PBML

The Prague Bulletin of Mathematical Linguistics

NUMBER 90 DECEMBER 2008 5–32

Netgraph Query Language for the Prague Dependency Treebank 2.0

Jiří Mírovský

Abstract

We study the annotation of the Prague Dependency Treebank 2.0 (PDT 2.0) and assemble a list of requirements on a query language that would allow searching for and studying all linguistic phenomena annotated in the treebank. We propose an extension to the query language of an existing search tool Netgraph 1.0 and show that the extended query language satisfies the list of requirements. We demonstrate how all principal linguistic phenomena annotated in the treebank can be searched for with the proposed query language and compare the query language to some other treebank search systems. The proposed query language has been implemented in the search tool Netgraph – we talk about features of a search tool that can simplify the searching and make it more powerful. We also present a table that shows the extent of usage of various features of the implemented query language by the users of Netgraph and mention several usages of Netgraph for other treebanks than PDT 2.0.

1. Introduction

Linguistically annotated treebanks play an essential role in modern computational linguistics. The more complex the treebanks become, the more sophisticated tools are required for using them, namely for searching in the data. A search tool helps extract useful information from the treebank, in order to study the language, the annotation system or even to search for errors in the annotation. The Prague Dependency Treebank 2.0 (Hajič et al. 2006), which is a sequel to the Prague Dependency Treebank 1.0 (Hajič et al. 2001), is one of the most advanced manually annotated treebanks in the linguistic world.

Three sides existed whose connection needed to be solved. First, it was the Prague Dependency Treebank 2.0 with its extensive annotation. Second, there existed a very limited but extremely intuitive search tool – Netgraph 1.0 (Ondruška 1998). Third, there were users longing for such a simple and intuitive tool that would be powerful enough to search in the Prague Dependency Treebank.

We study the annotation of the Prague Dependency Treebank 2.0 (PDT 2.0), especially on the tectogrammatical layer, which is by far the most complex layer of the treebank, and assemble a list of requirements on a query language that would allow searching for and studying all linguistic phenomena annotated in the treebank. We propose an extension to the query language of the existing search tool Netgraph 1.0 and show that the extended query language satisfies the list of requirements. We also demonstrate how all principal linguistic phenomena annotated in the treebank can be searched for with the proposed query language and compare the query language to some other treebank search systems. The proposed query language has also been implemented in the search tool Netgraph – we talk about features of a search tool that can simplify the searching and make it more powerful. We also present a table that shows the extent of usage of various features of the implemented query language by the users of Netgraph and mention several usages of Netgraph for other treebanks than PDT 2.0.

More details about the topics of this contribution can be found in Mírovský (2008e).

2. The Analysis of the Problem

We study linguistic phenomena annotated in PDT 2.0, in order to decide what features a query language of a search tool needs to have to allow searching for these phenomena and studying them (Mírovský 2008d). Afterwards, we summarize the features and formulate a concise list of linguistic requirements on a query language for PDT 2.0.

2.1. Linguistic Phenomena in PDT 2.0

PDT 2.0 has three layers of annotation: the morphological layer (Hana et al. 2005), the analytical layer (Hajič et al. 1997), and the tectogrammatical layer (Hajičová 1998). To be exact, there is one more layer – the word layer – which only keeps the tokenized original data and (apart from the tokenization) does not contain any annotation.

Our work is focused on the two structured layers – the analytical layer and the tectogrammatical layer. We intend to access the morphological information only from the higher layers, not directly. Since there is a 1:1 relation among nodes on the analytical layer (but for the technical root) and tokens on the morphological layer, the morphological information can be easily merged into the analytical layer – the nodes only get additional attributes. We study two ways of accessing the data of PDT 2.0:

- the analytical layer directly, the morphological and word layer information merged into the analytical layer; the tectogrammatical layer inaccessible,
- the tectogrammatical layer directly, the analytical layer "through" this layer, the morphological and word layer annotation merged into the analytical layer.

The difference between these two approaches is not only in the presence of the tectogrammatical layer, but also in the way of accessing the information from the lower layers, which is inevitably caused by the non-1:1 relation between the analytical and the tectogrammatical layer.

Since the tectogrammatical layer is by far the most complex layer in the treebank, we start our analysis with a study of the annotation manual for the tectogrammatical layer (t-manual, Mikulová et al. 2006) and focus also on the requirements on accessing the lower layers with non-1:1 relations. Afterwards, we add some requirements on the query language concerning the annotation of the lower layers – the analytical layer and the morphological layer.

During the studies, we have to keep in mind that we do not only want to search for a phenomenon, but also need to study it, which can be a much more complex task. Therefore, it is not sufficient e.g. to find a predicative complement, which is a trivial task, since the attribute `functor` of the complement is set to the value `COMPL`. In this particular example, we also need to be able in the query to specify properties of the node the second relation of the complement goes to, e.g. that it is an Actor.

2.1.1. The Tectogrammatical Layer

Basic Principles

The basic unit of annotation on the tectogrammatical layer of PDT 2.0 is a sentence as a basic means of conveying meaning (t-manual, page 8). The representation of the tectogrammatical annotation of a sentence is a rooted dependency tree. (More exactly, a tectogrammatical tree structure, TGTS, see Sgall (2001) for the differences between a TGTS and a theoretically substantiated tectogrammatical representation.) It consists of a set of nodes and a set of edges. One of the nodes is marked as the root. Each node is a complex unit accompanied by a set of attribute-value pairs. The edges express dependency relations between the nodes. The edges do not have their own attributes; attributes that logically belong to the edges (e.g. a type of the dependency) are represented as node-attributes (t-manual, page 9).

This implies the first and most basic requirement on the query language: one result of the search is one sentence along with the tree belonging to it. Also, the query language should be able to express the node evaluation and the tree dependency among nodes in the most direct way.

Valency

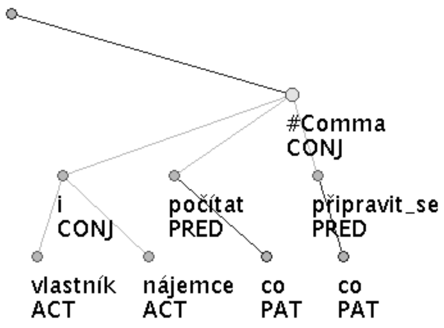
Valency (Hajičová, Panevová 1984) of verbs, valency of verbal nouns, valency of nouns that represent the nominal part of a complex predicate and valency of some adverbs are annotated fully in the trees (t-manual, pages 162-3). Since the valency of verbs is the most complete in the annotation and since the requirements on searching for valency frames of nouns are the same as those concerning verbs, we will (for the sake of simplicity in expressions) focus on the verbs only. Verbs usually have more than one meaning; each is assigned a separate valency frame. Every verb has as many valency frames as it has meanings (t-manual, page 105).

Therefore, the query language has to be able to distinguish valency frames and search for each one of them, at least as long as the valency frames differ in their members and not only in their index. (Two or more identical valency frames may represent different verb meanings (t-manual, page 105).) The required features include a presence of a son, its absence, and a possibility to control the number of sons of a node.

