SEARCHING IN THE PRAGUE DEPENDENCY TREEBANK

Jiří Mírovský

ÚSTAV FORMÁLNÍ
A APLIKOVANÉ LINGVISTIKY

STUDIES IN COMPUTATIONAL AND THEORETICAL LINGUISTICS

Jiří Mírovský

SEARCHING IN THE PRAGUE DEPENDENCY TREEBANK

Published by Institute of Formal and Applied Linguistics
as the 5th publication in the series
Studies in Computational and Theoretical Linguistics.

This book has been printed with the support of the project MSM0021620838 of The Ministry of Education of the Czech Republic.

ISBN 978-80-904175-6-4

Acknowledgement

First of all, my thanks belong to Jan Hajič, the chief of our department, for the support during my work on the book, and for his openness to my own suggestions and wishes about the direction of the research.

My great gratitude goes to Jarmila Panevová for her help with searching for interesting queries and for her admirable willingness to learn to use the tool and understand new features of the language.

I very much thank Petr Pajas for his help with the transformation of the data from PML format to FS format and for his kind readiness to answer my frequent requests for changes very quickly.

I would also like to thank Marie Mikulová for explaining details of annotation on the tectogrammatical layer to me, in which she is a great expert.

I am also grateful to Roman Ondruška for creating a good basis of the tool and for bringing up the first idea of the core of the query language in his Master Thesis.

And I thank Kiril Ribarov, my colleague, who finally persuaded me to start my PhD studies after a few years of my simple employment at the department.

I very much thank all users who have been using Netgraph for the inspiration from their feedback. Jiří Havelka helped me with non-projective constructions, Lucie Mladová turned my attention towards rhematizers, and there were many many others. I also thank Otakar Smrž for the Arabic screenshot of Netgraph, Marco Passarotti for the Latin screenshot, and Pavel Straňák for the Chinese screenshot.

I also want to thank my other colleagues at the Institute of Formal and Applied Linguistics for creating a very friendly atmosphere, and Eva Hajičová, its former director, for establishing such a pleasant team of researchers.

And I cannot but express my greatest gratitude to the nearest people of mine, my Mum and Dad, my late Grandma, and Dana, my girlfriend, for their constant support, for their firm belief in my abilities, and for their patience with the most undeserving person.

Jiří Mírovský

The research and work presented in the book were supported by the Ministry of Education, project Center for Computational Linguistics (No. LN00A063), the Grant Agency of the Academy of Sciences of the Czech Republic, project IS-REST (No. 1ET101120413), and the Institute of Formal and Applied Linguistics at Charles University in Prague.

Brief Contents

Contents

1 Introduction

1.1 Motivation

Linguistically annotated treebanks play an essential part 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.

Three sides existed whose connection is solved in this book. First, it was the Prague Dependency Treebank 2.0 (Hajič et al. 2006), one of the most advanced manually annotated treebanks in the linguistic world. Second, there existed a very limited but extremely intuitive search tool – Netgraph 1.0. Third, there were users longing for such a simple and intuitive tool that would be powerful enough to search in the Prague Dependency Treebank.

Our aim is to propose and implement a query system for this treebank that would not require programming skills from its users. A system that could be used by linguists without a knowledge of any programming language. A system that would fit the Prague Dependency Treebank 2.0 – it means to be powerful enough to search for all linguistic phenomena annotated in the data.

In the book, we study the annotation of the Prague Dependency Treebank 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 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 show how all principal linguistic phenomena annotated in the treebank can be searched for with the query language.

The proposed query language has also been implemented – we present the search tool as well and talk about the data format for the tool. The tool is freely available and can be downloaded from the internet, as described in the Appendix.

1.2 Outline of the Book

In the rest of this introductory chapter, we present very shortly the Prague Dependency Treebank 2.0, only for those who are not at all familiar with the treebank.

In Chapter “2 - The Problem Analysis”, we first mention some related work and present several existing search tools for treebanks, including Netgraph 1.0 – a basis for our own work. Afterwards, in Section “2.3 - Linguistic Phenomena in PDT 2.0”, we study annotation manuals for the Prague Dependency Treebank 2.0 and present linguistic phenomena that require our attention in creating a query language. We focus mainly on the tectogrammatical layer – the most complex layer of the treebank. In the subsequent section (“2.4 - Linguistic Requirements”), we summarize a list of requirements on a query language for the Prague Dependency Treebank 2.0.

In Chapter “3 - The Query Language”, we propose a query language that meets all requirements gathered in the previous chapter. It is an extension to the existing query language of Netgraph 1.0.

Chapter “4 - The Data” is dedicated to the description of the data used in Netgraph. The chapter not only describes the format of the data, but also shows that the query language is not independent of the data – it has some requirements on the data and the data can also help with some pre-computed information. Hidden nodes are presented in Section 4.6 as a way of accessing lower layers of annotation with non-1:1 relation among nodes of the layers.

In Chapter “5 - Using the Query Language”, we show that Netgraph Query Language, described in Chapter 3, fulfils the requirements stated in Chapter 2. We show that it meets the general requirements on a query language for the Prague Dependency Treebank 2.0, listed in Section 2.4, and how it can be used for searching for all linguistic phenomena from the treebank, gathered from the annotation manuals.

Chapter “6 - Notes on the Query Language” discusses some features of the query language. A comparison to several other query languages is also offered here (Section 6.5). Section 6.7 gives an example of how feedback from users influenced the development of the query language.

Chapter “7 - The Tool” introduces Netgraph – the tool that implements the query language.

Chapter “8 - Real World” shows to what extent the features of the query language are put to use by the users and what the users really do search for, by studying log files of the Netgraph server. Representative examples of real queries set by users are presented.

We conclude in Chapter “9 - Conclusion” by summarizing what has been done and proposing some future work on the query language and the tool.

Much additional information can be found in Appendixes. “Appendix A: Publications about Netgraph” enlists publications about Netgraph written or co-written by the author of this book. “Appendix B: FS File Format Description” describes formally the data format used in Netgraph. “Appendix C: FS Query Format Description” describes formally the syntax of the query language implemented in Netgraph. “Appendix D: List of Attributes in PDT 2.0” gives a list of all attributes of the Prague Dependency Treebank 2.0 used in Netgraph. “Appendix E: Other Usages of Netgraph” shows usages of Netgraph for some other treebanks. “Appendix F:

Installation and Usage of Netgraph – A Quick How-To" describes shortly how to install and use the Netgraph client.

1.3 The Prague Dependency Treebank 2.0

We very briefly describe the Prague Dependency Treebank 2.0, its properties and major attributes of the annotation. We focus on features that are important for basic understanding of the annotation of the treebank.

A more detailed description of all attributes of the Prague Dependency Treebank 2.0 is available in "Appendix D: List of Attributes in PDT 2.0".

The Prague Dependency Treebank 2.0 (PDT 2.0, see Hajič et al. 2006, Hajič 2004) is a manually annotated corpus of Czech. It is a sequel to the Prague Dependency Treebank 1.0 (PDT 1.0, see Hajič et al. 2001a, Hajič et al. 2001b).

