Supporting Universal Dependencies in Tree Editor TrEd

Jan Štěpánek[1]

[1] *Charles University, Faculty of Mathematics and Physics*

Abstract

The paper presents the tree editor TrEd and related tools that can be used to create, modify, browse, and search treebanks - large language corpora annotated with syntactic and/or semantic structure information. This might include not only phrase structure or dependencies, but also coreference, discourse analysis, and even inter-sentence relations.

The project started in the year 2000, and it has been in continuous use since then at various institutions all over the world. Most of the tools are written in Perl, which makes them available to all major operating systems.

For searching the treebanks, a query language was developed that describes sets of tree nodes and the relations between them. It also supports aggregation to produce quantitative outputs. There are two different implementations, one translates the queries into SQL statements, the other searches the data directly in the editor.

Originally, TrEd supported the PML data format used for the Prague Dependency Treebank. To process data in a different format, one first needed to convert the data into the PML format (and possibly convert the modified data data back to the initial format). Later, a versatile extension system was added to TrEd which made it possible to support other data formats directly.

We will show how this works on the example of Universal Dependencies. UD is a framework for grammar annotation across different human languages. The described extension allows TrEd (and some

other tools) to open the files in the original UD format natively, building the internal representation on the fly, and also serialise them back after editing.

1 Introduction

Corpus in the context of Natural Language Processing (NLP) is a collection of digital or digitalised texts. Corpora are often annotated, i.e. they have linguistic information added to them. The simplest kind of such an annotation is Part-of-Speech tagging, which assigns a part of speech to each word. Moreover, modern corpora often include more sophisticated pieces of information like a description of the relations between words.

Treebank is a corpus annotated with syntactic information. Usually, every sentence is represented as a tree (in the graph theory sense), while there exist two main groups of the syntactic structure being annotated: *phrase* and *dependency* structures. Phrase structures are based on Noam Chomsky's transformational grammar introduced in [1], whereas dependency structures elaborate Lucien Tesnière's ideas from [2]. The latter approach applied to English led to Stanford Dependencies [3], which combined with Google universal part-of-speech tags [4] and Interset interlingua for morphosyntactic tagsets [5] evolved into Universal Dependencies (UD, [6]) that will be described in detail below in Section 2.

The tree editor *TrEd* [7] was initially developed as the annotation tool for the Prague Dependency Treebank (PDT, [8]). In the beginning, it only supported the treebank's internal data format, but once the treebank switched to the XML-based format called Prague Markup Language (PML [9]), the editor was extended to support it, too. Furthermore, the modules implementing the new format were written in a general way, making it easier to add support for other formats used by other treebanks in future.

2 Universal Dependencies

Universal Dependencies is a framework for grammar annotation of text corpora in any human language, where "grammar" means part-of-speech tags, morphological features, and syntactic dependencies. It is an open community effort, with more than 600 contributors who have produced over 250

treebanks in more than 150 languages[1] since the first batch in 2015.

The main goal of the project is to facilitate development of multilingual parsers, cross-lingual learning, and research of the influence of language typology on parsing. It has also been used by linguists to study various language phenomena across different languages.

The dependency relations UD annotates hold between words, but there is no requirement for strict correspondence to orthographic or phonological words. Both 1:N and N:1 situations are supported: expanding contractions (e.g. French *au* can be represented as two syntactic words, *à* and *le*) or creating multi-token words (for example the abbreviation *e. g.* can enter dependency relations as a single word). The latter approach should be used scarcely, because a special technical dependency relation can often describe the same construction without complicating the tokenisation and word segmentation process.

