Does Netgraph Fit Prague Dependency Treebank?

Jiří Mírovský
Charles University in Prague
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics
Malostranské nám. 25, 118 00 Prague 1, Czech Republic
mirovsky@ufal.mff.cuni.cz

Abstract

On many examples we present a query language of Netgraph – a fully graphical tool for searching in the Prague Dependency Treebank 2.0. To demonstrate that the query language fits the treebank well, we study an annotation manual for the most complex layer of the treebank – the tectogrammatical layer – and show that linguistic phenomena annotated on the layer can be searched for using the query language.

1 Introduction

Netgraph is a client-server tool for searching in treebanks, designed to be as simple to use as possible, although with sufficient query power. It can be used both for dependency and constituent-structure types of treebanks, as long as the treebank is transformed to a suitable format.

In this paper, we show how well is Netgraph adapted for the Prague Dependency Treebank 2.0 (PDT 2.0, Hajič et al., 2006). We study the annotation manual for the tectogrammatical layer of PDT 2.0 (annotation manual, Mikulová et al., 2006), which is the most advanced and complex layer in the treebank, and show that linguistic phenomena described in the manual can be searched for with Netgraph.

In *section 1* (after this introduction) we give a short introduction to the Prague Dependency Treebank 2.0, to make the subsequent examples understandable.

In *section 2* we present the basics of Netgraph query language along with the idea of meta-attributes.

In *section 3* we create queries for linguistic phenomena from the annotation manual and present additional features of Netgraph query language, as they are needed.

We conclude in *section 4.*

1.1 Prague Dependency Treebank 2.0

The Prague Dependency Treebank 2.0 is a manually annotated corpus of Czech. The texts are annotated on three layers – morphological, analytical and tectogrammatical.

On the morphological layer, each token of every sentence is annotated with a lemma (attribute `m/lemma`), keeping the base form of the token, and a tag (attribute `m/tag`), keeping its morphological information.

The analytical layer corresponds to the surface syntax of the sentence; the annotation is a rooted dependency tree with labeled nodes. Attribute `afun` describes type of dependency between a dependent node and its governor. The left-right order of the nodes corresponds exactly to the surface order of tokens in the sentence (attribute `ord`).

The tectogrammatical layer, which we focus on in this paper, captures the linguistic meaning of the sentence in its context. Again, the annotation is a dependency tree with labeled nodes (Hajičová, 1998). The correspondence of the nodes to the lower layers is often not 1:1 (Mírovský (2006) about Netgraph addressing this issue).

Attribute `functor` describes the dependency between a dependent node and its governor. A tectogrammatical lemma (attribute `t_lemma`) is assigned to every node. 16 grammatemes (prefixed `gram`) keep additional annotation (e.g. `gram/verbmod` for verbal modality).

Topic and focus (Hajičová et al., 1998) are marked (attribute `tfa`), together with so-called deep word order reflected by the order of nodes in the annotation (attribute `deepord`).

Textual and grammatical coreference relations between nodes of certain category types are captured. Each node has a corpus-wide unique identifier (attribute `id`). Attributes `coref_text.rf` and `coref_gram.rf` contain `id`s of coreferential nodes of the respective types.

2 Netgraph Query Language

The query in Netgraph is a tree that forms a subtree in the result trees. The result of a search consists of trees that match the query – the query is found as a subtree of the result tree. The query can also consist of several trees joined either by `AND` or `OR` relation. In that case, all the query trees at the same time (or at least one of the query trees, respectively) are required to match the result tree.

The query has both a textual form and a graphical form. We will use both forms in the paper, the textual form for simple examples, the graphical form for more complex ones.

The syntax of the query language is very simple. In the textual form, square brackets enclose a node, attributes (pairs `name=value`) are separated by a comma, quotation marks enclose a regular expression in a value. Parentheses enclose a subtree of a node, brothers are separated by a comma. In multiple-tree queries, each tree is on a new line and the first line contains only a single AND or OR. Alternative values of an attribute, as well as alternative nodes, are separated by a vertical bar. It almost completes the description of the syntax, only one thing – references – will be added in the next subsection.

The following example shows a simple query, consisting of one node with a condition set on two attributes. We search for all nodes that are verbal but not Predicates:

```
[functor!=PRED,gram/sempos=v]
```

More interesting queries usually consist of several nodes, forming a tree structure. The following example query searches for trees containing a Predicate that directly governs an Actor and an Effect:

```
[functor=PRED]([functor=ACT],[func-
tor=EFF])
```

Please note that there is no condition in the query on the order of the Actor and the Effect, nor on their left-right position to their father. It does not prevent other nodes to be directly governed by the Predicate either.