Coordination and Apposition

A tree dependency is not always a linguistic dependency (t-manual, page 9). Coordination and apposition are examples of the phenomenon (t-manual, page 282). If a Predicate governs two coordinated Actors, these Actors depend on a coordinating node and this coordinating node depends on the Predicate. The query language should be able to skip such a coordinating node. In general, there should be a possibility to skip any type of node.

Skipping a given type of node helps but is not sufficient. The coordinated structure can be more complex, for example the Predicate itself can be coordinated too. Then, the Actors do not even belong to the subtree of any of the Predicates. In the following example, the two Predicates (“PRED”) are coordinated with conjunction (“CONJ”), as well as the two Actors (“ACT”). The linguistic dependencies go from each of the Actors to each of the Predicates but the tree dependencies are quite different:



In Czech: *S čím mohou vlastníci i nájemci počítat, na co by se měli připravit?*
In English: *What can owners and tenants expect, what should they get ready for?*

The query language should therefore be able to express the linguistic dependency directly. The information about the linguistic dependency, as well as many other phenomena, is annotated in the treebank by means of references (see Coreferences below).

Other Phenomena

Similarly, other phenomena annotated in the treebank are studied in Mírovský (2008e). For the lack of space in this paper, let us only briefly list the phenomena and their requirements.

Idioms (Phrasemes) etc.

The query language has to offer at least a basic searching procedure in the linear form of the sentences, to allow searching for any idiom or phraseme, regardless of the way it is or is not captured in the tectogrammatical tree. It can even help in a situation when the user does not know how a certain linguistic phenomenon is annotated on the tectogrammatical layer.

Complex Predicates

There are problematic cases of annotation of complex predicates where the expressed valency modification occurs in the same form in the valency frames of both components of the complex predicate (t-manual, page 362).

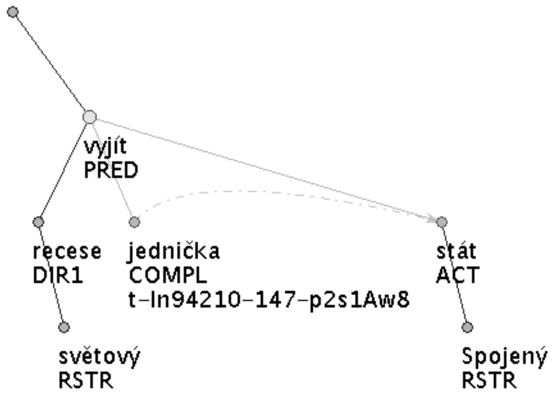
To study these special cases of valency, the query language has to offer a possibility to define that a valency member of the verbal part of a complex predicate is at the same time a valency member of the nominal part of the complex predicate, possibly with a different function. The identity of valency members is annotated by means of references, which is explained later (see Coreferences below).

Predicative Complement (Dual Relations)

The predicative complement is a non-obligatory free modification (adjunct) which has a dual semantic relation. It simultaneously modifies a noun and a verb (which can be nominalized). These two relations are represented by different means (t-manual, page 376):

- the relation to a verb is represented by means of an edge (which means it is represented in the same way as other modifications),
- the relation to a noun is represented by means of attribute `compl.rf`, the value of which is the identifier of the modified noun.

In the following example, the predicative complement (“`COMPL`”) has one relation to the verb (“`PRED`”) and another (dual) relation to the noun (“`ACT`”):



In Czech: *Ze světové recese vyšly jako jednička Spojené státy.*
In English: *The United States emerged from the world recession as number one.*

The second form of relation, represented once again with references (still see Coreferences just below), has to be expressible in the query language.

Coreferences

Grammatical coreference and textual coreference (Kučová et al. 2003) are annotated on the tectogrammatical layer. The current way of representing coreference uses references (t-manual, page 996).

Let us finally explain what references are. References are pointers, a technical way of expressing various relations between nodes[1]. They make use of the fact that every node of every tree has an identifier (the value of the attribute `id`), which is unique within PDT 2.0. If coreference, dual relation, or valency member identity is merely a link between two nodes (one node referring to another), it is enough to specify the identifier of the referred node in an appropriate attribute of the referring node. Reference types are distinguished by different referring attributes. Individual reference subtypes can be further distinguished by the value of another attribute.

The essential point in references (for the query language) is that at the time of forming a query, the value of the reference is unknown. For example, in the case of dual relation of the predicative complement, we know that the value of the attribute `compl.rf` of the complement must be the same as the value of the attribute `id` of the governing noun, but the value itself differs tree from tree and therefore is unknown at the time of creating the query. The query language has to offer a possibility to bind these unknown values.

Communicative Dynamism

Communicative dynamism (underlying order of nodes, cf. Hajičová et al. 1998) requires that the relative order of nodes in the tree from left to right can be expressed. The order of nodes is controlled by the attribute `deepord`, which contains a non-negative real (usually natural) number that sets the order of the nodes in the tree from left to right. Therefore, we will again need to refer to a value of an attribute of another node but this time with a relation other than "equal to".

Focus Proper and Quasi-Focus

Focus proper is the most dynamic and communicatively significant contextually non-bound part of the sentence. Focus proper is placed on the rightmost path leading from the effective root of the tectogrammatical tree, even though it occupies a different position in the surface structure. The node representing this expression will be placed rightmost in the tectogrammatical tree (t-manual, page 1129).

Quasi-focus is constituted by a contextually bound expression, on which the focus proper is dependent. The focus proper can immediately depend on the quasi-focus, or it can be a more deeply embedded expression. In the underlying word order, nodes representing the quasi-focus, although they are contextually bound, are placed to the right from their governing node.

[1] References are a technical term. They should not be confused with linguistic terms like coreferences.

Nodes representing the quasi-focus are therefore contextually bound nodes on the rightmost path in the tectogrammatical tree (t-manual, page 1130).

The ability of the query language to distinguish the rightmost node in the tree and the rightmost path leading from a node is therefore necessary.

Focalizers

Focalizers are expressions whose function is to signal the topic-focus articulation categories in the sentence, namely the communicatively most important categories – the focus and the contrastive topic.

Focalizers bring a further requirement on the query language – an ability to control the distance between nodes (in the terms of deep word order); at the very least, the query language has to distinguish an immediate brother and the relative horizontal position of nodes.

(Non-)Projectivity

Projectivity (Havelka 2007) is defined as follows: between a father and its son there can only be direct or indirect sons of the father (t-manual, page 1135).

The relative position of a node (node A) and an edge (nodes B, C) that together cause a non-projectivity forms four different configurations: ("B is to the left from C" or "B is to the right from C") x ("A is on the path from B to the root" or "it is not").

To be able to search for all configurations in one query, the query language should be able to combine several queries into one multi-query. We do not require that a general logical expression can be set above the single queries. We only require a general OR combination of the single queries.

2.1.2. Accessing Lower Layers

Studies of many linguistic phenomena require a multilayer access. For example, the query "find an example of a Patient that is more dynamic than its governing Predicate (with greater `deepord`) but on the surface layer is on the left side from the Predicate" requires information both from the tectogrammatical layer and the analytical layer (for the study of these phenomena, see Hajičová 2007).