The morphological information is composed of three parts:

- A *lemma*, a base form of the word. What a base form is is highly language-dependent, but no other information than the base form should be present in the lemma.
- A *part-of-speech tag* is one of the fixed 17 tags[2]. Some languages do not use all the tags, but adding new tags is not possible. All additional information belongs to the features. For example, `DET` designates a determiner, `VERB` and `NOUN` hopefully do not to be explained.
- A set of *features*. They typically describe the inflection of the word form (e.g. `Gender=Masc|Number=Sing`), but there can also be lexical features (e.g. the `PronType` feature categorises pronouns or pronominal adjectives, numerals, and adverbs as personal, reciprocal, interrogative, relative, demonstrative, etc.). Besides listed features, each language or treebank can define its own new features.

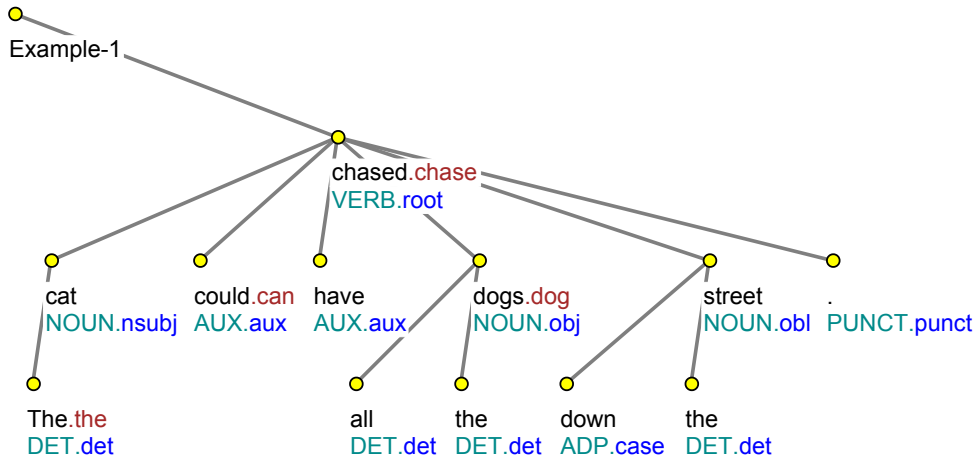
The syntactic annotation is formed of relations between words. Exactly one word is the head of the sentence, and each other word depends on another word in the sentence. The dependency relations hold primarily between content words (see Figure 1), function words "depend" on content words. This

[1] Numbers taken from UD version 2.14 from May 15, 2024.

[2] The list of all universal POS tags and their explanation can be found at https://universaldependencies.org/u/pos/index.html

is different from how function words are treated in many theories and dependency grammars, which also means the structure might need to be transformed non-trivially when translating a treebank from a different annotation schema into UD. The relations are *typed*, the set of so called *basic* types is fixed and language independent.

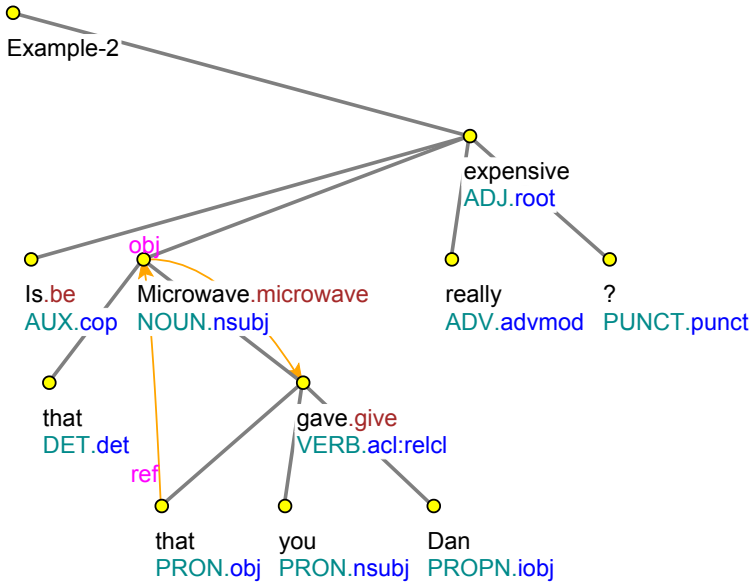
Figure 1: Basic dependencies in the sentence *The cat could have chased all the dogs down the street.* The relation is represented by a tree edge, the tree nodes correspond to words. Word forms are shown in black, lemmas in brown; to save horizontal space, only lemmas different to word forms are shown. Parts of speech are rendered in cyan. The type of a relation is displayed in blue at the depending word.