The texts in PDT 2.0 are annotated on three layers - the morphological layer, the analytical layer and the tectogrammatical layer. The corpus size is almost 2 million tokens (115 thousand sentences), although "only" 0.8 million tokens (49 thousand sentences) are annotated on all three layers. By "tokens" we mean word forms, including numbers and punctuation marks.

1.3.1 The Morphological Layer

On the morphological layer (Hana et al. 2005), 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. Sentence boundaries are annotated here, too. Attribute `m/form` keeps the form of the token from the sentence, with some possible corrections (like misprints in the source text).

1.3.2 The Analytical Layer

The analytical layer roughly corresponds to the surface syntax of the sentence; the annotation is a single-rooted dependency tree with labelled nodes (Hajič et al. 1997, Hajič 1998). The nodes on the analytical layer (except for technical roots of the trees) correspond 1:1 to the tokens of the sentences (more precisely about this in Section 2.3). The order of the nodes from left to right corresponds exactly to the surface order of tokens in the sentence. Non-projective constructions (that are quite frequent in Czech (Hajičová et al. 2004) and also in some other languages (Havelka 2007)) are allowed. Analytical functions are kept at nodes (attribute a/afun), but in fact they are names of the dependency relations between a dependent (son) node and its governor (father) node.

1.3.3 The Tectogrammatical Layer

The tectogrammatical layer captures the linguistic meaning of the sentence in its context. Again, the annotation is a rooted dependency tree with labelled nodes. The correspondence of the nodes to the lower layers is more complex here. It is often not 1:1, it can be both 1:N and N:1 (actually, even N:0, or M:N). It was shown in Mírovský (2006) how Netgraph deals with this issue. It is also discussed here in Section 4.6.

Many nodes found on the analytical layer disappear on the tectogrammatical layer (such as function words, e.g. prepositions, subordinating conjunctions, etc.). The information carried by these nodes is stored in attributes of the remaining (auto-semantic) nodes and can be reconstructed. On the other hand, some nodes representing for example obligatory positions of verb frames, deleted on the surface, and some other deletions, are regenerated on this layer (for a full list of deletions, see Mikulová et al. 2006).

The tectogrammatical layer goes beyond the surface structure and corresponds to the semantic structure of the sentence, replacing notions such as Subject and Object by functors like Actor, Patient, Addressee etc. (see Hajičová 1998, for a full list of functors, see Mikulová et al. (2006) and also "Appendix D: List of Attributes in PDT 2.0").

The attribute `functor` describes the dependency between a dependent node and its governor and is stored at the son-nodes. A tectogrammatical lemma (attribute `t_lemma`) is assigned to every node. Grammatemes are rendered as a set of 16 attributes grouped by the "prefix" `gram` (e.g. `gram/verbmod` for verbal modality).

The total of 39 attributes are assigned to every non-root node of the tectogrammatical tree, although (based on the node type) only a certain subset of the attributes is necessarily filled in.

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

Coreference relations between nodes of certain category types are captured (Kučová et al. 2003), distinguishing also the type of the relation (textual or grammatical). Each node has an identifier (attribute id) that is unique throughout the whole corpus. Attributes `coref_text.rf` and `coref_gram.rf` contain ids of the coreferential nodes of the respective types.

3 The Query Language

We introduce a query language that satisfies linguistic requirements stated in the previous section. We present the language informally on a series of examples. A formal definition of the textual form of the query language can be found in "Appendix C: FS Query Format Description". The query language is an extension of the existing query language of Netgraph 1.0, as presented in Section 2.2.

The proposed query language has two forms – a graphical form, which we call Netgraph Query Language, and a textual form, which we call FS Query Language. Netgraph Query Language is a graphical representation of FS Query Language. The query languages are equivalent. Each query in the textual form has its graphical counterpart and vice versa.

Users usually work with the graphical form of the query. It follows the idea "what you see is what you get", or rather "what you want to see in the result is what you draw in the query". The textual form cannot contain any formatting white characters. In this chapter, we always show both the graphical and the textual version of the query. In the subsequent chapters, we usually use only one of the versions, to save space. We present examples both from the analytical and the tectogrammatical layer; the attributes used in the query always show which of the layers is used (see "Appendix D: List of Attributes in PDT 2.0"). In the result analytical trees, usually the attributes `m/lemma` and `afun` are displayed, while in the tectogrammatical trees, usually the attributes `t_lemma` and `functor` are displayed.

The 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

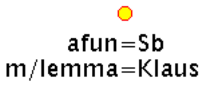
The simplest possible query is a simple node without any evaluation:

In the textual form, a node is enclosed in square brackets:

```
[]
```

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:



In the textual form, the attribute=value pairs are separated by a comma (","):

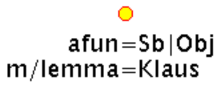
```
[m/lemma=Klaus,afun=Sb]
```

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

3.2 Alternative Values and Nodes

3.2.1 Alternative Values

Alternative values of attributes are separated by a vertical bar ("|"):



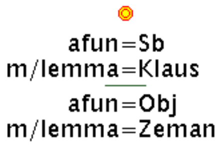
with the textual form:

```
[m/lemma=Klaus,afun=Sb|Obj]
```

This time, the node with lemma Klaus can either be a Subject ("Sb") or an Object ("Obj").

3.2.2 Alternative Nodes

It is possible to define an entire alternative set of values of attributes, like in the following example:



In the textual form, the alternative set of attributes, actually an alternative node, is separated by a vertical bar ("|"):

```
[m/lemma=Klaus,afun=Sb]|[m/lemma=Zeman,afun=Obj]
```

This query matches trees containing a node that is either a Subject with lemma Klaus, or an Object with lemma Zeman.

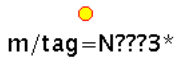
3.3 Wild Cards

Two wild cards can be used in values of attributes:

- "?" stands for any one character
- "*" stands for a sequence of characters (of length zero or greater)

The special meaning of these wild cards can be suppressed with a backslash ("\"). (To suppress the special meaning of a backslash, it can itself be escaped with another backslash.)

The following query searches for all trees containing a node that is a noun in the dative (the first position of the tag denotes part of speech, the fifth position denotes case)[2]:



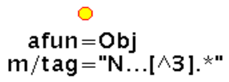
with the textual form:

```
[m/tag=N???3*]
```

To suppress the special meaning of these wild cards in the textual form of the query, two backslashes ("\\") must be used.

3.4 Regular Expressions

Beside the wild cards in values of attributes, a Perl-like regular expression (Hazel 2007) can be used as a whole value of an attribute. If a value of an attribute is enclosed in quotation marks, the value is considered an anchored[3] regular expression. The following query searches for all trees containing a node that is an Object, also a noun but not in the dative:



In the textual version, some characters (namely "[", "]", "(", ")", "=", "," and "|") have to be escaped with a backslash ("\"):

```
[afun=Obj,m/tag="N...\[^3\].*"]
```

Although regular expressions can fully replace wild cards introduced above, for backward compatibility and maybe for simplicity, the wild cards remain in the language. Moreover, references (see Section 3.9 below) cannot be a part of a regular expression[4] but they can be combined with the wild cards.