As we have already said, information from the lower layers can be easily compressed into the analytical layer, since there is a 1:1 relation among tokens/nodes of the layers (with some rare exceptions like misprints on the w-layer). The interrelationship between the tectogrammatical layer and the analytical layer is much more complex. Several nodes from the analytical layer may be (and often are) represented by one node on the tectogrammatical layer and new nodes without an analytical counterpart may appear on the tectogrammatical layer. It is necessary that the query language addresses this issue and allows access to the information from the lower layers.

2.1.3. The Analytical Layer (and Lower Layers)

Requirements (on a query language) of most linguistic phenomena annotated on the analytical layer have already been covered in the previous section, discussing the tectogrammatical layer. The lower layers only supplement a few additional requirements.

Morphological Tags

In PDT 2.0, morphological tags are positional. They consist of 15 characters, each representing a certain morphological category.

The query language has to offer a possibility to specify a part of the tag and leave the rest unspecified. It has to be able to set such conditions on the tag as "this is a noun", or "this is a plural in the accusative". Some conditions might include negation or enumeration, like "this is an adjective that is not in the accusative", or "this is a noun either in the dative or the accusative". This is best done with some sort of wild cards. The latter two examples suggest that such a powerful tool as regular expressions may be needed.

Agreement

There are several cases of agreement in the Czech language, like agreement in case, number and gender in attributive adjective phrases, agreement in gender, person and number between predicate and subject (though it may be complex), or agreement in case in apposition.

To study agreement, the query language has to allow to make a reference to only a part of a value of an attribute of another node, e.g. to the fifth position of the morphological tag for case.

Word Order

Word order is a linguistic phenomenon widely studied on the analytical layer, because this layer offers a perfect combination of word order (the same as in the sentence) and syntactic relations between the words. The same technique as with the deep word order on the tectogrammatical layer can be used here. The order of words (tokens) and also nodes in the analytical tree is controlled by the attribute `ord`.

The only new requirement on a query language is an ability to measure the horizontal distance between words, to satisfy linguistic queries like "find trees where a preposition and the head of the noun phrase are at least five words apart".

2.2. Linguistic Requirements

Let us summarize what features a query language has to have to suit PDT 2.0. We list the features from the previous section and also add some obvious requirements that have not been mentioned so far but are very useful generally, regardless of a corpus.

2.2.1. Complex Evaluation of a Node

- multiple attributes evaluation (an ability to set values of several attributes at one node)
- alternative values (e.g. to define that `functor` of a node is either a disjunction or a conjunction)
- alternative nodes (alternative evaluation of the whole set of attributes of a node)
- wild cards (regular expressions) in values of attributes (e.g. `m/tag=''N...4.*''` defines that the morphological tag of a node is a noun in the accusative, regardless of other morphological categories)
- negation (e.g. to express "this node is not an Actor")
- relations lower than ("<") , higher than (">") (for numerical attributes)

2.2.2. Dependencies Between Nodes (Vertical Relations)

- immediate, transitive dependency (existence, non-existence)
- vertical distance (from root, from one another)
- number of sons (zero for leaves)

2.2.3. Horizontal Relations

- precedence, immediate precedence (positive, negative)
- horizontal distance
- secondary edges, secondary dependencies, coreferences, long-range relations

2.2.4. Other Features

- multi-tree queries (combined with general `OR` relation)
- skipping a node of a given type (for skipping simple types of coordination, apposition etc.)
- skipping multiple nodes of a given type (e.g. for recognizing the rightmost path)
- references (for matching values of attributes unknown at the time of creating the query)
- accessing several layers of annotation at the same time with a non-1:1 relation (for studying relations between layers)
- searching in the surface form of the sentence

3. The Query Language

In this contribution, we only shortly and selectively introduce a query language that satisfies linguistic requirements stated in the previous section. We present the language informally on a series of examples. A full description of the query language, as well as a formal definition of the textual form of the query language, can be found in Mírovský (2008e). The query language is an extension to the existing query language of Netgraph 1.0 (Ondruška 1998).

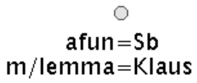
A query in Netgraph is always a tree (or a multi-tree, see below) that forms a subtree in the result trees. The treebank is searched tree by tree and whenever the query is found as a subtree of a tree, the tree becomes a part of the result.

3.1. The Basics

The simplest possible query is a simple node without any evaluation:

This query matches all nodes of all trees in the treebank, each tree as many times as how many nodes there are in the tree.

Values of attributes of the node can be specified in the form of `attribute=value` pairs:



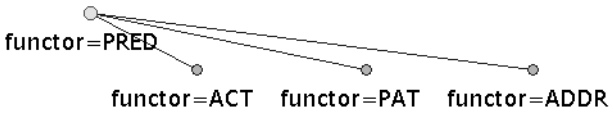
The query searches for all trees containing a node evaluated as Subject (“`Sb`”) with lemma Klaus.

3.2. Regular Expressions

A Perl-like regular expression (Hazel 2007) can be used as a whole value of an attribute. If the value of an attribute is enclosed in quotation marks, the value is considered an anchored regular expression.

3.3. Dependencies Between Nodes

Dependencies between nodes are expressed directly in the syntax of the query language. Since the result is always a tree, the query also is a tree (or a multi-tree, see Section below) and the syntax does not allow non-tree constructions. The following query searches for Predicates (“`PRED`”) that directly govern an Actor (“`ACT`”), a Patient (“`PAT`”) and an Addressee (“`ADDR`”):



It is important to note that the query does not prevent other nodes in the result being sons of the Predicate and that the order of the sons as they appear in the query can differ from their order in the result trees.

3.4. Meta-Attributes

Meta-attributes are attributes that are not present in the corpus, yet they pretend to be ordinary attributes and users can treat them the same way as normal attributes. There are eleven meta-attributes, each adding some power to the query language, enhancing its semantics, while keeping the syntax of the language on the same simple level. To be easily recognized, names of the meta-attributes start with an underscore ("_").

3.4.1. _transitive

This meta-attribute defines a transitive edge. It has two possible values: the value `true` means that a node may appear anywhere in the subtree of a node matching its query-father, the value `exclusive` means, in addition, that the transitive edge cannot share nodes in the result tree with other exclusively transitive edges.

3.4.2. _optional

The meta-attribute `_optional` defines an optional node. It may but does not have to be in the result tree at a given position. Its father and its son (in the query) can be the direct father and son in the result. Only the specified node can appear (once or more times as a chain) between them in the result tree. Possible values are:

- `true` – There may be a chain of unlimited length (even zero) of nodes matching the optional node in the result tree between nodes matching the query-father and the query-son of the optional node.
- *a positive integer* – There may be a chain of length from zero up to the given number of nodes matching the optional node in the result tree between nodes matching the query-parent and the query-son of the optional node.

3.4.3. _#sons

The meta-attribute `_#sons` ("number of sons") controls the exact number of sons of a node in the result tree.

3.4.4. _#hsons

The meta-attribute `_#hsons` ("number of hidden sons") is similar to the meta-attribute `_#sons`. It controls the exact number of hidden sons of a node in the result tree. Hidden sons are used to access lower layers of annotation from the tectogrammatical layer, see Section .