Besides basic dependencies there are also *enhanced* dependencies. They annotate implicit relations between words like additional subject relation in control and raising constructions or coreference in relative clauses (see Figure 2); they can express relations to elided words; propagation of dependencies into or from conjuncts; or further specify a basic relation (e.g. adding preposition and/or case to a nominal modifier `nmod` or adnominal clause `acl` can help disambiguate its semantic role). Enhanced dependencies do not have to form a tree and their types can bear more detailed information than the fixed basic dependency types. The enhanced graph is not required to contain all the basic dependencies, but many of them are usually part of it.

UD uses the so called CoNLL-U format to serialise the data. Every node is represented by a line, tabs separate the line into columns corresponding to

Figure 2: Enhanced dependencies in the sentence *Is that Microwave that you gave Dan really expensive?*: *that* references *Microwave* which in turn is an object of *give*.



the identifier, form, lemma, part of speech, language-specific part of speech, feature set, head identifier, dependency type, enhanced dependency, and miscellaneous. For example, the word *that* in Figure 2 is represented by the line shown in Figure 3. The enhanced dependency column contains both the basic dependency and the coreference separated by a vertical bar, it encodes both the head of the dependency (3 in the enhanced case, i.e. *Microwave*) and the type (`ref` here), separated by a colon.

Figure 3: Columns representing *that* in Figure 2.

```
ID  FORM  LEMMA UPOS XPOS FEATS        HEAD DEPREL DEPS        MISC
4   that  that  PRON WDT  PronType=Rel 6    obj    3:ref|6:obj _
```

3 Tree Editor TrEd

3.1 The Tree Editor

The editor is written in Perl[3]. It makes it available for all major operating systems of personal computers. The released code is available for download and licensed under GNU GPL 2. The largest part of the source code has been developed in a private SVN repository—but it is currently moving to git and GitHub, where some other parts or related projects are located already. Unfortunately, this change influences the release process, which makes the transition more complex.

For the GUI, the project uses the Tk[4] modules. Tk::Canvas turned out to be quite flexible and powerful when it comes to connecting circles by lines or curves of various colours and styles, as well as moving objects around with a mouse.

By dragging tree nodes, annotators can change the structure of trees: by dropping one node over another, they make the former a child of the latter.

TrEd can display and edit much more than just trees: many treebanks also contain coreference links, references between elements of different layers of annotation, or groups of words (multi-word expressions). Different types of links between nodes can be created by dropping the nodes while holding `Shift`, `Control`, `Alt`, or their combinations. Annotators' most frequent actions can be assigned to keyboard shortcuts, speeding up the process and making it less repetitive.

TrEd can be used to annotate data manually, but it also has a "batch" version without the GUI. It can run Perl code against data files to perform changes in the data, search the data, or aggregate information extracted from the data. The process can be parallelised on a multi-CPU machine or in a computer cluster, enabling fast processing of large data sets.

Inspired by Perl's `-n` and `-p` switches, the `btred` (batch TrEd) command implemented `-T` and `-N` which automatically run the given code for each tree and node, respectively. The shell command in Listing 1 searches all the requested UD files for nominal subjects whose parents are not verbs (not really interesting from a linguistic point of view, the parents are most probably adjectives, nouns, and pronouns in copula constructions, for example the word *expensive* in Figure 2):

[3] https://perl.org
[4] https://metacpan.org/pod/Tk

```
btred -NTe '
    FPosition()
        if $this->{deprel} eq "nsubj"
        && $this->parent->{upostag} ne "VERB"
    ' data/*.conllu | tred -l-
```

Listing 1: Processing UD files from the command line.

