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.

2 The Problem Analysis

In the first part of this chapter, in Section 2.1, we focus on the related work. We mention some more or less theoretical papers about query languages for treebanks and also present several existing search tools for treebanks.

In Section 2.2, we describe Netgraph 1.0 – an existing tool for searching in PDT 1.0. It was a basis for further development in my research.

Afterwards, in Section 2.3, we focus on linguistic phenomena annotated in PDT 2.0 and requirements on the query language, posed by the phenomena and linguistic research needs.

Finally, in Section 2.4, we formulate a concise list of linguistic requirements on the query language for PDT 2.0.

2.1 Related Work

2.1.1 More or Less Theoretical Papers

In Lai, Bird (2004), the authors name seven linguistic queries they consider important representatives for checking a sufficiency of a query language power. They study several query tools and their query languages and compare them on the basis of their abilities to express these seven queries. In Bird et al. (2005), the authors use a revised set of seven key linguistic queries as a basis for forming a list of three expressive features important for linguistic queries. The features are: immediate precedence, subtree scoping and edge alignment. In Bird et al. (2006), another set of seven linguistic queries is used to show a necessity to enhance XPath (a standard query language for XML, Clark, DeRose 1999) to support linguistic queries.

Cassidy (2002) studies adequacy of XQuery (a search language based on XPath, Boag et al. 1999) for searching in hierarchically annotated data. Requirements on a query language for annotation graphs used in speech recognition is also presented in Bird et al. (2000). A description of linguistic phenomena annotated in the Tiger Treebank, along with an introduction to a search tool TigerSearch, developed especially for this treebank, is given in Brants et al. (2002), nevertheless without a systematic study of the required features.

Laura Kallmeyer (Kallmeyer 2000) studies requirements on a query language based on two examples of complex linguistic phenomena taken from the NEGRA corpus and the Penn Treebank, respectively.

To handle alignment information, Merz and Volk (2005) study requirements on a search tool for parallel treebanks.

All the work mentioned above can be used as an ample source of inspiration, though it cannot be applied directly to PDT 2.0. A thorough study of the PDT 2.0 annotation is needed to form conclusions about requirements on a search tool for this dependency tree based corpus, consisting of several layers of annotation and having an extremely complex annotation scheme.

2.1.2 Existing Search Tools

Manatee/Bonito

Manatee/Bonito (Rychlý 2000) is the first tool that needs to be mentioned. It is a well known search tool used for the Czech National Corpus (CNC), a huge corpus of Czech texts annotated automatically with morphological information (Čermák 1997), and also for many other linearly annotated linguistic corpora. Manatee/Bonito is a client-server oriented program. Many clients (Bonitos) can connect simultaneously to a server (Manatee), while the server performs the searching.

The query language is quite simple yet powerful for searching in the linear data. Let us give an example of a query:

```
[lemma="jaro" & tag="NNN.6.+" & word="j.+"]
```

will return all occurrences of words that have lemma "jaro", are in locative (both plural and singular since the position of number in the tag is filled with a dot), and begin with a lowercase character.

Manatee/Bonito is a very advanced tool for searching in linear linguistic data (such as morphologically annotated texts). Its usage for searching in structural data is naturally limited, since it is not intended for such a task.

The way of annotation of CNC is very similar to the way the morphological layer of PDT 2.0 is annotated. Manatee/Bonito can very well be (and actually is) used for linear searching in the morphological annotation of PDT 2.0.

TGrep

TGrep (Pito 1994) is a traditional line-based search tool developed primarily for the Penn Treebank (Marcus et al. 1993; Marcus et al. 1994). It can be used for any treebank where each node is evaluated with only one symbol – either a non-terminal or a token. Regular expressions can be used for matching node symbols. A set of predicates is used for expressing relations between nodes. A query example:

```
S <1 /^NP/ < (VP < (NP $.. NP))
```

means: Get all `S`s that start with an `NP` and that dominate a `VP` that in turn has two `NP` sons. The predicates used in this example mean:

`<1`	immediate dominance, first child
`<`	immediate dominance
`$..`	brotherhood, precedence

TGrep2

TGrep2 (Rohde 2005) is a sequel to TGrep. It is almost completely backward compatible with TGrep but brings a number of new features, from which we select:

- nodes can have full Boolean expressions of relationships to other nodes
- nodes can be given unique labels and may then be referred to by those labels in the pattern specification
- patterns are no longer restricted to simple tree architectures; the use of node labels and segmented patterns allows links in a pattern to form back-edges as well, permitting cycles of links
- multiple search patterns may be specified and one can retrieve the first subtree matching any pattern, the first subtree matching each pattern, or all subtrees matching any pattern, or all matches between subtrees and patterns
- several new predicates have been introduced
- macros can be defined and used to simplify pattern specification

Introduction of Boolean expressions allows setting such complex query patterns as:

```
A [< B | ![. C !, F]] | ![< D !.. E]
```

which means: (`A` has son `B` or it does not (immediately precede `C` and not immediately follow `F`)) or (`A` does not (have son `D` and is not followed by `E`)).

TigerSearch

TigerSearch (Lezius 2002) is a graphically oriented search tool for the Tiger Treebank (Brants et al. 2002). The query language consists of three levels. On the node level, nodes can be described by Boolean expressions over feature-value pairs:

```
[word="lacht" & pos="VVFIN"]
```

On the node relation level, descriptions of two or more nodes are combined by a relation. There are two basic relations - immediate precedence ("`.`") and immediate dominance ("`>`"). There are also derived node relations such as underspecified dominance, brotherhood etc. A labelled dominance is used in the following example:

```
[cat="NP"] >RC [cat="S"]
```

Finally, on the graph description level, Boolean expressions over node relations, without negation, are allowed, and variables can be used to express coreference of nodes or feature values, as shown in the next example (a node with category `S` is assigned to variable `#n` and used again in the second expression (as the very same node)):

```
(#n:[cat="S"] > [pos="PRELS"]) & (#n > [pos="VVFIN"])
```

It is important to add that all node expressions in the query are existentially quantified.

Oraculum

Oraculum (Ljubopytnov et al. 2002) is a tool developed for searching in the Prague Dependency Treebank, although it can be used for other treebanks, too. It is a client-server application, with the client part web-browser based. Oraculum is able to combine several data sources in one query and use the full power of logical programming in the queries. Making queries is a combination of clicking on buttons and writing logical formulas. Writing more complex queries requires a knowledge of logical programming in Prolog. To demonstrate the complexity of such queries, let us copy an example from the paper mentioned above, without detailed explanation. As the authors say, the following code finds all tectogrammatical trees whose head clause is a verb having either ″agens″ or ″patiens″valency actant and an actant whose morphological tag is not the same as of some descendant of the ″agens″/″patiens″ actant:

```
query([],[]).
query([Tree|Trees],Output) :-
  (struct(Tree,
    [[x, central, [left-any-eq-y, y-any-eq-z, z-any-eq-right],[y-z],
[('tag','V*')]],
     [y, [left-any-eq-right], [('afun','agens'),(or),
('afun','patiens')]],
     [z, [left-any-eq-right], [('tag',V)]], Matching_struct ]),
     not ( struct([u, ('tag',V)]), path(u, y, ['vu',(1,INF)]) )
     -> Output = [Tree, NextTrees] ; Output = [NextTrees]
  ),
   query(Trees, NextTrees).
```

Oraculum is a product of a student project and its development stopped shortly after the project had been defended.

TrEd

Tred (Pajas 2007) has been developed for the Prague Dependency Treebank since the year 2000. It is primarily a tool for editing trees but has been widely used for searching, especially during post-annotation corrections. Users can write complex queries in Perl programming language and access tree structures in object-oriented way. The search can be parallelized. The data can be processed non-interactively using scripts, which can also change the content of the data. The creation of a query requires at least a limited knowledge of Perl programming language. The following example shows a function for printing sentences containing a patient in plural dependent on a negated verb, regardless on any combination of coordination in the structure:

```
sub pluralpat() {
  if ($this->attr('gram/number') eq "pl" and $this->{functor} eq
"PAT"){
    foreach my $eparent (PML_T::GetEParents($this)) {
      if (grep {$_->{t_lemma} eq "#Neg"}
PML_T::GetEChildren($eparent)) {
        print "($this->{t_lemma})
".PML_T::GetSentenceString($root)."\n";
      }
    }
  }
}
```

All components including the treebank must reside at the same computer, or at least a local network.[1]

VIQTORYA

Viqtorya (Steiner, Kallmeyer 2002) is a search tool developed for the Tübingen Treebanks (Hinrichs et al. 2000). It has a graphical interface, but without a visual depiction of the query. A first order logic without quantification is chosen as a query language, with some restrictions. The following example query searches for a preposition von linearly preceding a preposition bis and, moreover, a prepositional phrase (with syntactic category PX) that dominates both prepositions:

```
token(1)=von & token(2)=bis & 1..2 & cat(3)=PX & 3>>1 & 3>>2
```

Natural numbers are used as variables, ”`..`” means a linear precedence, and ”>>” marks a transitive dominance.

Fsq

Finite structure query (fsq, Kepser 2003) is another query language developed for the Tübingen Treebanks. It uses the full first-order logic (with quantification), with LISP-like syntax. The following example query searches for trees without a subject in a subordinate clause and all its subclauses (written in in-fix syntax):

```
∃x∃y SIMPX(x) ∧ SIMPX(y) ∧ (x >> y) ∧ (x != y) ∧ (∀z !((y >> z) ∧
ON(z)))
```

`SIMPX` is a predicate expressing a clause node, `ON` denotes an Object in the nominative, ”>>” means a transitive dominance, and ”`!`” means negation.

In Chapter 6, in Section “6.5 - Comparison to Other Treebank Query Systems”, we show to what extent some of these other tools fulfill the requirements of PDT 2.0 and how they compare to our proposed system. Let us present now a starting point of the development of our own query system.

1 Recently, an extension to TrEd allowing advanced graphical searching has been introduced. It is called PML Tree Query, for details see `http://ufal.mff.cuni.cz/~pajas/pmltq/`.

2.2 Netgraph 1.0 – The Starting Point

The development of Netgraph started in 1998 as Roman Ondruška's Master Thesis (Ondruška 1998). We describe the result of his work in this section, in other words, what had been done before the work on the topic of this book began.

Netgraph 1.0 was being developed along with the annotation of PDT 1.0 as a search tool for the analytical layer of the corpus. It was a client/server application working in the internet environment. The server was written in C Programming Language and worked on Linux, the client was written in Java 1.0 as an applet for a web browser.

The core architecture of the tool was set and has not since then significantly changed. The server used FS File Format, which was one of two formats used during the work on PDT 1.0, both for treebank files and as a query language. Multiple users could connect to the server simultaneously. The user could choose files for searching, enter a query in the textual form and browse the result trees, displayed along with the sentences. It was possible to select attributes that would be displayed at nodes in the result trees. An individual node could be selected and all its attributes were displayed in a table. In queries, wild cards could be used ("?" stood for one character, "*" for a sequence of characters). Unfortunately, the client only supported ASCII characters; Czech accented characters could not be entered in the query, nor displayed in the trees.

Except for the wild cards, the query language was identical to FS File Format. A formal description of the format, still used in Netgraph, is given in Appendix B: FS File Format Description. Informally, the query language allowed defining tree structure and set values of attributes of individual nodes, using alternative values of attributes and nodes. Given the query tree, the search algorithm performed a subtree matching on the trees of the corpus.

In the query language of Netgraph 1.0, square brackets enclose a node, parentheses enclose a subtree. The following example query in Netgraph 1.0 searches for a Predicate governing directly an Object in the accusative:

```
[afun=Pred]([afun=Obj,tag=????4*])
```

For several limitations (like missing support for Czech accented characters), Netgraph 1.0 had never been really used for searching. Nevertheless, the core of the tool was well designed and proved to be a sound basis for further development. Also the query language was extremely intuitive and proved to be a good basis for a simple and full-featured query language for PDT 2.0.

2.3 Linguistic Phenomena in PDT 2.0

In this section, we make a list of linguistic phenomena that are annotated in PDT 2.0 and that determine the necessary features of the query language (partially published in Mírovský 2008d).

PDT 2.0 has three layers of annotation: the morphological layer, the analytical layer, and the tectogrammatical layer. To be exact, there is one more layer – the word layer – that 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. For using the morphological layer exclusively and directly, a very good search tool Manatee/Bonito, described in Section 2.1.2, can be used.

We intend to access the morphological information only from the higher layers, not directly. Since there is relation 1:1 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.

There is also almost 1:1 relation among tokens on the word layer (the layer of segmented text) and tokens on the morphological layer. The only exceptions are misprints in the input text. They do not cause any troubles in merging the word layer information into the morphological information, since the data format allows using alternative values for the case of merging two (or more) tokens from the word layer into one token on the morphological layer (the morphological token then has two (or more) counterparts on the word layer, represented as alternative values of respective attributes). In case of dividing one word token into two (or more) morphological tokens, the two (or more) morphological tokens simply refer to the same word token.

It is worth noting that the word layer only needs to be accessed if these particular misprints are studied. Otherwise, the corrected word layer information is present on the morphological layer. This area of studies is well outside our interest and scope of this work.

We therefore 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.

In other words, we either see/search in/study the analytical layer with all information from the lower layers available, or the tectogrammatical layer, also with all the information from the lower layers available. 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 non-1:1 relation between the analytical and 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 relation. Afterwards, we add some requirements on the query language set by 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 attribute `functor` of the complement is set to value `COMPL`. In this particular example, we also need to be able to specify in the query properties of the node the second dependency of the complement goes to, e.g. that it is an Actor.

2.3.1 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. It consists of a set of nodes and a set of edges. One of the nodes is marked as a root. Each node is a complex unit consisting of 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).

It 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

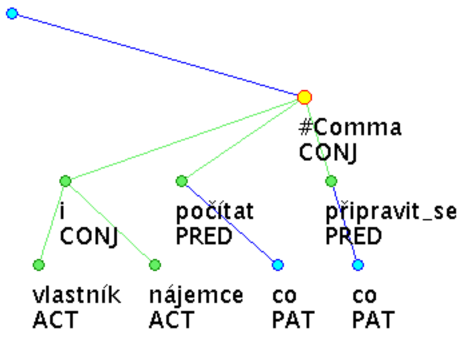
Valency of semantic verbs, valency of semantic verbal nouns, valency of semantic nouns that represent the nominal part of a complex predicate and valency of some semantic adverbs are annotated fully in the trees (t-manual, pages 162-3). Since the valency of verbs is most complete in the annotation and since the requirements on searching for valency frames of nouns are the same as of 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

The tree dependency is not always the linguistic dependency (t-manual, page 9). Coordination and apposition are examples of such a 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 they should 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).

Idioms (Phrasemes) etc.

Idioms/phrasemes (idiomatic/phraseologic constructions) are combinations of two or more words with a fixed lexical content, which together constitute one lexical unit with a metaphorical meaning (which cannot be decomposed into meanings of its parts) (t-manual, page 308). Only expressions which are represented by at least two auto-semantic nodes in the tectogrammatical tree are captured as idioms (functor DPHR). One-node (one-auto-semantic-word) idioms are not represented as idioms in the tree. For example, in the expression "chlapec k pohledání" ("a boy to look for"), the prepositional phrase (in Czech) gets functor RSTR, and it is not indicated that it is an idiom.

Secondary prepositions are another example of a linguistic phenomenon that can be easily recognized in the surface form of the sentence but is difficult to find in the tectogrammatical tree.

Therefore, the query language should also offer a basic searching in the linear form of the sentence, 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

A complex predicate is a multi-word predicate consisting of a semantically empty verb which expresses the grammatical meanings in a sentence, and a noun (frequently denoting an event or a state of affairs) that carries the main lexical meaning of the entire phrase

(t-manual, page 345). Searching for a complex predicate is a simple task and does not bring new requirements on the query language. It is valency of complex predicates that requires our attention, especially dual function of a valency modification. The nominal and verbal components of the complex predicate are assigned the appropriate valency frame from the valency lexicon. By means of newly established nodes with `t_lemma` substitutes, those valency modification positions not present at surface layer are filled. There are problematic cases 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 again by means of references, which is explained later (see Coreferences below).

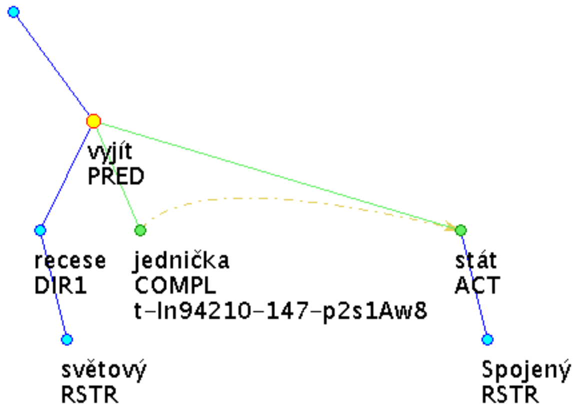
Predicative Complement (Dual Dependency)

On the tectogrammatical layer, also cases of the so-called predicative complement are represented. The predicative complement is a non-obligatory free modification (adjunct) which has a dual semantic dependency relation. It simultaneously modifies a noun and a verb (which can be nominalized).

These two dependency relations are represented by different means (t-manual, page 376):

- the dependency on a verb is represented by means of an edge (which means it is represented in the same way as other modifications),
- the dependency on a (semantic) 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 dependency on a verb (PRED) and another (dual) dependency on a noun (ACT):



In Czech: *Ze světové recese vyšly$_1$ jako_jednička$_2$ Spojené státy$_3$.*
In English: *The United States$_3$ emerged$_1$ from the world recession as_number_one$_2$.*

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

Coreferences

Two types of coreferences are annotated on the tectogrammatical layer:

- grammatical coreference
- textual coreference

The current way of representing coreference uses references (t-manual, page 996).

Let us finally explain what references are. References 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 dependency, or valency member identity is 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 dependency of predicative complement, we know that the value of attribute `compl.rf` of the complement must be the same as the value of 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.

Topic-Focus Articulation

On the tectogrammatical layer, also the topic-focus articulation (TFA) is annotated. TFA annotation comprises two phenomena:

- contextual boundness, which is represented by values of the attribute `tfa` for each node of the tectogrammatical tree.
- communicative dynamism, which is represented by the underlying order of nodes.

Annotated trees therefore contain two types of information – on the one hand, the value of contextual boundness of a node and its relative ordering with respect to its brother nodes reflects its function within the topic-focus articulation of the sentence, on the other hand, the set of all the TFA values in the tree and the relative ordering of subtrees reflect the overall functional perspective of the sentence, and thus enable to distinguish in the sentence the complex categories of topic and focus (however, these are not annotated explicitly) (t-manual, page 1118).

While contextual boundness itself does not bring any new requirement on the query language, 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 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 relation other than "equal to".

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. If the focus proper is constituted by an expression represented as the effective root of the tectogrammatical tree (i.e. the governing predicate is the focus proper), there is no right path leading from the effective root (t-manual, page 1129).

Quasi-Focus

Quasi-focus is constituted by (both contrastive and non-contrastive) contextually bound expressions, 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. 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.

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 the contrastive topic.

The position of rhematizers in the surface word order is quite loose, however they almost always stand right before the expressions they rhematize, i.e. the expressions whose being in the focus or the contrastive topic they signal (t-manual, pages 1165-6).

The guidelines for positioning rhematizers in tectogrammatical trees are simple (t-manual, page 1171):

- 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.
- if a rhematizer constitutes the focus proper, it is placed according to the guidelines for the position of the focus proper – i.e. on the rightmost path leading from the effective root of the tectogrammatical tree.

Rhematizers therefore 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 relative horizontal position of nodes.

(Non-)Projectivity

Projectivity of a tree is defined as follows: if two nodes B and C are connected by an edge and C is to the left from B, then all nodes to the right from B and to the left from C are connected with the root via a path that passes through at least one of the nodes

B or C. In short: 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 on the left from C” or “B is on the right from C”) x (“A is on the path from B to the root” or “it is not”). Each of the configurations can be searched for using properties of the language that have been required so far by other linguistic phenomena. Four different queries search for four different configurations.

