An Implementation of Combined Partial Parser and Morphosyntactic Disambiguator

Aleksander Buczyński
Institute of Computer Science
Polish Academy of Sciences
Ordona 21, 01-237 Warszawa, Poland
olekb@ipipan.waw.pl

Abstract

The aim of this paper is to present a simple yet efficient implementation of a tool for simultaneous rule-based morphosyntactic tagging and partial parsing formalism. The parser is currently used for creating a treebank of partial parses in a valency acquisition project over the IPI PAN Corpus of Polish.

1 Introduction

1.1 Motivation

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

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.

Following the observation that both tasks, morphosyntactic tagging and partial constituency parsing, involve similar linguistic knowledge, a formalism for simultaneous tagging and parsing was proposed in (Przepiórkowski, 2007). This paper presents a revised version of the formalism and a simple implementation of a parser understanding rules written according to it. The input to the rules is a tokenised and morphosyntactically annotated XML text. The output contains disambiguation annotation and two new levels of constructions: syntactic words and syntactic groups.

2 The Formalism

2.1 Terminology

In the remainder of this paper we call the smallest interpreted unit, i.e., a sequence of characters together with their morphosyntactic interpretations (lemma, grammatical class, grammatical categories) a *segment*. A *syntactic word* is a non-empty sequence of segments and/or syntactic words. Syntactic words are named entities, analytical forms, or any other sequences of tokens which, from the syntactic point of view, behave as single words. Just as basic words, they may have a number of morphosyntactic interpretations. By a *token* we will understand a segment or a syntactic word. A *syntactic group* (in short: group) is a non-empty sequence of tokens and/or syntactic groups. Each group is identified by its syntactic head and semantic head, which have to be tokens. Finally, a *syntactic entity* is a token or a syntactic group; it follows that syntactic groups may be defined as a non-empty sequence of entities.

2.2 The Basic Format

Each rule consists of up to 4 parts: Match describes the sequence of syntactic entities to find; Left and Right — restrictions on the context; Actions — a sequence of morphological and syntactic actions to be taken on the matching entities.

For example:

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

means:

- find a sequence of two tokens such that the first token is an unambiguous preposition ([pos~~"prep"]), and the second token is a possible form of the lexeme CO 'what' or KTO 'who' ([base~"co|kto"]),
- if there exist interpretations of these two tokens with the same value of case, reject all interpretations of these two tokens which do not agree in case (cf. unify(case,1,2));
- if the above unification did not fail, mark 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));

Left and Right parts of a rule may be empty; in such a case the part may be omitted.

2.3 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 (Janus and Przepiórkowski, 2006) corpus search engine; in particular there is a distinction between certain and uncertain information — 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 as proposed in (Przepiórkowski, 2007), e.g., [semh=[pos~~"subst"]] specifies a syntactic group whose semantic head is a 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.

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

An example of a possible value of Left, Match or Right might be:

```
[pos~"adv"] ([pos~~"prep"]
[pos~"subst"] ns?  [pos~"interp"]?
se | [synh=[pos~~"prep"]])
```

2.4 Actions

The Actions part contains a sequence of morphological and syntactic actions to be taken when a matching sequence of syntactic entities is found. While morphological actions delete some interpretations of specified tokens, syntactic actions group entities into syntactic words or syntactic groups. The actions may also include conditions that must be satisfied in order for other actions to take place, for example case or gender agreement between tokens.

The actions may refer to entities matched by the specifications in Left, Match and Right by numbers. These specifications are numbered from 1, counting from the first specification in Left to the last specification in Right. For example, in the following rule, 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. unify(case,1,4,5)), as well as case agreement (possibly of a different case) between the adjective and noun in the match (cf. unify(case,2,3)).

```
Left:  [pos~~"adj"]
Match: [pos~~"adj"][pos~~"subst"]
```

```
Right: [pos~~"adj"][pos~~"subst"]
Actions: unify(case,2,3);
         unify(case,1,4,5)
```

The exact repertoire of actions still evolves, but the most frequent are:

- `agree(<cat>,...,<tok>,...)` - check if the grammatical categories (`<cat>,...`) of entities specified by subsequent numbers (`<tok>,...`) agree;
- `unify(<cat>,...,<tok>,...)` - as above, plus delete interpretations that do not agree;
- `delete(<cond>,<tok>,...)` - delete all interpretations of specified tokens matching the specified condition (for example `case~"gen|acc"`)
- `leave(<cond>,<tok>,...)` - leave only the interpretations matching the specified condition;
- `nword(<tag>,<base>)` - create a new syntactic word with given tag and base form;
- `mword(<tag>,<tok>)` - create a new syntactic word by copying and appropriately modifying all interpretations of the token specified by number;
- `group(<type>,<synh>,<semh>)` - create a new syntactic group with syntactic head and semantic head specified by numbers.

The actions `agree` and `unify` take 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 entity specifications that should be taken into account when checking agreement.

A reference to entity specification refers to all entities 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.