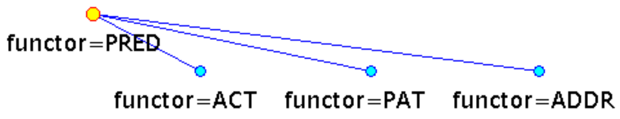
2 See "Appendix D: List of Attributes in PDT 2.0" for a description of positions of the attribute m/tag.

3 "Anchored" means that it must match the whole value of the attribute in the result tree (not only its substring).

4 A regular expression has to be compiled before it can be matched with a string. The compilation is only made once for each regular expression in the query. If it could contain references, it would have to be compiled each time a value is substituted for the reference, i.e. many times for each searched tree.

3.5 Dependencies Between Nodes

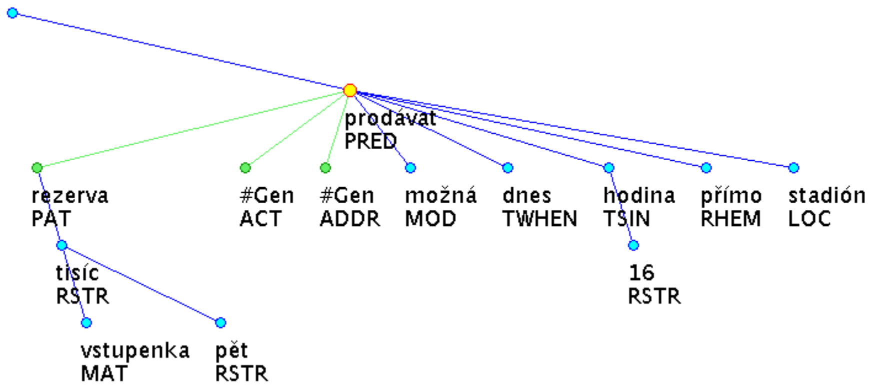
Dependencies between nodes are expressed directly in the syntax of the query language. Since the result is always a tree, the query also is a tree (or a multi-tree, see Section 3.10 below) and the syntax does not allow non-tree constructions. The following query searches for Predicates ("PRED") that directly govern an Actor ("ACT"), a Patient ("PAT") and an Addressee ("ADDR"):



In the textual version, sons of a node are separated by a comma (","), together they are enclosed in parentheses ("(", ")") and follow directly their father:

```
[functor=PRED]([functor=ACT],[functor=PAT],[functor=ADDR])
```

The following tree is a possible result for this query:



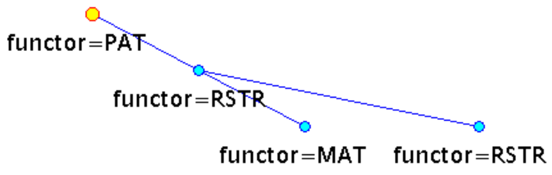
In Czech: *Rezerva$_1$ pěti tisíc vstupenek se_možná_bude_prodávat$_2$ dnes od 16 hod. přímo na stadionu.*
In English: *A_reserve$_1$ of five thousand tickets may_be_sold$_2$ today from 4 pm. directly at the stadium.*

The subtree matching the query is highlighted with green, the node matching the root of the query is highlighted with the yellow colour and slightly enlarged.

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 tree.

To make quite clear how to stack dependencies in the textual form of the query, let us give another example. The following query searches for a Patient ("PAT") that governs a Restriction ("RSTR") that governs a Material ("MAT") and another Restriction ("RSTR").

The result tree given above matches this query too:



With the textual version:

```
[functor=PAT]([functor=RSTR]([functor=MAT],[functor=RSTR]))
```

3.6 Arithmetic Expressions

Some attributes contain numeric values. Simple arithmetic expressions can be used in values of these attributes, namely addition ("+") and subtraction ("-"). Since it is impossible to give a meaningful example now, we postpone giving an example until after references are introduced in Section 3.9.

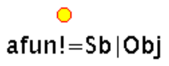
3.7 Other Relations

In setting values of attributes, the following relations can be used:

- equal to ("=")
- not equal to ("!=")
- less than ("<")
- less than or equal to ("<=")
- greater than (">")
- greater than or equal to (">=")

For numeric values, the relations are understood in their mathematical meaning. For textual values, alphabetical ordering is used. For each attribute, the relation can only be set once. It is therefore common for all alternative values of the attribute. If alternative values are used with relation "not equal to", the meaning is "the value is neither of these values".

The following query searches for all nodes that are neither Subjects, nor Objects:



With the textual form:

```
[afun!=Sb|Obj]
```

3.8 Meta-Attributes

The query language presented so far offers no possibility to set more complex negation, restrict the position of the query tree in the result tree or the size of the result tree. Nor the order of nodes can be controlled. Meta-attributes bring additional power to the query system.

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 like normal attributes. To be easily recognized, names of the meta-attributes start with an underscore ("_"). 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:

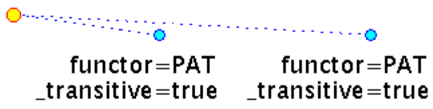
- `_transitive` – defines a transitive edge
- `_optional` – defines an optional node
- `_#sons` – defines number of sons of a node
- `_#hsons` – defines number of hidden sons of a node
- `_depth` – defines a depth of a node
- `_#descendants` – defines number of descendants of a node
- `_#lbrothers` – defines number of left brothers of a node
- `_#rbrothers` – defines number of right brothers of a node
- `_#occurrences` – defines number of occurrences of a node
- `_name` – names a node for a later reference
- `_sentence` – contains the linear form of the sentence

The following subsections offer a detailed description of the individual meta-attributes.

3.8.1 _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[5].

A truly transitive edge merely expresses the fact that a node belongs to the subtree of another node. The following query searches for a tree containing two Patients anywhere in the tree:

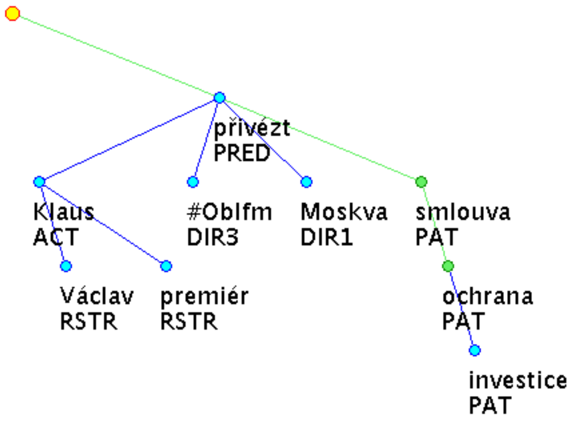


5 In Netgraph, alternative values cannot be defined for meta-attribute `_transitive`.

With the textual version:

```
[]([functor=PAT,_transitive=true],[functor=PAT,_transitive=true])
```

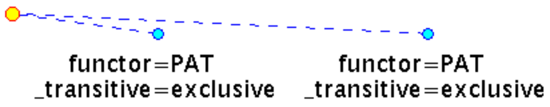
The following tree is a possible result for this query:



In Czech: *Premiér Václav Klaus přivezl z Moskvy smlouvu$_1$ o_ochraně$_2$ investic.*
In English: *Prime minister Václav Klaus has brought an_agreement$_1$ about_a_protection$_2$ of investments from Moscow.*

The root of the result tree matches the root of the query. Please note that both Patients matching the query, although in this particular result one depends on the other, are in the subtree of the root (in the result tree), which is exactly what the query requires.