The variable `$this` is assigned each node in each tree in each file. The function `FPosition` prints a file name, a sentence number, and a node number in a special format that TrEd understands, so we can directly browse the matching nodes.

As a by-product of the treebank creation, various aspects of the annotation process have been studied, e.g. inter-annotator agreement and its development over time, or the influence of automatic pre-annotation on the speed and accuracy of the annotation.

3.2 The Data Format

Internally, TrEd uses the PML format for its data. The distribution implementing it can be found on CPAN as Treex::PML[5]. Each specific annotation project that uses PML first defines its own PML schema that describes how the general PML data structures are organised in the data. The schema is used by Treex::PML to precompile fast data loading code in Perl. By default, XML is used as the format of both the schema and data, but Treex::PML provides means to change even this aspect of the data. The project can consist of several interconnected layers of annotation, each layer can be a simple linear annotation, or can contain dependency or constituency trees.

PML defines the following data types:

- A *document* represents one collection of annotated related sentences, usually a list or sequence (see below) of trees plus some metadata.
- A *node* represents a node in a tree (usually corresponding to a word in a sentence). The whole sentence is represented by its root node. Each node can have a different type, which describes from what simpler types it should be composed.

[5] https://metacpan.org/pod/Treex::PML

- A *list* corresponds to an array, it consists of any number of elements of the same type.
- An *alternative* is similar to a list, but its semantics is different: while a list represents all its elements as valid values, an alternative represents just one of the presented values, but it is not clear which one it should be.
- A *sequence* is again similar to a list, but it does not require its elements to be of the same type (the possible combinations of types can be restricted, though).
- A *structure* is similar to a Perl hash. Some of its keys might be marked as obligatory, each key must have a corresponding value type specified by the schema.
- A *container* is a special type of a structure with only one central value of a given type, possibly annotated with other name-value pairs with atomic values.
- An *atomic* value is a literal string, but its format might be further specified in the schema (e.g. an integer, date, identifier, reference to an identifier).

So called *roles* are assigned to particular object definitions in the schema, telling TrEd (or other applications) how to interpret the data. For example, the `#NODE` role tells TrEd it should build a tree node from the objects corresponding to the given schema definition.

If we compare the UD data format based on table separated values with very limited internal structure to the PML format, we can see that they are substantially different, which makes the conversion between them a bit challenging.

3.3 Searching

Trivial searches are possible in the data directly from TrEd, e.g. you can specify a regular expression for an atomic value. To search for complex structures (e.g. a subject of a passive verb in past tense with more than one temporal adverbials), one can write a subroutine in Perl returning the

positions of matching nodes and run it in the batch mode. Unfortunately, not all linguists can write complex Perl code.

Therefore, PML Tree Query (PML-TQ, [10]) was developed in 2009. It consists of three components:

- A *query language* that can express relations between nodes, cross-layer relations, and arbitrary boolean combinations of statements. It also includes a sub-language to quantitative analysis of the results.
- There are three *client interfaces*: a GUI client implemented directly in TrEd with graphical query builder, a command line interface, and a web interface that provides some assistance in building a query.
- There are two *engines* that evaluate the queries. One translates the query into SQL and searches the data saved to a relational database, the second one is implemented in Perl and searches the data files directly. The latter is significantly slower, but suitable for data in the process of annotation that change frequently, as the conversion to a database is even more time consuming.

For example, to get a picture of what parts of speech are the non-verb parents of nominal subjects (cf. Listing 1), one can use the query shown in Figure 4. Note that nodes must be named if their attributes are later referred to in the first line of the analytical part, any other line refers to the columns of the preceding line by their numbers. The answer for all the English UD data in version 2.14 is shown in Table 1.

