take place. These specifications are numbered from 1, counting from the first specification in Left to the last specification in Right. For example, in the rule given below, there should be case agreement between the adjective specified in the left context and the adjective and the noun specified in the right context (cf. agree(case,1,4,5)), as well as case agreement (possibly of a different case) between the adjective and noun in the match (cf. agree(case,2,3)).

```
Left:  [pos~~adj]
Match: [pos~~adj] [pos~~subst]
Right: [pos~~adj] [pos~~subst]
Cond:  agree(case,2,3), agree(case,1,4,5)
```

In the case of the more realistic rule given next, the reference 2 refers to *all* tokens matched by the specification [pos~~adj]*, so the condition agree(case,2,3) requires case agreement between tokens whose number is unknown at the time of reading and parsing the rule.

```
Left:  sb
Match: [pos~~adj]* [pos~~subst]
Cond:  agree(case,2,3)
```

The repertoire of conditions that may appear in the Cond part still evolves, with agree being the most frequent condition. It takes a variable number of arguments: the initial arguments, such as case or gender, specify the grammatical categories that should *simultaneously* agree, so the condition agree(case,gender,1,2) is properly stronger than the sequence of conditions: agree(case,1,2),agree(gender,1,2). Subsequent arguments of agree are natural numbers referring to token specifications that should be taken into account when checking agreement.

2.4 'Morph'

The Morph part contains a sequence of action specifications that should be carried out on morphosyntactic interpretations of appropriate tokens. Similarly to Cond, a reference to token specification refers to all tokens matched by that specification, so, e. g., in case 1 refers to specification [pos~~adj]*, unify(case,1) means that all adjectives matched by that specification must be rid of all interpretations whose case is not shared by all these adjectives.

The set of actions currently contains unify, with a variable number of arguments and with semantics analogous to the condition agree, so, e. g., unify(case,gender,1,2) may lead to the rejection of a larger number of interpretations than the action sequence unify(case, 1,2), unify(gender,1,2). Other actions implemented in the prototype are delete (e. g.: delete(prep,nom,2), i. e., from the token(s) found by the specification 2 delete those interpretations, according to which these tokens are prepositions subcategorising for a nominative argument) and leave (e. g., leave(subst,1,2), i. e., for tokens found by specifications 1 and 2, leave only nominal interpretations).

We assume that the actions specified in Morph may only refer to single tokens; when a reference in such an action refers to a syntactic group, the action is performed on the syntactic head

of that group. For example, assuming that the following rule finds a sequence of a nominal segment, a multi-segment syntactic word and a nominal group, the action `unify(case,1)` will result in the unification of case values of that nominal segment, of the syntactic word as a whole and of the syntactic head of the group.

```
Match: ([pos~~subst] | [synh=[pos~~subst]])+
Morph: unify(case,1)
```

2.5 'Synt'

While actions specified in `Morph` delete some interpretations, actions in `Synt` group syntactic entities into syntactic words or syntactic groups; the two operators used for this purpose are, correspondingly, `word` and `group`.

Syntactic Word

Any specification of an action aimed at creating a syntactic word is of the form `word(<spec>)` where `<spec>` is the specification of the action that should be carried out. Marking a syntactic word is more difficult than marking a syntactic group as it requires specifying the orthographic form of the word (i. e., `<orth>` in XML terms) and its possible interpretations (`<lex>`).

We assume that the orthographic form of the syntactic word is always a simple concatenation of all orthographic forms of all tokens immediately contained in that syntactic word, taking into account information about space or its lack between consecutive tokens; see the end of section 2.6 for an example.

The creation of morphosyntactic interpretations of syntactic words involves more sophisticated operations. For example, a rule identifying negated verbs, such as the following one, may require that the interpretations of the whole syntactic word be the same as the interpretations of the verbal segment, but with neg added to each interpretation.

```
Left: ([pos!~"prep"] | [case!~"acc"])
Match: [orth~"[Nn]ie"] [pos~~"praet|fin|impt|imps|inf"]
       (ns [pos~~aglt])?
Synt:  word(neg(3))
Morph: leave(2,qub)
```

In this case, the specification `neg(3)` is used to copy and appropriately modify all interpretations of the segment matched by the specification `[pos~~"praet|fin|impt|imps|inf"]`. The result of the application of this rule for the sequence *nie było* 'wasn't', lit. 'not was', will be as shown in listing 1 (leaving out disambiguation information and some interpretations of *nie*).

```
<syntok>                                    the code in italics was added by the rule
  <orth>nie było</orth>
  <lex>
    <base>nie być</base>
    <ctag>neg:praet:sg:n:imperf</ctag>
  </lex>
  <tok>
    <orth>nie</orth>
    <lex><base>nie</base><ctag>qub</ctag></lex>

                    pronominal interpretations of nie omitted here

  </tok>
  <tok>
    <orth>było</orth>
    <lex>
      <base>być</base>
      <ctag>praet:sg:n:imperf</ctag>
    </lex>
  </tok>
</syntok>
```

Listing 1: An example output of the negation rule on p. 85

Other action specifications currently in use include:

- `aff(<num>)`: creates interpretations of non-negated verbs (by adding `aff` to interpretations of the verbal segment);
- `copy(<num>)`: copies interpretations of one of the tokens immediately within the word.