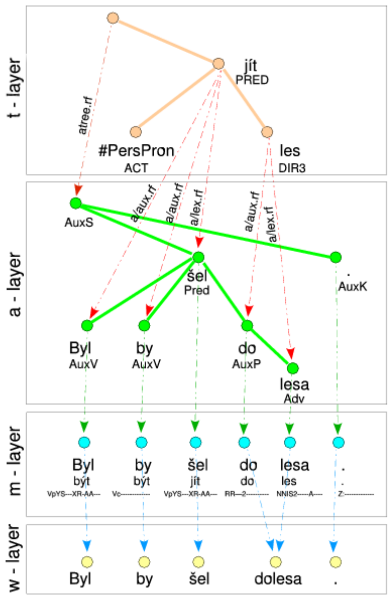
To be able to search for all configurations in one query, the query language should be able to combine several queries into one multi-query. We do not require that a general logical expression can be set above the single queries. We only require a general OR combination of the single queries.

2.3.2 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.

The following picture is taken from the PDT 2.0 guide and shows the typical relation among layers of annotation for a sentence:



In Czech: *Byl by šel do lesa.*

In English: *He would have gone to the forest.*

As we have already said, information from the lower layers can be easily compressed into the analytical layer, since there is relation 1:1 among tokens/nodes of the layers (with some rare exceptions like misprints in the w-layer). The situation 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.

2.3.3 The Analytical Layer (and Lower Layers)

Here, we focus on linguistic phenomena annotated on the analytical layer (or any lower layer) that bring a new requirement on the query language (that has not been set in the studies of the tectogrammatical layer).

The analytical layer is much less complex than the tectogrammatical layer. The basic principles are the same as on the tectogrammatical layer – the representation of the structure of a sentence is rendered in the form of a dependency tree, whose nodes are labelled with complex symbols (sets of attributes). The edges are not labelled (in the technical sense). The information logically belonging to an edge is represented in attributes of the depending node. One node is marked as a root.

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, e.g. the first position represents part of speech, the third position represents gender, the fourth position represents number, the fifth position represents case. For a full description of the morphological tags, please consult Appendix D: List of Attributes in PDT 2.0.

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 includenegation 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 strong 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 person, gender 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 it offers a perfect combination of a 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 attribute `ord`. Non-projective constructions are much more often and interesting here than on the tectogrammatical layer. Nevertheless, they appear also on the tectogrammatical layer and their contribution to the requirements on a query language has already been mentioned.

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".

2.4 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.

2.4.1 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 less than ("<") , greater than (">") (for numerical attributes)

2.4.2 Dependencies Between Nodes (Vertical Relations)

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

2.4.3 Horizontal Relations

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

2.4.4 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 non-1:1 relation (for studying relation between layers)
- searching in the surface form of the sentence

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

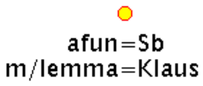
The simplest possible query is a simple node without any evaluation:

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 (","):

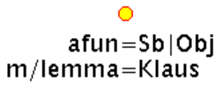
```
[m/lemma=Klaus,afun=Sb]
```

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 ("|"):



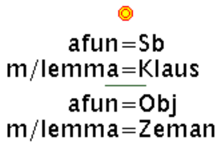
with the textual form:

```
[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.

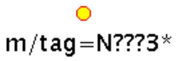
3.3 Wild Cards

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]:



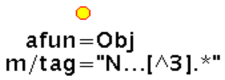
with the textual form:

```
[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.

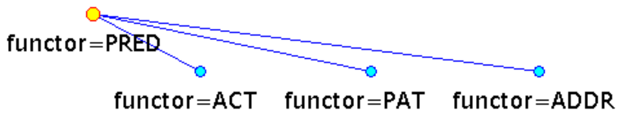
2 See "Appendix D: List of Attributes in PDT 2.0" for a description of positions of the attribute `m/tag`.

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

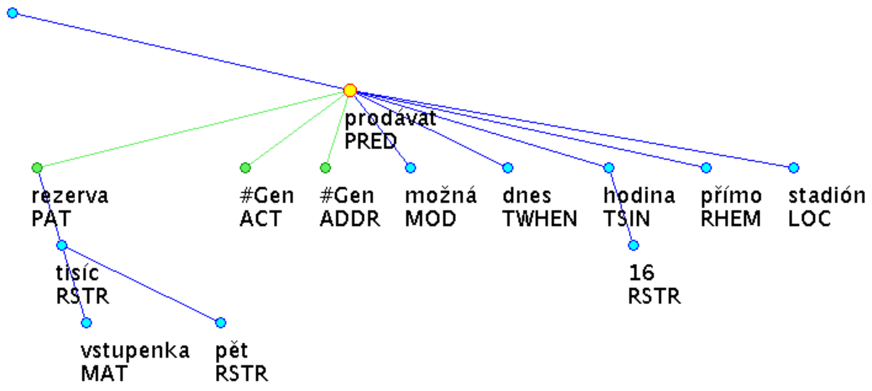
Dependencies between nodes are expressed directly in the syntax of the query language. Since the result is always a tree, the query also is a tree (or a multi-tree, see Section 3.10 below) and the syntax does not allow non-tree constructions. The following query searches for Predicates ("PRED") that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



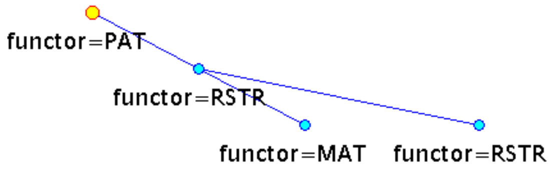
In Czech: *Rezerva$_1$ pěti tisíc vstupenek se_možná_bude_prodávat$_2$ dnes od 16 hod. přímo na stadionu.*
In English: *A_reserve$_1$ of five thousand tickets may_be_sold$_2$ today from 4 pm. directly at the stadium.*

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.

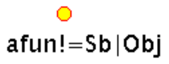
3.7 Other Relations

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:

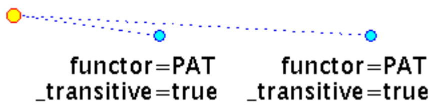
- `_transitive` – defines a transitive edge
- `_optional` – defines an optional node
- `_#sons` – defines number of sons of a node
- `_#hsons` – defines number of hidden sons of a node
- `_depth` – defines a depth of a node
- `_#descendants` – defines number of descendants of a node
- `_#lbrothers` – 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:

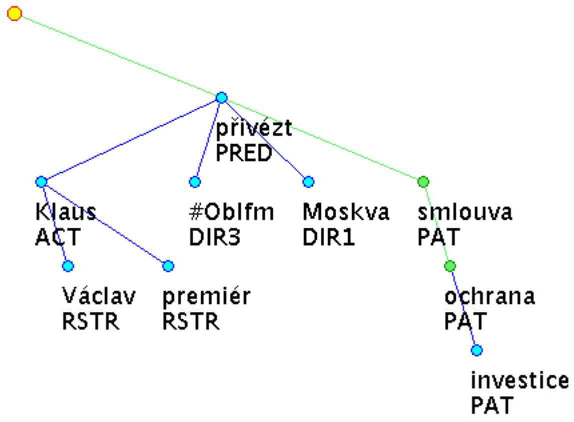


5 In Netgraph, alternative values cannot be defined for meta-attribute `_transitive`.

With the textual version:

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

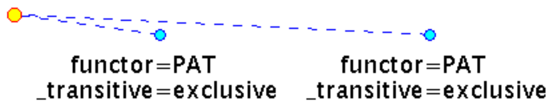
The following tree is a possible result for this query:



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:

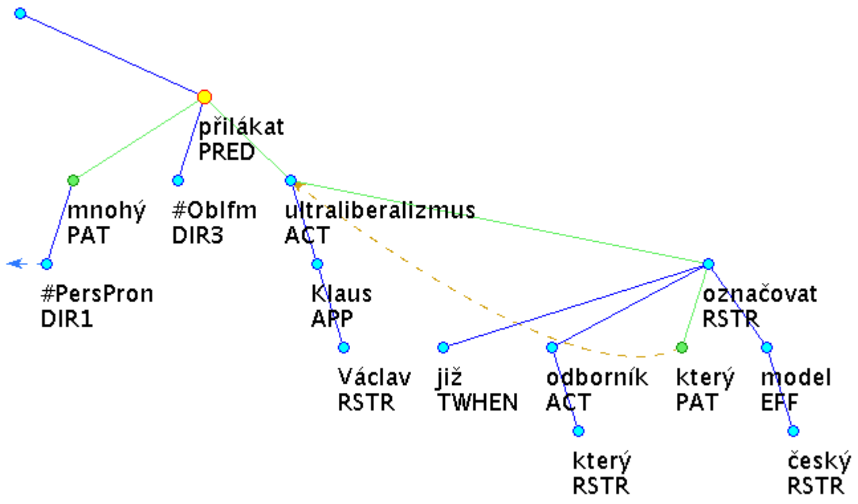


With the textual version:

```
[]([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: *Mnozí*$_1$ *z nich byli_přilákáni*$_2$ *ultraliberalismem Václava Klause, který*$_3$ *již někteří odborníci označují jako „český model".*
In English: *Many*$_1$ *of them were_attracted*$_2$ *by the ultra-liberalism of Václav Klaus, which*$_3$ *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).

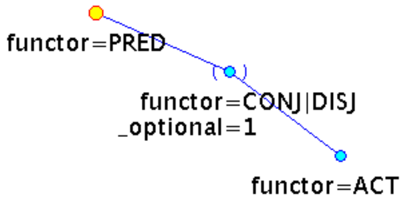
3.8.2 _optional

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.

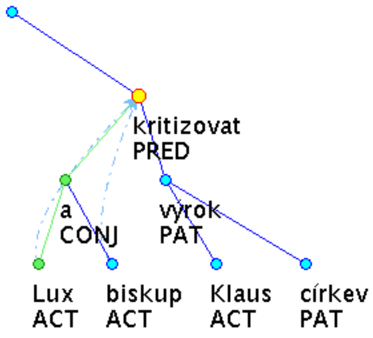
The following query searches for trees containing a Predicate that either directly governs an Actor, or there is a Conjunction or a Disjunction node between the Predicate and the Actor:



With the textual version:

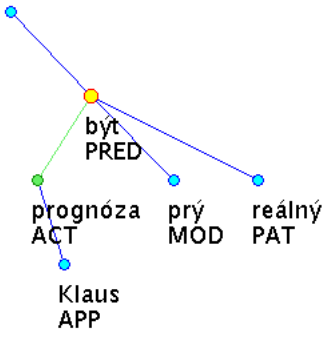
```
[functor=PRED]([functor=CONJ|DISJ,_optional=1]([functor=ACT]))
```

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:



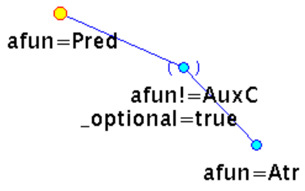
In Czech: *Lux$_1$ a$_2$ biskupové kritizovali$_3$ Klausovy výroky o církvi.*
In English: *Lux$_1$ and$_2$ bishops criticized$_3$ Klaus's statements about the Church.*

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.*

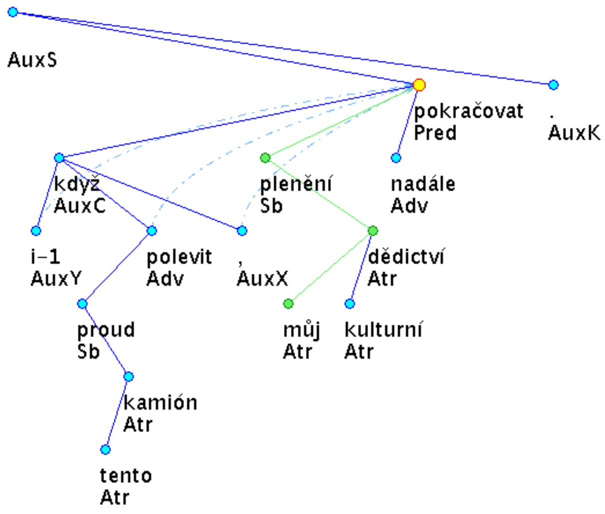
The following query demonstrates the usage of the meta-attribute `_optional` with the value `true`. It searches for Attributes ("`Atr`") anywhere in the subtree of a Predicate ("`Pred`") but does not allow a subordinating conjunction ("`AuxC`") appear on the path from the Predicate to the Attribute:



With the textual version:

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

The following tree is a possible result for this query:



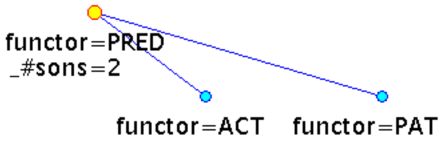
In Czech: *I když proud těchto kamionů polevil, plenění našeho$_1$ kulturního dědictví nadále pokračuje$_2$.*
In English: *Even though the stream of these lorries slackened, the plundering of our$_1$ cultural heritage still continues$_2$.*

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

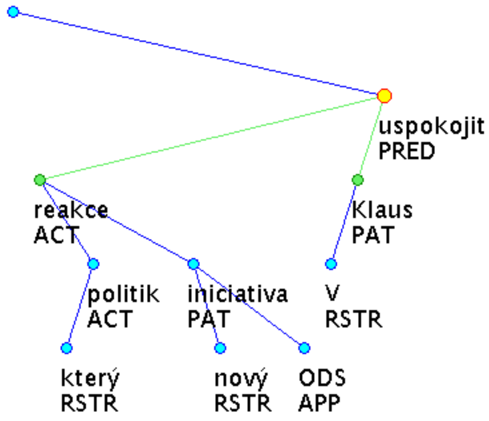
The meta-attribute _#sons ("number of sons") controls the exact number of sons of a node in the result tree. The following query searches for a Predicate governing an Actor and a Patient and nothing else:



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.*

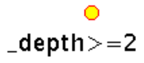
The meta-attribute _#sons prevented the Predicate from having more than two sons in the result tree. The predicate could not have less than two sons in the result also because there were two sons in the query.

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

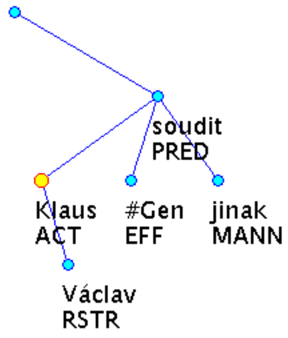
The meta attribute `_depth` controls the distance of a node in the result tree from the root of the result tree. The following query searches for all nodes that are at level 2 or greater – their distance from the root is at least 2:



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:

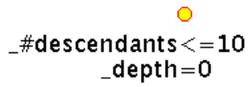


In Czech: *Václav Klaus$_1$ soudí jinak.*
In English: *Václav Klaus$_1$ thinks otherwise.*

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`)):

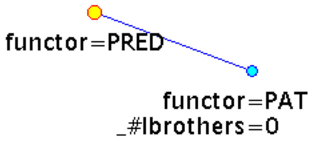


With the textual version:

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

3.8.7 _#lbrothers

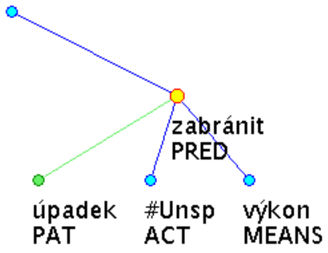
The meta-attribute `_#lbrothers` (“number of left brothers”) controls the exact number of left brothers of a node in the result tree. The following query searches for a Predicate that governs a Patient as its first son:



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.*

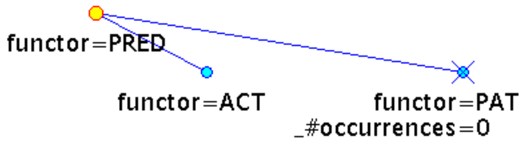
3.8.8 _#rbrothers

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).

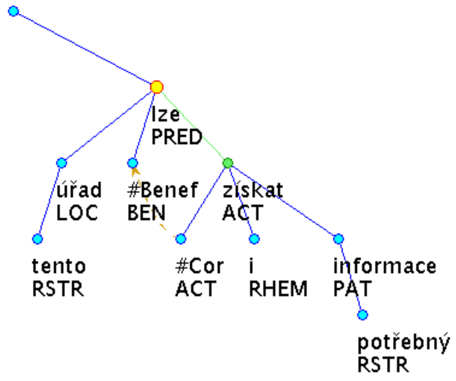
The following query searches for Predicates that govern (directly) an Actor but not a Patient:



With the textual form:

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

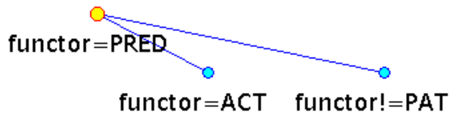
The following tree is a possible result for this query:



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").

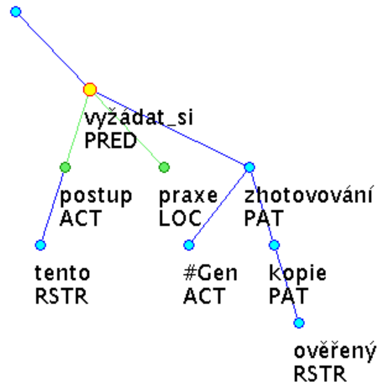
Please note that the following query has quite a different meaning:



With the textual version:

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

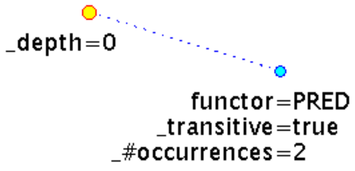
The following tree is a possible result for the query:



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").

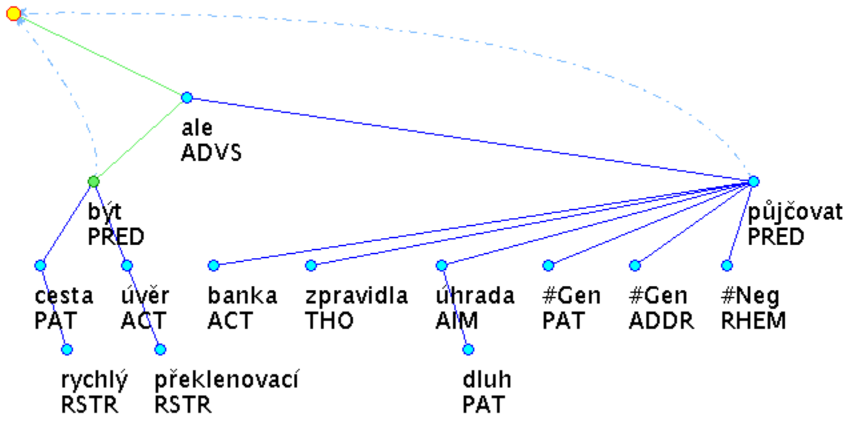
The meta-attribute `_#occurrences` can be combined with the meta-attribute `_transitive` set to the value `true` for the transitive meaning of the occurrences; then, it controls how many nodes of the kind can occur in the whole subtree of the father of the node in the result tree (excluding the father). 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.*

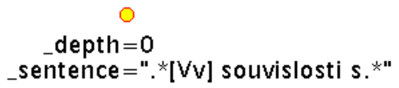
Since only one Predicate is actually drawn in the query, only one is highlighted in the result.

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

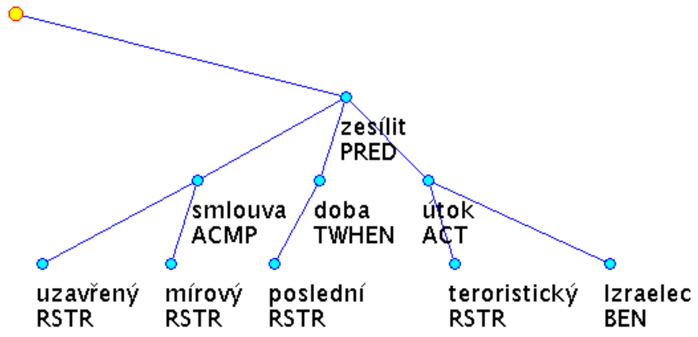
The meta-attribute `_sentence` can be used to search in the linear surface form of the trees – in the sentences. The following query searches for all trees (sentences) that contain the expression “v souvislosti s” (“in connection with”), regardless of its position in the sentence. To avoid matching each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

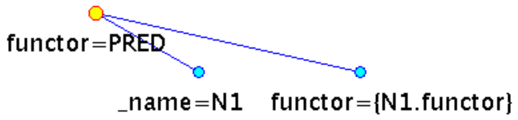


In Czech: *V_souvislosti_s*$_1$ *uzavřenými mírovými smlouvami v poslední době zesílily teroristické útoky proti Izraelcům.*
In English: *In_connection_with*$_1$ *the signed treaties of peace, terrorist attacks towards Israelis recently intensified.*

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:

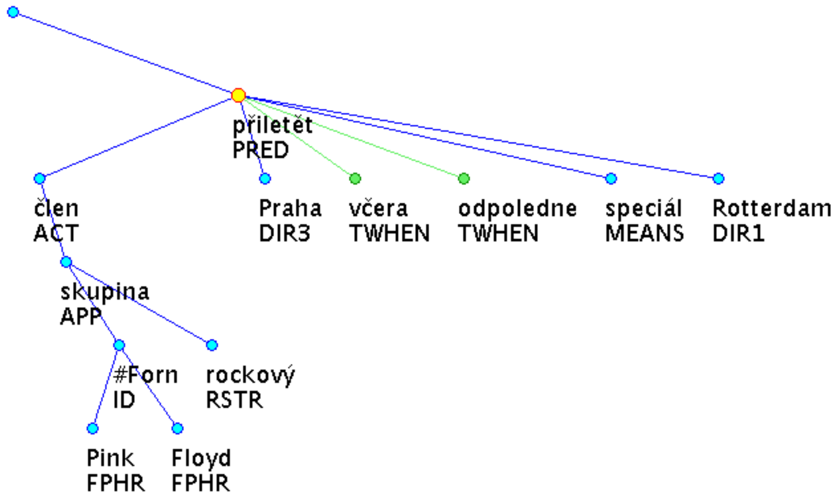


With the textual form:

```
[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`:

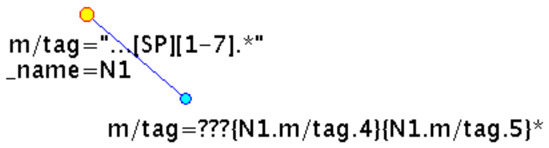


In Czech: *Členové rockové skupiny Pink Floyd přiletěli$_1$ do Prahy včera$_2$ odpoledne$_3$ speciálem z Rotterdamu.*
In English: *Members of the rock group Pink Floyd arrived$_1$ in Prague yesterday$_2$ afternoon$_3$ with a special flight from Rotterdam.*

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`):

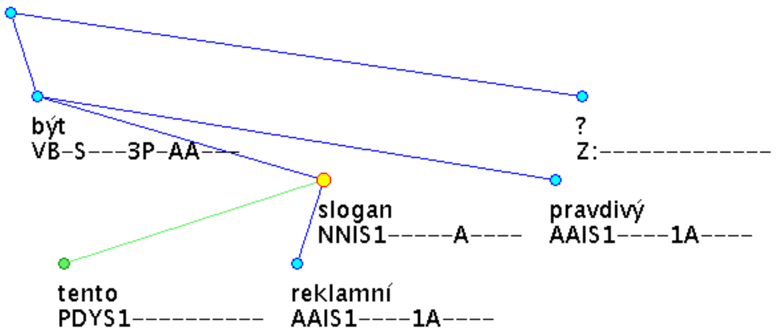


With the textual version:

```
[_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:

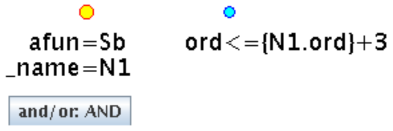


In Czech: *Je tento$_1$ reklamní slogan$_2$ pravdivý?*
In English: *Is this$_1$ advertising slogan$_2$ honest?*

A reference cannot be a part of a regular expression.

3.10 Multi-Tree Queries

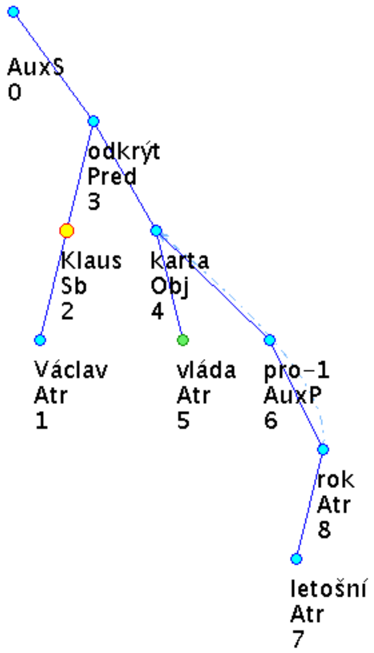
A multi-tree query consists of several trees combined either with a general `AND` or a general `OR`. In the case of `AND`, all the query trees are required to be found in the result tree at the same time (different nodes in the query cannot be matched with one node in the result), while in the case of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

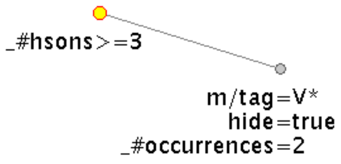
3.11 Hidden Nodes

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".

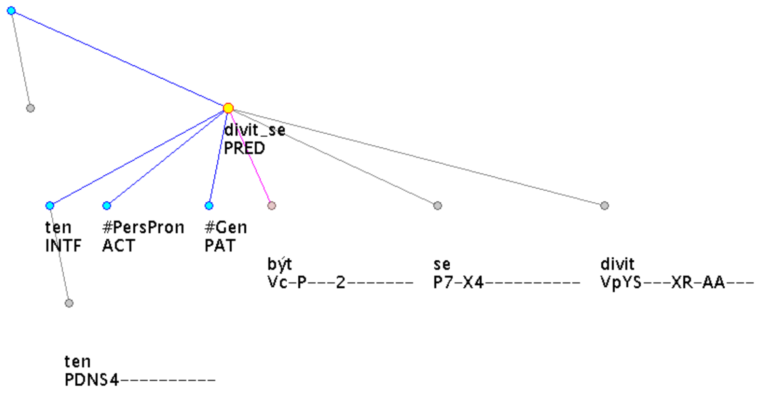
The following query searches for a node that has at least three hidden sons, two of which are verbs (their morphological tag starts with "V"):



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.

4 The Data

Before we proceed to using the proposed query language, we need to describe the data used in the tool that implements the query language, because the language is not independent of the data and has some requirements on the data. We first talk about the file format (Section 4.1), then we mention the definition of corpus-specific features in the header of the files (Section 4.2). Section 4.3 shows that some additional information in the data can help the tool from needless computing. Section 4.4 talks about realization of references in the data and Section 4.5 describes one very corpus-specific property of the data that required an adaptation of the tool. Section 4.6 elaborates hidden nodes – a way of accessing lower layers of annotation in cases with non-1:1 relation among nodes on the layers.

4.1 The Format

Netgraph uses FS File Format for storing the treebank. FS File Format was first used in the tree editor Graph (Křen 1996) during the work on first versions of the Prague Dependency Treebank and was one of two main formats used in the final production of PDT 1.0 (along with CSTS (Hajič et al. 2001a)). By the way, the name "Netgraph" was also inspired by the tree editor Graph.

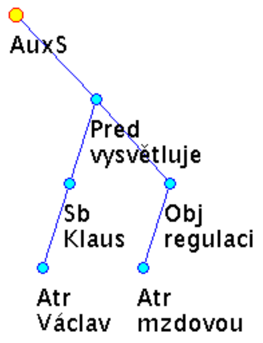
FS format is a very simple text format. It consists of two parts: a header and a set of trees. The header defines attributes and properties of the attributes that are later used in the set of trees. The trees follow the header, each tree is on one line of the file.

A detailed formal description of the format is given in "Appendix B: FS File Format Description". Let us give only a simple (very simplified) example of a header and one tree here:

```
@E UTF-8
@P afun
@L afun|AuxS|Adv|AdvAtr|Apos|Atr|AtrAdv|AtrAtr|AtrObj|Atv|AtvV|AuxC|
AuxG|AuxK|AuxO|AuxP|AuxR|AuxT|AuxV|AuxX|AuxY|AuxZ|Coord|ExD|Obj|ObjA-
tr|Pnom|Pred|Sb
@P ord
@O ord
@N ord
@V w/token