To prevent the possibility of the Patients to depend on one another, the exclusive transitivity can be used in the query:

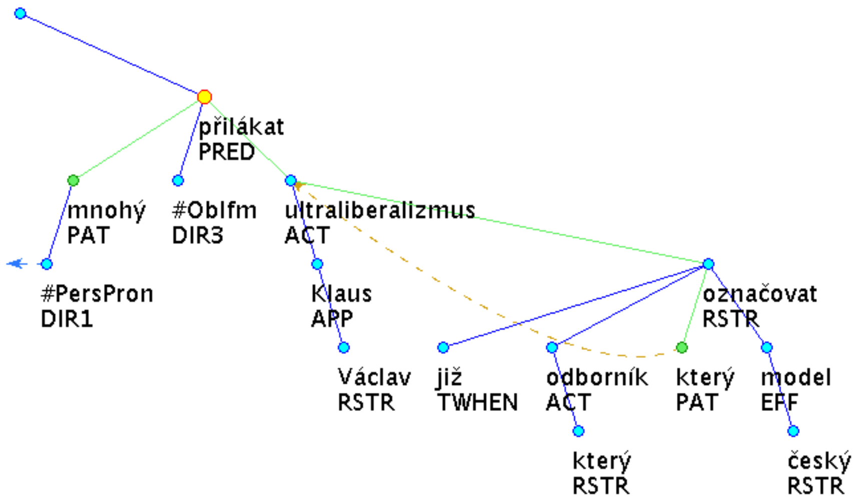


With the textual version:

```
[]([functor=PAT,_transitive=exclusive],[functor=PAT,_transitive=exclu-
sive])
```

Exclusively transitive edges cannot share nodes in the result tree and therefore make sure that neither of the Patients in the example query can belong to the subtree of the other Patient.

The following result tree matches this query:



In Czech: *Mnozí₁ z nich byli_přilákáni₂ ultraliberalismem Václava Klause, který₃ již někteří odborníci označují jako „český model".*
In English: *Many₁ of them were_attracted₂ by the ultra-liberalism of Václav Klaus, which₃ some experts already term as "Czech model".*

While both result trees match the first query (the query with two truly transitive edges), only the second result tree matches the second query (the query with two exclusively transitive edges).

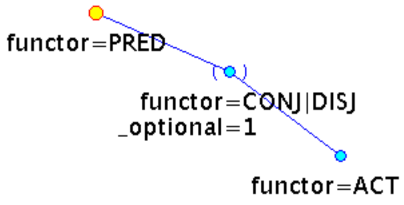
3.8.2 _optional

The meta-attribute `_optional` defines an optional node[6]. 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.

6 In Netgraph, the meta-attribute `_optional` can only be defined once at a node. If there are alternative nodes defined, it can be used in any of the sets of attributes. It can only be used with the relation equal ("="). It cannot use alternative values. It cannot be used at the root of the query.

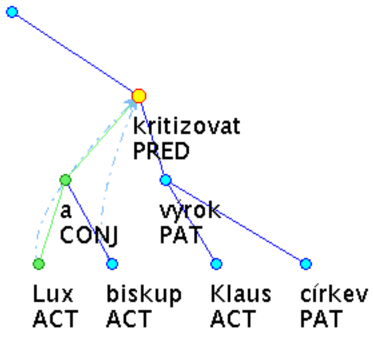
The following query searches for trees containing a Predicate that either directly governs an Actor, or there is a Conjunction or a Disjunction node between the Predicate and the Actor:



With the textual version:

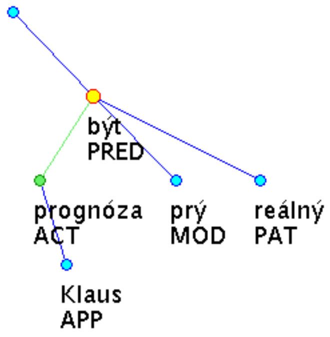
```
[functor=PRED]([functor=CONJ|DISJ,_optional=1]([functor=ACT]))
```

There are two possible types of result trees for this query (with or without the optional coordinating node). The following tree represents results with the optional coordinating node:



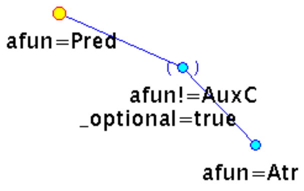
In Czech: *Lux$_1$ a$_2$ biskupové kritizovali$_3$ Klausovy výroky o církvi.*
In English: *Lux$_1$ and$_2$ bishops criticized$_3$ Klaus's statements about the Church.*

The next tree represents results without the optional coordinating node:



In Czech: *Klausovy prognózy$_1$ jsou$_2$ prý reálné.*
In English: *Klaus's forecasts$_1$ are$_2$ allegedly realistic.*

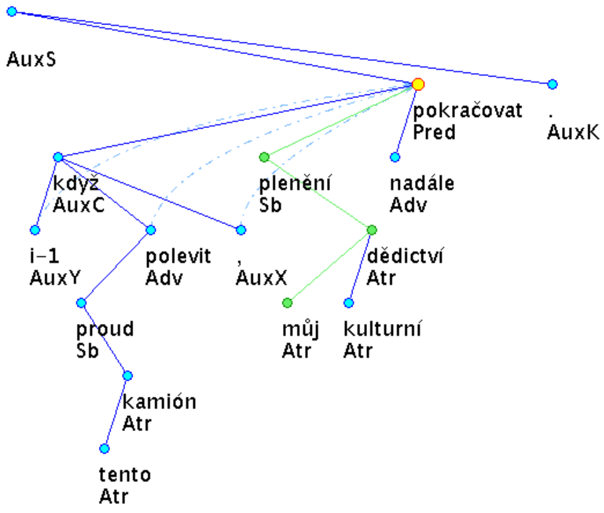
The following query demonstrates the usage of the meta-attribute `_optional` with the value `true`. It searches for Attributes ("`Atr`") anywhere in the subtree of a Predicate ("`Pred`") but does not allow a subordinating conjunction ("`AuxC`") appear on the path from the Predicate to the Attribute:



With the textual version:

```
[afun=Pred]([afun!=AuxC,_optional=true]([afun=Atr]))
```

The following tree is a possible result for this query:



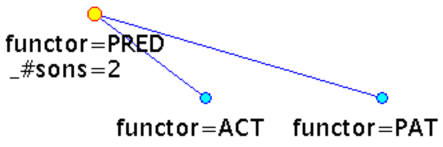
In Czech: *I když proud těchto kamionů polevil, plenění našeho$_1$ kulturního dědictví nadále pokračuje$_2$.*
In English: *Even though the stream of these lorries slackened, the plundering of our$_1$ cultural heritage still continues$_2$.*

In this particular result, the nodes `plenění(Sb)` and `dědictví(Atr)` match the optional node from the query, and the node `můj(Atr)` matches the `Atr` node from the query. The three Attributes ("`Atr`") on the right side of the tree can match the Attribute from the query, while the two Attributes on the left side of the tree cannot, because of the `AuxC` node lying on the path from the Attributes to the Predicate ("`Pred`").[7]

7 The node dědictví(Atr) can also match the Atr node from the query; Together with pokračovat(Pred) and plenění(Sb), these three nodes match the whole query and form another result.