3.4 Extensions

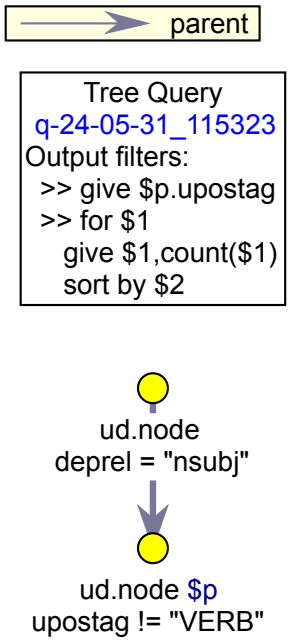
TrEd can be enhanced by extensions. Extensions are plugins that can provide additional Perl modules, resources (e.g. PML schemata), style sheets, and/or annotation modes (usually adjusting keyboard and mouse behaviour for a specific annotation project). TrEd also supports minor modes (inspired by a similar feature of GNU Emacs[6]) which can change its behaviour across annotation modes (for example, there is a minor mode for displaying neighbouring sentences).

Extensions are organised into repositories which are accessible by the HTTP protocol. Upon request, TrEd can download all available extensions

[6] https://www.gnu.org/software/emacs/

Figure 4: Example query in the query language and its graphical representation.

```
ud.node [
    deprel = "nsubj",
    parent ud.node $p := [
        upostag != "VERB"
    ]
];
>> give $p.upostag
>> for $1
   give $1, count($1)
   sort by $2
```



and let the user choose which ones to install, or upgrade the installed ones if newer versions are found. Currently, there are 55 extensions available in the official TrEd repositories[7].

4 UD in TrEd

Most existing extensions for non-PML data formats included conversion scripts from the native format to PML and back (see for example the Penn Treebank extension[8]). When developing the extension for UD, we decided to use a different strategy.

TrEd makes it possible to write a *backend* for a non-PML data format. The backend is a module that implements a given interface, most importantly the `read` and `write` functions. The functions are responsible for converting

[7] https://ufal.mff.cuni.cz/tred/extensions/
[8] https://ufal.mff.cuni.cz/tred/extensions/core/ptb/documentation/

PUNCT	2
X	9
PART	11
INTJ	26
DET	54
ADP	130
SYM	131
NUM	299
PROPN	605
ADV	735
AUX	929
PRON	1264
NOUN	5622
ADJ	7169

Table 1: Output of the query from Figure 4: Parts of speech of non-verbal parents of nominal subjects in the English UD data.

the data format into TrEd's internal representation and serialising the internal representation back to the original data format.

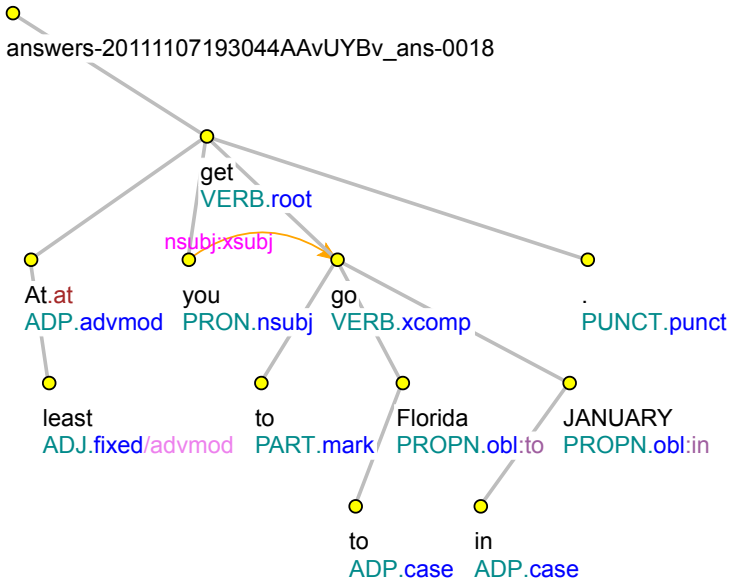
At the beginning, no annotation tool for UD was needed, as most data were converted to UD from existing treebanks in other data formats. Therefore, the extension implemented no advanced editing methods. Later on, when UD became more prominent, new annotation projects started using UD natively, leading to incorporation of more complex functionalities.