When a reference 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 the first segment, the syntactic word as a whole and the syntactic head of the group.

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

The only exception to this rule is the semantic head parameter in the `group` action; when it references a syntactic group, the semantic, not syntactic, head is inherited.

For `mword` and `nword` actions we assume that the orthographic form of the created 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.

The `mword` action is used to copy and possibly modify all interpretations of the specified token. For example, a rule identifying negated verbs, such as the rule below, 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~~"verb"]
       (ns [orth~"by[mś]?"])?
       (ns [pos~~"aglt"])?
Actions: leave(pos~"qub", 2);
         mword(neg,3)
```

The `nword` action creates a syntactic word with a new interpretation and a new base form (lemma). For example, the rule below 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~"że"]
Actions: leave(pos~"prep",1);
```

```
          leave(pos~"subst",2);
          nword(conj, mimo że)
```

The `group(<type>,<synh>,<semh>)` action creates a new syntactic group, 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 following 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"]
Actions: 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 entities, e.g., `([pos~"subst"]|[synh=[pos~"subst"]])` but not `[case~"nom"]+`

3 The Implementation

3.1 Objectives

The goal of the implementation was a combined partial parser and tagger that would be reasonably fast, but at the same time easy to modify and maintain. At the time of designing and implementing the parser, neither the set of rules, nor the specific repertoire of possible actions within rules was known, hence, the flexibility and modifiability of the design was a key issue.

3.2 Input and Output

The parser currently takes as input the version of the XML Corpus Encoding Standard (Ide et al., 2000) assumed in the IPI PAN Corpus of Polish (`korpus.pl`). The tagset is configurable, therefore the tool can be possibly used for other languages as well.

Rules may modify the input in one of two ways. Morphological actions may delete certain interpretations of certain tokens; this fact is marked by the attribute `disamb="0"` added to `<lex>` elements representing these interpretations. On the other hand, syntactic actions modify the input by adding `<syntok>` and `<group>` elements, marking syntactic words and groups.

3.3 Algorithm Overview

During the initialisation phase, the parser loads the external tagset specification and the ruleset, and converts the latter to a set of compiled regular expressions and actions. Then input files are parsed one by one (for each input file a corresponding output file containing parsing results is created). To reduce memory usage, the parsing is done by chunks defined in the input files, such as sentences or paragraphs. In the remainder of the paper we assume the chunks are sentences.

During the parsing, a sentence has dual representation:

1. object-oriented syntactic entity tree, used for easy manipulation of entities (for example disabling certain interpretations or creating new syntactic words) and preserving all necessary information to generate the final output;
2. compact string for quick regexp matching, containing only the informations important for these rules which have not been applied yet.

The entity tree is initialised as a flat (one level deep) tree with all leaves (segments and possibly special entities, like no space, sentence beginning, sentence end) connected directly to the root. Application of a syntactic action means inserting a new node (syntacting word or group) to the tree, between the root and a few of the existing nodes. As the parsing processes, the tree changes its shape: it becomes deeper and narrower.

The string representations is consistently updated to always represent the top level of the tree (the children of the root). Therefore, the searched string's length tends to decrease with every action applied (as opposed to increasing in a naïve implementation, with single representation and syntactic / disambiguation markup added). This is not a strictly monotonous process, as creating new syntactic entities containing only one segment may temporarily increase the length, but the increase is offset with the next rule applied to this entity (and generally the point of parsing is to eventually find groups longer than one segment).

Morphological actions do not change the shape of the tree, but also reduce the string representation length by deleting from the string certain interpretations. The interpretations are preserved in the tree to produce the final output, but are not interesting for further stages of parsing.

3.4 Representation of Sentence

The string representation is a compromise between XML and binary representation, designed for easy, fast and precise matching, with the use of existing regular expression libraries.

The representation describes the top level of the current state of the sentence tree, including only the informations that may be used by rule matching. For each child of the tree root, the following informations are preserved in the string: type (token / group / special) and identifier (allowing to find the entity in the tree in case an action should be applied to it). The further part of the string depends on the type — for token it is orthografic forms and a list of interpretations; for group — number of heads of the group and lists of interpretations of syntactic and semantic head.

Every interpretation consists of a base form and a morphosyntactic tag (part of speech, case, gender, numer, degree, etc.). Because the tagset used in the IPI PAN Corpus is intended to be human readable, the morphosyntactic tag is fairly descriptive (long values) and, on the other hand, compact (may have many parts ommited, for example when the category is not applicable to the given part of speech). To make pattern matching easier, the tag is converted to a string of fixed width. In the string, each character corresponds to one morphological category from the tagset (first part of speech, then number, case, gender etc.) as for example in the Czech positional tag system (Hajič and Hladká, 1997). The characters — upper- and lowercase letters, or 0 (zero) for categories non-applicable for a given part of speech — are assigned automatically, on the basis of the external tagset definition read at initialisation. A few examples are presented in table 1.