Additionally, it is possible to create a new interpretation and a new base form (lemma). For example, the following rule will create, for a sequence like *mimo tego, że* or *Mimo że* 'in spite of, despite', a syntactic word with the base form *mimo że* and the conjunctive interpretation.

```
Match: [orth~"[Mm]imo"] [orth~"to|tego"]?
       (ns [orth~","])? [orth~%{\.z}e%]
Synt:  word("conj","mimo %{\.z}e%")
Morph: leave(1,prep), leave(2,subst)
```

Syntactic Group

Any specification of an action aimed at creating a syntactic group has the form `group(<type>,<synh>,<semh>)`, where `<type>` is the categorial type of the group (e. g., `PG`), while `<synh>` and `<semh>` are references to appropriate token specifications in the `Match` part. For example, the next rule may be used to create a numeral group, syntactically headed by the numeral and semantically headed by the noun.

```
Left:  [pos~~prep]
Match: [pos~~num] [pos~~adj]* [pos~~subst]
Synt:  group(NumG,2,4)
```

Of course, the rules should be constructed in such a way that references <synh> and <semh> refer to specifications of single tokens, e. g., [case~~nom] or [pos~~num], but not [case~~nom]+ or ([pos~~subst] | [synh=[pos~~subst]]); otherwise the rules parser should signal an error.

2.6 Input and Output: Example

The prototype tool implementing the specification above currently takes as input the version of the XML Corpus Encoding Standard (XCES, Ide et al., 2000) assumed in the IPI PAN Corpus (Przepiórkowski, 2004). The rules operate sequentially, so the input to one rule may be the output of another rule.

Rules may modify the input in one of two ways. Specifications in Morph may delete certain interpretations of certain tokens; this fact is marked by the attribute disamb:sh="0" added to <lex> elements representing these interpretations. For example, the result of the application of the first example rule (p. 82) to the following input sequence of two tokens *Po co* 'why, what for', lit. 'for what' in listing 2 should have the effect as in listing 3.[1]

```
<tok id="tA5">
  <orth>Po</orth>
  <lex disamb="1">
    <base>po</base>
    <ctag>prep:acc</ctag>
  </lex>
  <lex><base>po</base><ctag>prep:loc</ctag></lex>
</tok>
<tok id="tA6">
  <orth>co</orth>
  <lex><base>co</base><ctag>conj</ctag></lex>
  <lex><base>co</base><ctag>prep:acc</ctag></lex>
  <lex><base>co</base><ctag>qub</ctag></lex>
  <lex><base>co</base><ctag>subst:sg:nom:n</ctag></lex>
  <lex disamb="1">
    <base>co</base>
    <ctag>subst:sg:acc:n</ctag>
 </lex>
</tok>
```

Listing 2: Input to the rule shown on p. 82

The disamb="1" specification comes from the tagger used to annotate the IPI PAN Corpus of Polish, but any input attributes of <lex> are copied to output, so the output may look as

[1] Note that this rule is not optimal, as it should also reject the prep:acc interpretation of *co*.

```
<tok id="tA5">
  <orth>Po</orth>
  <lex disamb="1">
    <base>po</base>
    <ctag>prep:acc</ctag>
  </lex>
  <lex disamb:sh="0">
    <base>po</base>
    <ctag>prep:loc</ctag>
  </lex>
</tok>
<tok id="tA6">
  <orth>co</orth>
  <lex disamb:sh="0">
    <base>co</base>
    <ctag>conj</ctag>
  </lex>
  <lex><base>co</base><ctag>prep:acc</ctag></lex>
  <lex disamb:sh="0">
    <base>co</base>
    <ctag>qub</ctag>
  </lex>
  <lex disamb:sh="0">
    <base>co</base>
    <ctag>subst:sg:nom:n</ctag>
  </lex>
  <lex disamb="1">
    <base>co</base>
    <ctag>subst:sg:acc:n</ctag>
  </lex>
</tok>
```

Listing 3: Simplified output of the rule shown on p. 82

follows, which reflects a disagreement between the previous tagger and the decision of the rule: `<lex disamb="1" disamb:sh="0">...</lex>`.

The other way in which a rule may modify the input consists of grouping syntactic entities according to specifications in `Synt`. For example, the specification `group(PG,1,2)` applied to the above sequence *Po co* will result in adding a `<group>` element as shown in listing 4.