3.4.5. _depth

The meta attribute `_depth` controls the distance of a node in the result tree from the root of the result tree.

3.4.6. _#descendants

The meta-attribute `_#descendants` ("number of descendants") controls the exact number of all descendants of a node (number of nodes in its subtree), excluding the node itself.

3.4.7. _#lbrothers

The meta-attribute `_#lbrothers` ("number of left brothers") controls the exact number of left brothers of a node in the result tree.

3.4.8. _#rbrothers

Similarly, the meta-attribute `_#rbrothers` ("number of right brothers") controls the exact number of right brothers of a node in the result tree.

3.4.9. _#occurrences

The meta-attribute `_#occurrences` ("number of occurrences") specifies the exact number of occurrences of a particular node at a particular place in the result tree. It controls how many nodes of the kind can occur in the result tree as sons of the father of the node (including the node itself). Zero value can be used to express that a particular type of node is not among sons of a node.

3.4.10. _name

The meta-attribute `_name` is used to name a node for a later reference, see Section "???-??¿ below.

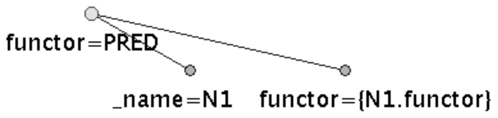
3.4.11. _sentence

The meta-attribute `_sentence` can be used to search in the linear surface form of the trees – in the sentences.

3.5. References

References are used in the queries to refer to values of attributes in the result trees, to values unknown at the time of creating the query. We use the word "references" as a technical term that simply means "a pointer". It should not be confused with linguistically motivated terms like "coreference".

First, a node in the query has to be named using the meta-attribute `_name`. Then, references to values of attributes of this node can be used at other nodes of the query. The following query searches for a Predicate with two sons with the same functor in the result tree, whatever the functor may be:



References can refer to the whole value (as shown above) or only to one character of the value. The required position is separated from the name of the attribute with another dot (“`.`”). It is also possible that references only form a substring of a defined value and appear several times in a definition of an attribute (but they cannot be a part of a regular expression).

3.6. Multi-Tree Queries

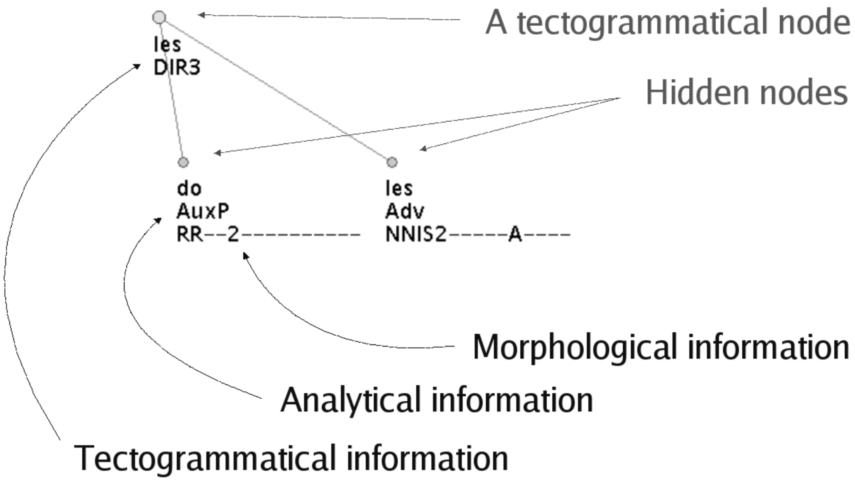
A multi-tree query consists of several trees combined either with a general `AND` or a general `OR`. In the case of `AND`, all the query trees are required to be found in the result tree at the same time (different nodes in the query cannot be matched with one node in the result), while in the case of `OR`, at least one of the query trees is required to be found in the result tree.

3.7. Hidden Nodes

Hidden nodes are nodes that are marked as hidden by setting the attribute `hide` to `true`. Their visibility in result trees can be switched on and off. Hidden nodes can serve as a connection to the lower layers of annotation with non-1:1 relations or generally to any external source of information.

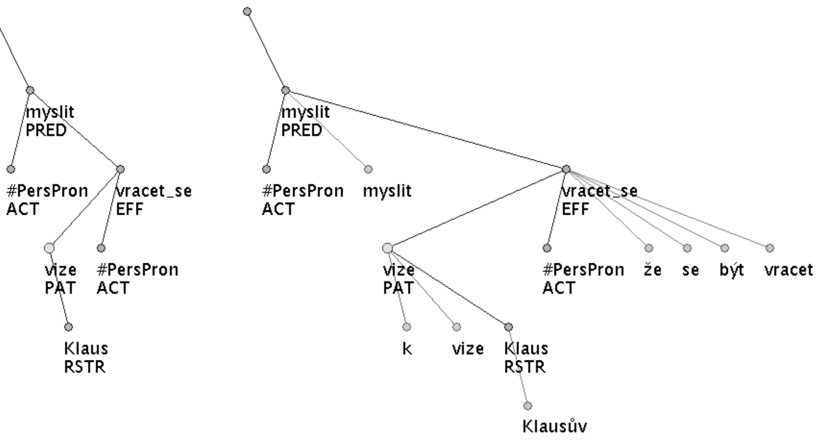
Netgraph uses the hidden nodes as a connection to the lower layers of annotation with non-1:1 relations. It presents all the available information in one tree (Mírovský 2006). The tectogrammatical nodes contain only the tectogrammatical information, while all the information from the lower layers is kept at the hidden nodes. Each tectogrammatical node has as many hidden sons as there are analytical nodes corresponding to the tectogrammatical node. (Hidden nodes were first introduced with the Prague Dependency Treebank 1.0; they were used in a slightly different way there.)

The principle of using hidden nodes for representing information from several layers of annotation in one tree is demonstrated in the following picture, which shows how the phrase “do lesa” (“to the forest”) is annotated on several layers of annotation and how it is represented using the hidden nodes:



One node on the tectogrammatical layer with `t_lemma=les` ("the forest") and `functor=DIR3` (representing the direction "to") has two hidden sons, representing a preposition `do` ("to") and an adverbial `les` ("the forest"). The information from the morphological layer is merged into the analytical layer.

The hidden nodes are usually not displayed – they are "hidden". The following picture demonstrates two possible ways of displaying a tectogrammatical tree in Netgraph. On the left side, there is a tectogrammatical tree with the hidden nodes hidden. In the same tree on the right side, the hidden nodes are displayed:



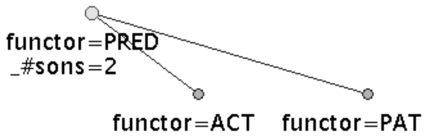
In Czech: *Myslím, že ke Klausově vizi se budeme vracet.*
In English: *I think that to Klaus's vision we will get back.*

4. Using the Query Language

Let us show a few representative examples of searching for some of the linguistic phenomena annotated in PDT 2.0. It was shown in much more detail in Mírovský 2008e that Netgraph Query Language fulfils the requirements stated in Section : The query language meets the general requirements on a query language for PDT 2.0, listed in Section ; it can be used for searching for all linguistic phenomena from PDT 2.0 listed in Section (Mírovský 2008c).