3.8.3 _#sons

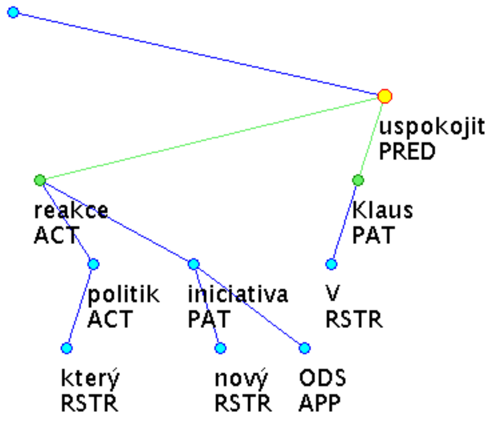
The meta-attribute `_#sons` ("number of sons") controls the exact number of sons of a node in the result tree. The following query searches for a Predicate governing an Actor and a Patient and nothing else:



With the textual version:

```
[functor=PRED,_#sons=2]([functor=ACT],[functor=PAT])
```

The following tree is a possible result for this query:



In Czech: *Reakce$_1$ některých politiků na novou iniciativu ODS V. Klause$_2$ uspokojily$_3$.*
In English: *V. Klaus$_2$ was_satisfied$_3$ with responses$_1$ of some politicians to the new initiative of ODS.*

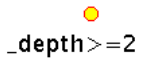
The meta-attribute `_#sons` prevented the Predicate from having more than two sons in the result tree. The predicate could not have less than two sons in the result also because there were two sons in the query.

3.8.4 _#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. Let us postpone giving an example of this meta-attribute until after the hidden nodes have been introduced in Section "3.11 - Hidden Nodes".

3.8.5 _depth

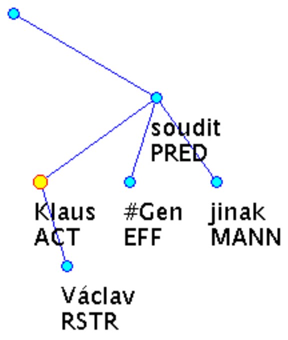
The meta attribute `_depth` controls the distance of a node in the result tree from the root of the result tree. The following query searches for all nodes that are at level 2 or greater – their distance from the root is at least 2:



With the textual version:

```
[_depth>=2]
```

All nodes in the following tree but the root and the Predicate match the query; the first result in the tree is displayed:

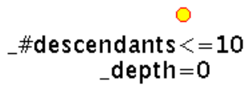


In Czech: *Václav Klaus$_1$ soudí jinak.*
In English: *Václav Klaus$_1$ thinks otherwise.*

3.8.6 _#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.

The following query searches for all trees consisting of at most 10 nodes (plus the technical root that matches the query node (because of `_depth=0`)):

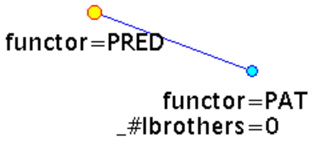


With the textual version:

```
[_depth=0,_#descendants<=10]
```

3.8.7 _#lbrothers

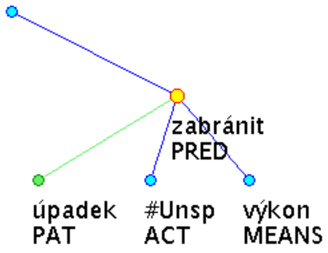
The meta-attribute `_#lbrothers` ("number of left brothers") controls the exact number of left brothers of a node in the result tree. The following query searches for a Predicate that governs a Patient as its first son:



With the textual version:

```
[functor=PRED]([functor=PAT,_#lbrothers=0])
```

The following tree is a possible result for the query:



In Czech: *Úpadku$_1$ zabránili$_2$ výkonem.*
In English: *They prevented$_2$ bankruptcy$_1$ with effort.*

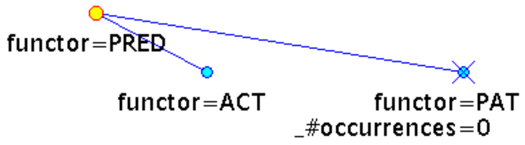
3.8.8 _#rbrothers

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

3.8.9 _#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).

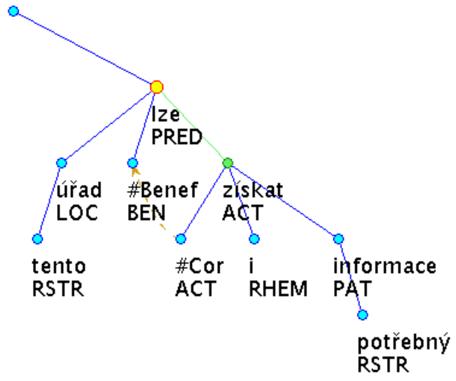
The following query searches for Predicates that govern (directly) an Actor but not a Patient:



With the textual form:

```
[functor=PRED]([functor=ACT],[functor=PAT,_#occurrences=0])
```

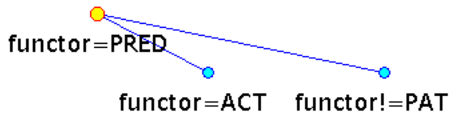
The following tree is a possible result for this query:



In Czech: *Na tomto úřadě lze$_1$ získat$_2$ i potřebné informace.*
In English: *Even useful information can$_1$ be_obtained$_2$ at this office.*

The Predicate ("PRED") in the result tree can have other sons than the Actor ("ACT"). Nevertheless, non of them can be a Patient ("PAT").

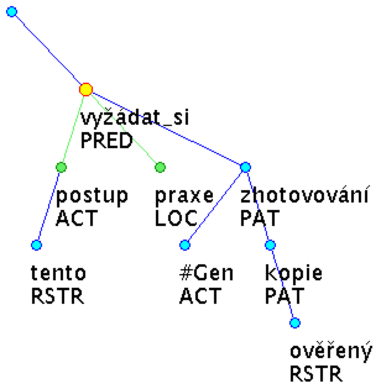
Please note that the following query has quite a different meaning:



With the textual version:

```
[functor=PRED]([functor=ACT],[functor!=PAT])
```

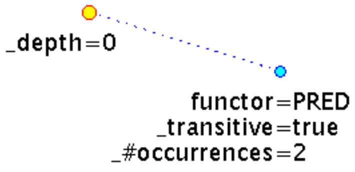
The following tree is a possible result for the query:



In Czech: *Tento postup$_1$ si_vyžádá$_2$ v_praxi$_3$ zhotovování ověřených kopí.*
In English: *In_practice$_3$, this procedure$_1$ will_require$_2$ production of certified copies.*

The "non-Patient" node from the query matches the Locative ("LOC") in the result tree and does not prevent another son from being a Patient ("PAT").

