Prolog-based query interface to syntactic dependencies extracted from biomedical literature

Techical Report ifi-2006.04

Kaarel Kaljurand and **Fabio Rinaldi** and **Gerold Schneider**
Institute of Computational Linguistics
University of Zurich
Email: {kalju,rinaldi,gschneid}@ifi.unizh.ch

Abstract

We describe a Natural Language Processing pipeline (NLPPL) which includes a syntactic dependency parser (Pro3Gres) as its main component. In addition, we describe a simple Prolog-based query language, which provides a user-friendly interface to the pipeline output.

Our work is focused on parsing texts from the biomedical domain. The query interface is designed to allow further experiments with the resulting syntactic data by both the linguists and the biomedical domain specialists. The final goal of our work is to explore the language used in the biomedical domain in attempt to extract semantic dependencies (e.g. biological pathway descriptions) from the biomedical literature.

1 Introduction

Researchers in the biomedical domain typically explore new results in the field using the PubMed[1] keyword search over the published articles and/or interfaces to manually compiled ontologies, such as the Gene Ontology[2], or pathway databases, such as KEGG[3], which classify existing domain knowledge. There are also commercial tools, such as MedScan (Daraselia et al., 2004) which attempt to automatically construct pathway descriptions from published articles, thus speeding up the process of creating semantically searchable data. Given the complexity of the later task, the output of such tools is often unreliable, or alternatively very scarce, i.e. only a few semantic relations are output for which the confidence is very high.

In the following we describe a Natural Language Processing pipeline (NLPPL), which is built around a deep-linguistic statistical dependency parser Pro3Gres (Schneider et al., 2004; Schneider, 2004), and its application to biomedical literature, which we conduct within the OntoGene project[4].

We focus on a Prolog-based query language and its HTML front-end that helps to explore the syntactic dependencies extracted by the pipeline. We hope that the query language could serve as a middle ground, enabling communication between biomedical domain specialists on the one hand, and linguists on the other hand, and that it might help to track down the relevant linguistic structures that are used to describe biological pathways (e.g. gene-protein interaction).

Our query language can be used to conduct a simple keyword search. Still, one can go further, by querying for linguistic structures, such as subjects, objects, prepositional phrases, etc. Of course, the final goal is to extract and present the complete pathways as described in the existing literature, since this is the representation that domain specialists eventually like to work with.

Our approach focuses on a small amount of data, i.e. we don't rely on redundancy in the input documents to find the relevant relations. Instead we are interested in fine-tuning our syntactic methods to cover a wide variety syntactic constructs used to talk about biomedical concepts and relations.

[1] `http://www.ncbi.nlm.nih.gov/entrez`
[2] `http://www.geneontology.org`
[3] `http://www.genome.jp/kegg`
[4] `http://www.ontogene.org`

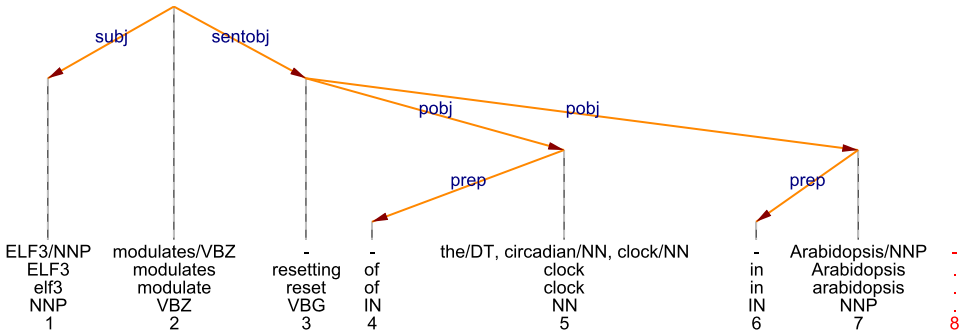

Figure 1: Tree of syntactic dependencies in the sentence "ELF3 modulates resetting of the circadian clock in Arabidopsis" along with other linguistic annotations

2 Pro3Gres and NLPPL

Pro3Gres is a fast, deep-linguistic statistical dependency parser which has recently been applied to biomedical parsing (Rinaldi et al., 2006).