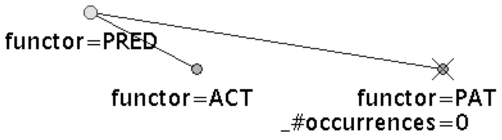
Valency

We present two queries for studying valency. The first query searches for Predicates governing an Actor, a Patient and nothing else (the Actor and the Patient are members of the valency frame, no other member is present):



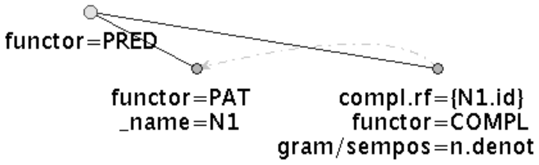
The meta-attribute `_#sons` makes sure that there are no other sons of the Predicate in the result trees.

The second query searches for Predicates governing an Actor and not governing a Patient. Since Patient has to be the second inner participant of any valency frame that has at least two inner participants (t-manual, page 102), the query searches for occurrences of Predicates with only one inner participant in its valency frame – the Actor:



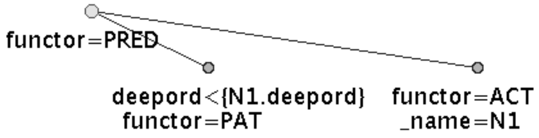
Predicative Complement (Dual Relation)

The example query uses references, the referential information is stored in the attribute `compl.rf`. The query searches for those cases of the predicative complement where the second relation goes to a Patient:



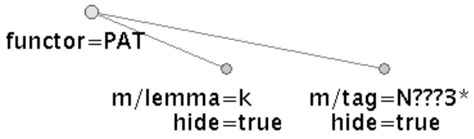
Topic-Focus Articulation

The communicative dynamism requires that the relative order of nodes in the tree from left to right can be expressed. The order of nodes is controlled by the attribute `deepord`, which contains a non-negative real (usually natural) number that sets the order of nodes from left to right. The following query demonstrates searching for a Predicate governing an Actor and a Patient, the Patient less dynamic (on the left side in the tree) than the Actor:



Accessing Lower Layers

Let us present an example query that accesses the lower layers of annotation from the tectogrammatical layer. It searches for Patients (on the tectogrammatical layer) that are expressed with a preposition "k" and a noun in the dative on the morphological layer:



The Patient has (at least) two hidden sons, the former with lemma "k", the latter with a morphological tag that states that the node is a noun in the dative.

5. Comparison to Other Treebank Query Systems

To show the power of FS Query Language, we use an indirect approach of comparing the language to four other query languages, languages of TGrep (Pito 1994), TGrep2 (Rohde 2005), TigerSearch (Brants et al. 2002), and fsq (Kepser 2003). We present a table showing to what extent the five tools (Netgraph and the other four tools) fulfil the requirements stated in Section 2.2. Please note that the table is biased in favour of Netgraph, because Netgraph has been designed to fulfil the requirements. The table does not contain query language features that do not belong to the requirements. The other tools have been designed for different corpora and may implement features that Netgraph does not. A detailed unbiased comparison of the expressive power of Netgraph Query Language and the query languages of TGrep, TGrep2 and TigerSearch is presented in Mírovský (2008a) and Mírovský (2008e).

In the table, the following marks are used:

+ ... the feature is supported

- ... the feature is not supported

* ... the feature is partially supported

N/A ... the feature is not applicable to the query language

Complex Evaluation of a Node	**TGrep**	**TGrep2**	**TigerSearch**	**fsq**	**Netgraph**
multiple attributes evaluation (the ability to set values of several attributes at one node)	–	–	+	+	+
alternative values (e.g. to define that functor of a node is either a disjunction or a conjunction)	+	+	+	+	+[I]
alternative nodes (alternative evaluation of the whole set of attributes of a node)	N/A	+	+	+	+
wild cards (regular expressions) in values of attributes	+	+	+	+	+
negation (e.g. to express “this node is not an Actor”)	+	+	+	+	+
relations lower than (<) , higher than (>)	–	–	–	–	+

Dependencies Between Nodes (Vertical Relations)	**TGrep**	**TGrep2**	**TigerSearch**	**fsq**	**Netgraph**
immediate, transitive dependency (existence, non-existence)	+	+	*[II]	+	+
vertical distance (from root, from one another)	–	–	*[III]	*[III]	+
number of sons (zero for leaves)	+	+	+	+	+

Horizontal Relations	**TGrep**	**TGrep2**	**TigerSearch**	**fsq**	**Netgraph**
precedence, immediate precedence (positive, negative)	+	+	*[IV]	+	+
horizontal distance	–	–	*[V]	*[V]	+
secondary edges, secondary dependencies, coreferences, long-range relations	*[VI]	*[VI]	+	+	+

Other Features	**TGrep**	**TGrep2**	**TigerSearch**	**fsq**	**Netgraph**
multi-tree queries (combined with the general OR relation)	–	+[VII]	+[VIII]	+[IX]	+[X]
skipping a node of a given type (for skipping simple types of coordination, apposition etc.)	–	+[XI]	+[XII]	+	+
skipping multiple nodes of a given type (e.g. for recognizing the rightmost path)	–[XIII]	–[XIII]	–[XIV]	+	+
references (for matching values of attributes unknown at the time of creating the query)	–	+	+	–	+
accessing several layers of annotation for studying relations between layers with non-1:1 relations	N/A	N/A	N/A	N/A	+
searching in the surface form of the sentence	+[XV]	+[XV]	+[XV]	+	+

Notes referred to from the table:

I: Only OR relation is supported.

II: Variables (nodes in the query) are existentially quantified. If the query specifies that A does not dominate B, then B must appear somewhere else in the tree.

III: Vertical distance can only be measured for nodes that are in the transitive dependency relation.

IV: Variables (nodes in the query) are existentially quantified. If the query specifies that A does not precede B, then B must appear somewhere else in the tree.

V: Horizontal distance can be measured for leaf nodes.

VI: Only one type of dependency can be set (although multiple times at a node, expressing relations to several nodes).

VII: Full Boolean expressions on patterns are supported.

VIII: Boolean expressions without negation on patterns are supported.
IX: At least first-order logic formula can be used.
X: Only the general OR or general AND are supported.
XI: Thanks to general Boolean expressions on patterns.
XII: Thanks to Boolean expressions on patterns.
XIII: But there are special predicates for the rightmost/leftmost descendant of a node.
XIV: But there are special predicates for the rightmost/leftmost leaf descendant of a node.
XV: Using predicates for precedence and immediate precedence on terminals.

6. The Tool

We have implemented Netgraph Query Language in a search tool called Netgraph. As a basis, we used Netgraph 1.0. We present a list of features that we consider important for a search tool for a treebank, especially for the Prague Dependency Treebank 2.0. We do not include general features that can be expected from any graphically oriented tool, like saving or printing capability. We rather focus on features that are connected with searching in treebanks. All these features have been implemented in Netgraph, so we present them this way. Some of the features have been implemented on a request from users:

- **client-server architecture**

With the client-server architecture, data can reside at one place in the Internet. Multiple users (clients) can access the server simultaneously (Mírovský, Ondruška 2002a, Mírovský et al. 2002b). The version control has been implemented in the tool, in order to keep the server and the client compatible.