The extension also contains a *style sheet* that configures how trees are drawn in TrEd (Figures 1 and 2 show simple examples).

- The word form is displayed as a black label of each node, the label is grey for empty nodes. The part of speech tag is displayed in dark cyan.
- As each non-root node must have a single parent, the dependency type is shown as a blue label of the child node.
- The lemma is displayed only if it is different from the word form to narrow the trees.
- If the enhanced dependency is the same as the basic one, it is not shown at all. If there is an additional dependency, it is shown as an

orange curved arrow with the dependency type shown in magenta near the start of the arrow. If the basic dependency is not duplicated in the enhanced one, the tree edge is coloured in cyan instead of grey and is thinner. If the enhanced dependency has the same parent as the basic one but a different type, the type can be displayed in two ways: if it begins with the basic type, only the extra characters are displayed in blue violet, otherwise, it is displayed full in violet after a slash (see Figure 5: the prepositions *to* and *in* extend the basic dependency type `obl`, while the basic dependency type `fixed` is replaced by the type `advmod` in the enhanced dependency graph).

Figure 5: Enhanced dependencies to parent, but with a type different to the basic one: *At least you get to go to Florida in JANUARY.*

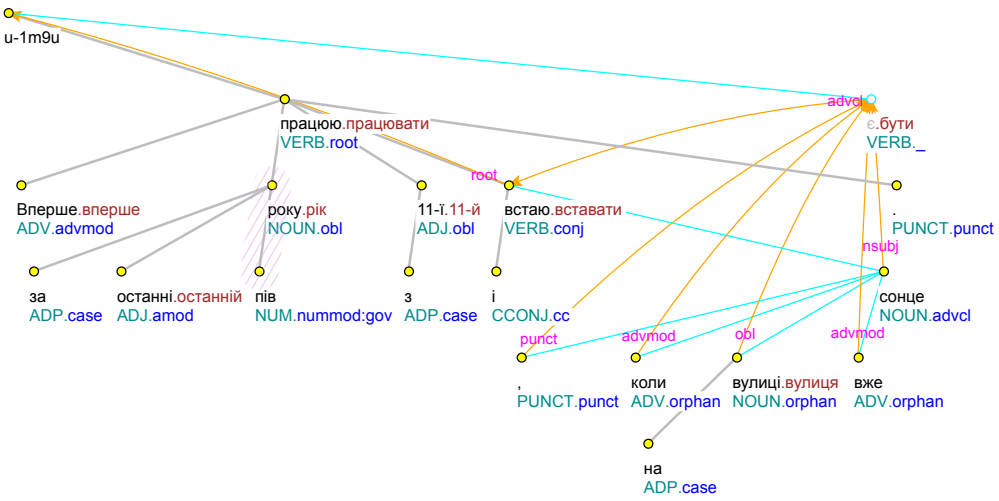


- An expanded contraction is represented by a dashed area covering the individual words (see Figure 6 for a bit more complex example).

Moreover, the extension implements basic editing functionality:

- Dropping node *a* on node *b* attaches node *a* to node *b*. An enhanced dependency is created together with the basic one, if an enhanced dependency existed to the old or new parent, it is removed.