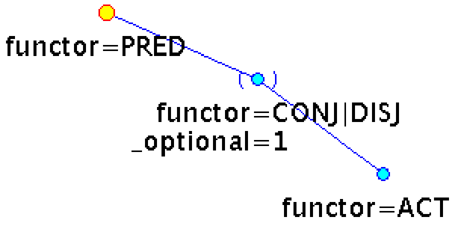
2.1 Meta-Attributes

Meta-attributes add more power to this simple query language. They allow to use real negation, restrict the position of the query in the result tree and the size of the result tree, or control the order of nodes. Meta-attributes are not present in the corpus but they pretend to be ordinary attributes and the user uses them the same way like normal attributes. Their names start with an underscore. We show only an example of their usage now and present more of them in the next section, whenever we need them. A detailed description of all meta-attributes was given in Mírovský (2008).

Meta-attribute _optional marks an optional node. The node then may but does not have to be in the result. It can be used for skipping coordination. If we are interested, for example, in Predicates governing Actors and want to get both cases (with coordination and without it) in one query, we can use this query:

```
[functor=PRED]([functor=CONJ|
DISJ,_optional=1]([functor=ACT])).
```

The coordination (Conjunction or Disjunction) becomes optional. If there is a node between the Predicate and its Actor in the result tree, it has to be the coordinating node. But the Actor may also be a direct son of the Predicate, omitting the optional coordination. Since we set meta-attribute _optional to 1, only one such optional node may appear in the result tree. The picture shows the graphical representation of the query:



It is also possible to set relations (other than dependency) between nodes in the result trees (such as order, agreement, coreference). All this can be done using meta-attribute _name (which names a node) and a system of references.

Curly brackets enclose a reference to a value of an attribute of another node (with a given name) in the result tree.

In the following example (knowing that attribute deepord controls the order of nodes in the tree from left to right), we search for an Actor that is on the right side from a Predicate and is dependent on it:

```
[functor=PRED,_name=N1]
([functor=ACT,deepord>{N1.deepord}]
)
```

We have named the Predicate node N1 and specified that deepord of the Actor node should be bigger than deepord of node N1.

3 Fitting the Annotation Manual

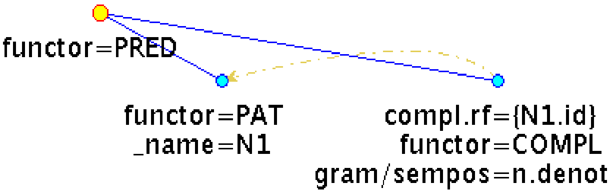
After we have presented the basics of the query language, we can proceed to showing how to search for linguistic phenomena described in the annotation manual. Since it is impossible to list everything from the manual here (the manual contains more than a thousand pages), we will focus on the interesting, important and complex parts.

3.1 Trivia

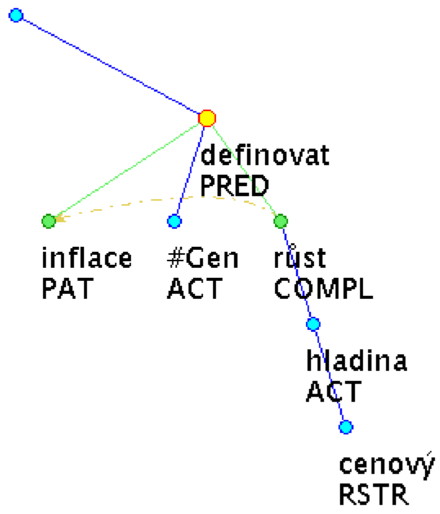
Most parts of the annotation manual are covered with instructions how to annotate linguistic phenomena in the means of one node. It is of course trivial to search for any combination of values of attributes of one node. Many other cases only require to create a simple tree-structure as a query. We shall rather concentrate on more complex cases, where a more complex query-tree (and usually usage of a meta-attribute) is required.