- **authentication of users**

In order to protect the data, the authentication of users is available. Each user gets a login name and a password to access the server. Different users can have different permissions (maximum number of found trees, a permission to change the password, a permission to save the result trees to the local disc).

- **graphical creation of the query**

Especially for non-programmers, a graphical creation of the query, in our case a full implementation of Netgraph Query Language, is important.

- **browsing the result trees**

Obviously, users have to be able to browse the result of a query. A graphical representation of the trees is again an important feature. It includes displaying coreferential arrows and other references, as well as hidden nodes on request.

- **access to context trees**

Since the annotation on the tectogrammatical layer captures the linguistic meaning of the sentence in its context, the context of the sentence has to be accessible as well. The tool allows displaying context trees in both directions (forward and backward).

- **chained queries**

To refine a result of a query, another query can be set on top of the previous query. The second query searches only in the result of the previous query. This way, queries can be chained unlimitedly.

- **inverted search**

Some queries can be much simpler if the inversion of matching is available. We can simply define a query that represents a phenomenon that we do not want in the result trees and invert the search. Only trees that do not match the query become a part of the result.

- **search only for the first occurrence in each tree**

If we are only interested in the result trees and not in multiple occurrences of a query in the result trees, a possibility to search only for the first occurrence in the result trees can be useful. Although the tool allows the user to browse the result trees in such a way that multiple occurrences of a query in one tree are skipped, they are still searched for (thus the search slows down); searching only for the first occurrence makes the search faster. It is also very useful for chained queries if the subsequent query does not search in several same trees representing multiple occurrences of the previous query in one tree.

- **removing trees from the result**

Sometimes, it is difficult to refine a query further to obtain the exact set of result trees a user wishes. Therefore, a possibility to remove an unwanted result tree from the result is available (e.g. before the result is saved to the local disc).

- **right-left trees**

Some languages, like Arabic, require right-left ordering of nodes in the trees, as well as of the tokens in the sentence. The tool has to offer this feature.

- **multi-language support**

UTF-8 has become a standard in coding characters of natural languages. Thanks to this universal coding, all major languages are supported in Netgraph, even at the same time (in one corpus).

- **basic statistics**

The tool has to provide at least the most basic statistics about the result. It provides the following numbers: number of searched trees, number of found (result) trees, number of found occurrences in the found trees, and also number of the actually displayed tree/occurrence.

- **external command**

For further processing of the found tree, an external command can be run from the tool. Several variables for identifying the file, the tree and the position in the tree are substituted before the external command is launched.

- **speed/portability**

For the server, speed is the most important factor. Therefore, C programming language (Herout 2002) has been chosen for the implementation.

On the other hand, the most important factor for choosing the programming language for the client is portability. Java 2 (Eckel 2006) belongs to the best portable programming languages and it has also a very good support for various natural languages; it uses its own fonts and supports UTF-8 very naturally. Therefore, Java 2 has been chosen as a programming language for the client.

6.1. Changes since Version 1.0

The actual version of Netgraph is 1.93. We call the original version of Netgraph, programmed by Roman Ondruška, Netgraph 1.0. Here, we describe the main changes that have been done to the tool since this 1.0 version.

Let us start with several numbers representing code lines. The Netgraph client 1.0 had 1 526 lines of code. The Netgraph client 1.93 consists of more than 21 thousand lines. The Netgraph server 1.0 had 3 973 lines of code. The Netgraph server 1.93 has more than 11 thousand lines.

The following lists contain the most important changes that have been done since the version 1.0. The first list describes extensions to the query language, the second list describes changes in the tool.

6.1.1. Main Extensions to the Query Language

- Meta-attributes have been introduced to the system.
- References to values of attributes of (other) nodes can be set in the query.
- Regular expressions in values of attributes can be used.
- Other relations than equation can be used for setting values of attributes.
- Arithmetic operations in numerical values of attributes can be used.
- Multi-tree queries are supported.
- Support for hidden nodes has been added.

6.1.2. Main Extensions to the Tool

- The tool now supports the tectogrammatical trees (with hidden nodes and coreferences), both in searching and displaying; a configuration file defining how to display individual references is available.
- Authentication of users has been implemented.
- Queries can be chained.
- The matching of a query can be inverted.
- History of queries is created; queries or the whole history can be saved to the local disc; a list of selected files for searching can also be saved.
- Result trees can be printed or saved to the local disc.
- The tool now supports the UTF-8 encoding.
- Right-left trees are supported.
- Version control has been implemented.
- A query is created in a fully graphical way.
- Basic statistics about the search are provided.

- Context trees can be displayed.
- Individual trees can be removed from the result.
- An external command with variables substitution can be launched from the tool.

7. Real World

7.1. Netgraph Query Language and PDT 2.0

After we have presented Netgraph Query Language and shown what can be searched for with the language, it might be interesting to know to what extent the features have been put to use by the users and what the users really do search for. There are about 40 registered users and an anonymous access to the server for PDT 2.0 is also available.

Since October 2002, the Netgraph server stores all queries to a log file. By then, only the analytical trees were searched through in Netgraph. Since February 2005, also the tectogrammatical trees (though not publicly released yet) have been made available in Netgraph for the internal usage of our institute, and later (after PDT 2.0 publication) the tectogrammatical trees were made available for the registered public users, too.

From these two servers (the analytical and the tectogrammatical trees), all queries entered by users have been stored in log files. However, we have not had access to queries that had been processed on local installations of the Netgraph server, e.g. on notebooks, which are quite numerous. All the following numbers come only from the two public servers mentioned above (from the dates stated above up to March 24, 2008). For obvious reasons, before any statistics were counted, we excluded all queries that we had entered.

Number of:	Total	Analytical Trees	Tectogrammatical Trees
all queries	16 870	10 299	6 571
one-node queries	10 146	7 180	2 966
structured queries (more than one node)	6 724	3 119	3 605
queries without a meta-attribute	15 575	9 989	5 586
queries with a meta-attribute	1 295	310	985
`_transitive`	174	81	93
`_optional`	172	18	154
`_#sons`	91	22	69
`_#hsons`	36	–	36
`_depth`	51	11	40
`_#descendants`	103	24	79
`_#lbrothers`	35	25	10
`_#rbrothers`	11	0	11
`_#occurrences`	197	12	185
`_name`	397	116	281
`_sentence`	28	1	27
queries with a reference	363	110	253
queries with a hidden node	1 194	–	1 194
queries with an alternative value	884	314	570
queries with an alternative node	94	19	75

The table shows numbers of queries using various features of the query language, both on the analytical layer and on the tectogrammatical layer. The total usage is also counted.

Some values in the table should be equal but they are not. The number of queries that use the meta-attribute `_name` should be equal to the number of queries that use a reference. The discrepancy is caused by errors in some queries (e.g. queries that contain a named node but the name is never used).

A representative selection of queries put in by the users can be found in Mírovský (2008e).

7.2. Other Usages of Netgraph

The query tool Netgraph and its query language are general enough to be used with other treebanks than PDT 2.0. Netgraph can be used both for dependency trees and for constituent structure trees, provided the treebank is transformed to FS File Format (Mírovský 2008e), and also other kinds of usage are possible.