[afun=AuxS,ord=0]([afun=Pred,ord=3,w/token=vysvětluje]
([afun=Sb,ord=2,w/token=Klaus]([afun=Atr,ord=1,w/token=Václav]),
[afun=Obj,ord=5,w/token=regulaci]([afun=Atr,ord=4,w/token=mzdovou])))
```

It is a representation of the following tree:



In Czech: *Václav Klaus vysvětluje mzdovou regulaci*
In English: *Václav Klaus explains the wage restraint*

All attributes that can be used in the trees are defined in the header, some with all possible values. The second character on each line defines some property of the attribute (e.g. "`O`" = obligatory). In the trees, a node is enclosed in square brackets ("`[`", "`]`"), followed by its subtree in parentheses ("`(`", "`)`"). Brothers are separated by a comma ("`,`"), just as different attributes of one node are. The attribute `ord` is used to control the left-right order of the nodes in the tree (as defined by its property "`N`" in the header). Thus, crossing edges are allowed in the data.

It can be easily seen that FS Query Format is an extension of this format.

The first line in the header says that the file is encoded in UTF-8[10]. Thus, the support for all major languages is ensured and even various languages can co-exist in one file, if required. UTF-8 is the only encoding supported in Netgraph.

There are several reasons why Netgraph uses FS File Format. The main reason is probably historical. The format has been used in Netgraph from the beginning and it has never proved unsuitable. In fact, it is very convenient for the purpose. It can be easily read both by people and programs, is space-saving and programs can read it very quickly. It is also a general format that can be easily adopted to various treebanks. The treebank-related information is stored in the header.

4.2 Corpus-Specific Features in the Header

FS File Format can be used for various treebanks. In the header of each FS file, several important attributes can (some of them should) be defined. The attributes can have arbitrary names, their function is defined by a property in the header:

- Nodes order attribute (property "`N`") – this attribute controls the order of nodes in the tree from left to right; non-negative real numbers are allowed

10 UTF-8 and Unicode Standards: `http://www.utf-8.com/`

- Words order attribute (property `"W"`) – this attribute controls the order of words in the sentence from left to right (if not defined in the header, attribute with the property `"N"` is used); non-negative real values are allowed
- Words value attribute (property `"V"`) – values of this attribute are used to assemble the original sentence (the tokens are ordered according to values of the attribute with property `"W"`)
- Hiding attribute (property `"H"`) – the attribute with this property is used to distinguish hidden nodes

4.3 How Data Can Help

Things that can be pre-computed can be stored in the data so that the tool can be simpler and does not have to waste time. In PDT 2.0, there are several such pre-computed attributes.

The attribute `eparents` keeps an identifiers of a linguistically effective father of each node (but the root)[11]. The algorithm that finds the effective father is quite complex (Štěpánek 2006). Thanks to the pre-computation, Netgraph does not have to implement it.

Attribute `eparents_diff` is another supplemental attribute. It keeps the same information as `eparents` but only if the effective father of a node differs from the technical father of the node. This fact could be determined in the query but this way the information is directly in the data, easily accessible, making some queries simpler.

Another pre-computed attribute in the tectogrammatical trees in Netgraph is the attribute `sentence`[12]. It is only filled-in at the root of each tree and keeps the whole sentence the tree belongs to. The reason for this is that it would be very difficult to assemble the original sentence from the information stored in the tectogrammatical tree, even with the hidden nodes present (see Section 4.6), because there is no representation of punctuation in the data[13].

4.4 References

This section discusses a rather technical feature – how to adapt the tool for references (secondary edges etc.) that are annotated in the data, in order to display them properly.

What references are annotated in the data is closely corpus-related. Even in PDT 2.0, different references are used on the analytical layer and on the tectogrammatical layer.

11 If there are more than one effective father, the single references are kept as alternative values of the attribute.

12 Not the meta-attribute `_sentence`!

13 The situation is different on the analytical layer, where all tokens of the sentence are represented in the tree and the sentence is assembled from values of the attribute `w/token`.

On the analytical layer, there are only two references in the data:

- effective parentage of all nodes (the attribute `eparents`)
- effective parentage of nodes where the effective father differs from the technical father (the attribute `eparents_diff`)

On the tectogrammatical layer, the following references are annotated in the data:

- effective parentage of all nodes (the attribute `eparents`)
- effective parentage of nodes where the effective father differs from the technical father (the attribute `eparents_diff`)
- grammatical coreference (the attribute `coref_gram.rf`)
- textual coreference (the attribute `coref_text.rf`)
- predicative complement (the attribute `compl.rf`)

And at the hidden nodes:

- effective parentage of all hidden nodes (the attribute `a/eparents`)
- effective parentage of all hidden nodes where the effective father differs from their technical father on the analytical layer (the attribute `a/eparents_diff`)

Netgraph can display all these references in the depicted trees. For each corpus, references and the way how to display them can be defined in a special textual file at the server side. A list of the references is created in the client after the connection to the server is established. Then, the user can switch on and off displaying of the individual references.

4.5 Attribute m/lemma

There is a very special way of treating the attribute `m/lemma` implemented in Netgraph. It is a completely PDT 2.0-specific feature of the tool. The attribute `m/lemma` keeps a morphological lemma of a token – a base form of the token. Without elaborating the details, we can say that different words can have the same lemma. The lemmas are then distinguished by a variant, which is often followed by a comment, explaining the nature of the word. Let us give an example. Lemma "stát" represents five different words and has five variants:

- stát-1_^(státní_útvar) ... (in English: state, country)
- stát-2_^(něco_se_přihodilo) ... (in English: to happen)
- stát-3_^(někdo/něco_stojí,_např._na_nohou) ... (in English: to stand (e.g. on feet))
- stát-4_^(něco_stojí_peníze) ... (in English: to cost (money))
- stát-5_^(sníh) ... (in English: to melt out)

Users cannot be supposed to know all variants of all lemmas or even the comments too. Netgraph allows searching for a lemma without specifying the variant or the comment. The expression `m/lemma=stát` searches for all five variants of the lemma.

This behaviour can be switched on and off in the menu. It is nevertheless always possible to specify the particular variant of a lemma in the query, e.g. `m/lemma=stát-2`, to search for that particular meaning of the lemma.

4.6 Hidden Nodes

Hidden nodes were first introduced with the Prague Dependency Treebank 1.0 in the sample of two hundred sentences annotated on the tectogrammatical layer (and all lower layers), as a way of representing information from several layers of annotation with non-1:1 relation among nodes in one tree structure. Each tectogrammatical node with some counterpart on the analytical layer contained additional attributes representing the analytical node with the greatest contribution to the lexical meaning of the tectogrammatical node. All other analytical nodes belonging to the tectogrammatical node appeared as hidden sons of the tectogrammatical node (their attribute `hide` was set to `hide` (yes, really `hide`)).

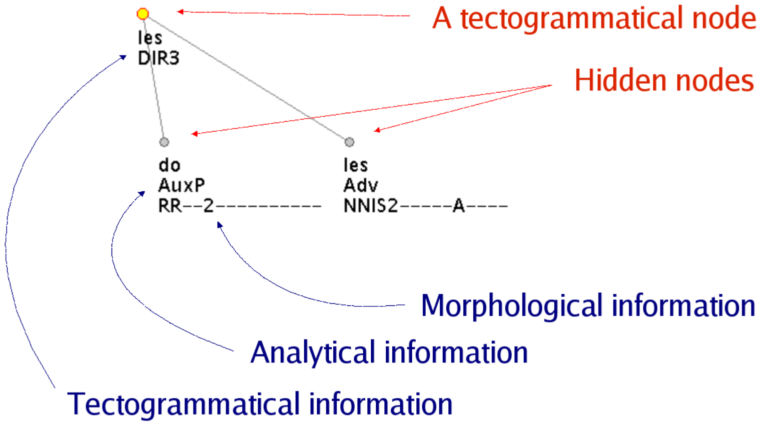
In PDT 2.0, a new data format has been introduced – Prague Mark-up Language (PML, Pajas, Štěpánek 2005). Each layer of annotation is annotated in its own file, the files are interlinked in order to preserve relations between the contents (Pajas, Štěpánek 2006). There are no hidden nodes any more.

Netgraph, on the other hand, presents all the available information in one tree (Mírovský 2006). For this purpose, we decided to use the hidden nodes in a slightly different way. In our approach, 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. (There may be zero, one or several such nodes belonging to one tectogrammatical node.) This way, logically different information is kept at logically different places. Moreover, the search algorithm does not take the hidden nodes into account, unless a node is explicitly specified in the query as hidden. It is therefore no longer necessary that the set of attributes of the hidden nodes differs entirely from the set of the tectogrammatical attributes (although it is still true in the data). Technically, in the data, the hidden nodes are distinguished by the value `true` of the attribute `hide`.

The hidden nodes are not a part of the tectogrammatical layer, they only provide a connection to the lower layers. All the nodes from the analytical layer (except for the technical root), both auto-semantic and non-auto-semantic, become the hidden nodes on the tectogrammatical layer in Netgraph. Non-hidden nodes on the tectogrammatical layer do not carry any information from the lower layers. This information is only accessible through the hidden nodes.

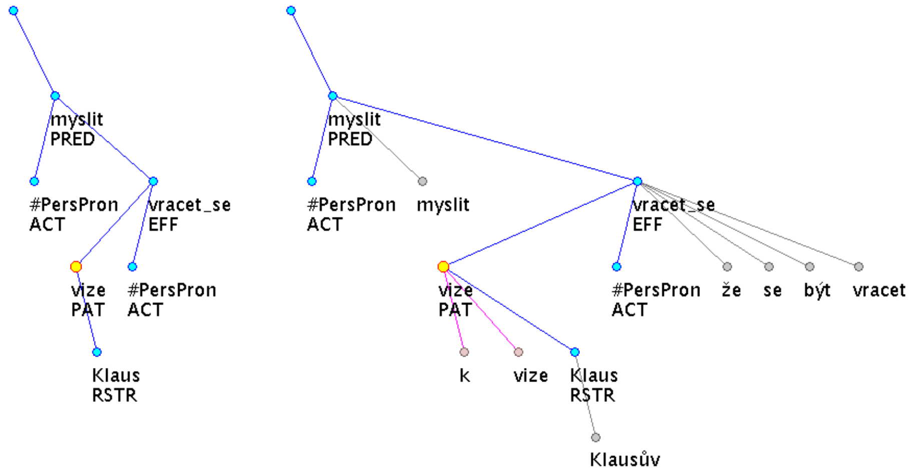
As mentioned above, meta-attributes treat the hidden nodes in accordance with the definition of the hidden nodes. Some meta-attributes do not take them into account at all (like `_#sons`), others are specifically focused on them (`_#hsons`).

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$_1$ Klausově vizi$_2$ se budeme vracet.*
In English: *I think that to$_1$ Klaus`s vision$_2$ we will get back.*

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 PDT 2.0, listed in Section 2.4 at the end of Chapter 2, and how it can be used for searching for all linguistic phenomena from PDT 2.0 listed in the chapter in Section 2.3. (Parts of this chapter were published in Mírovský 2008c.)

5.1 General Requirements

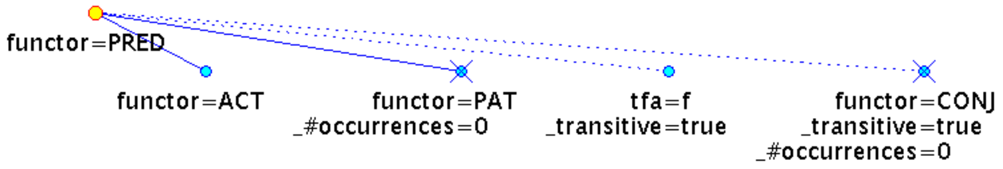
We show that Netgraph Query Language (graphical representation of FS Query Language) fulfils the general requirements on a query language for PDT 2.0, listed at the end of Chapter 2 in Section 2.4.

5.1.1 Complex Evaluation of a Node

It can be directly seen that Netgraph Query Language fits the requirements for the complex evaluation of a single node. The definition of the language from Chapter 3 follows almost exactly the points of the complex evaluation.

5.1.2 Dependencies Between Nodes (Vertical Relations)

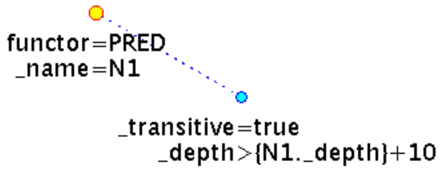
The positive immediate dependency can be directly specified in the query, since the language can directly form a tree structure. The positive transitive dependency can be specified using the meta-attribute `_transitive`. Both cases (immediate and transitive) can be used in the negative sense with a help of the meta-attribute `_#occurrences`, set to zero. All these types of dependency appear in the following example:



The first two sons represent the positive and negative immediate dependency, the third and the fourth sons represent the positive and the negative transitive dependency. The query searches for Predicates governing directly an Actor, not governing directly a Patient, governing transitively a node in focus, and not governing transitively any Conjunction.

Please note that two positively defined nodes in the query cannot be merged into one matching result node. Therefore, if the Actor (the first son) was the only transitive descendant of the Predicate in focus, the third son (and therefore the whole query) would not match.

The vertical distance from the root in the result tree can be simply defined with the meta-attribute `_depth`. The vertical distance between two nodes can be defined with the meta-attribute `_depth` and references, like in the following example that searches for a node transitively dependent on a Predicate, at the vertical distance from the Predicate greater than 10:

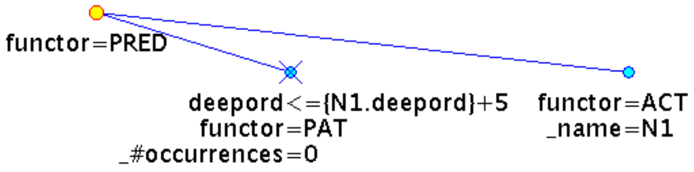


Number of sons of a node in the result tree can be directly controlled with the meta-attribute `_#sons`.

5.1.3 Horizontal Relations

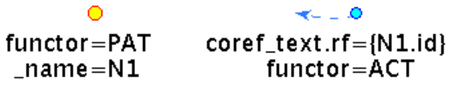
The precedence and immediate precedence, as well as the horizontal distance, all in the positive and negative senses, can be specified using references to an attribute controlling the horizontal order of nodes in the tree, which has to be present in the data.

Let us give only one example to demonstrate the definition of such a query. The following query searches for a Predicate governing an Actor. It also states that if there is a Patient, it must be on the right side from the Actor and at least at distance 6. The heuristic algorithm ordering nodes in the graphical form of the query may have chosen rather unintuitive ordering here (at least at the first sight):



All secondary edges, secondary dependencies, coreferences, and other long-range relations can be expressed using references. Each type of the long-range relations requires a dedicated attribute in the data, containing an identifier of the target node. Therefore, a unique identifier of nodes is also required. It can be common for all purposes.

The following query serves as an example of queries with a secondary edge. It searches for an Actor with the textual coreferential relation to a Patient. Both the Actor and the Patient can appear anywhere in the result tree.



The logical expression AND is used in the query.

5.1.4 Other Features

FS Query Language supports multi-tree queries combined either with the logical AND or the logical OR. This simple combination seems to be sufficient for required purposes.

The meta-attribute `_optional` servers for skipping node(s) of a given type. Its usage has been demonstrated in Chapter 3.

Access to lower layers of annotation with non-1:1 relation among nodes is achieved with the hidden nodes. Their description has been given in Chapters 3 and 4.

The meta-attribute `_sentence` can be directly used for searching in the linear surface form of the sentences.

5.2 Using the Query Language for Searching in PDT 2.0

We show that (and how) the linguistic phenomena described in Chapter 2 in Section 2.3 can be searched for using Netgraph Query Language. We list the phenomena again and present representative queries for them.

5.2.1 The Tectogrammatical Layer

Basic Principles

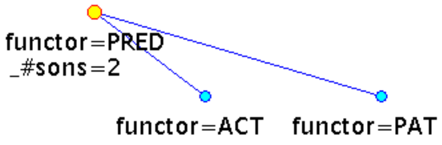
The language should be able to express the node evaluation and the tree dependency among nodes in the most direct way.

We believe that we have shown the capability of the language to express the complex node evaluation and the basic dependencies among nodes in the previous text and will not bother the reader by repeating the same examples again.

Valency

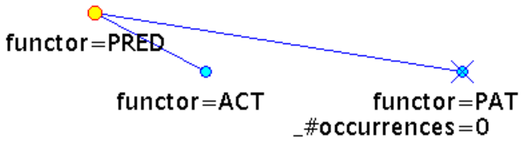
The query language has to be able to distinguish valency frames. The required features include a presence of a son, its absence, as well as controlling number of sons of a node.

Let us show two representative 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.

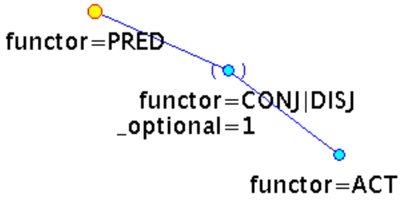
The second query searches for Predicates governing an Actor and not governing a Patient. Since Patient has to be the second inner participant of any valency frame having at least two inner participants (t-manual, page 102), the query searches for occurrences of Predicates with only one inner participant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



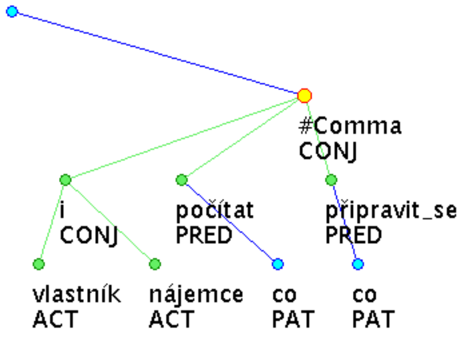
Coordination and Apposition

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

The meta-attribute `_optional` can be used directly to skip a node. Let us only repeat the example given in Chapter 3, searching for a Predicate governing an Actor with an optional coordinating node in between:

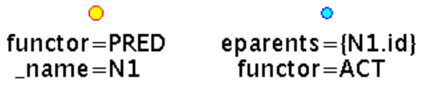