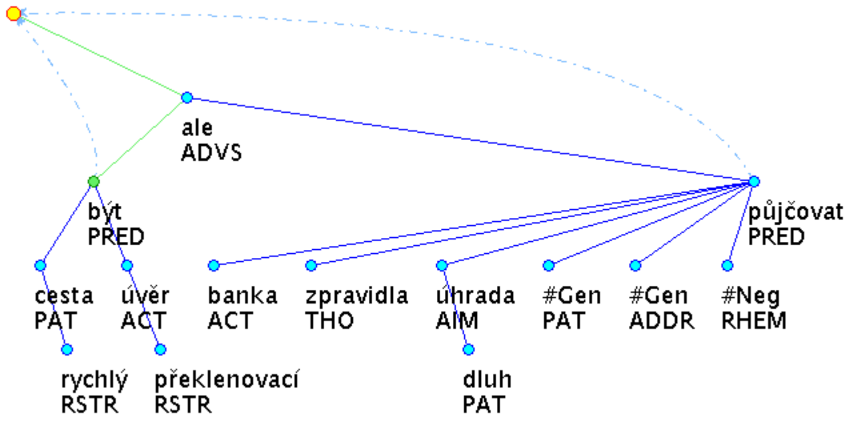
The meta-attribute `_#occurrences` can be combined with the meta-attribute `_transitive` set to the value `true` for the transitive meaning of the occurrences; then, it controls how many nodes of the kind can occur in the whole subtree of the father of the node in the result tree (excluding the father). The following query searches for trees that contain exactly two Predicates (in the whole tree; the technical root cannot be a Predicate):



With the textual version:

```
[_depth=0]([functor=PRED,_transitive=true,_#occurrences=2])
```

Note: If the meta-attribute `_#occurrences` is combined with `_transitive=true`, the father node in the query may even be omitted and the query may consist only of the node defining the Predicate, with the same result. It may be simpler but probably is less intuitive. The following tree is a possible result for the query:



In Czech: *Nejrychlejší cestou by_byl$_1$ překlenovací úvěr, ale banky zpravidla na úhradu dluhů nepůjčují.*
In English: *The bridging loan would_be$_1$ the fastest way but banks usually do not lend money for settlement of debt.*

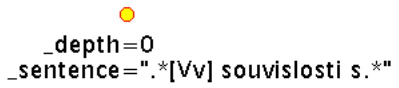
Since only one Predicate is actually drawn in the query, only one is highlighted in the result.

3.8.10 _name

The meta-attribute `_name` is used to name a node for a later reference, see Section “3.9 - References” below.

3.8.11 _sentence

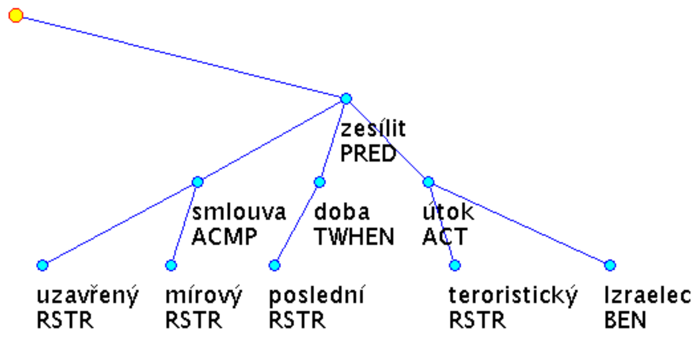
The meta-attribute `_sentence` can be used to search in the linear surface form of the trees – in the sentences. The following query searches for all trees (sentences) that contain the expression “v souvislosti s” (“in connection with”), regardless of its position in the sentence. To avoid matching each node in these trees, we use the meta-attribute `_depth`. It makes sure that only the root will match the query node:



With the textual version:

```
[_sentence=".*\[Vv\] souvislosti s.*",_depth=0]
```

The following tree is a possible result for the query:

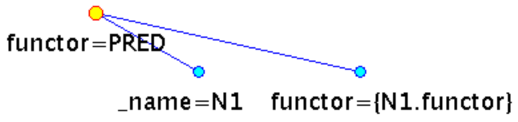


In Czech: *V_souvislosti_s$_1$ uzavřenými mírovými smlouvami v poslední době zesílily teroristické útoky proti Izraelcům.*
In English: *In_connection_with$_1$ the signed treaties of peace, terrorist attacks towards Israelis recently intensified.*

Since the expression “v souvislosti s” is considered a secondary preposition and not an auto-semantic word(s), it is not represented with a node on the tectogrammatical layer. Thanks to the meta-attribute `_sentence`, it can still be easily found.

3.9 References

References serve to refer in the query to values of attributes in the result trees, to values unknown at the time of creating the query. First, a node in the query has to be named using the meta-attribute _name.[8] 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:

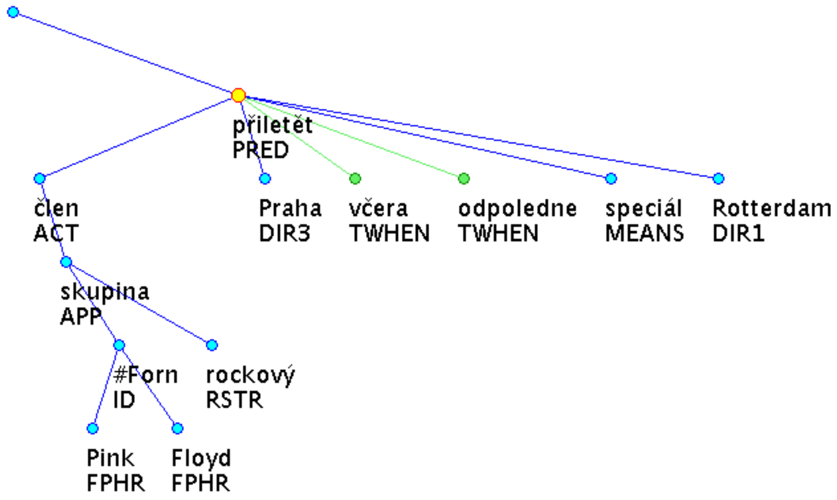


With the textual form:

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

The reference is enclosed in braces ("{", "}") and the name of the node that is referred to is separated from the name of the attribute with a dot ("."). The first son is named N1, the functor of the second son is set to the same value as the functor of the node N1 in the result tree.

The following tree is a possible result for the query. In this case, the functor of the two sons is TWHEN:

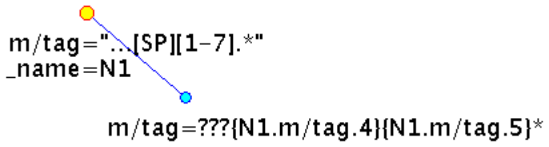


In Czech: *Členové rockové skupiny Pink Floyd přiletěli$_1$ do Prahy včera$_2$ odpoledne$_3$ speciálem z Rotterdamu.*
In English: *Members of the rock group Pink Floyd arrived$_1$ in Prague yesterday$_2$ afternoon$_3$ with a special flight from Rotterdam.*

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 an-

8 In Netgraph, the meta-attribute _name can only be defined once at a node. If there are alternative nodes defined, the meta-attribute _name can only be used in the first set of attributes. It can only be used with the relation equal ("="). It cannot use alternative values.

other 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. The following query searches for a father and a son that agree in case and number (which are the fourth and fifth position of the morphological tag (attribute `m/tag`):