Figure 6: Different basic and enhanced dependencies: *Вперше за останні півроку працюю з 11-ї і встаю, коли на вулиці вже сонце.*



- Dropping node *a* on node *b* while pressing `Control` toggles existence of an enhanced dependency of node *a* on node *b*. If a dependency was created, the user is asked for its type. This works even when *a* is already *b*'s parent.
- Dropping node *a* on node *b* while pressing `Shift` attaches node *a* to node *b* without creating an enhanced dependency. If an enhanced dependency existed to the old or new parent, it is removed.
- Pressing `Insert` inserts a null child node after the current one. The user is asked for its form.
- Pressing `Control`+→ or `Control`+← adds the current node to a contraction group to the right or left from the node. If the neighbouring node is not in a group, a new group is created and the user is asked for its form. If the nodes are already in a group, the group is deleted.
- Pressing `d` opens a dialog box to edit the current node's dependency type. If previously an enhanced dependency of the same type as the basic dependency existed, it is changed together with it, otherwise, it stays unchanged.

- Pressing `p`, `f`, and `l` makes it possible to edit `upostag`, `feats`, and `lemma`, respectively.

By making all the UD data readable by TrEd, we also made them searchable by both the engines of PML-TQ directly without any intermediate conversion steps. The Perl engine can be run locally, the database engine is available at a dedicated LINDAT/CLARIAH-CZ page[9].

5 Conclusion

Annotation projects that use the Universal Dependencies framework natively can use the tree editor TrEd as a tool for manual annotation. It should be now available on any major computer operating system. Moreover, it offers an expressive searching language with the editor itself used as a graphical client.

Acknowledgement

This paper has been supported by the Ministry of Education, Youth and Sports of the Czech Republic, Project No. LM2023062 LINDAT/CLARIAH-CZ.

References

[1] Chomsky N. Syntactic Structures. The Hague/Paris: Mouton & Co.; 1957.

[2] Tesnière L. Éléments de syntaxe structurale. Paris: Klincksieck; 1959.

[3] de Marneffe MC, Manning CD. The Stanford Typed Dependencies Representation. In: Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation. Manchester, UK: Coling 2008 Organizing Committee; 2008. p. 1–8.

[4] Petrov S, Das D, McDonald R. A Universal Part-of-Speech Tagset. In: Proceedings of the 8th International Conference on Language Resources

[9] https://lindat.mff.cuni.cz/services/pmltq

and Evaluation (LREC 2012). İstanbul, Turkey: European Language Resources Association; 2012. p. 2089–2096.

[5] Zeman D. Reusable Tagset Conversion Using Tagset Drivers. In: Proceedings of the 6th International Conference on Language Resources and Evaluation (LREC 2008). Marrakech, Morocco: European Language Resources Association; 2008. p. 213-8.

[6] de Marneffe MC, Manning CD, Nivre J, Zeman D. Universal Dependencies. Computational Linguistics. 2021;47(2):255-308.

[7] Pajas P, Štěpánek J. Recent Advances in a Feature-Rich Framework for Treebank Annotation. In: Scott D, Uszkoreit H, editors. The 22nd International Conference on Computational Linguistics - Proceedings of the Conference. vol. 2. Manchester, UK: Coling 2008 Organizing Committee; 2008. p. 673-80.

[8] Hajič J, Böhmová A, Hajičová E, Vidová-Hladká B. The Prague Dependency Treebank: A Three-Level Annotation Scenario. In: Abeillé A, editor. Treebanks: Building and Using Parsed Corpora. Amsterdam: Kluwer; 2000. p. 103-27.

[9] Pajas P, Štěpánek J. XML-Based Representation of Multi-Layered Annotation in the PDT 2.0. In: Hinrichs RE, Ide N, Palmer M, Pustejovsky J, editors. Proceedings of the LREC Workshop on Merging and Layering Linguistic Information (LREC 2006). Genova, Italy: ELRA; 2006. p. 40-7.

[10] Pajas P, Štěpánek J. System for Querying Syntactically Annotated Corpora. In: Proceedings of the ACL-IJCNLP 2009 Software Demonstrations. Suntec, Singapore: Association for Computational Linguistics; 2009. p. 33-6.