Let us recall the tree where the coordinated structure is more complex and skipping a node does not help. The two Predicates are coordinated with Conjunction, and so the two Actors are. The linguistic dependencies go from each of the Actors to each of the Predicates but the tree dependencies are quite different:



In Czech: *S čím mohou vlastníci*$_1$ *i*$_2$ *nájemci*$_3$ *počítat*$_4$*, na co by_se_měli_připravit*$_5$*?*
In English: *What can owners*$_1$ *and*$_2$ *tenants*$_3$ *expect*$_4$*, what they should_get_ready*$_5$ *for?*

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



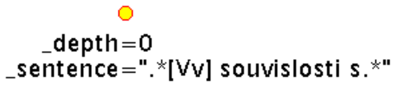
```
functor=PRED     eparents={N1.id}
 _name=N1          functor=ACT
```

The 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 use the attribute `eparents_diff` instead, which keeps identifiers of all effective linguistic fathers of a node only if they differ from its technical father.

Idioms (Phrasemes) etc.

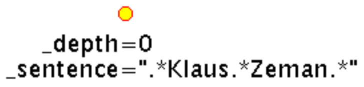
Some idioms/phrasemes and secondary prepositions are linguistic phenomena that can be easily recognized in the surface form of the sentence but may be difficult to find in the tectogrammatical tree. The meta-attribute `_sentence` can be used to search directly in the linear form of the sentences, regardless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

Let us repeat an example query from Chapter 3 and present one more. The first query searches for the phrase “v souvislosti s” (“in connection with”), regardless of the position of the phrase in the sentence. To avoid matching each node in the tree, the meta-attribute `_depth` is added:



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

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



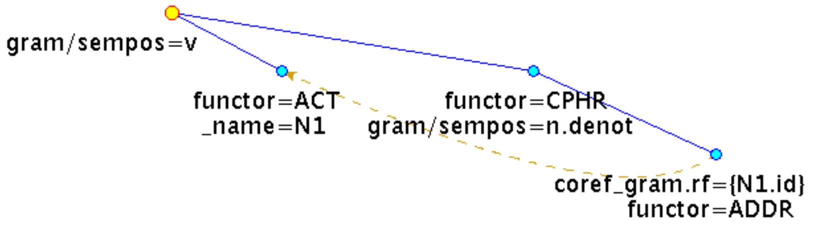
```
   _depth=0
_sentence=".*Klaus.*Zeman.*"
```

Complex Predicates

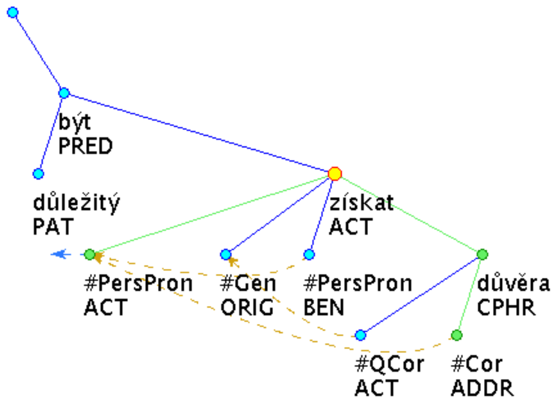
Let us recall that the complex predicate is a multi-word predicate consisting of a semantically empty verb which expresses the grammatical meanings in a sentence, and a noun (frequently denoting an event or a state of affairs) which carries the main lexical meaning of the entire phrase (t-manual, page 345). The `functor` of the nominal part of the complex predicate is assigned the value `CPHR`.

We are interested in cases with dual function of a valency modification 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).

The following query follows the definition of the complex predicate and takes advantage of the fact that the dual function of a valency modification is expressed with the grammatical coreference – the attribute `coref_gram.rf` at a valency member of the nominal part contains an identifier of a valency member of the verbal part of the complex predicate. In this query, we search for those cases where a valency member of the nominal part is an Addressee ("`ADDR`") and refers to a valency member of the verbal part that is an Actor ("`ACT`"):



The following tree is a possible result for the query:

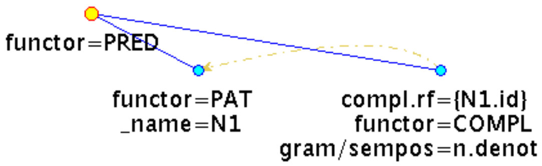


In Czech: *Důležité je, aby si získala$_1$ důvěru$_2$.*
In English: *It is important that she$_0$ gains$_1$ confidence$_2$.*

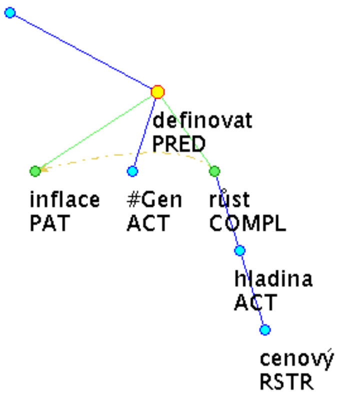
Predicative Complement (Dual Dependency)

The predicative complement is a non-obligatory free modification (adjunct) which has a dual semantic dependency relation. It simultaneously modifies a noun and a verb (which can be nominalized). The second dependency (the dependency on the (semantic) noun) is represented by means of the attribute `compl.rf`, the value of which is the identifier of the modified noun (t-manual, page 376).

The query uses references, just like in the previous subsection with complex predicates. This time, the referential information is stored in the attribute `compl.rf`. The query searches for those cases of the predicative complement where the second dependency goes to a Patient:



And the following tree is a possible result for the query:

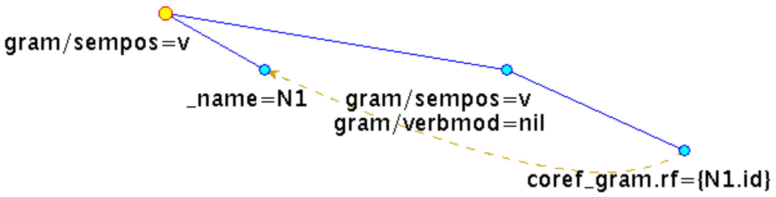


In Czech: *Inflace$_1$ je_definována$_2$ jako_růst$_3$ cenové hladiny.*
In English: *Inflation$_1$ is_defined$_2$ as_an_increase$_3$ of the prices level.*

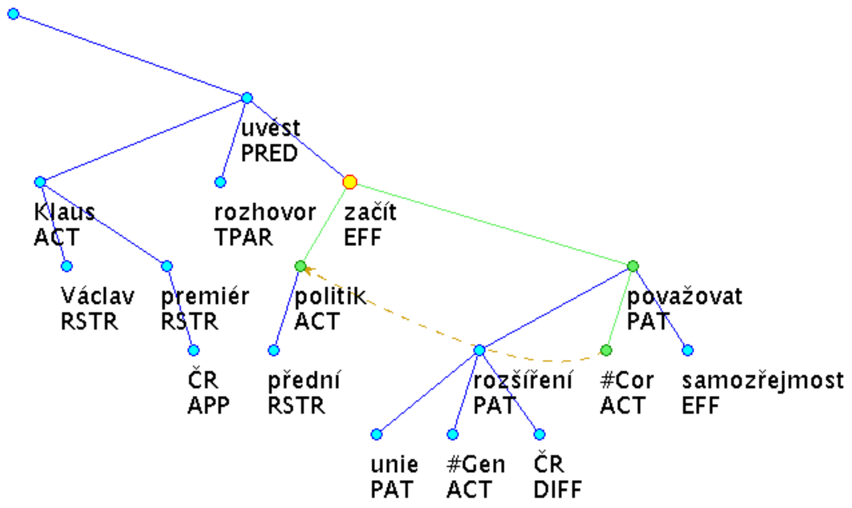
Coreferences

There are two types of coreferences annotated on the tectogrammatical layer – the grammatical coreference and the textual coreference. Like other long-range dependencies, they are annotated using referential attributes, the grammatical coreference uses the attribute `coref_gram.rf`, and the textual coreference uses the attribute `coref_text.rf`.

Let us give one representative example, searching for type-1 control constructions, which is a type of grammatical coreference where an infinitive depends on a control verb; this time, we do not set any other condition on the nodes:



And a result tree:

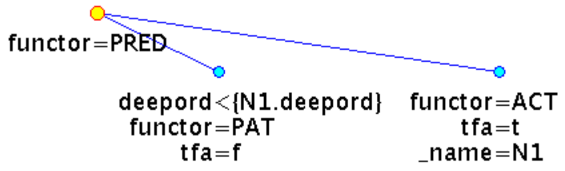


In Czech: *Přední politici$_1$ začali$_2$ rozšíření unie o ČR považovat$_3$ za samozřejmost, uvedl během rozhovorů premiér ČR Václav Klaus.*
In English: *Prominent politicians$_1$ started$_2$ to_take$_3$ the extension of the union for granted, the prime minister of ČR Václav Klaus pointed out during the discussions.*

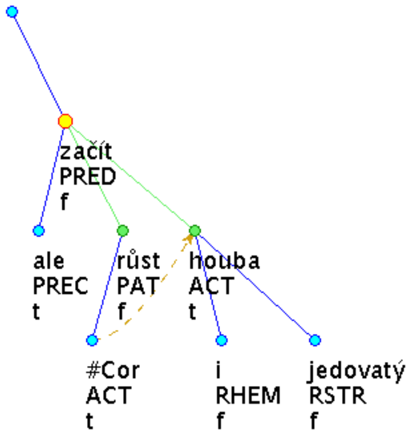
Topic-Focus Articulation

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. Therefore, we will again need to refer to a value of an attribute of another node with relation other than “equal to”.

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:

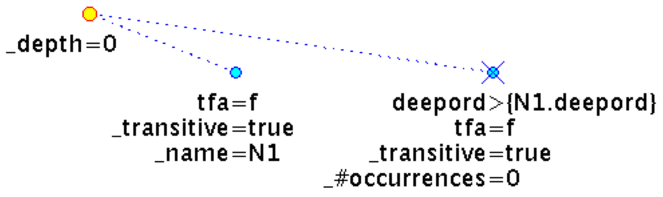


In Czech: *Začaly$_1$ ale růst$_2$ i houby$_3$ jedovaté.*
In English: *But also poisonous mushrooms$_3$ started$_1$ to_grow$_2$.*

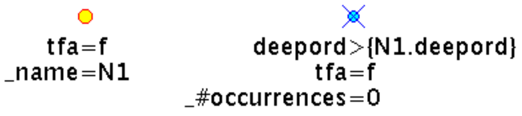
Focus Proper

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 at the surface structure. The node representing this expression is placed rightmost in the tectogrammatical tree.

The following query searches for the focus proper:

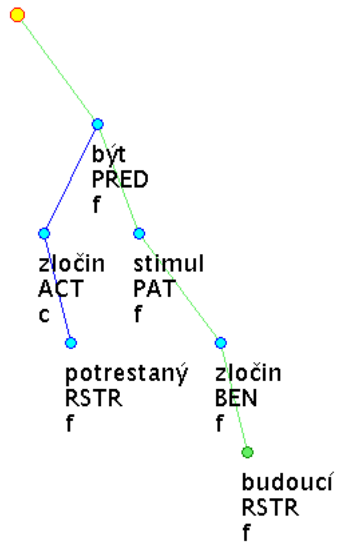


The same query can be expressed with a multi-tree query with the logical expression AND:



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$_1$.*
In English: *An unpunished crime is a stimulant for future$_1$ crimes.*

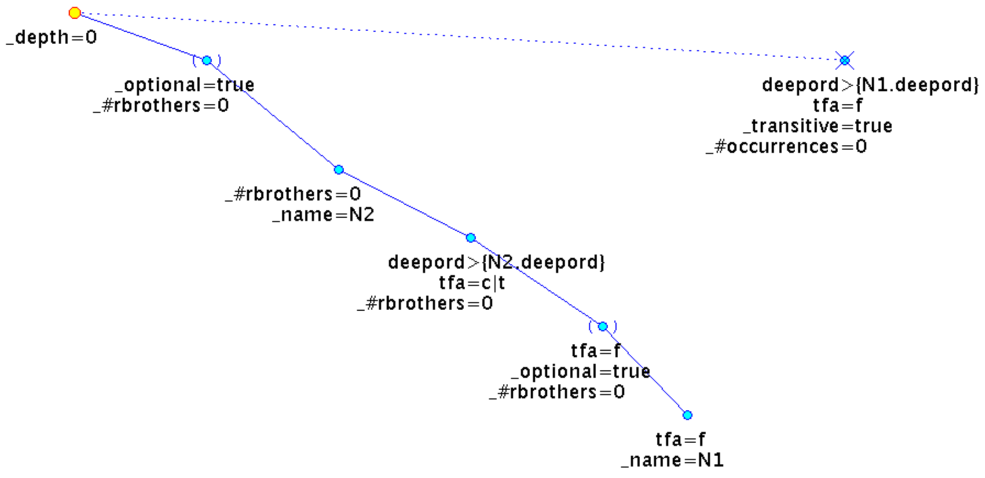
Quasi-Focus

Quasi-focus is constituted by (both contrastive and non-contrastive) contextually bound expressions, 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.

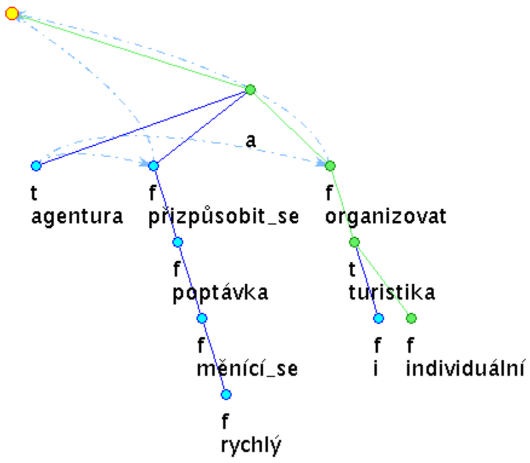
In the underlying word order, nodes representing the quasi-focus, although they are contextually bound, are placed to the right from their governing node. Nodes representing the quasi-focus are therefore contextually bound nodes on the rightmost path in the tectogrammatical tree (t-manual, page 1130).

The query searching for the quasi focus is one of the most complex queries we present, and yet, it follows the definition of the quasi focus, which is quite complex itself.

The first node under the technical root represents nodes on the rightmost path (`_#rbrothers=0`) that lie above the quasi focus. The node named N2 represents the node lying immediately above the quasi focus. Its son is the quasi focus (it is on the right side from its father and has no right brothers, it also is in topic or contrastive topic). The optional son of the quasi focus is defined as a part of the focus and represents the continuation of the rightmost path, that should all be in focus, until the focus proper is reached (named N1). The transitive son of the root makes sure that the node N1 really is the focus proper:



The following tree represents a possible result for the query:



In Czech: *Agentura se přizpůsobila rychle se měnící poptávce a organizuje i turistiku$_1$ individuální.*
In English: *The agency has adapted to the quickly evolving demand and organizes also individual tourism$_1$.*

Although all nodes on the rightmost path from the root are highlighted as matching nodes and therefore the quasi focus must be identified by values of attribute `tfa` of the nodes, the important thing is that the quasi focus can be identified in the query and additional conditions can be set on it.[14]

14 The tectogrammatical manual states that the quasi focus can consist of more than one node (t-manual, page 1131). The query we have presented searches for its most dynamic node.

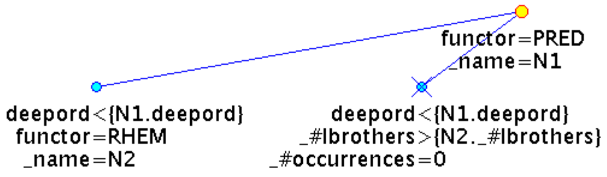
Rhematizers

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 the contrastive topic.

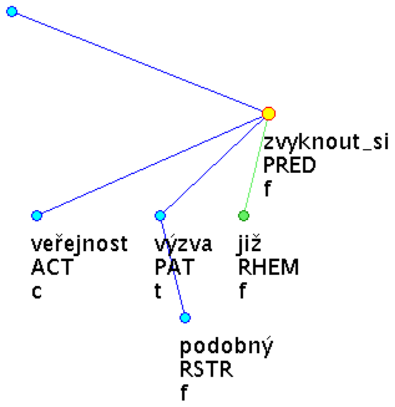
There are two cases of rhematizers that we need to distinguish:

- The 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; the governing predicate is not in the scope of the rhematizer.
- If the scope of the 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 right-most left son of the Predicate:



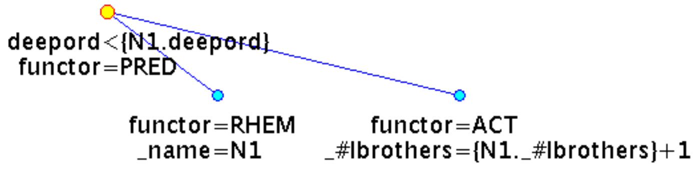
The query defines that there is not a node that lies to the left from the Predicate and to the right from the rhematizer. Since we cannot set two different conditions with two different relations on one attribute, we have to use the meta-attribute `_#lbrothers` to define that the undesired node is on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



In Czech: *Veřejnost si na podobné výzvy již$_1$ zvykla$_2$.*

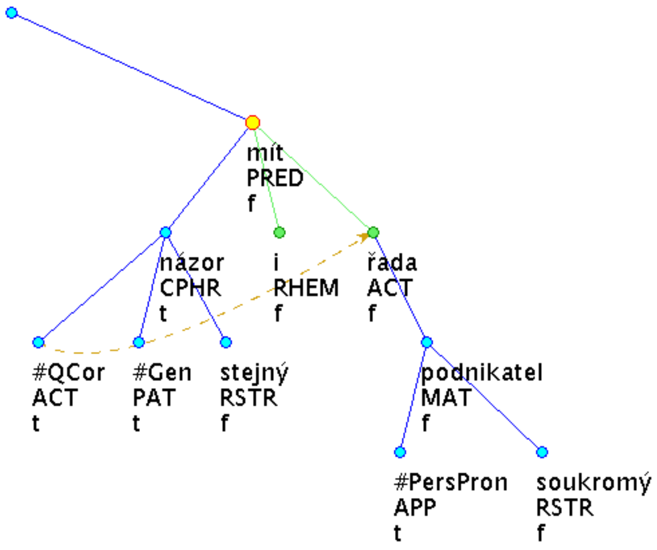
In English: *The public has already$_1$ got_accustomed$_2$ to such calls.*

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:

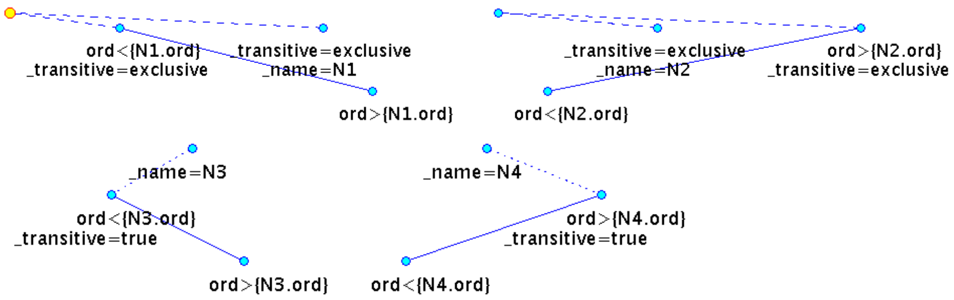


In Czech: *Stejný názor má$_1$ i$_2$ řada$_3$ našich soukromých podnikatelů.*
In English: *Also$_2$ a_number$_3$ of our private investors have$_1$ the same opinion.*

(Non-)Projectivity

Let us recall a simple definition of projectivity of a tree: between a father and its son (in the left-right order) there can only be direct or indirect sons of the father (t-manual, page 1135). We present a query that searches for non-projective trees. It consists of four trees (combined with the logical expression OR). Each tree represents one of the four possible configurations of nodes causing the non-projectivity. Since the (non-)projectivity is much more important, interesting and frequent on the analytical layer than on the tectogrammatical layer, the query searches on the analytical layer (and therefore uses the attribute `ord`, which controls the order of nodes on the analytical layer).

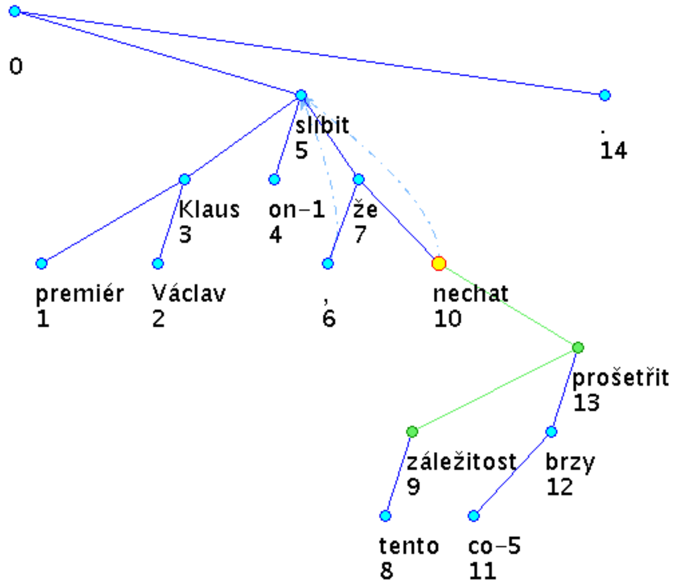
The query is too wide to fit the page, therefore it is split into two rows:



The former two trees represent non-projective configurations where the node proving the non-projectivity is not on the path from the non-projective edge to the root of the tree. The latter two trees represent non-projective configurations where it is on the path. The exclusive transitivity is used to make sure that node N1 (or N2) cannot appear in the subtree of the non-transitive edge (as it might if the true transitivity was used; then, the edge might be projective).

Note: If we used the attribute `deepord` instead of `ord`, we might use the same query on the tectogrammatical layer.

The following tree is a possible result for the query; attributes `m/lemma` and `ord` are displayed:

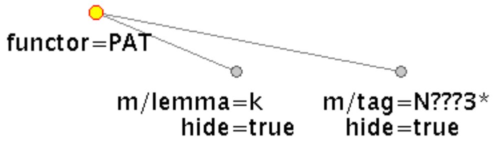


In Czech: *Premiér Václav Klaus mu slíbil, že tuto záležitost*$_1$ *nechá*$_2$ *co nejdříve prošetřit*$_3$.
In English: *The prime minister Václav Klaus has promised him that he will_have*$_2$ *the affair*$_1$ *investigated*$_3$.

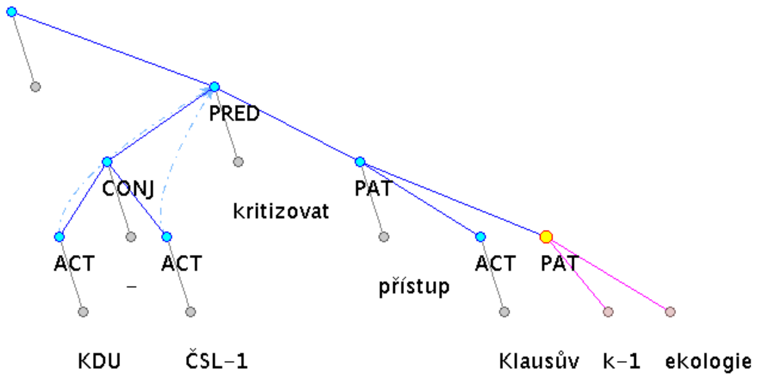
5.2.2 Accessing Lower Layers

Let us present three examples of queries that access the lower layers of annotation from the tectogrammatical layer.

The first query searches for Patients (on the tectogrammatical layer) that are expressed with a preposition “k” and a noun in the dative on the morphological layer:

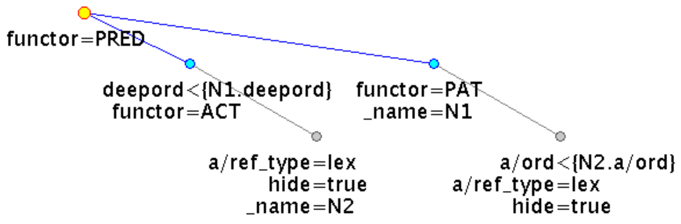


The Patient has (at least) two hidden sons, the former with lemma “k”, the latter with a morphological tag that states that the node is a noun in the dative. The following tree is a possible result for the query. Both tectogrammatical and hidden nodes are displayed. The attribute `functor` is displayed at the tectogrammatical nodes, the attribute `m/lemma` is displayed at the hidden nodes. For saving space, the attribute `m/tag` is not displayed (the node with lemma “ekologie” has the morphological tag “NNFS3-----A----”):

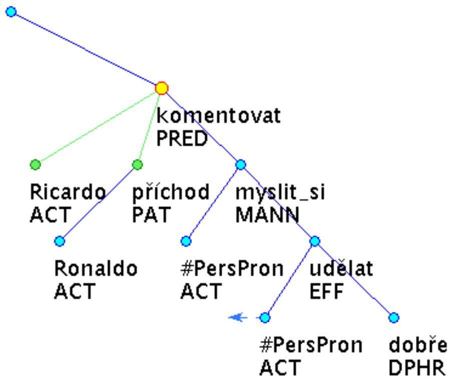


In Czech: *KDU-ČSL kritizuje Klausův přístup k*$_1$ *ekologii*$_2$.
In English: *KDU-ČSL criticizes Klaus’s attitude towards*$_1$ *ecology*$_2$.

The second query searches for an Actor less dynamic than a Patient (on the left side from it in the tectogrammatical tree), but with the opposite order of respective lexical nodes on the analytical layer (and therefore also on the surface – in the sentence):

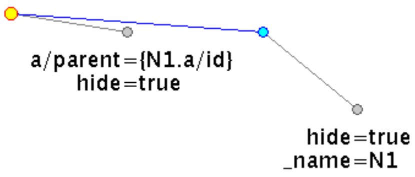


The attribute `a/ref_type` set to the value `lex` makes sure that the two hidden nodes represent the lexical counterparts of the Actor and the Patient. References to the attributes `deepord` and `a/ord` ensure the required order of the nodes. The following tree is a possible result for the query; the hidden nodes are not displayed:

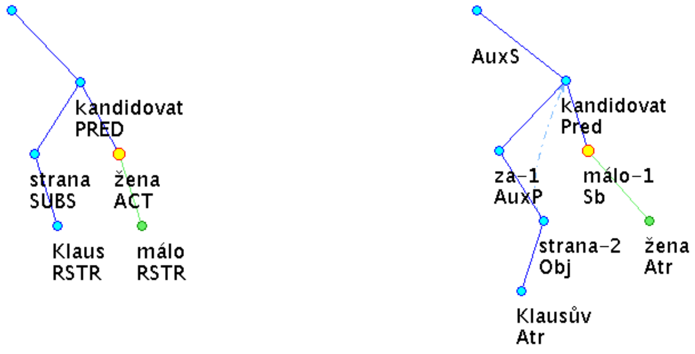


In Czech: *Myslím si, že udělal dobře, komentuje příchod$_1$ Ronalda Ricardo$_2$.*
In English: *I think that he did well, Ricardo$_2$ says about Ronald's coming$_1$.*

The third query shows how to study a difference in the structure of the tectogrammatical and the analytical tree. It searches for the tectogrammatical father and son whose lexical counterparts on the analytical layer have the opposite dependency relation:



The attribute `a/parent` keeps the identifier of the father of the node on the analytical layer. The following tree is a possible result for the query:



In Czech: *Za Klausovu stranu kandiduje málo$_1$ žen$_2$.*
In English: *Not_enough$_1$ women$_2$ candidate for Klaus's party.*

To show the difference in the structure, the respective analytical tree is displayed as well (it was found with another query on the analytical layer). The attributes `t_lemma` and `functor` are displayed in the tectogrammatical tree, the attributes `m/lemma` and `afun` are displayed in the analytical tree:

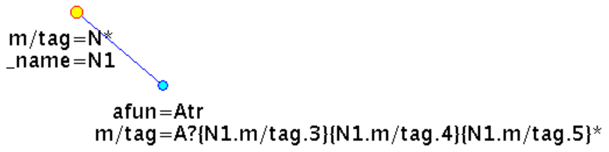
5.2.3 The Analytical Layer

Morphological Tags

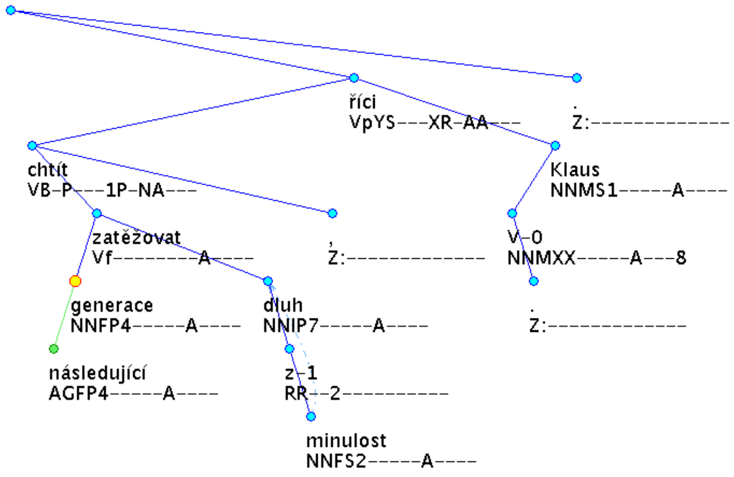
Regular expressions are a powerful tool and allow complex searching for an underspecified morphological tag. Such an example query has been given in Section 3.4.

Agreement

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. Since regular expressions cannot contain references, we have to use the old-style wild cards. The following query searches for a noun (`m/tag=N*`) with an attributive adjective (a dependent adjective that agrees with the noun in gender, number and case, which are at the third, fourth and fifth position of the morphological tag):



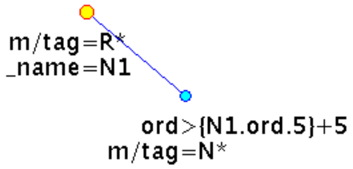
The following tree is a possible result for the query. The attributes `m/lemma` and `m/tag` are displayed:



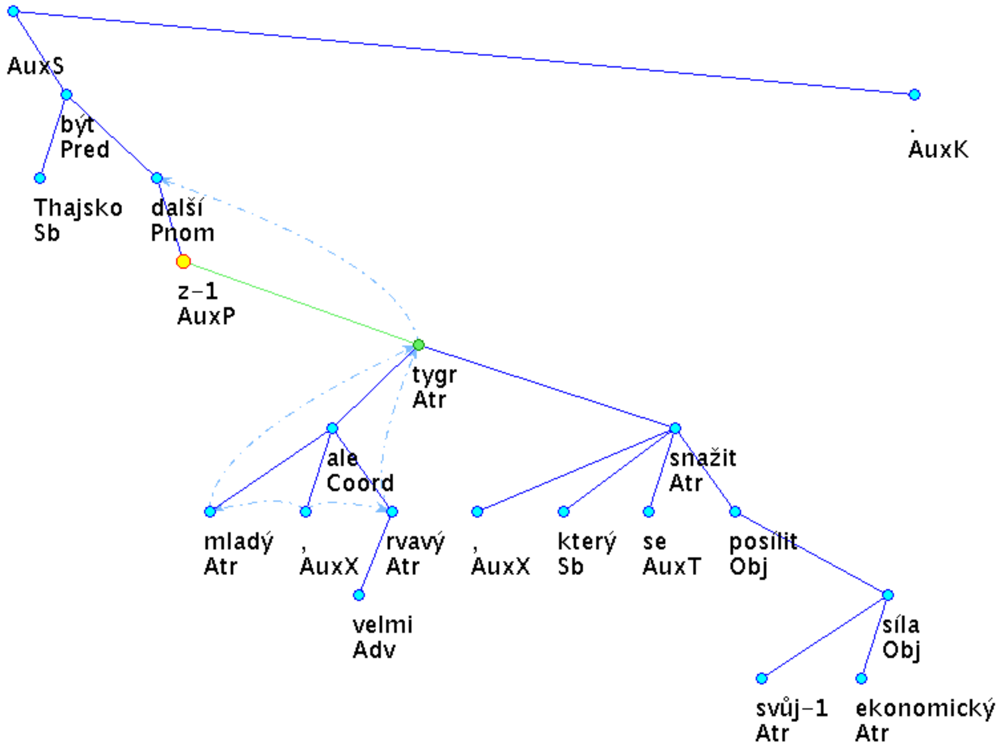
In Czech: *Nechceme následující*$_1$ *generace*$_2$ *zatěžovat dluhy z minulosti, řekl V. Klaus.*
In English: *We do not want to burden next*$_1$ *generations*$_2$ *with debts from the past, said V. Klaus.*

Word Order

The only new requirement on a query language that studies of word order on the analytical layer bring is an ability to measure the horizontal distance between words. The following query searches for trees where a preposition and a noun head of the prepositional phrase are at least five words apart:



The following tree is a possible result for the query; the attributes `m/lemma` and `afun` are displayed:



In Czech: *Thajsko je dalším z$_1$ mladých, ale velmi rvavých tygrů$_2$, kteří se snaží posílit svoji ekonomickou sílu.*
In English: *Thailand is another one of$_1$ young but very combative tigers$_2$ that try to strengthen their economic power.*

6 Notes on the Query Language

6.1 Netgraph Query Language vs. FS Query Language

Netgraph Query Language is a graphical representation of the textual FS Query Language. They are equivalent, every query in the textual form has its graphical counterpart and vice versa. Therefore, we sometimes mix these terms in the text.

6.2 Trees Only

The syntax of some search languages allows defining queries that are not trees – queries that contain a cycle, although their primary purpose may be to search in a corpus of trees where no cycle can occur. For example, it is very easy to make a cycle in TGrep2:

```
VP=v << (NP <<  =v)
```

The query says that a `VP` (named `v` for a later reference) dominates (transitively) an `NP` that in turn dominates (transitively) the same `VP` (referred to as `v`).

Even in much simpler TGrep, where no cycle can be defined, a nonsensical construction is easily created:

```
VP < (NP > NP)
```

The query says that a `VP` immediately dominates an `NP`, which is immediately dominated by another `NP`. But obviously, we do not want a node to have two fathers.

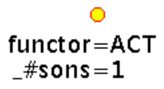
A query in Netgraph Query Language is also supposed to be a tree (or a multi-tree). An important property of the syntax of this query language is that the syntax itself does not allow to create any other structure than trees. It is a simple way how to avoid needless mistakes.

Please note that only the primary dependency structure has to be a tree in Netgraph Query Language. Secondary edges and all other "secondary/long-range relations" are expressible using the references (Section 3.9).

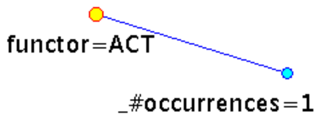
6.3 Redundancy

It can be easily shown that the features of the presented query language are redundant. It means that there are often several ways of creating a certain query. In other words, there are often several queries that do the same thing – search for the same trees (generally, regardless of the corpus) – using different features of the query language.

Let us give a simple example. The following two queries both search for Actors that have exactly one son (of any kind). The first query uses the meta-attribute `_#sons`:

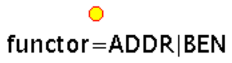


The second query uses the meta-attribute `_#occurrences` at a node without any specification:

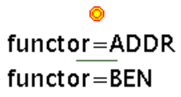


Both the queries find exactly the same trees and the same occurrences in the trees (see Section 6.4 below about a difference between result trees and result occurrences).

Even one node queries that do not use any meta-attribute can show the redundancy of the query language. The following two queries are quite equivalent, both of them search for a node that is either an Addressee or a Benefactor. The first query uses alternative values of an attribute:



The second query uses an alternative node:



The redundancy in the query language can be (and has been) used for testing the tool. If there are two or more different queries that should theoretically find the same number of result trees (or result occurrences), it can be easily checked if they really do so.

6.3.1 Two Types of Redundancy

There are two primary reasons for adding features to the query language, possibly causing two types of redundancy:

- simplification of the query language – a feature is added that does not increase the power of the query language but simplifies some queries; it can be completely substituted by a combination of other features
- increasing the power of the query language – a feature is added that increases the power of the query language; nevertheless, it is often the case that some particular queries using this feature can be substituted by a combination of other features

Both reasons for adding features have been exercised during the development of Netgraph Query Language, although the second reason has been much more frequent.

There are three features worth noticing that simplify the queries and do not increase the power of the query language:

- alternative values of an attribute – it is always possible to express alternative values of an attribute using alternative nodes. Nevertheless, it is much simpler to use three alternative values for one attribute and three alternative values for another attribute instead of nine combinations of these values if we could only use alternative nodes.
- multi-tree queries with trees combined with logical AND – this type of multi-tree queries can be expressed with one-tree queries with the transitive dependency on the root (provided that there is always a technical root that we are not interested in in the queries). For example, searching for two nodes without a specified relation between them can be accomplished with two transitive sons of the root or with a multi-tree query with relation AND.
- exclusive transitivity – as demonstrated later in Section 6.7, the exclusive transitivity can be substituted by a much more complex expression using only the true transitivity. After new values to the meta-attribute `_optional` were added to the query language, it is also possible to use expressions with the meta-attribute `_optional=true` to substitute the exclusive transitivity, yet the exclusive transitivity is still simpler.

The other features that have been added to the query language increase its power. Non of these features can be removed from the language without weakening it. But of course, in some cases several different queries can search for the same thing.

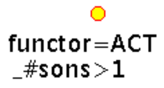
6.4 Result Trees and Result Occurrences

A query can match a result tree more than once, at different places in the result tree or with different configurations of the nodes. We call each configuration of the nodes of the query in the result tree an occurrence of the query in the result tree, or shortly an occurrence[15].

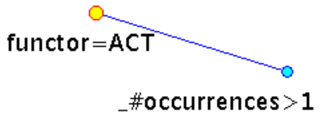
The following three queries are equivalent in the sense that they find exactly the same result trees, but they each match different times – the numbers of occurrences the queries match are different.

15 The term ”occurrence” used in this sense should not be confused with the meta-attribute `_#occurrences` (number of occurrences).

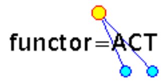
The queries search for Actors that have at least two sons. The first query uses the meta-attribute `_#sons`:



This query matches only once for each Actor with at least two sons. The second query uses the meta-attribute `_#occurrences`:



This query matches for each Actor with at least two sons as many times as how many sons the Actor has. The third query uses two son nodes without any specification. It defines that the Actor has two sons but it does not specify their order:



This query matches for each Actor with at least two sons as many times as how many combinations of matching two query-sons with the result-sons of the Actor there are.

6.5 Comparison to Other Treebank Query Systems

Since FS Query Language (Netgraph Query Language) belongs neither to path based query languages nor to logic based query languages, which are well understood, it may be difficult to assess its exact expressive power.

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, TGrep2, TigerSearch, and fsq (see Section "2.1.2 - Existing Search Tools").[16]

6.5.1 A Biased Table

Let us first offer a table showing to what extent the five tools (Netgraph and the other four tools) fulfil the requirements stated in Section "2.4 - Linguistic Requirements". 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 un-

16 We were unable to find information about Viqtoria sufficient to include this tool into the comparison. The development of Oraculum has long ago been discontinued and TrEd is not meant as a tool for searching.

biased comparison of the expressive power of Netgraph Query Language and the query languages of TGrep, TGrep2 and TigerSearch follows in the subsequent subsections.[17]

complex evaluation of a node	*TGrep*	*TGrep2*	*TigerSearch*	*fsq*	*Netgraph*
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)	+	+	+	+	+[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 less than (<) , greater 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 at the same time with non-1:1 relation (for studying relation between layers)	N/A	N/A	N/A	N/A	+
searching in the surface form of the sentence	+[XV]	+[XV]	+[XV]	+	+

17 A detailed comparison to fsq could not be written since the available user manual for fsq is not detailed enough (`http://tcl.sfs.uni-tuebingen.de/fsq/fsq-userman.pdf`)

These marks have been used in the table:

+ ... 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

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 but multiple times at a node.

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.

6.5.2 Comparison to TGrep

As we presented in Mírovský (2008a), all predicates of TGrep can be translated to FS Query Language. Let us show only a few examples of the translation here. We use the textual version of the translated queries[18]; labels A and B stand for any evaluation of the node possible in Tgrep:

"A immediately dominates B":

In TGrep: `A < B`
In Netgraph: `[A]([B])`

18 The graphical version would have to be faked, because in the graphical interface of Netgraph, a node cannot be marked only with label A or B. Therefore, the translations of the queries cannot be directly copied to Netgraph. The labels A and B would have to be replaced by concrete evaluations of the nodes.

“B is the X-th son of A”:

In TGrep: `A <X B`
In Netgraph: `[A]([B,_#lbrothers=X-1])`

“A dominates B”:

In TGrep: `A << B`
In Netgraph: `[A]([B,_transitive=true])`

“B is the leftmost (rightmost) descendant of A:

In TGrep: `A <<, B`
In Netgraph: `[A]([B,_transitive=true,_name=N1],[_transi-tive=true,ord<{N1.ord},_#occurrences=0])`

B is a transitive descendant of A and there is no transitive descendant of A that has smaller ord than B. The rightmost descendant is similar (`ord>{N1.ord}`).

And a few translations of negative predicates:

“A does not immediately dominate B”:

In TGrep: `A !< B`
In Netgraph: `[A]([B,_#occurrences=0])`

“A does not dominate B”:

In TGrep: `A !<< B`
In Netgraph: `[A]([B,_transitive=true,_#occurrences=0])`

“B is not the X-th son of A”:

In TGrep: `A !<X B`
In Netgraph: `A([B,_#lbrothers!=X-1])`

But note that it also means that B is a son of A. Using the meta-attribute `_#occur-rences` again, we may have another try on this example with a different meaning:

In Netgraph: `[A]([B,_#lbrothers=X-1,_#occurrences=0])`

Here, B still may be a son of A, but not necessarily, and in any case not the X-th one.

This way, all TGrep predicates, as they are listed in the TGrep manual (Pito 1994), can be translated to FS Query Language, as we presented in the cited paper (Mírovský 2008a). It was not shown, however, whether any combination of the predicates in

TGrep can also be translated. It is possible that there might be a combination of negative TGrep predicates (whose translation leads to more complex expressions in FS Query Language) that cannot be translated. Nevertheless, we have not found any such combination, partly because TGrep manual does not state clearly the semantics of the single negative predicates and does not say anything about the semantics of their combination.

As stated in Section 6.2, TGrep also allows to define constructions where a node has two fathers. Since such constructions are undesirable, it can hardly be considered a disadvantage that Netgraph cannot create them.

On the other hand, there is no difficulty in finding a query in Netgraph that cannot be translated to TGrep, as was also shown in Mírovský (2008a). Let us put aside the fact that TGrep is a one attribute searcher (it is designed for treebanks where every node of the trees has only one attribute with one value) and let us focus on the structure of trees. Since TGrep always searches for one pattern only, it cannot reproduce multi-tree queries from Netgraph, combined with the expression OR. The meta-attribute `_optional` also represents a type of an OR-expression on the tree structure and even the following simple example cannot be reproduced in TGrep:

```
[A]([B,_optional=1]([C]))
```

Therefore, we can conclude that (at least in most aspects) FS Query Language is more powerful than the query language of TGrep.

6.5.3 Comparison to TGrep2

TGrep2 brings several new predicates in comparison with TGrep. Most of them can be translated to Netgraph, one cannot:

"B is the only child of A":

In TGrep2: `A <: B`
In Netgraph: `[A,_#sons=1]([B])`

"There is a single path of descent from A and B is on it":

In TGrep2: `A <<: B`
In Netgraph: `[A,_#sons=1]([_#sons=1,_optional=true]([B]))`

"A has the same name as B":

In TGrep2: `A ~ B`
In Netgraph: independently of the structure of the whole query, this predicate can always be expressed with a reference from node B to node A, referring to the principle attribute, e.g. `[_name=N1]([afun={N1.afun}])`.

In TGrep2, node A immediately precedes node B if the last terminal symbol produced by A immediately precedes the first terminal symbol produced by B. In the following rather complex translation to Netgraph, we assume that values of the attribute `ord` at the leaf nodes are identical to the left-right order of the nodes (which should be true for the constituent-structure trees TGrep2 is designed for):

In TGrep2: `A . B`

In Netgraph: `[_depth=0]([A,_transitive=true]([_transitive=true,_#sons=0,_name=N1],[_transitive=true,ord>{N1.ord},_#occurrences=0,_#sons=0]),[B,_transitive=true]([_transitive=true,_#sons=0,_#occurrences=0,ord<{N2.ord}],[_transitive=true,_#sons=0,_name=N2,ord={N1.ord}+1]))`

" A is the same node as B":

In TGrep2: `A = B`

This predicate cannot be generally translated to Netgraph, where two nodes in the query cannot match one node in the result tree at the same time. The equal sign is usually used together with another predicate, e.g. `A <<= B` means that B is either dominated by A, or B is equal to A. The only possibility to translate these constructions to Netgraph is using multi-tree queries with logical OR. More complex patterns in TGrep2 with more than one such predicate with equal sign therefore cannot be translated to Netgraph.

As stated in Section 6.2, TGrep2 allows defining a cycle in edges connecting the nodes. This ability, though usually not useful, also makes TGrep2 query language more powerful in certain aspect than FS Query Language.

One of the major additions in TGrep2 (in comparison with TGrep) is the ability to specify Boolean expressions over the relationships between nodes. Thus, very complex queries can be made:

```
A [< B | ![. C !, F]] | ![< D !.. E]
```

The example is taken from TGrep2 User Manual (Rohde 2005) and it means: (A has child B or it does not (immediately precede C and not immediately follow F)) or (A does not (have child D and is not followed by E)). Such complex queries cannot be reproduced in Netgraph.

On the other hand, queries in Netgraph can be found that cannot be translated to TGrep2, even if we put aside the fact that (just like TGrep) TGrep2 is designed only for treebanks with nodes evaluated with one attribute.

One of such queries combines the meta-attribute `_optional` with the meta-attribute `_#sons`. It searches for a node A with node B in its subtree and only with nodes with exactly two sons on the path from A to B:

```
[A]([_optional=true,_#sons=2]([B]))
```

There is a special predicate for paths with nodes that have exactly one son in TGrep2, but the query with the path with nodes with two sons cannot be reproduced.

The reason is not in the combination of the meta-attributes but already in the usage of the meta-attribute `_optional` with the value `true`. TGrep2 has no feature to substitute this meta-attribute, to set a condition on a path of nodes of an arbitrary length.

In TGrep2, it is also impossible to substitute references in general. For example, the following query in Netgraph cannot be translated to TGrep2:

```
[]([_name=N1],[_#sons={N1._#sons}])
```

It searches for two brothers that have the same number of sons (unspecified in the query).

And also other constructions can be found in Netgraph, untranslatable to TGrep2.

We do not claim that the queries (either in Netgraph or in TGrep2) that cannot be translated to the other query language are linguistically relevant. We only wanted to compare the power of the two query languages.

As shown in the previous paragraphs, neither of the query languages (TGrep2 or FS Query Language) is unambiguously superior to the other. Neither all queries from TGrep2 can be translated to Netgraph, nor all queries from Netgraph can be translated to TGrep2. In some areas, TGrep2 is more powerful than Netgraph, in other areas Netgraph is more powerful than TGrep2. We could also say that the powers of the tools are not comparable.

6.5.4 Comparison to TigerSearch

Node Description

In TigerSearch, on the node level, nodes can be described by Boolean expressions over attribute-value pairs, where each value can also be expressed as a Boolean expression over single values.

Netgraph uses alternative nodes and alternative values of attributes for the description of a node. Thus, it has a slightly lesser power in expressing a node evaluation than TigerSearch. The only drawback of Netgraph is that it cannot set more than one condition on one attribute with relation "AND", i.e. set two conditions on one attribute that should be valid at the same time.

On the other hand, in contrast to Netgraph, TigerSearch cannot use relations less than ("<") and greater than (">") in setting values of attributes.

Both tools allow using regular expressions as single values.

Node Relations

TigerSearch uses a similar set of predicates like TGrep2. Most of the predicates can be translated to Netgraph. Let us show the translation of predicates that are not present in TGrep or Tgrep2:

"A dominates B directly with a labelled dominance":

In TigerSearch: `A >L B`
In Netgraph: `[A]([B,afun=L])`

All labelled versions of TigerSearch predicates can be translated this way (the label of the edge is moved to an attribute of the son-node).

"A dominates B with a distance between `m` and `n` (0<m<n)":

In TigerSearch: `A >m,n B`
In Netgraph: `[A,_name=N1]([B,_transitive=true,_depth={N1._depth}+m|...|{N1._depth}+n])`

"B is the leftmost (rightmost) terminal successor of A":

In TigerSearch: `B >@l A`
In Netgraph: `[B]([A,_transitive=true,_name=N1,_#sons=0],[_transitive=true,ord<{N1.ord},_#occurrences=0,_#sons=0])`

It is very similar to TGrep predicate "B is the leftmost descendant of A" (`A <<, B`), we only added `_#sons=0` here to make sure the descendants are leaves. The rightmost version only differs in the relation at the attribute `ord`.

The definition of precedence for non-terminals in TigerSearch is: a node A precedes a node B if the left corner (the leftmost terminal successor) of A precedes the left corner of B. Quite a complex query has to be used in Netgraph to translate this type of precedence, nevertheless, for the sake of comparing the power of the query languages, it can be done:

"A precedes B with at least distance 1":

In TigerSearch: `A .* B`
In Netgraph: `[_depth=0]([A,_transitive=true]([_transitive=true,_#sons=0,_name=N1],[_transitive=true,ord<{N1.ord},_#occurrences=0,_#sons=0]),[B,_transitive=true]([_transitive=true,_#sons=0,_#occurrences=0,ord<{N2.ord}],[_transitive=true,_#sons=0,_name=N2,ord>{N1.ord}]))`

"A precedes B with a distance at least `n` (n>0)":

In TigerSearch: `A .n B`
In Netgraph: as above, with `ord={N1.ord}+n` in the last line; we assume that the values of the attribute `ord` increase by one for the terminals.

"A precedes B with a distance between `m` and `n` (0<m<n)":

In TigerSearch: `A .m,n B`
In Netgraph: similarly, with `ord<{N2.ord}+n` in the first line and `ord>{N1.ord}+m` in the last line

"There is a secondary edge from A to B"

In TigerSearch: `A >~ B`
In Netgraph: `[_depth=0]([B,_name=N1,_transitive=true],[A,_transitive=true,s.rf={N1.id}])`

Where `s.rf` is a referential attribute for the secondary edge.

There are several predicates for the declaration of brothers in TigerSearch. These can be easily translated to Netgraph, both in the positive and negative sense, by creating a mutual father of the nodes, or respectively, by creating two different fathers of the nodes. A combination of brotherhood and precedence can also be transformed, similarly to the predicate for precedence ("`.*`") above.

Negation

All variables/node patterns in TigerSearch are existentially quantified. Therefore, the expression "A does not directly dominate B" (`A !> B`) means "A and B appear in the tree but A does not directly dominate B". The full negation cannot be expressed in TigerSearch. This "weak" type of negation can be translated to Netgraph using the "real" negation and the existence of the node B somewhere else in the tree. Let us give one example of the translation:

"A does not directly dominate B":

In TigerSearch: `A !> B`
In Netgraph: `[_depth=0]([A,_transitive=true]([B,_#occurrences=0]),[B,_transitive=true])`

Graph Description

TigerSearch uses restricted Boolean expressions over node relations and node descriptions for a further description of the query. Negation is not allowed, only conjunction ("&") and disjunction ("|") are supported.

Since negation is not allowed, Netgraph can translate graph descriptions from TigerSearch in their disjunctive normal form without negation using multi-tree queries with relation OR. For the "inner" relation AND it can use the transitive dependency on the root-node, for example:

In TigerSearch: `(A & B) | (C & D)`
In Netgraph: `OR`
```
[_depth=0]([A,_transitive=true],[B,_transitive=true])
[_depth=0]([C,_transitive=true],[D,_transitive=true])
```

In the terms of the power of the graph description, the two tools are equal.

Variables

TigerSearch uses variables to bind values of attributes of different nodes. It can be translated to Netgraph using references.

Graph Predicates

TigerSearch defines several graph predicates; it uses variables for identifying a node that the predicate applies to; for the sake of simplicity, we use labels like A and B in the previous examples:

"A is the root":

In TigerSearch: `root(A)`
In Netgraph: `[A,_depth=0]`

"A has from `m` to `n` sons":

In TigerSearch: `arity(A,m,n)`
In Netgraph: `[A,_#sons=m|...|n]`

"A dominates from `m` to `n` leaves":

In TigerSearch: `tokenarity(A,m,n)`
In Netgraph: `[A]([_transitive=true,_#sons=0,_#occurrences=m|...|n])`

" A only dominates leaves that form a continuous string" :

In TigerSearch: `continuous(A)`

This predicate cannot be translated to Netgraph.

" A dominates leaves that do not form a continuous string" :

In TigerSearch: `discontinuous(A)`

This predicate cannot be translated to Netgraph.

Let us finish the comparison of the expressive power of the two tools. We have shown that TigerSearch has a few features that cannot be translated to Netgraph. Let us look on the opposite direction – what the disadvantages of TigerSearch in comparison with Netgraph are.

The most serious disadvantage of TigerSearch is without a doubt its lack of real negation. All nodes used with negative predicates have to appear somewhere else in the tree.

Also other examples of queries in Netgraph that cannot be translated to TigerSearch can be found. Just like with TGrep2, it is impossible to set a condition on a path of nodes of an arbitrary length in TigerSearch, i.e. generally translate queries from Netgraph with the meta-attribute `_optional` set to the value `true`.

We can conclude, similarly to the comparison with TGrep2, that neither of the query languages (TigerSearch or FS Query Language) is superior to the other. Neither all queries from TigerSearch can be translated to Netgraph, nor all queries from Netgraph can be translated to TigerSearch.

6.5.5 Why Is It So Complex in Netgraph?

Some of the translations from the other tools to Netgraph may seem very complex, sometimes much more complex than the original expressions in the other tools.

The main reason is that we matched the predicates from the other tools. It is clear that Netgraph that uses a different set of features cannot be as straightforward as these tools in mimicking their predicates. For our purpose of comparing the expressive power, it is sufficient that the translation exists. We also believe that in Netgraph, even the complex expressions remain well readable when displayed in the graphical form (to save space, we always used the textual form in this section).

If we tried to translate simple Netgraph expressions to the other tools, we might get similarly complex translations. For example, searching for nodes A that have two or three sons is quite straightforward in Netgraph (since we have a convenient meta-attribute at our disposal), while in TGrep, we have to rephrase it indirectly and much less intuitively by defining that there are two sons of A of any kind but there are not four sons of A of any kind:

In Netgraph: `[A,_#sons=2|3]`
In TGrep: `A <2 __ !<4 __`

6.6 Universality

Netgraph has been primarily developed for the Prague Dependency Treebank. Nevertheless, it can be used for any other linguistic treebank, as long as the treebank is converted to FS File Format (described shortly in Section "4.1 - The Format" and in detail in "Appendix B: FS File Format Description"), and as long as its size does not substantially exceed the size of PDT 2.0 (for the sake of the search speed; see the discussion of the search speed in Subsection 9.2.2). The features of the query language are general enough for other dependency treebanks, and as shown in the previous section (6.5), it can also be used for constituent-structure treebanks.

We have described in Chapter "4 - The Data" how to adapt the tool for a treebank – by a declaration of attributes of the treebank in the file header, by creating a configuration file for all references (secondary edges etc.) in the data, and by adding some necessary information to the data (like an attribute for left-right order of nodes in the tree etc.).

Several examples of usage of Netgraph for other treebanks are given in "Appendix E: Other Usages of Netgraph".

6.7 Feedback From Users

During the years of development, Netgraph has been used by many users. Their feedback influenced the way the query language and also the tool developed.

Several seminars have been organized during the past years with frequent users of Netgraph. They had prepared linguistic phenomena they wanted to search for, and during the seminars, we tried to create queries in Netgraph that would search for those phenomena. If it was not possible, we discussed what new features might be introduced to Netgraph Query Language in order to satisfy the requirements. If it was possible to create a query but the query was too complex, we also tried to figure out what new feature of the query language would make the query simpler.

A lot of inspiration has also come from using Netgraph for other treebanks than PDT 2.0. Netgraph has been used both for dependency and constituent structure treebanks, and for several languages, e.g. Arabic, Chinese, Latin, Slovak, English etc. Some of these usages for other treebanks are related in "Appendix E: Other Usages of Netgraph".

Let us give an example of a feature introduced to simplify the query language at a request from the users. The following query searches for all non-projective constructions on the analytical layer:

```
OR
[]([_transitive=exclusive,_name=N1],[_transi-
tive=exclusive,ord<{N1.ord}]([ord>{N1.ord}]))
[]([_transitive=exclusive,_name=N2],[_transi-
tive=exclusive,ord>{N2.ord}]([ord<{N2.ord}]))
[_name=N3]([_transitive=true,ord<{N3.ord}]([ord>{N3.ord}]))
[_name=N4]([_transitive=true,ord>{N4.ord}]([ord<{N4.ord}]))
```

The graphical representation of the query was given in Chapter 5. The query consists of four trees, representing possible configurations of a node and an edge causing the non-projectivity.

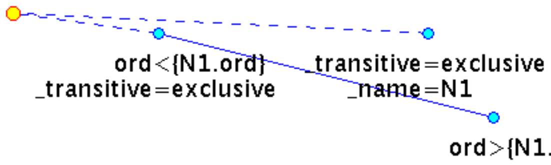
The query is not simple, yet before the value `exclusive` of the meta-attribute `_transitive` was introduced, it consisted of ten trees and was much more complex:

```
OR
[]([ord<{N1a.ord}]([ord>{N1a.ord}]),[_name=N1a])
[]([]([_transitive=true,ord<{N1b.ord}]([ord>{N1b.ord}])),[_name=N1b])
[]([ord<{N1c.ord}]([ord>{N1c.ord}]),[]([_transitive=true,_name=N1c]))
[]([]([_transitive=true,ord<{N1d.ord}]([ord>{N1d.ord}])),[]([_transi-
tive=true,_name=N1d]))
[]([ord>{N2a.ord}]([ord<{N2a.ord}]),[_name=N2a])
[]([]([_transitive=true,ord>{N2b.ord}]([ord<{N2b.ord}])),[_name=N2b])
[]([ord>{N2c.ord}]([ord<{N2c.ord}]),[]([_transitive=true,_name=N2c]))
[]([]([_transitive=true,ord>{N2d.ord}]([ord<{N2d.ord}])),[]([_transi-
tive=true,_name=N2d]))
[_name=N3]([_transitive=true,ord<{N3.ord}]([ord>{N3.ord}]))
[_name=N4]([_transitive=true,ord>{N4.ord}]([ord<{N4.ord}]))
```

Let us focus on the first tree of the first query:

```
[]([_transitive=exclusive,_name=N1],[_transi-
tive=exclusive,ord<{N1.ord}]([ord>{N1.ord}]))
```

With the graphical form:

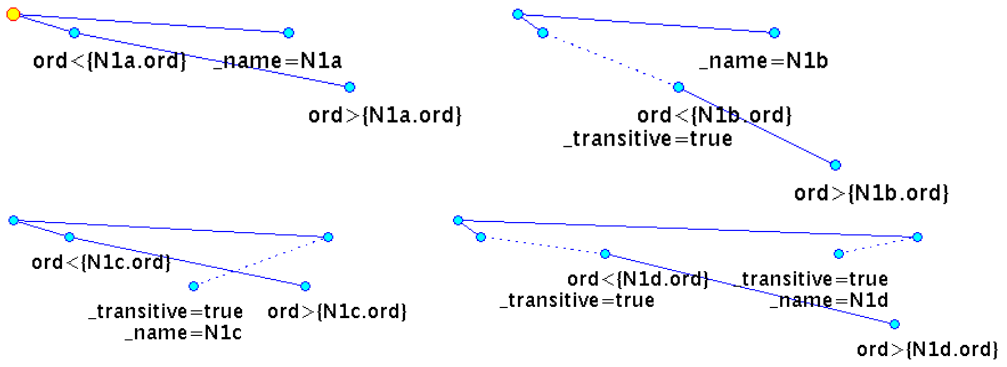


It represents the configuration of a node and an edge forming a non-projective construction where the node (N1) does not lie on the path from the edge to the root and the left node of the edge is the father of the right node. The value `exclusive` of the meta-attribute `_transitive` makes sure that no nodes of the two transitive edges are shared. Therefore, the node N1 cannot belong to the subtree of the non-projective edge.

With only the value `true` of the meta-attribute `_transitive` available, this is much more complicated to achieve. If we simply used `_transitive=true` instead of `_transitive=exclusive`, the node N1 might be a son of any of the two nodes of the non-projective edge (the edge might not be non-projective then), because the true transitivity would only say that the node N1 could appear anywhere in the subtree of the root of the query. In fact, four query trees must be used instead of the one with the exclusive transitivity, to make sure that this cannot happen:

```
[]([ord<{N1a.ord}]([ord>{N1a.ord}]),[_name=N1a])
[]([]([_transitive=true,ord<{N1b.ord}]([ord>{N1b.ord}])),[_name=N1b])
[]([ord<{N1c.ord}]([ord>{N1c.ord}]),[]([_transitive=true,_name=N1c]))
[]([]([_transitive=true,ord<{N1d.ord}]([ord>{N1d.ord}])),[]([_transi-
tive=true,_name=N1d]))
```

With the graphical form:



These four trees substitute one tree with the exclusive transitivity. The root of the query has always two non-transitive sons that make sure that their transitive subtrees are disjoint. It can also happen that any of these two sons is already a part of the non-projective edge or the non-projective node. Since a transitive edge in Netgraph cannot have zero length (the father and the son of a transitive edge cannot merge into one node in the result tree), four trees with four different configurations are needed, as presented.

The introduction of the value `exclusive` of the meta-attribute `_transitive` makes the query not only much simpler but also much more intuitive.

7 The Tool

We have implemented Netgraph Query Language in a search tool called Netgraph. As a basis, we used Netgraph 1.0, a simple tool described in Chapter 2 in Section "2.2 - Netgraph 1.0 – The Starting Point".

A short description of the installation and usage of the tool can be found in "Appendix F: Installation and Usage of Netgraph – A Quick How-To". The tool itself, as well as a detailed manual and the technical documentation, can be found on the Netgraph home page[19].

In this chapter, we concentrate on the properties that are important for a search tool for PDT 2.0. We also discuss changes since version 1.0 of the tool.

7.1 Properties of the Tool

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 were 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.

19 `http://quest.ms.mff.cuni.cz/netgraph`

- **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 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[20] 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 (re-

20 UTF-8 (`http://www.utf-8.com`) stands for Unicode Transformation Format-8. It is an octet (8-bit) lossless encoding of Unicode characters (`http://www.unicode.org`).

sult) trees, number of found occurrences in the found trees (see Section 6.4), 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.[21]
- **speed/portability**
  For the server, speed is the most important factor. Therefore, C programming language[22] (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[23] (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.

7.2 Changes since Version 1.0

The actual version of Netgraph is 1.95. We call the original version of Netgraph programmed by Roman Ondruška (described in Chapter 2 in Section 2.2) 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. Netgraph Client 1.0 had 1 526 lines of code. Netgraph Client 1.95 consists of more than 21 thousand lines. Netgraph Server 1.0 had 3 973 lines of code. Netgraph Server 1.95 has more than 11 thousand lines.

The following lists contain the most principle 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.

7.2.1 Main Extensions to the Query Language

- Meta-attributes have been introduces 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.

21 With a suitable configuration, e.g. the analytical tree corresponding to the actually depicted tectogrammatical tree can be opened in TrEd using the external command.

22 GCC compiler of C programming language has been used (`http://gcc.gnu.org/`)

23 `http://java.sun.com`

- Multi-tree queries are supported.
- Support for hidden nodes has been added.

7.2.2 Main Extensions to the Tool

- The tool now supports the tectogrammatical trees (hidden nodes, 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.

8 Real World

After we have presented all features of Netgraph Query Language and shown what can be searched for with the language, it might be interesting to know to what extent the features are 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

number of:	*total*	*analytical trees*	*tectogrammatical trees*
_#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).

8.1 The Queries

We present a representative selection of queries put in by the users. Examples from the analytical layer are typed in italic, examples from the tectogrammatical layer are typed in the regular font.

8.1.1 One-Node Queries

Most of one-node queries on the analytical layer are also one-attribute queries, queries setting only one attribute, most often `m/form` or `m/lemma`, occasionally `m/tag` or `afun`, e.g.:

[m/form=chlapec]

[m/form=kluk]

[m/form=nejspíš]*

[m/lemma=plzeňské]

[m/lemma=plzeňský]

[m/tag=Vf]*

[afun=AtvV]

One node queries that combine several attributes, mostly used for studies of word class (POS) conversion, also use mainly the attributes `m/form`, `m/lemma`, `m/tag`, and `afun`:

```
[m/lemma=vedoucí,m/tag=NN*]

[m/lemma=vedoucí,m/tag=A*]

[m/lemma=vedoucí,m/tag=N*,afun=Atr*]

[m/lemma=vedoucí,m/tag=A*,afun=Atr*]

[m/tag=N*,m/form=vzhledem]

[m/tag=R*,m/form=vzhledem]

[m/lemma=večer,m/tag=N*]

[m/lemma=večer,m/tag=D*]

[m/lemma=večer,afun!=Atr|Adv]

[afun=AtvV,m/tag=A*]

[afun=AtvV,m/lemma=sám]
```

The attribute `t_lemma` is the most often used attribute in one-node queries on the tectogrammatical layer. Also the attribute `functor` and various grammatemes are frequently used:

```
[t_lemma=proměnit]|[t_lemma=proměňovat]

[t_lemma=původní,functor=TWHEN]

[t_lemma=podobný|stejný,functor=PREC]

[t_lemma=sám,functor!=COMPL|RSTR]

[functor=APPS|CONFR|CONJ|CONFR|CONTRA|CSQ|DISJ|GRAD|OPER|REAS|ADVS]

[functor=ACT,is_generated!=1,gram/sempos=v]

[gram/sempos=v,gram/aspect=cpl,gram/tense=ant,gram/verbmod=nil,gram/pe
rson=3]
```

8.1.2 Structured Queries without Meta-Attributes

Structured queries on the analytical layer much more often use the attributes `m/tag` and `afun` and less the attributes `m/form` and `m/lemma`. The following examples are typical queries without meta-attributes.

a noun valency:

```
[m/lemma=vzkaz]([m/tag=N???3*])
```

infinitive constructions, dependent on atypical verbs:

```
[m/tag=A*,afun=Pnom]([m/tag=Vf*,afun=Obj])
```

```
[m/tag=Vf*,m/lemma!=být|lze|muset|moci|chtít|umět|smět|dovést|potřebo-
vat|začít|začínat|přestat|nechat|hodlat|jet|jít|odmítat|potřebuju|při-
jet|přijít|chodit|dokázat|dát|dávat|mít|stačit|nechávat|umožňovat]
([m/tag=Vf*]([m/lemma=se]))
```

comparative constructions:

```
[m/form=*ěji]([m/form=než])
```

coordination:

```
[m/tag="Vp.*"]([afun=Coord]([afun=Sb,m/tag="NNF.*"],
[afun=Sb,m/tag="NNM.*"]))
```

On the tectogrammatical layer, the attribute `functor` is undoubtedly the most often used attribute in the structured queries. Grammatemes and the attribute `t_lemma` are also widely used. Other attributes are used less frequently, depending on the phenomena they describe. The following examples represent queries without meta-attributes on the tectogrammatical layer:

topic-focus articulation and systemic ordering:

```
[functor=PRED]([functor!=ADVS|APPS|CONFR|CONJ|CSQ|DISJ|GRAD|OPER]
([tfa=c]))
```

```
[]([functor=TWHEN,tfa=f],[functor=LOC,tfa=f])
```

coordination:

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

multiple adverbial time complement, combination of time modifications:

```
[functor=TWHEN]([functor=TWHEN,gram/sempos=adj.denot])
```

```
[gram/sempos=v]([functor=TSIN,gram/sempos!=n.quant.def]([gram/sempos!
=n.quant.def]),[functor=TTILL,gram/sempos!=n.quant.def]([gram/sempos!
=n.quant.def]))
```

valency of verbs, co-occurrence of valency members:

```
[gram/sempos=v]([functor=ACT],[functor=ADDR],[functor=EFF],
[functor=ORIG],[functor=PAT])
```

8.1.3 Queries with Meta-Attributes

The following examples show typical queries (put in by users) that use meta-attributes, both on the analytical layer and on the tectogrammatical layer. Sometimes, interesting examples of queries could only be found on one of the layers. The queries are divided into sections by the principal meta-attribute; nevertheless, many queries use several meta-attributes at once. Again, examples from the analytical layer are typed in italic, while examples from the tectogrammatical layer are typed in the regular font:

_transitive

The meta-attribute `_transitive` is most often used to express possibly non-direct dependencies between nodes.

non-projectivity:

`[_name=n1]([ord<{n1.ord},_transitive=true]([ord>{n1.ord}]))`

relative position of a noun and an enclitic in a subordinate clause:

`[m/form=že]([]([_transitive=true,m/tag=N.*,_name=N1],[m/form=by|se|mu|mi|si|ho,ord>{N1.ord},_transitive=true]))`

surface word order:

`[afun=Pred]([_name=N1,afun=Adv,_transitive=true],[ord<{N1.ord},_#occurrences>=1,_transitive=true])`

possibly deep nested modifier:

`[functor=PRED,t_lemma=být]([]([t_lemma=těžký,_transitive=true]))`

nodes without the adnominal adjunct in their subtree:

`[functor!=RSTR]([functor=RSTR,_#occurrences=0,_transitive=true])`

grammatical coreference:

```
[functor=PRED,gram/sempos=v]([_name=N1,_transitive=true],[_transi-
tive=true,gram/sempos=v,gram/verbmod=nil]
([functor=ACT,coref_gram.rf={N1.id}]))
```

_optional

On the analytical layer, the meta-attribute `_optional` is generally used to skip one node, most often a preposition, a coordination or an apposition:

valency on surface (three objects with prepositions (in two cases optional)):

`[m/tag=V*]([afun=AuxP,m/lemma=za]([afun=Obj]),[afun=AuxP,_optional=1]([afun=Obj]),[afun=AuxP,_optional=1]([afun=Obj]))`

coordination/apposition:

```
[m/tag=N*]([afun=Coord,_optional=1]|[afun=Apos,_optional=1]
([m/tag=N???3*]))
```

On the tectogrammatical layer, also usually one node of certain type is skipped (the value `true` has only recently been introduced to the language and has not yet been widely used by the users, at least on the public server[24]):

coordination etc.:

```
[t_lemma=zájem]([functor=CONJ,_optional=1]([functor=PAT]))
```

```
[functor=PRED]([nodetype=coap,_optional=1]
([functor=CNCS,gram/sempos=v]))
```

topic-focus articulation:

```
[functor=PRED]([functor=ADVS|APPS|CONFR|CONTRA|CONJ|GRAD|CSQ|REAS|
OPER,_optional=true]([]([tfa=f])))
```

_#sons

The meta-attribute `_#sons` is often used to study “extreme cases” of how a type of node can be modified.

valency of verbs:

```
[gram/sempos=v]([_#sons>5])
```

leaf of the tree:

```
[functor=LOC]([_#sons=0,functor=PAR])
```

type of node (phraseme) with any modification:

```
[nodetype=dphr,_#sons>0]
```

_depth

The meta-attribute `_depth` is usually used as an auxiliary attribute, e.g. with the meta-attribute `_sentence`, to avoid an unwanted multiplication of results. Only occasionally, users are directly interested in some levels in the result trees.

topic-focus articulation at certain levels:

```
[tfa=c,_depth=2]
```

```
[functor=ADVS|CONFR|CONJ|CSQ|DISJ|GRAD|OPER|REAS|APPS|CONTRA]
([tfa=c,_depth=3])
```

24 To be exact, the expression `_optional=true` used to have the same meaning as `_optional=1` has now and it was the only possible usage of the meta-attribute. The old examples of queries have been modified to comply with the recent semantics of the query language.

technical usage with the meta-attribute `_sentence`:

```
[_sentence=".*na základě .*",_depth=0]
```

with the meta-attribute `_#descendants` to search for small results with a given functor:

```
[_#descendants<=11,_depth=1]([functor=AIM,gram/sempos=v])]
```

_#descendants

The meta-attribute `_#descendants` is most often used to set the minimal or maximal (sometimes exact) size of the whole tree (as in the previous example with the meta-attribute `_depth`), or of a subtree of a certain node, representing a linguistic phenomenon.

exact size of a subtree:

```
[afun=Pred,_#descendants=5|6|7|8]([afun=Obj]([m/lemma=svůj]))
```

leaf of the tree:

```
[functor=CONJ]([functor=ADDR,t_lemma=#PersPron,_#descendants=0])
```

small trees or subtrees containing specific information:

```
[functor=PRED,_#descendants<=10]([nodetype=coap,_optional=1]
([functor=CNCS,gram/sempos=v]))
```

```
[t_lemma=vidět]([functor=PAT,gram/sempos=v,_#descendants<=3])
```

big trees or subtrees:

```
[_#descendants>12,functor=PRED]([sempos=v,functor=AIM])
```

_#lbrothers, _#rbrothers

The meta-attributes `_#lbrothers` and `_#rbrothers` are used to study phenomena related to the left-right order of sons of a node; on the analytical layer, it corresponds closely to the surface word order, on the tectogrammatical layer, the order of nodes reflects the communicative dynamism.

position of an enclitic:

```
[]([afun!=AuxX|AuxG|AuxC,_#lbrothers=0],[m/lemma=se,_#lbrothers>1])
```

topic-focus articulation:

```
[_depth=1]([tfa=c,_#lbrothers>0])
```

rhematizers, their position:

```
[functor=PRED]([functor=RHEM,t_lemma=také,_#lbrothers=0])
```

```
[]([_name=N1,functor=RHEM],[_#occur-
rences=0,deepord<{N2.deepord},_#lbrothers>0],
[_name=N2,deepord>{N1.deepord}])
```

```
[functor=PRED,_name=N1]
([deepord<{N1.deepord},_name=N2,functor=RHEM,t_lemma!=#Neg],
[deepord<{N1.deepord},_#rbrothers<{N2._#rbrothers},_#occurrences=0])
```

_#occurrences

The meta-attribute `_#occurrences` is most often used to study valency of words or classes of words. It is most frequently (but not only) used to forbid a presence of a certain son (`_#occurrences=0`) of a node.

surface valency:

```
[]([afun=Obj,_#occurrences>1])
```

```
[afun!=AuxS,m/form!=že|aby|ať|zda|ač|ačkoli*|když|jako|jestliže|je-
likož|kdyby|když|neboť|pokud|protože|přestože|takže|zatímco]([_option-
al=1,afun=Coord|Apos]([m/tag=VB*|?c*|?e*|?i*|?m*|?p*|?q*|?s*|?t*]
([m/tag=?K*|?u*|?Y*|?4*|?J*|?E*|?z*|?Q*|TT*,_#occurrences=0],
[m/form=jak|kam|kde|kudy|proč,_#occurrences=0])))
```

valency, co-occurrence of related functors:

```
[gram/sempos=v]([functor=LOC,_#occurrences>=3])
```

```
[]([functor=DIR1],[functor=DIR3,_#occurrences=0])
```

systemic ordering near the verb (in combination with references):

```
[functor=PRED,_name=N1]([deepord<{N1.deepord},_name=N2],
[deepord<{N1.deepord},_#rbrothers<{N2._#rbrothers},_#occurrences=0])
```

_name

See Subsection “8.1.4 - Queries with References” below.

_#hsons

See Subsection “8.1.5 - Queries with Hidden Nodes” below.

_sentence

As expected, the meta-attribute `_sentence` is used to search in the linear form of the sentence for a sequence of words, mainly on the tectogrammatical layer.