Pro3Gres assumes that its input has already undergone sentence splitting, tokenization, lemmatization, and noun and verb group chunking, i.e. Pro3Gres focuses only on finding the dependencies between the heads of the chunks. This leaves a lot of room for experimentation with different off-the-self part-of-speech taggers, chunkers, terminology extractors, and allows us to choose the preprocessing tools which are optimized for a certain domain (news articles, biomedical literature).

We have implemented a pipeline which performs the following tasks required by Pro3Gres.

- Sentence splitting by MXTERMINATOR (Reynar and Ratnaparkhi, 1997)
- Tokenization by the Penn treebank tokenizer[5]
- Part-of-speech tagging by MXPOST (Ratnaparkhi, 1996)
- Lemmatization by morpha (Minnen et al., 2001)
- Terminology detection by matching the tokens against existing term lists from biomedical ontologies
- Noun and verb group chunking by LTCHUNK (Mikheev, 1997) followed by the detection of chunk heads by a simple pattern matching over the part-of-speech tags of the tokens

[5] `http://www.cis.upenn.edu/~treebank/tokenization.html`

For each pipeline component, a wrapper has been written which expects an XML input and produces an XML output, so that tools which are not XML-aware can be used seamlessly together.

The pipeline itself is implemented as an Apache Ant[6] build file. It calls the modules sequentially and manages their inputs and outputs.

When NLPPL finishes, we have gained the following annotation of the input sentences (figure 1 shows the annotation graphically):

- Sentences are tokenized and their borders are detected. Each sentence and each token has been assigned an ID.
- Each token is lemmatized.
- Tokens are grouped into chunks. Each chunk has a type (NP or VP) and a head token.
- Tokens are grouped into terms. Each term has a normal-form and a semantic type.
- Each sentence is described as a syntactic dependency structure. Each dependency occurs between two tokens and has a type.

3 Queries over syntactic dependencies

While improving on each of the pipeline components, the obvious next step would be to detect semantic relations in the sentences by attempting to solve anaphoric links, nominalizations, truth values of clauses, etc. However, we believe that the current representation is already good enough to be shared with domain specialists, so that they could help us define what further improvement is needed and what kind of linguistic processing it might depend on, to render the extracted annotations more useful for the end-users.