3.2 References

We use a technical term "references" to cover all phenomena represented in the annotation by a reference to an identifier of another node. It includes linguistic coreference (textual and grammatical), predicative complement and effective parentage. A dedicated attribute exists for each of these relations. The following example searches for predicative complements and uses attribute compl.rf, which points from the complement to the governing noun ("second dependency"). The query searches for those cases of predicative complement where the second dependency goes to a Patient:



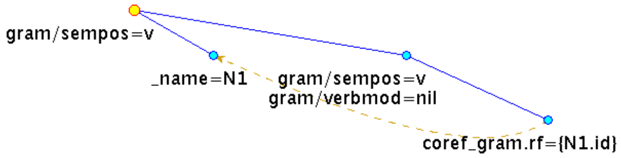
The following tree is a possible result for the query:



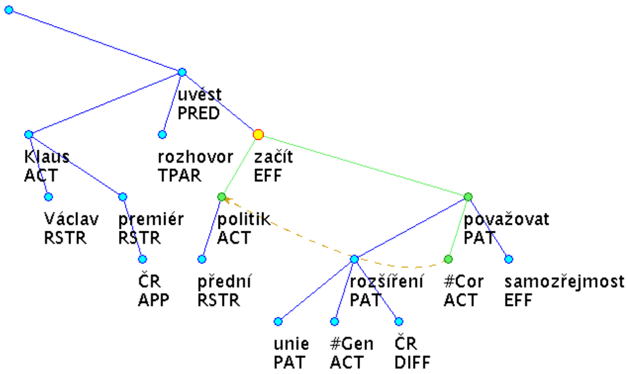
In Czech: *Inflace je definována jako růst cenové hladiny.*

In English: *Inflation is defined as an increase of the prices level.*

A similar technique can be used for all other types of references. The following example shows how to search for type-1 control constructions, a type of grammatical coreference where an infinitive (`gram/sempos=v`, `gram/verbmod=nil`) depends on a control verb (`gram/sempos=v`). This time, the referential attribute is `coref_gram.rf` . We do not set any other condition on the nodes:



And one of the result trees:

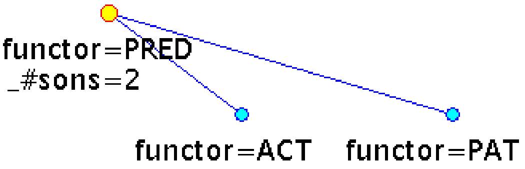


In Czech: *Přední politici začali rozšíření unie o ČR považovat za samozřejmost, uvedl během rozhovorů premiér ČR Václav Klaus.*

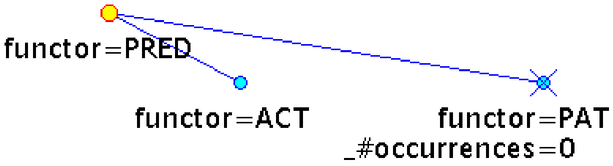
In English: *Prominent politicians started to take the extension of the union for granted, the prime minister of CR Václav Klaus pointed out during the discussions.*

3.3 Valency

A simple example of searching for a Predicate governing an Actor and an Effect was given in section 2. For real studies of valency, we need a way of restricting number and type of dependent nodes of a Predicate (or another node with valency). Meta-attribute `_#sons` controls the exact number of sons of a node in the result tree. In the following example, we search for a Predicate that governs an Actor and a Patient and nothing else:



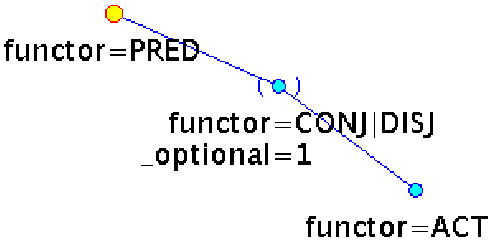
To control the type of sons of a particular node, there is meta-attribute `_#occurrences` available. It is used in the next example, where we require the Predicate to govern an Actor but not a Patient. It may have other sons, though:



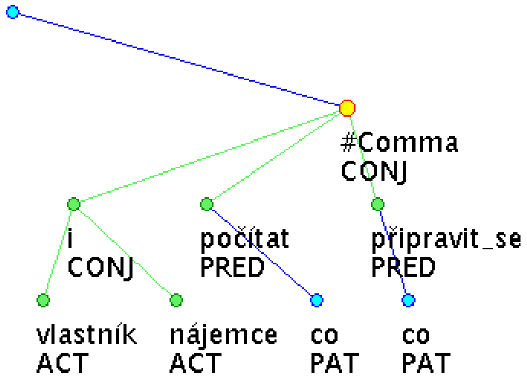
3.4 Coordination and Apposition

The query language should be able to skip a coordinating node. In general, there should be a possibility to skip any type of node.