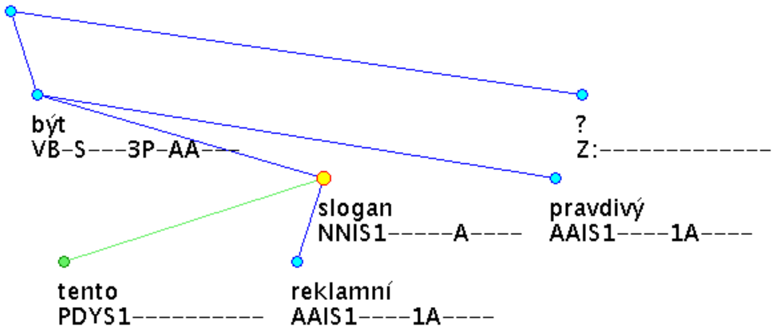


With the textual version:

```
[_name=N1,m/tag="...\[SP\]\[1-7\].*"]([m/tag=???{N1.m/tag.4}
{N1.m/tag.5}*])
```

The definition of the tag of the father ensures that the tag is defined and sets which values are acceptable at the fourth and fifth positions. The definition of the tag of the son makes sure that the fourth and fifth positions of the two tags are the same, regardless of other positions.

The following tree is a possible result for the query:

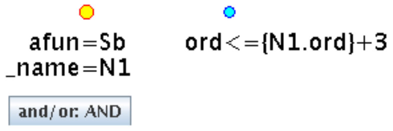


In Czech: *Je tento$_1$ reklamní slogan$_2$ pravdivý?*
In English: *Is this$_1$ advertising slogan$_2$ honest?*

A reference cannot be a part of a regular expression.

3.10 Multi-Tree Queries

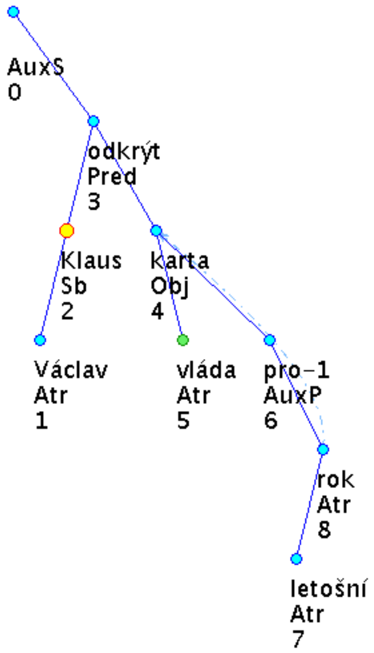
A multi-tree query consists of several trees combined either with a general `AND` or a general `OR`. In the case of `AND`, all the query trees are required to be found in the result tree at the same time (different nodes in the query cannot be matched with one node in the result), while in the case of `OR`, at least one of the query trees is required to be found in the result tree. The following query also demonstrates a usage of an arithmetic expression. It takes advantage of the fact that the attribute `ord` controls the horizontal order of nodes in the analytical trees. The query searches for a Subject and a node that can either be anywhere to the left from the Subject or, if to the right, at the distance at most three:



In the textual version, the required boolean combination (`AND` or `OR`) is on the first line and each tree is placed separately on the subsequent lines:

```
AND
[_name=N1,afun=Sb]
[ord<={N1.ord}+3]
```

The following tree shows a possible result for the query. Attributes `m/lemma`, `afun` and `ord` are displayed:



In Czech: *Václav Klaus*$_1$ *odkryl karty vlády*$_2$ *pro letošní rok*
In English: *Václav Klaus*$_1$ *revealed cards of the_government*$_2$ *for this year*

The horizontal order of nodes is displayed in the tree. The leftmost node is the root (`ord=0`). The node `Václav(Atr)` follows with `ord=1`, then `Klaus(Sb)` with `ord=2` and so on. The node `letošní(Atr)` is the rightmost but one (with `ord=7`), `rok(Atr)` with `ord=8` is the rightmost node in the tree.

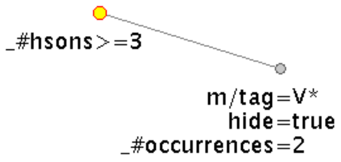
3.11 Hidden Nodes

Hidden nodes are nodes that are marked as hidden by setting the attribute `hide` to `true`.[9] Their visibility in result trees can be switched on and off. Hidden nodes serve as a connection to the lower layers of annotation or generally to any external source of information.

The search algorithm ignores the hidden nodes entirely unless a node in the query is explicitly marked as hidden. Some meta-attributes do not take the hidden nodes into account either. The meta-attribute `_#descendants` only counts non-hidden nodes in a subtree, as well as the meta-attribute `_#sons`. The meta-attribute `_#occurrences`, on the other hand, if used at a hidden node, treats hidden nodes as normal nodes. The meta-attribute `_#hsons` counts a number of hidden sons of a node.

Netgraph uses the hidden nodes as a connection to the lower layers of annotation with non-1:1 relation, as described later in Section “4.6 - Hidden Nodes”.

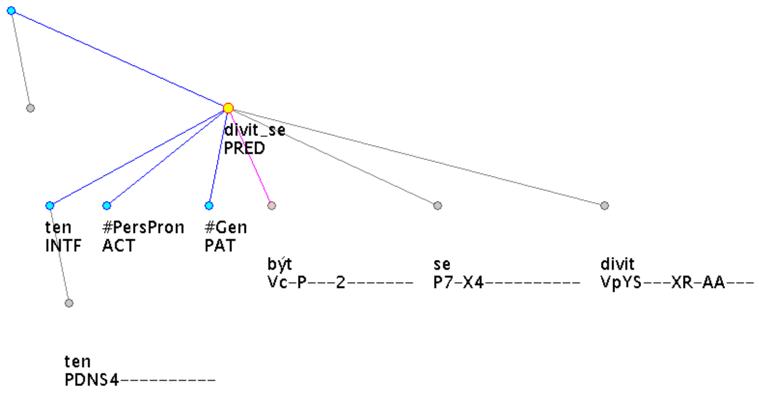
The following query searches for a node that has at least three hidden sons, two of which are verbs (their morphological tag starts with ”V”):



With the textual form:

```
[_#hsons>=3]([hide=true,m/tag=V*,_#occurrences=2])
```

The following tree is a possible result for the query:



In Czech: *To byste_se_divil*$_1$.
In English: *You would_be_surprised*$_1$.

9 In Netgraph, the attribute `hide` can only be defined once at a node. If there are alternative nodes defined, the attribute hide can only be used in the first set of attributes. It can only be used with the relation equal (”=”).

The nodes with the labels directly below the circles are nodes belonging to the tectogrammatical layer. All other nodes are the hidden nodes (now displayed), providing connection to the lower layers of annotation. The attributes `t_lemma` and `functor` are displayed at the tectogrammatical nodes, the attributes `m/lemma` and `m/tag` are displayed at the hidden nodes. The tectogrammatical node `divit_se(PRED)` has three tectogrammatical sons and three hidden sons.

Summary

Three sides existed whose connection is solved in this book. First, it was the Prague Dependency Treebank 2.0, one of the most advanced treebanks in the linguistic world. Second, there existed a very limited but extremely intuitive search tool – Netgraph 1.0. Third, there were users longing for such a simple and intuitive tool that would be powerful enough to search in the Prague Dependency Treebank.