3.5 Rule Matching

The conversion from the `Left`, `Match` and `Right` parts of the rule to a regular expression over the string representation is fairly straightforward. Two exceptions — regular expressions as morphosyntactic category values and the distinction between ex-

IPI PAN tag	fixed length tag
`adj:pl:acc:f:sup`	`UBDD0C0000000`
`conj`	`B000000000000`
`fin:pl:sec:imperf`	`bB00B0A000000`

Table 1: Examples of tag conversion between human readable and inner positional tagset.

istential and universal quantification over interpretations — will be described in more detail below.

First, the rule might be looking for a token whose grammatical category is described by a regular expresion. For example, `[gender~"m."]` should match human masculine (`m1`), animate masculine (`m2`), and inanimate masculine (`m3`) tokens; `[pos~"ppron[123]+|siebie"]` should match various pronouns; `[pos!~"num.*"]` should match all segments except for main and collective numerals; etc. Because morphosyntactic tags are converted to fixed length representations, the regular expressions also have to be converted before compilation.

To this end, the regular expression is matched against all possible values of the given category. Since, after conversion, every value is represented as a single character, the resulting regexp can use square brackets to represent the range of possible values.

The conversion can be done only for attributes with values from a well-defined, finite set. Since we do not want to assume that we know all the text to parse before compiling rules, we assume that the dictionary is infinite (this is different from Poliqarp, where dictionary is calculated during compilation of corpus to binary form). The assumption makes it difficult to convert requirements with negated `orth` or `base` (for example `[orth!~"[Nn]ie"]`). As for now, such requirements are not included in the compiled regular expression, but instead handled as an extra condition in the `Action` part.

Another issue that has to be taken into careful consideration is the distinction between certain and uncertain information. A segment may have many interpretations and sometimes a rule may apply only when all the interpretations meet the specified condition (for example `[pos~~"subst"]`), while in other cases one matching interpretation should be enough to trigger the rule (`[pos~"subst"]`). The aforementioned requirements translate respectively to the following regular expressions:[1]

- `(<N[^<>]+)+`
- `(<[^<>]+)*(<N[^<>]+)(<[^<>]+)*`

Of course, a combination of existential and universal requirements is a valid requirement as well, for example: `[pos~~"subst" case~"gen|acc"]` (all interpretations noun, at least one of them in genitive or accusative case) should translate to:

```
(<N[^<>]+)*(<N.[BD][^<>]+)
(<N[^<>]+)*
```

3.6 Actions

When a match is found, the parser runs a sequence of actions connected with the given rule, described in 2.4. Each action may be condition, morphological action, syntactic action or a combination of the above (for example unify is both a condition and a morphological action). The parser executes the sequence until it encounters an action which evaluates to false (for example, unification of cases fails).

The actions affect both the tree and the string representation of the parsed sentence. The tree is updated instantly (cost of update is constant or linear to match lenght), but the string update (cost linear to sentence length) is delayed until it is really needed (at most once per rule).

4 Conclusion and Future Work

Althought morphosyntactic disambiguation rules and partial parsing rules often encode the same linguistic knowledge, 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 presents a formalism and a working prototype of a tool implementing simultaneous rule-based disambiguation and partial parsing.

Unlike other partial parsers, the tool does not expect a fully disambiguated input. The simplicity of the formalism and its implementation makes it possible to integrate a morphological analyser into parser and allow a greater flexibility in input formats.

On the other hand, the rule syntax can be extended to take advantage of the metadata present in the corpus (for example: style, media, or date of publishing). Many rules, both morphological and syntactic, may be applicable only to specific kinds of texts — for example archaic or modern, official or common.

[1]< and > were chosen as convenient separators of interpretations and entities, because they should not happen in the input data (they have to be escaped in XML).

References

Jan Hajič and Barbara Hladká. 1997. Tagging of inflective languages: a comparison. In *Proceedings of the ANLP'9y*, pages 136–143, Washington, DC.

Nancy Ide, Patrice Bonhomme, and Laurent Romary. 2000. XCES: An XML-based standard for linguistic corpora. In *Proceedings of the Linguistic Resources and Evaluation Conference*, pages 825–830, Athens, Greece.

Daniel Janus and Adam Przepiórkowski. 2006. Poliqarp 1.0: Some technical aspects of a linguistic search engine for large corpora. In Jacek Waliński, Krzysztof Kredens, and Stanisław Goźdź-Roszkowski, editors, *The proceedings of Practical Applications of Linguistic Corpora 2005*, Frankfurt am Main. Peter Lang.

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.

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

Adam Przepiórkowski. 2007. A preliminary formalism for simultaneous rule-based tagging and partial parsing. In Georg Rehm, Andreas Witt, and Lothar Lemnitzer, editors, *Datenstrukturen für linguistische Ressourcen und ihre Anwendungen – Proceedings der GLDV-Jahrestagung 2007*, Tübingen. Gunter Narr Verlag.

Adam Przepiórkowski. 2007. On heads and coordination in valence acquisition. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing (CICLing 2007)*, Lecture Notes in Computer Science, Berlin. Springer-Verlag.

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