Meta-attribute `_optional` can be used directly to skip a node or a chain of nodes of a certain property. Let us repeat the example given in section 2.1, searching for a Predicate governing an Actor with an optional coordinating node in between:



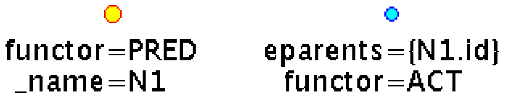
The query skips simple cases of coordination but cannot find the required relation between the nodes in a more complex structure. The following picture shows a tree where the structure of coordinations is more complex and skipping a node does not help. The two Predicates are coordinated with Conjunction, as well as the two Actors. 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 they should get ready for?*

Since the information about the linguistic dependency is annotated in the treebank (by the means of references), there is no problem in creating a general query skipping any possible combination of coordinations (the same applies to apposition):

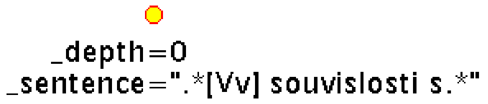


We do not presume any special relative position of the two nodes in the tree and therefore use a multi-tree query (the trees are combined with logical expression AND). Attribute `eparents` keeps identifiers of all effective linguistic fathers of a node. If we wanted to search only for the cases where the linguistic father(s) differ(s) from the technical father, we might instead use attribute `eparents_diff`, which keeps identifiers of all effective linguistic fathers of a node only if they differ from its technical father.

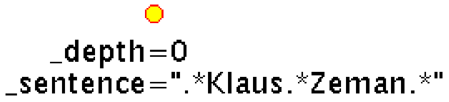
3.5 Idioms (Phrasemes) etc.

Some idioms/phrasemes and secondary prepositions are linguistic phenomena that can be easily recognized in the surface form of the sentence but may be difficult to find in the tectogrammatical tree. Meta-attribute `_sentence` can be used to search directly in the linear form of the sentences, regardless of the way a phenomenon is or even is not captured in the tectogrammatical tree.

Let us present two examples. The first query searches for a phrase "v souvislosti s" ("in relation to"), regardless of its position in the sentence. To avoid matching each node in the tree, meta-attribute `_depth` is added; it controls distance from the root in the result tree:

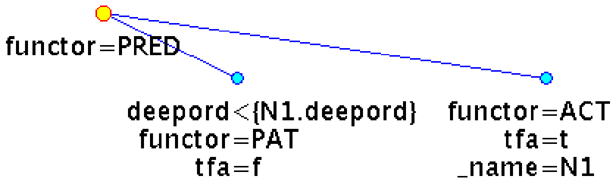


The second query searches for sentences containing words "Klaus" and "Zeman", in this order, anywhere in the sentence, even in forms like "Klause" or "Zemanovi":

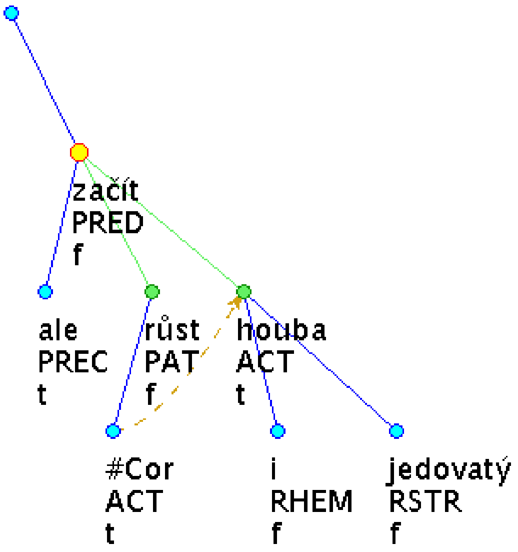


3.6 Topic-Focus Articulation

To study topic-focus articulation and communicative dynamism, it is essential to be able to control the order of nodes, in combination with setting values of attribute `tfa` (`f`=focus, `t`=topic, `c`=contrastive topic). As was shown in section 2, a system of references and attribute `deepord` allow setting the order of nodes. The following query demonstrates searching for a Predicate governing an Actor and a Patient, the Patient in focus and less dynamic (on the left side in the tree) than the Actor in topic:



And a possible result tree:



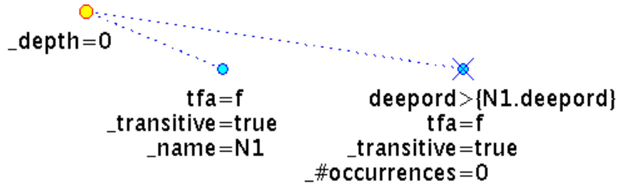
In Czech: *Začaly ale růst i houby jedovaté.*

In English: *But also poisonous mushrooms started to grow.*

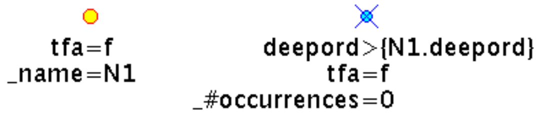
3.6.1 Focus Proper

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 is at a different position in the surface structure. The node representing this expression will be placed rightmost in the tectogrammatical tree.

The following query searches for focus proper; the first version of the query uses two transitive edges (meta-attribute `_transitive`) to place the two sons of the root anywhere in the result tree:

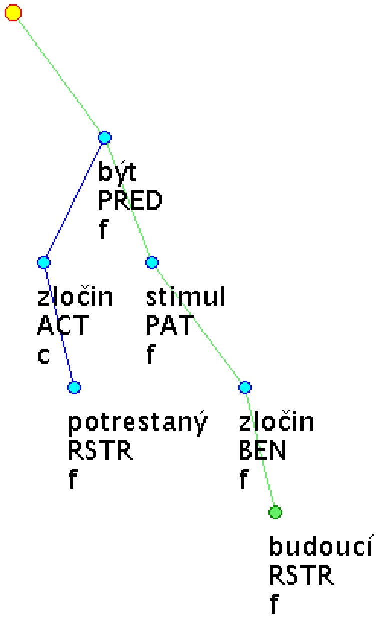


The same query can be expressed with a multiple-tree query with logical expression AND:



In both cases, we search for a node in focus named N1, which is the focus proper, by defining that there cannot be a node in focus on the right side from N1 anywhere in the tree.

The following tree is a possible result for both the queries; yet, the highlighted nodes show that the first version was used:



In Czech: *Nepotrestaný zločin je stimulem pro zločiny budoucí.*

In English: *An unpunished crime is a stimulant for future crimes.*

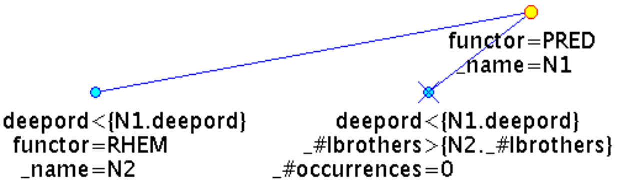
3.6.2 Rhematizers

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

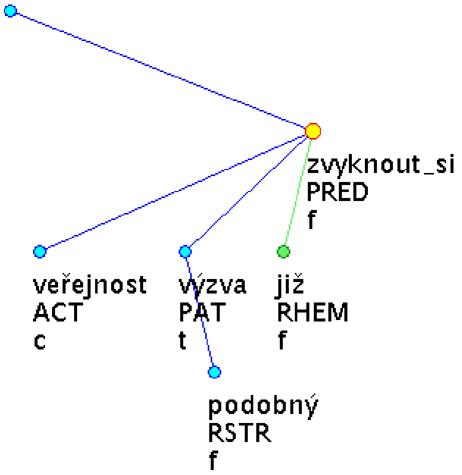
There are two cases of rhematizers that we need to distinguish:

- a rhematizer (i.e. the node representing the rhematizer) is placed as the closest left brother (in the underlying word order) of the first node of the expression that is in its scope.
- if the scope of a rhematizer includes the governing predicate, the rhematizer is placed as the closest left son of the node representing the governing predicate.