In the book, we study the annotation of the Prague Dependency Treebank 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 show how all principal linguistic phenomena annotated in the treebank can be searched for with the query language.

The proposed query language has also been implemented – we present the search tool as well and talk about the data format for the tool. The tool is freely available and can be downloaded from the internet, as described in the Appendix.

Chapter 1 offers the introduction to the content of the book. In Chapter 2, related work and several existing search tools for treebanks are discussed. The annotation manuals for the Prague Dependency Treebank 2.0 are studied here as well and a list of requirements on a query language for the treebank is assembled. In Chapter 3, a query language that meets the requirements is presented. Chapter 4 is dedicated to the description of the data used in Netgraph. Hidden nodes are presented as a way of accessing lower layers of annotation. Chapter 5 shows that Netgraph Query Language fulfils the requirements stated in Chapter 2. Chapter 6 discusses some features of the query language. A comparison to several other query languages is also offered here. Chapter 7 introduces Netgraph – the tool that implements the query language. Chapter 8 shows to what extent the features of the query language are put to use by the users and what the users really do search for. Representative examples of real queries set by users are presented here. Chapter 9 concludes and summarizes what has been done and suggests what can be done in the future.

Much additional information can be found in Appendixes, including a list of publications about Netgraph, a formal description of the data format used in Netgraph, a formal description of the syntax of the query language implemented in Netgraph, also a comprehensive list of all attributes of the Prague Dependency Treebank 2.0 used in Netgraph. Usages of Netgraph for some other treebanks are presented as well and short installation and usage instructions for Netgraph are offered.

Bibliography

Bird et al. (2000): Towards A Query Language for Annotation Graphs. *In: Proceedings of the Second International Language and Evaluation Conference, Paris, ELRA, 2000.*

Bird et al. (2005): Extending Xpath to Support Linguistc Queries. *In: Proceedings of the Workshop on Programming Language Technologies for XML, California, USA, 2005. .*

Bird et al. (2006): Designing and Evaluating an XPath Dialect for Linguistic Queries. *In: Proceedings of the 22nd International Conference on Data Engineering (ICDE), pp 52-61, Atlanta, USA, 2006.*

Boag et al. (1999): XQuery 1.0: An XML Query Language. *IW3C Working Draft, http://www.w3.org/TR/xpath, 1999.*

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

Cassidy S. (2002): XQuery as an Annotation Query Language: a Use Case Analysis. *In: Proceedings of the Third International Conference on Language Resources and Evaluation, Canary Islands, Spain, 2002*

Clark J., DeRose S. (1999): XML Path Language (XPath). *http://www.w3.org/TR/xpath, 1999.*

Čermák, F. (1997): Czech National Corpus: A Case in Many Contexts. *International Journal of Corpus Linguistics 2, 1997, 181–197.*

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. (1998): Building a Syntactically Annotated Corpus: The Prague Dependency Treebank. *In Issues of Valency and Meaning, Karolinum, Praha 1998, pp. 106-132.*

Hajič J. (2004): Complex Corpus Annotation: The Prague Dependency Treebank. *Jazykovedný ústav Ĺ. Štúra, SAV, Bratislava, 2004.*

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

Hajič J., Pajas P. and Vidová-Hladká B. (2001b): The Prague Dependency Treebank: Annotation Structure and Support. *In IRCS Workshop on Linguistic databases, 2001, pp. 105-114.*

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., Panevová J., Buráňová E., Urešová Z., Bémová A. (1999): Annotations at analytical level, Instructions for annotators. *Available from http://ufal.mff.cuni.cz/pdt2.0/doc/pdt-guide/en/html/ch05.html; also available on PDT 2.0 CD-ROM (Hajič et al. 2006), 1999.*

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

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.*

Hajičová E., Havelka J., Sgall P., Veselá K., Zeman D. (2004): Issues of Projectivity in the Prague Dependency Treebank. *MFF UK, Prague, 81, 2004.*

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.*

Hinrichs E. W., Bartels J., Kawata Y., Kordoni V., Telljohann H. (2000): The VERBMOBIL Treebanks. *In Proceedings of KONVENS, 2000.*

Kallmeyer L. (2000): On the Complexity of Queries for Structurally Annotated Linguistic Data. *In Proceedings of ACIDCA'2000, Corpora and Natural Language Processing, Tunisia, 2000, pp. 105-110.*

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.*

Křen M. (1996): Editor grafů. *Master Thesis, Charles University in Prague, 1996.*

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.*

Lai C., Bird S. (2004): Querying and updating treebanks: A critical survey and requirements analysis. *In: Proceedings of the Australasian Language Technology Workshop, Sydney, Australia, 2004.*

Lezius W. (2002): Ein Suchwerkzeug für syntaktisch annotierte Textkorpora. *PhD. Thesis IMS, University of Stuttgart, 2002.*

Ljubopytnov V., Němec P., Pilátová M., Reschke J., Stuchl J. (2002): Oraculum, a System for Complex Linguistic Queries. *In: Proceedings of SOFSEM 2002, Student Research Forum, Milovy, 2002.*

Lopatková M., Žabokrtský Z., Benešová V. (2006): Valency Lexicon of Czech Verbs VALLEX 2.0. *Tech. Report No. 2006/34, UFAL MFF UK, 2006.*

Marcus M., Santorini B., Marcinkiewicz M. A. (1993): Building a large annotated corpus of English: the Penn Treebank. *In: Computational Linguistics, 19, 1993.*

Marcus M., Kim G., Marcinkiewicz M. A., MacIntyre R., Bies A., Ferguson M., Katz K., & Schasberger B. (1994): The Penn Treebank: annotating predicate argument structure. *In Proceedings of the human language technology workshop. Morgan Kaufmann Publishers Inc, 1994.*

Merz Ch., Volk M. (2005): Requirements for a Parallel Treebank Search Tool. *In: Proceedings of GLDV-Conference, Bonn, Germany, 2005.*

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. (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/*

Pajas P., Štěpánek J. (2006): XML-Based Representation of Multi-Layered Annotation in the PDT 2.0. *In: Proceedings of the LREC Workshop on Merging and Layering Linguistic Information (LREC 2006), Paris, France, 2006, pp. 40-47.*

Pajas P., Štěpánek J. (2005): A Generic XML-Based Format for Structured Linguistic Annotation and Its Application to Prague Dependency Treebank 2.0. *In: ÚFAL Technical Report, 29, MFF UK, Prague, 2005.*

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

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

Rychlý P. (2000): Korpusové manažery a jejich efektivní implementace. PhD. Thesis, Brno, 2000.

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.

Steiner I., Kallmeyer L. (2002): VIQTORYA – A Visual Tool for Syntactically Annotated Corpora. In: Proceedings of the Third International Conference on Language Resources and Evaluation (LREC), Las Palmas, Gran Canaria, 2002, pp. 1704-1711.

Štěpánek J. (2006): Závislostní zachycení větné struktury v anotovaném syntaktickém korpusu (nástroje pro zajištění konzistence dat). PhD. Thesis, Prague, 2006.

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*

Index