7.2.1. Czech Academic Corpus 1.0 and 2.0

During the work on the re-annotation of the Czech academic corpus (Králík and Hladká 2006), Netgraph was used for searching for errors in the process of re-annotation of the data from the original annotation scheme to a PDT-like annotation scheme. The first version of the “new” Czech academic corpus contained only the morphological annotation (Vidová-Hladká et al. 2007). During its preparation, the data was searched for errors on the morphological layer. Since there is no structure in the morphological annotation (but Netgraph only works with trees), flat morphological “trees” were used, in which a technical root had all the words of the sentence as its sons.

During the preparation of the second version of the Czech Academic Corpus (version 2.0), which is going to be released in LDC in the Fall of 2008, the morphologically annotated files were first automatically parsed on the analytical layer (Ribarov et al. 2006). Netgraph was then used for searching for errors on the analytical layer. The annotation was almost identical to the analytical layer of PDT 2.0, therefore a similar set of checks as for PDT 2.0 (Štěpánek 2006) was used.

7.2.2. Latin IT Treebank

Index Thomisticus (IT) Treebank is an ongoing project, which is a part of the Lessico Tomistico Biculturale (LTB) project by Father Roberto Busa.[2] IT-Treebank wants to make IT a Treebank.

The annotation on the analytical layer is performed on the basis of the annotation guidelines for the Prague Dependency Treebank and according to guidelines specifically written for Latin, shared and developed with the Latin Dependency Treebank of the Perseus Project in Boston. Presently, IT-Treebank is composed of 32 880 tokens, for a total of 1 479 syntactically parsed sentences from the Scriptum super Sententiis Magistri Petri Lombardi.

During the development of the Latin treebank, Netgraph is used for browsing the data and searching in the data.

7.2.3. Arabic Trees

In the year 2003, Netgraph was installed in LDC (Linguistic Data Consortium) in Philadelphia, University of Pennsylvania[3], to be used with their Arabic treebank. In cooperation with LDC, the Prague Arabic Dependency Treebank (Smrž et al. 2005) was developed at ÚFAL (Institute of Formal and Applied Linguistics) at Charles University in Prague[4]. Netgraph was used during the annotation work for studying the treebanks. Right-left ordering of nodes in trees was implemented for purposes of the Arabic treebanks.

7.2.4. Chinese Treebank

Netgraph has also been used for a work on a Chinese treebank at ÚFAL. Since Java supports Chinese language and Netgraph works with files encoded in UTF-8, no adaptation of the tool

[2]`http://gircse.marginalia.it/ passarotti/`. IT is considered as the pathfinder of Computer Sciences applications in the Humanities; it retains the opera omnia by Thomas Aquinas (118 texts), plus works by other 61 authors related to Thomas (61 texts). It is a corpus of around 11 millions of tokens (150.000 types; 20.000 lemmas).

[3]LDC – `http://www.ldc.upenn.edu/`

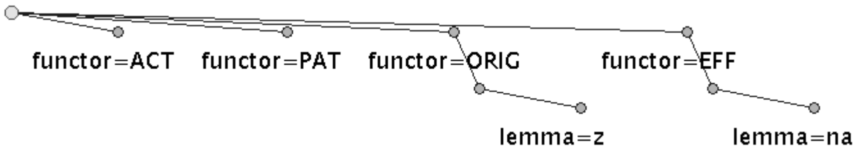
[4]ÚFAL – `http://ufal.mff.cuni.cz`

was necessary. This is an example of the use of Netgraph with constituent-structure trees.

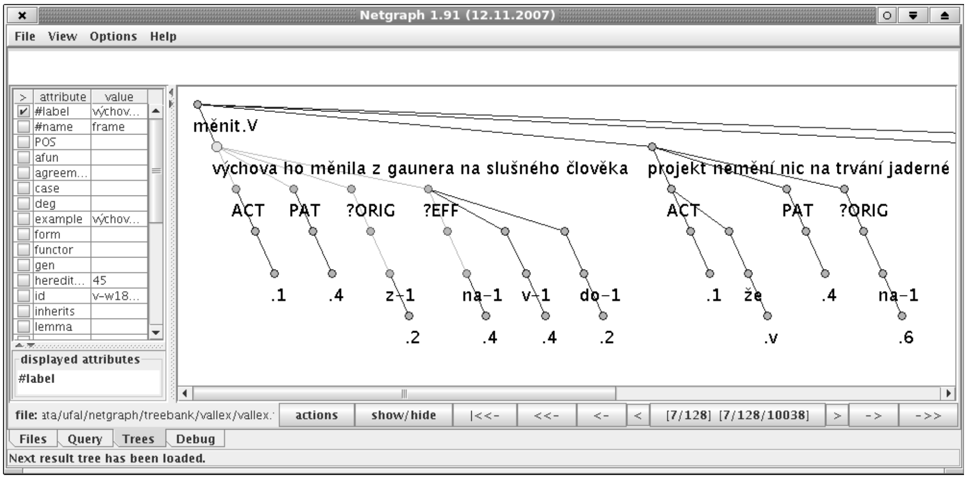
7.2.5. Vallex

Vallex is a valency lexicon of Czech (Žabokrtský and Lopatková 2007). A recent usage of Netgraph for sophisticated searching in this "treebank" belongs to interesting applications of the tool. Thanks to Petr Pajas and his tool TrEd (Pajas 2007), Vallex has been transformed to FS File Format and can be searched through with Netgraph.

The following query searches for valency frames of the type "přešila panenku z kašpárka na čerta" ("she altered the puppet from the Punch to the devil"), i.e. valency frames consisting of an Actor, a Patient, an Origin and an Effect. The query also requires that on the surface, the Origin is expressed with the preposition "z" and the Effect is expressed with the preposition "na":



The following picture shows one of the results in Netgraph:



In Czech: *výchova ho měnila z gaunera na slušného člověka*
In English: *education was changing him from a scrounger to a decent man*

8. Conclusion

In the paper, we have studied the Prague Dependency Treebank 2.0 and created a list of linguistic phenomena annotated in the treebank that bring a requirement on a query language for searching in the treebank. We have assembled a list of requirements that any query language should satisfy in order to fit the Prague Dependency Treebank 2.0.

We have proposed Netgraph Query Language – a simple to use and graphically oriented language that meets the requirements.

The proposed query language is an extension to an existing query language – a query language of Netgraph 1.0. The following three features are the most important additions to the query language:

- *meta-attributes* – for setting complex types of relation between nodes and complex properties of the nodes
- *hidden nodes* – for accessing lower layers of annotation with non-1:1 relation among nodes
- *references* – for setting relations between values of attributes of nodes that are unknown at the time of creating the query

The proposed query language meets the requirements on a query language for the Prague Dependency Treebank 2.0 (Mírovský 2008e).

We have compared the proposed query language to some other query languages.

We have also studied to what extent the features of the query language have been put to use by real users

The proposed query language has been implemented in Netgraph, which is also an extension to the existing search tool – Netgraph 1.0. Thus, a comfortable, simple to use and fully graphically oriented client-server system for searching in the Prague Dependency Treebank 2.0 has been created.