We present two queries to show how to study rhematizers. The first query searches for rhematizers with the Predicate in its scope, i.e. for a rhematizer that is the rightmost left son of the Predicate:



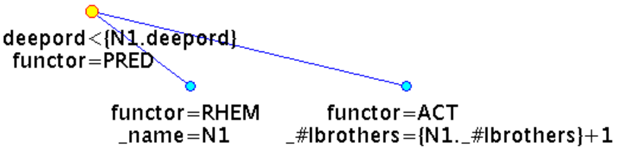
The query defines that there is not a node that is left from the Predicate but right from the rhematizer. Since we cannot set two different conditions with two different relations on one attribute, we have to use meta-attribute `_#lbrothers` to define that the undesired node is on the right side from the rhematizer. The following tree is a possible result for the query:



In Czech: *Veřejnost si na podobné výzvy již zvykla.*

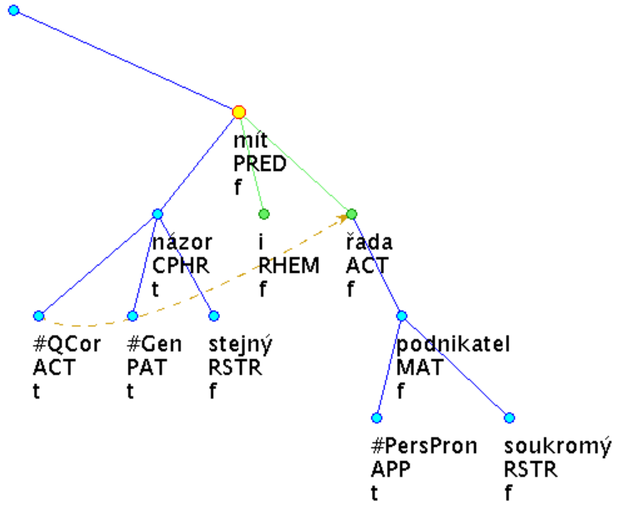
In English: *The public has already got accustomed to such calls.*

The second query searches for the cases where the Predicate is not in the scope of the rhematizer. The query also states that the first rhematized node is an Actor:



This time, the Predicate is on the left side from the rhematizer and the Actor is an immediate right brother of the rhematizer.

The following tree is a possible result for the query:

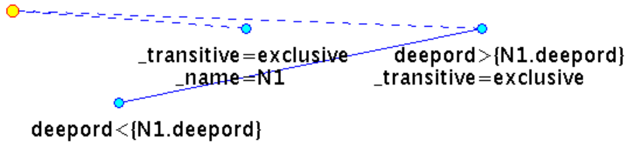


In Czech: *Stejný názor má i řada našich soukromých podnikatelů.*

In English: *Also a number of our private investors have the same opinion.*

3.6.3 (Non-)Projectivity

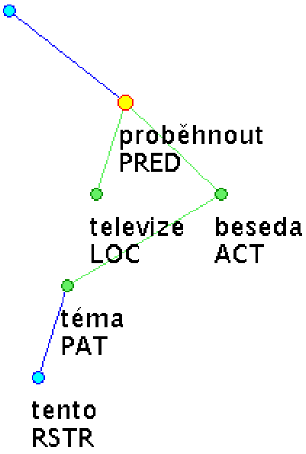
Non-projective constructions (studied for many languages in Havelka (2007)) are not frequent on the tectogrammatical layer of PDT 2.0, yet they are allowed and present. Projectivity of a tree is defined very simply: between a father and its son (in left-right order) there can only be direct or indirect sons of the father. To capture all types of non-projective edges in one query, we have to combine four query-trees with OR-logical expression that represent four non-projective configurations: the node causing the non-projectivity is or is not on the path from the non-projective edge to the root of the tree, and the father-node of the non-projective edge is on the left side from its son or on the right side. The following query is one of those trees (the other three are similar). The query uses meta-attribute `_transitive`, which defines a transitive edge. Its special value `exlusive` ensures that no nodes are shared by two exclusively transitive edges:



If we used attribute `ord` instead of `deepord`, the same query might be used for searching for non-projective constructions on the analytical layer, where they are much more often. The following tree is a possible result on the tectogrammatical layer. It represents the sentence:

In Czech: *Na toto téma by měla v televizi proběhnout beseda.*

In English: *A discussion on this theme should take place on the TV.*



4 Conclusion

We have shown that the most complex linguistic phenomena annotated in PDT 2.0 can be searched for with Netgraph query language with quite simple queries. Of course, it is even easier to search for the other phenomena, which are less complex and more numerous. We can conclude then that Netgraph fits PDT 2.0 well.

5 Acknowledgment

This research was supported by the Grant Agency of the Academy of Sciences of the Czech Republic, project IS-REST (No. 1ET101120413).

6 References

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

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

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

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

Mikulová M. et al. (2006). Annotation on the tectogrammatical level in the Prague Dependency Treebank. Annotation manual. Tech. Report 30, ÚFAL MFF UK, 2006.

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