[6] `http://ant.apache.org`

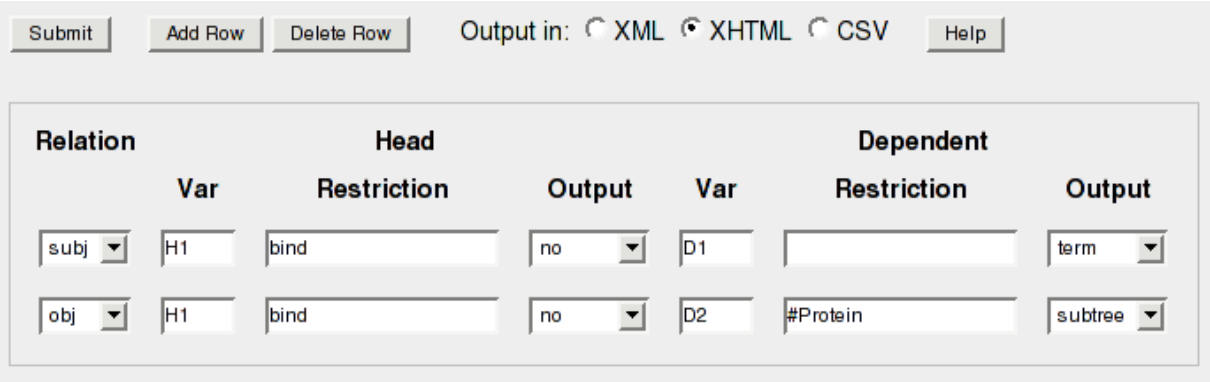

Figure 2: HTML front-end to the query language

NLPPL uses XML to represent its intermediate inputs and outputs and there is an increasing number of formalisms (such as XQuery, XSLT, etc.) that allow querying and modification of XML-formatted content. Still, for the time being, we have chosen Prolog to represent and query the extracted data. Prolog has a simple and powerful built-in query mechanism. Also, it is a well-known language among linguists.

First we convert the data (i.e. all the linguistic annotations of the input texts) into the following Prolog facts.

```
token(Id, Token, Lemma, Tag).
term(Id, Lex, Type, [Id1,...]).
chunk(Id, CHeadId, CType, [Id1,...]).
hd(HeadId, DepId, Rel, SId).
```

where *Id*, *HeadId* and *DepId* are token IDs by which the data is indexed; *token/4* maps a token ID to the actual token (*Token*), its lemmatized form (*Lemma*) and its part-of-speech tag (*Tag*); *term/4* maps a token ID to the term (i.e. a list of token IDs) to which it belongs; each term has a normalized form (*Lex*) and a semantic type (e.g. Gene, Compound, Process, etc); *chunk/4* maps a token ID to the chunk (i.e. a list of token IDs) to which it belongs; each chunk has a head (*CHeadId*) and type (*CType*) which is either a noun group or a verb group; finally, *hd/4* describes the directed dependency relation with type *Rel* between tokens with IDs *HeadId* and *DepId* in sentence with an ID *SId*.

Writing Prolog queries over this data is a trivial task, e.g. to extract all the subject-object pairs for a given verb (e.g. *bind*), we can write:

```
:- hd(HeadId, DepId1, subj, SId),
   hd(HeadId, DepId2, obj, SId),
   token(HeadId, _, bind, _),
   token(DepId1, _, SubjectLemma, _),
   token(DepId2, _, ObjectLemma, _).
```

and then restrict the output to *SubjectLemma* and *ObjectLemma*.

We have further constrained and simplified the query language to have the form:

```
:- tree([
    branch(Rel1, H1, D1, HR1, DR1),
    branch(Rel2, H2, D2, HR2, DR2),
    ...
], OutputConfiguration).
```

where *HR* and *DR* are restrictions on the lemma, term type or part-or-speech tag of the tokens. The first argument to the predicate *tree* is a list of relations (branches in the dependency tree) which must all be satisfied. Further arguments to the predicate *tree* (denoted here by *OutputConfiguration*) specify which of the instantiated variables to present in the final output. Also, we can specify how the variable must be represented in the output: either by its corresponding lemma, token, term, chunk or the whole dependency subtree that it heads. The later is important, since e.g. for the sentence in figure 1, one is probably not interested in the relation *modulate(elf3,reset)*, but instead the node corresponding to *resetting* should be expanded to include its prepositional modifiers (*of the circadian clock* and *in Arabidopsis*).

Given the final representation, a query which asks for the subject of the *bind*-relation where the syntactic object is a protein, might look like the following:

```
:- tree([
    branch(subj, H1, D1, lemma:bind, _),
    branch(obj, H1, D2, _, type:protein)
], OutputConfiguration).
```

The final query representation can be easily mapped to a web interface based on HTML forms (see figure 2). Each relation is represented by a triple *(Relation,Head,Dependent)* and the sharing of arguments by means of connecting relations

(i.e. branches of the tree) to each other.

The result of the query is presented in XML or in a tabular format (either in XHTML or CSV), where each row corresponds to one instantiation of the query variables. In addition, a link to the complete dependency tree of the sentence is provided. The tabular format enables further automatic analysis (e.g. filtering and sorting of the output which the interface currently does not allow) via e.g. spreadsheet tools.