Acknowledgement

The research and work presented in the paper were supported by the Grant Agency of the Academy of Sciences of the Czech Republic, project IS-REST (No. 1ET101120413).

9. References

Brants S. et al. (2002): The TIGER Treebank. *In: Proceedings of TLT 2002, Sozopol, Bulgaria, 2002.*

Eckel B. (2006): Thinking in Java (4th edition). *Prentice Hall PTR, 2006.*

Hana J., Zeman D., Hajič J., Hanová H., Hladká B., Jeřábek E. (2005): Manual for Morphological Annotation, Revision for PDT 2.0. *ÚFAL Technical Report TR-2005-27, Charles University in Prague, 2005.*

Hajič J., Vidová-Hladká B., Panevová J., Hajičová E., Sgall P., Pajas P. (2001): Prague Dependency Treebank 1.0 (Final Production Label). *CD-ROM LDC2001T10, LDC, Philadelphia, 2001.*

Hajič J. et al. (1997): A Manual for Analytic Layer Tagging of the Prague Dependency Treebank. *ÚFAL Technical Report TR-1997-03, Charles University in Prague, 1997.*

Hajič J. et al. (2006): Prague Dependency Treebank 2.0. *CD-ROM LDC2006T01, LDC, Philadelphia, 2006.*

Hajičová E. (2007): Information Structure from the Point of View of the Relation of Function and Form. *In: The Prague Bulletin of Mathematical Linguistics 88, 2007, pp. 53-71.*

Hajičová E. (1998): Prague Dependency Treebank: From analytic to tectogrammatical annotations. *In: Proceedings of 2nd TST, Brno, Springer-Verlag Berlin Heidelberg New York, 1998, pp. 45-50.*

Hajičová E, Panevová J. (1984): Valency (case) frames. *In P. Sgall (ed.): Contributions to Functional Syntax, Semantics and Language Comprehension, Prague, Academia, 1984, pp. 147-188.*

Hajičová E., Partee B., Sgall P. (1998): Topic-Focus Articulation, Tripartite Structures and Semantic Content. *Dordrecht, Amsterdam, Kluwer Academic Publishers, 1998.*

Havelka J. (2007): Beyond Projectivity: Multilingual Evaluation of Constraints and Measures on Non-Projective Structures. *In: Proceedings of ACL 2007, Prague, pp. 608-615.*

Hazel P. (2007): PCRE (Perl Compatible Regular Expressions) Manual Page. *Available from http://www.pcre.org/*

Herout P. (2002): Učebnice jazyka C. *Kopp 2002.*

Kepser S. (2003): Finite Structure Query – A Tool for Querying Syntactically Annotated Corpora. *In Proceedings of EACL 2003, pp. 179-186.*

Králík J., Hladká B. (2006): Proměna Českého akademického korpusu (The transformation of the Czech Academic Corpus). *In: Slovo a slovesnost 3/2006, pp. 179-194.*

Kučová L., Kolářová-Řezníčková V., Žabokrtský Z., Pajas P., Čulo O. (2003): Anotování koreference v Pražském závislostním korpusu. *ÚFAL Technical Report TR-2003-19, Charles University in Prague, 2003.*

Mikulová M., Bémová A., Hajič J., Hajičová E., Havelka J., Kolářová V., Kučová L., Lopatková M., Pajas P., Panevová J., Razímová M., Sgall P., Štěpánek J., Urešová Z., Veselá K., Žabokrtský Z. (2006): Annotation on the tectogrammatical level in the Prague Dependency Treebank. Annotation manual. *Tech. Report 30, ÚFAL MFF UK, 2006.*

Mírovský J. (2008e): Netgraph – a Tool for Searching in the Prague Dependency Treebank 2.0. *PhD Thesis, Charles University in Prague, 2008.*

Mírovský J. (2008d): PDT 2.0 Requirements on a Query Language. *In: Proceedings of ACL 2008, Columbus, Ohio, USA, 16th - 18th June 2008, pp. 37-45.*

Mírovský J. (2008c): Does Netgraph Fit Prague Dependency Treebank? *In: Proceedings of the Sixth International Language Resources and Evaluation (LREC 2008), Marrakech, Marocco, 28th - 30th May 2008.*

Mírovský J. (2008a): Towards a Simple and Full-Featured Treebank Query Language. *In: Proceedings of ICGL 2008, Hong Kong, 9th - 11th January 2008, pp. 171-178.*

Mírovský J. (2006): Netgraph: a Tool for Searching in Prague Dependency Treebank 2.0. *In Proceedings of TLT 2006, Prague, pp. 211-222.*

Mírovský J., Ondruška R., Průša D. (2002b): Searching through Prague Dependency Treebank

- Conception and Architecture. *In Proceedings of The First Workshop on Treebanks and Linguistic Theories, Sozopol, 2002, pp. 114—122.*

Mírovský J., Ondruška R. (2002a): NetGraph System: Searching through the Prague Dependency Treebank. *In: The Prague Bulletin of Mathematical Linguistics 77, 2002, pp. 101-104.*

Ondruška R. (1998): Tools for Searching in Syntactically Annotated Corpora. *Master Thesis, Charles University in Prague, 1998.*

Pajas P. (2007): TrEd User's Manual. Available from http://ufal.mff.cuni.cz/ pajas/tred/

Pito R. (1994): TGrep Manual Page. *Available from http://www.ldc.upenn.edu/ldc/online/treebank/*

Ribarov et al. (2006): When a Statistically Oriented Parser Was More Efficient Than a Linguist: A Case of Treebank Conversion. *In: The Prague Bulletin of Mathematical Linguistics 86, 2006, pp. 21-38.*

Rohde D. (2005): TGrep2 User Manual. *Available from http://www-cgi.cs.cmu.edu/ dr/TGrep2/ tgrep2.pdf*

Sgall P. (2001): Underlying Structures in Annotating Czech National Corpus. *In: Current Issues in Formal Slavic Linguistics, edited by G. Zybatow, U. Junghanns, G. Mehlhorn and L. Szucsich, Peter Lang, Frankfurt a/Main, 2001, pp. 499-505.*

Smrž O., Pajas P., Žabokrtský Z., Hajič J., Mírovský J., Němec P. (2005): Learning to Use the Prague Arabic Dependency Treebank. *In: Elabbas Benmamoun. Proceedings of Annual Symposium on Arabic Linguistics (ALS-19). Urbana, IL, USA, Apr. 1-3: John Benjamins, 2005.*

Štěpánek J. (2006): Post-Annotation Checking of Prague Dependency Treebank 2.0 Data. *In: The Prague Bulletin of Mathematical Linguistics 85, 2006, pp. 23-33.*

Vidová-Hladká B., Hajič J., Hana J., Hlaváčová J., Mírovský J., Votrubec J. (2007): Czech Academic Corpus 1.0 Guide. *Karolinum - Charles University Press, 2007, ISBN: 978-80-246-1315-4*

Žabokrtský Z., Lopatková M. (2007): Valency Information in VALLEX 2.0. *In: The Prague Bulletin of Mathematical Linguistics 88, 2007, pp. 41-59.*