The values of attributes `synh` and `semh` are values of corresponding `id`s of the syntactic head and the semantic head. Another example, of the result of a `word(...)` specification in the `Synt` part, is given in listing 1 (page 85).

```
<group synh="tA5" semh="tA6" type="PG">
  <tok id="tA5">
    <orth>Po</orth>
    <lex disamb="1">
      <base>po</base><ctag>prep:acc</ctag>
    </lex>
    <lex disamb:sh="0">
      <base>po</base><ctag>prep:loc</ctag>
    </lex>
  </tok>
  <tok id="tA6">
    <orth>co</orth>
    <lex disamb:sh="0">
      <base>co</base>
      <ctag>conj</ctag>
    </lex>
    <lex><base>co</base><ctag>prep:acc</ctag></lex>
    <lex disamb:sh="0">
      <base>co</base>
      <ctag>qub</ctag>
    </lex>
    <lex disamb:sh="0">
      <base>co</base>
      <ctag>subst:sg:nom:n</ctag>
    </lex>
    <lex disamb="1">
      <base>co</base>
      <ctag>subst:sg:acc:n</ctag>
    </lex>
  </tok>
</group>
```

Listing 4: Fuller output of the rule shown on p. 82

3 Conclusion

The starting point of this paper was the observation that morphosyntactic disambiguation rules and partial parsing rules often encode the same linguistic knowledge. While many partial parsers expect a fully disambiguated input and some are interleaved with morphosyntactic disambiguators, we are not aware of any partial (or shallow) parsing systems accepting morphosyntactically ambiguous input and disambiguating it with the same rules that are used for parsing. This paper proposes a formalism for writing such partial parsing/morphosyntactic disambiguation rules.

The work presented here is to some extent work in progress. The exact syntax and semantics of the formalism still evolve, and currently only a prototype implementation of a tagger/parser based on such rules is available. Although the ultimate test of the usefulness of this approach will be the quality of the partial treebank of Polish to be annotated with the use of this formalism

(Przepiórkowski, 2006), we hope to have convinced the reader about the initial viability of the idea of simultaneous rule-based disambiguation and partial parsing.

Bibliography

Grover, Claire and Tobin, Richard (2006): "Rule-Based Chunking and Reusability". In: *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC 2006)*.

Hajič, Jan; Krbec, Pavel; Květoň, Pavel; Oliva, Karel and Petkevič, Vladimir (2001): "Serial Combination of Rules and Statistics". In: *Proceedings of ACL '01*. pp. 260–267.

Hinrichs, Erhard W. and Trushkina, Julia (2002): "Forging Agreement: Morphological Disambiguation of Noun Phrases". In: *Proceedings of TLT 2002*. Sozopol, Bulgaria, pp. 78–95.

Ide, Nancy; Bonhomme, Patrice and Romary, Laurent (2000): "XCES: An XML-based Standard for Linguistic Corpora". In: *Proc. of the Ling. Resources and Evaluation Conference*. Athens, pp. 825–830.

Janus, Daniel and Przepiórkowski, Adam (2006): "POLIQARP 1.0: Some technical aspects of a linguistic search engine for large corpora". In: *The Proceedings of Practical Applications of Linguistic Corpora 2005*, edited by Waliński, J.; Kredens, K. and Goźdź-Roszkowski, S. Frankfurt/Main: Peter Lang.

Karlsson, F.; Voutilainen, A.; Heikkilä, J. and Anttila, A. (editors) (1995): *Constraint Grammar: A Language-Independent System for Parsing Unrestricted Text*. Berlin: de Gruyter.

Müller, Frank Henrik (2006): *A Finite State Approach to Shallow Parsing and Grammatical Functions Annotation of German*. Ph.D. thesis, Universität Tübingen. Pre-final Version of March 11, 2006.

Neumann, Günter; Braun, Christian and Piskorski, Jakub (2000): "A Divide-and-Conquer Strategy for Shallow Parsing of German Free Texts". In: *Proceedings of ANLP 2000*. Seattle, pp. 239–246.

Neumann, Günter and Piskorski, Jakub (2002): "A Shallow Text Processing Core Engine". *Journal of Computational Intelligence* 18 (3): pp. 451–476.

Oliva, Karel and Petkevič, Vladimir (2002): "Morphological and Syntactic Tagging of Slavonic Languages". A lecture at *Empirical Linguistics and NLP*, Sozopol, Bulgaria, September 2002.

Piasecki, Maciej (2006): "Hand-Written and Automatically Extracted Rules for Polish Tagger". In: *Proceedings of Text, Dialogue and Speech (TSD) 2006*.

Przepiórkowski, Adam (2004): *The IPI PAN Corpus: Preliminary version*. Warsaw: Institute of Computer Science, Polish Academy of Sciences.

Przepiórkowski, Adam (2006): "On Heads and Coordination in a Partial Treebank". In: *Proceedings of TLT 2006*, edited by Hajič, Jan and Nivre, Joakim. Prague, pp. 163–174.

Rudolf, Michał (2004): *Metody automatycznej analizy korpusu tekstów polskich*. Warsaw: Uniwersytet Warszawski, Wydział Polonistyki.

Schiehlen, Michael (2002): "Experiments in German Noun Chunking". In: *Proceedings of the 19th International Conference on Computational Linguistics (COLING 2002)*. Taipei.