Note that our query language can also be used to conduct a simple keyword search. The results would display the subclauses that contain the given keywords, thus hiding the rest of the sentence which is likely not to be directly relevant. In this sense, we make a small step from a traditional keyword search, while producing a richer output and preserving its reliability.

Our current implementation is based on SWI-Prolog[7]. The complete dataset, along with a small library that implements the query layer, is compiled into a Prolog saved state. A front-end which is based on HTML and Javascript communicates with the saved state over HTTP common gateway interface (CGI). Such a lightweight system is easy to deploy and is accessible from any computer which has access to the internet.

4 Future work

The current query language presupposes a good knowledge not only of the linguistic concepts such as subject, prepositional phrase, subordinated clause, but also of the expected structure of the Pro3Gres output. In order to explore a relation between two biological concepts, which are not in a direct head-dependent relation, one has to guess the possible tree-configurations that lead from one concept to the other. To express queries such as *How is a protein X related to the protein Y?*, the user would like to focus on the given concepts and not on the tree structure and find all the possible connecting syntactic relations. We believe that while restricting ourselves to our current data and preserving the simplicity of the query interface, it is possible to support more powerful queries.

One the other side, we must communicate with domain specialists to find out which of the linguistic concepts they are willing to accept as part of the query language, i.e. what is the most optimal way to make compatible the different ways how linguists and biologists talk about the biomedical concepts and functional relationships between them.

[7] `http://www.swi-prolog.org`

5 Acknowledgments

We would like to thank Christos Andronis, Andreas Persidis and Ourania Konstanti from Biovista[8] for contributing their document collection and domain expertise to this research.

[8] `http://www.biovista.com`

References

Nikolai Daraselia, Sergei Egorov, Andrey Yazhuk, Svetlana Novichkova, Anton Yuryev, and Ilya Mazo. 2004. Extracting Protein Function Information from MEDLINE Using a Full-Sentence Parser. In Tobias Scheffer, editor, *Second European Workshop on Data Mining and Text Mining for Bioinformatics*, pages 11–18, Pisa, Italy, September. ECML/PKDD.

Andrei Mikheev. 1997. Automatic rule induction for unknown word guessing. *Computational Linguistics*, 23(3):405–423.

Guido Minnen, John Carroll, and Darren Pearce. 2001. Applied morphological processing of English. *Natural Language Engineering*, 7(3):207–223.

Adwait Ratnaparkhi. 1996. A Maximum Entropy Part-Of-Speech Tagger. In *Proceedings of the Empirical Methods in Natural Language Processing Conference*. University of Pennsylvania, 17–18 May.

Jeffrey C. Reynar and Adwait Ratnaparkhi. 1997. A Maximum Entropy Approach to Identifying Sentence Boundaries. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, Washington, D.C., March 31–April 3. University of Pennsylvania.

Fabio Rinaldi, Gerold Schneider, Kaarel Kaljurand, Michael Hess, and Martin Romacker. 2006. An Environment for Relation Mining over Richly Annotated Corpora: the case of GENIA. In *SMBM 2006: 2nd International Symposium on Semantic Mining in Biomedicine*, Jena, Germany, April. Accepted for publication.

Gerold Schneider, Fabio Rinaldi, and James Dowdall. 2004. Fast, deep-linguistic statistical dependency parsing. In Geert-Jan M. Kruijff and Denys Duchier, editors, *COLING 2004 Recent Advances in Dependency Grammar*, pages 33–40, Geneva, Switzerland, 28th August. COLING.

Gerold Schneider. 2004. Combining shallow and deep processing for a robust, fast, deep-linguistic dependency parser. In Erhard Hinrichs and Kiril Simov, editors, *ESSLLI 2004 Workshop on Combining Shallow and Deep Processing for NLP*, pages 41–50, Nancy, France, August.