```
[_sentence=".*\[Nn\]a základě.*"]
```

```
[_sentence="Česká televize.*",_depth=0]
```

8.1.4 Queries with References

References are widely used by the users, more often on the tectogrammatical layer, as there are more complex phenomena annotated there.

binding a form and a lemma together:

```
[_name=N1,lemma={N1.form}]
```

word order:

```
[afun=Pred,_name=N1]([afun=Sb,ord>{N1.ord}])
```

```
[m/tag=N*,_name=N1]([m/lemma=ten,ord={N1.ord}-3])
```

```
[ord>{N2.ord},m/tag="Vf.*"]([m/tag="Vf.*",_name=N2,ord>1])
```

non-projectivity:

```
[_name=n1]([ord<{n1.ord},_transitive=true]([ord>{n1.ord}]))
```

```
[]([_transitive=exclusive,_name=N1,t_lemma!=#*],[_transitive=exclu-
sive,deepord>{N1.deepord}]([deepord<{N1.deepord}]))
```

topic-focus articulation:

```
[functor=PRED]([tfa=t,_name=N1],[deepord={N1.deepord}+1,tfa=f])
```

```
[]([is_member=1,_name=N1],[is_member=1,tfa!={N1.tfa}])
```

rhematizers:

```
[]([_name=N1,functor=RHEM],[_#occur-
rences=0,deepord<{N2.deepord},_#lbrothers>0],
[_name=N2,deepord>{N1.deepord}])
```

communicative dynamism in conditional expressions:

```
[functor=PRED]([functor=COND,deepord<{N1.deepord},gram/sempos!=v],
[functor=PAT,_name=N1])
```

re-generated node with the same `t_lemma`:

```
[]([_name=N1,is_generated!=1],[is_generated=1,t_lemma={N1.t_lemma}])
```

coreference:

```
[]([functor=ACT,_name=N1],[]([functor=ACT,coref_gram.rf={N1.id},t_lem-
ma=#Cor]))
```

8.1.5 Queries with Hidden Nodes

Of course, queries with hidden nodes only appear on the tectogrammatical layer. Most queries combine attributes from several layers, fewer queries only use attributes from the hidden nodes. In all the queries, users are either interested in the surface representation of a tectogrammatical phenomenon, or in a linguistic meaning (tectogrammatical annotation) of a surface expression.

specific words deleted on the tectogrammatical layer:

```
[m/lemma=zatímco,hide=true]
```

conditional expressions:

```
[gram/sempos=n.denot]([functor=COND]([hide=true,m/form=pokud]))
```

```
[functor=PRED]([functor=COND]([m/lemma=kdyby,hide=true]),
[t_lemma=#Gen,functor=ACT])
```

subtype of subject clauses:

```
[t_lemma=být,gram/verbmod!=cdn]([functor=PAT,gram/sempos=adj.denot],
[gram/verbmod=ind|cdn,functor=ACT]([m/lemma=aby,hide=true]))
```

surface form with (or without) a given meaning (correlative expressions):

```
[t_lemma=ten,_#hsons=1,functor!=MEANS|MANN]([hide=true,m/form=tím])
```

reference to a preceding context with a specific dependency:

```
[functor!=PRED,nodetype!=coap,_#hsons=1]([functor=PREC],
[hide=true,m/tag!=V*])
```

re-generated verb with the same `t_lemma`:

```
[]([_name=N1,is_generated!=1]([hide=true,m/tag=V*]),
[is_generated=1,t_lemma={N1.t_lemma}])
```

specific functor (cause, location) expressed with a given number of surface words:

```
[functor=CAUS,_#hsons>3]([a/afun=AuxC,hide=true])
```

```
[functor=LOC,_#hsons=3]
```

topic-focus articulation of specific words (pronouns with the stress):

```
[tfa=t]([hide=true,m/form=jemu|jeho|mne|mně|tebe|tobě|sebe|sobě,m/tag!
=PS*])
```

topic-focus articulation at a specific position in the sentence:

```
[tfa=c]([hide=true,a/ord=12])
```

time expression expressed with the accusative:

```
[functor=TFHL|TFRWH|THL|THO|TOWH|TPAR|TTILL|TWHEN]
([hide=true,m/tag=\"....4.*\"])
```

specific time expression:

```
[t_lemma=hodina]([hide=true,m/form=před])
```

Actor expressed as a subject in the genitive:

```
[functor=ACT]([hide=true,m/tag="....2.*",a/afun=Sb])
```

specific expression of a Patient:

```
[gram/sempos=v]([functor=PAT]([hide=true,m/tag!
="....4.*"|"V.*"|"R.*"|"J.*",a/afun!=Pnom|Sb]))
```

specific type of expression on the surface, noun valency:

```
[t_lemma=obchod]([]([hide=true,m/form=s],[hide=true,m/tag="N...7.*"]))
```

9 Conclusion

9.1 What Has Been Done

In the book, 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

We have shown that the proposed query language really meets the requirements on a query language for the Prague Dependency Treebank 2.0.

We have discussed properties of the data for the query language and 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 and presented representative examples of real-world queries that use the features.

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.

9.2 Future Work

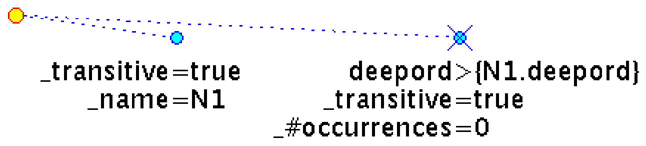
9.2.1 The Query Language

We present several ideas about future work on Netgraph Query Language and on the tool as well. Obviously, no change can be made in the language without changing the tool too. On the other hand, the tool can be improved while preserving the same query language.

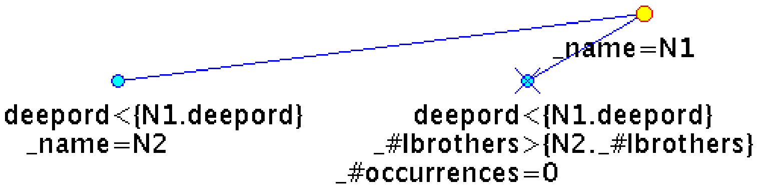
Simplification

Searching for complex phenomena inevitably leads to complex queries. It is always possible to extend the query language to support a special operation in a simple way, at the cost of making the query language more extensive.

Constructions searching for the left-/rightmost node of certain kind can serve as an example. Let us recall two queries. The query searching for the rightmost descendant of a node in a tectogrammatical tree:



and the query searching for the rightmost left son of a node in a tectogrammatical tree:

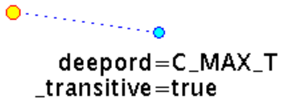


Both construction are defined in a negative way, there has to be a definition of an undesired node. If we added several special constants to the query language, it might be possible to create these queries positively and more simply. The constants might be:

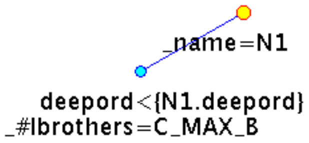
name	*description*
C_MAX_T	Conditioned maximum possible value in the tree
C_MAX_B	Conditioned maximum possible value among brothers
C_MIN_T	Conditioned minimum possible value in the tree
C_MIN_B	Conditioned minimum possible value among brothers
U_MAX_T	Unconditioned maximum possible value in the tree
U_MAX_B	Unconditioned maximum possible value among brothers
U_MIN_T	Unconditioned minimum possible value in the tree
U_MIN_B	Unconditioned minimum possible value among brothers

Conditioned constants mean that the maximum value is selected from nodes matching all other attributes defined at the node, taking also the position of the node in the query into account (for *_T constants). Unconditioned constants mean that the maximum value is selected regardless of other attributes of the node, i.e. from all nodes in the tree, or from all sons of the father of the node (not taking the position of the node in the query into account for *_T constants).

Using the constants, the two queries from above could be considerably simplified. The first query would search for the rightmost descendant of a node:



The second query would search for the rightmost left son of a node:



If we used the unconditioned constant `deepord=U_MAX_T` in the first query, it would search for those cases where the rightmost node in the tree is a descendant of the father-node from the query.

If we used the unconditioned constant `_#lbrothers=U_MAX_B` in the second query, it would search for those cases where the rightmost left son of a node is also the rightmost son of the node.

Further Extensions

More conditions on values of one attribute

One of extensions that might be useful is a possibility do define more conditions on values of one attribute that should be true at the same time, possibly with different relations. It would be a counterpart to alternative values of an attribute. This way, we might, for example, create a query that would search for a node with the morphological tag that is a noun but is not in the accusative, without using a regular expression. We would specify two conditions that should be true at the same time: `m/lemma=N* & m/lemma!=????4*`. It is only a simple example, the query can be actually created using a regular expression without any extension: `m/lemma="N...\[^4\].*"`. Yet, there might be queries where such an extension would prove necessary, like defining complex references among nodes. In all tasks in searching in PDT 2.0, we managed to find another way of defining the required query, nevertheless it is true that in the current state of the query language we cannot directly define that node A is on the left side from node B and on the right side from node C. We must define e.g. that A is on the left side from B and C is on the left side from A. There might be a reasonable query that cannot be defined this way and the possibility of setting two conditions on one attribute would help.

More complex logical combinations of trees in a multi-tree query

We tried to make the query language, especially its graphical representation, as simple as possible. We also had to take into account that the research presented in this book was not only theoretical but the proposed query language would have to be implemented. Therefore, and since the simple AND or OR logical expression combining trees in multi-tree queries proved sufficient for searching in PDT 2.0, we did not propose more complex logical combinations in the query language.

Yet, they might be sometimes useful. Purely for technical reasons, implementation of the disjunctive normal form[25] without negation would be simplest and it might be the first step towards allowing full logical expressions in combination of trees in a multi-tree queries in the future. The conjunctive normal form[26] would require more fundamental changes in the search algorithm.

Corpus-Wide Comparing and Statistics

Netgraph query language has no support for corpus-wide searching in the sense of comparing different trees in the corpus. It is not possible to search e.g. for a tree with the greatest number of nodes in the corpus. Or for a tree with the longest path from the root to a leaf in the corpus. Yet, some linguists might be interested in such a kind of working with the treebank. It is of course already possible to set a series of queries, searching first for trees with more than e.g. 50 nodes, increase the number in subsequent queries and thus finally find the biggest tree. Nevertheless, a more direct method would be nice.

Also the statistics that are acquired during the searching might be richer. The language might have support for specifying a part of the query that further statistics might be acquired about. The query tool might then provide statistics e.g. what types of nodes appear (and at what counts) at a certain position in the trees. (Thus providing e.g. a list of all possible sons of a Predicate along with numbers how often they appear.)

25 A logical formula is considered to be in the disjunctive normal form (DNF) if and only if it is a disjunction of one or more conjunctions of one or more literals. The only propositional operators in DNF are AND, OR, and NOT. The NOT operator can only be used as a part of a literal. In our case, a literal means a tree.

26 A logical formula is considered to be in the conjunctive normal form (CNF) if and only if it is a conjunction of one or more disjunctions of one or more literals. The only propositional operators in CNF are AND, OR, and NOT. The NOT operator can only be used as a part of a literal.

9.2.2 Speed

The linear searching implemented in Netgraph is quite sufficient for searching in PDT 2.0. Most queries are processed within 30 seconds or less (on the Netgraph public server). Only complex underspecified queries (presenting nodes without definite evaluation of their attributes) need more processing time.

PDT 2.0 consists of approx. 1.5 million tokens (on the analytical layer). It is unlikely that a manually annotated corpus might be of a higher-order size. Nevertheless, an automatically annotated corpus can easily be much larger. For example, the Czech National Corpus (Čermák 1997) consists of approx. 300 million[27] tokens. Simple arithmetic shows that searching in such a large corpus (if it was automatically parsed on the analytical layer) might take (300/1.5) * 30 seconds = 6000 seconds, which is almost 2 hours. Such a time is of course unacceptable.

There are two possible solutions of the problem and can be implemented separately or simultaneously:

- parallelization
  Since the searching is performed tree by tree independently, there is no problem in splitting searching of the entire data in many sub-parts.
- indexing
  A set of candidate trees from the corpus can be significantly reduced using indexing of some attributes.

Non of the methods, nor their combination, can solve the problem entirely. Parallelization is expensive and we can hardly expect to achieve 200 parallel searching processes for each user (which is the approximate number that would decrease the time of searching through the parsed Czech National Corpus back to 30 seconds). Indexing can be extremely effective for queries that specify indexed attributes but becomes useless for underspecified queries searching for structural phenomena.

9.2.3 Further Improvements

There are many other possible improvements, mainly to the tool, which are often wishes from users that have been collected during the years of development and usage of Netgraph and have not yet been implemented. The full To-Do list is much longer, let us only demonstrate the type of improvements to the tool that users wish, in a short selection from the To-Do list, without a special order:

- displaying list of found sentences
- saving/exporting trees in other formats than only FS
- highlighting the words in the sentence corresponding with the nodes matching the query

27 Czech National Corpus version SYN2006PUB

- command-line interface without GUI
- better support for external data sources (dictionaries etc.)
- support for scripts (plug-ins)
- cut and paste in the query
- auto-scrolling a large result tree so that a node matching a specified query node is displayed (especially useful for flat morphological “trees”, trees without a structure where all nodes depend directly on the root)

10 Appendixes

The following appendixes have been enclosed to this work:

- Appendix A: Publications about Netgraph
- Appendix B: FS File Format Description
- Appendix C: FS Query Format Description
- Appendix D: List of Attributes in PDT 2.0
- Appendix E: Other Usages of Netgraph
- Appendix F: Installation and Usage of Netgraph – A Quick How-To

10.1 Appendix A: Publications about Netgraph

This is a list of publications about Netgraph (or mentioning Netgraph) written or co-written by the author of this book, ordered from the most recent to older ones. A short description of the content of each paper is offered.

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.*

Linguistic phenomena annotated on all layers of PDT 2.0 are studied in the paper and a list of requirements on a query language is formulated here.

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.*

This paper presents the most complex linguistic phenomena annotated on the tectogrammatical layer of PDT 2.0 and shows how it can be searched for and studied with Netgraph.

Mírovský J. (2008b): Netgraph - Making Searching in Treebanks Easy. *In: Proceedings of the Third International Joint Conference on Natural Language Processing (IJCNLP 2008), Hyderabad, India, 8th - 10th January 2008, pp. 945-950.*

The paper presents Netgraph query language and shows how its advanced techniques can be used for searching for important linguistic phenomena.

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.*

Netgraph query language is presented in the paper and all meta-attributes are listed here. A comparison to TGrep is offered, all TGrep predicates are translated to Netgraph query language.

Mírovský J., Panevová J. (2008): Learning to Search in Prague Dependency Treebank. *In: Proceedings of Grammar and Corpora 2007, Liblice, Czech Republic, 25th - 27th September 2007, pp. 105-111.*

In this paper, we demonstrate how the Prague Dependency Treebank can be queried with Netgraph. New meta-attributes are introduced.

Mírovský J. (2006): Netgraph: a Tool for Searching in Prague Dependency Treebank 2.0. *In Proceedings of The Fifth International Treebanks and Linguistic Theories conference (TLT 2006), Prague, pp. 211-222.*

In this paper, Netgraph query language is presented along with a detailed description of meta-attributes. Hidden nodes, as well as references, are first introduced here.

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.*

This paper (among other topics) shows the usage of Netgraph for searching in the Prague Arabic Dependency Treebank.

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, Sofia, Bulgaria and Tuebingen, Germany, Sozopol, Bulgaria, 20th and 21st September 2002, pp. 114-122.*

It offers an introduction to the inner architecture of the Netgraph server. It also presents the query language and introduces first meta-attributes.

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.*

This paper introduces the client-server architecture of Netgraph and the basics of the query language.

10.2 Appendix B: FS File Format Description

The origin of this description of the syntax of FS File Format has been taken from CD-ROM Prague Dependency Treebank 1.0 (Hajič et al. 2001a). It has been updated to the current state of the format, used in Netgraph (and in TrEd (Pajas 2007)).

FS files serve for encoding the tree annotation of sentences in a natural language. Each FS file contains a sequence of trees, which represent the sentences. Each node is described by a set of attributes.

The names and data types of particular attributes are not a part of the FS format. Rather, each FS file has a header that defines attributes for its tree nodes locally.

10.2.1 Notes on Metasyntax

The non-terminal symbols are enclosed in "<" and ">" characters, terminal symbols or strings of terminal symbols are enclosed in double quotes. A C-like notation is used inside the quotes, thus "\t" means the character with code 9, i.e. HTAB. The character "\n" represents the end of line regardless of the platform, i.e. it matches not only real "\n" in its C sense, but also "\r\n" (DOS-Windows EOL), or even "\r".

The unary postfix operators "*", "+" and "?" mean that the operand appears n-times in a row, where n>=0 for *, n>0 for +, and n is 0 or 1 for ?.

In contexts where a non-terminal can be interpreted as a set, the binary operator "-" can be used. It denotes a difference of two sets.

10.2.2 The FS File Structure

The FS file contains a header with node attribute definitions, and a sequence of trees. Anything following the trees is considered a configuration for an editor and is ignored in Netgraph.

```
<fs-file> ::=
     <encoding-line>? <definition-line>+ "\n"+ (<tree> "\n")+ <editor-
     configuration>?
<encoding-line> ::=
     "@E " <encoding>
<encoding> ::=
     "utf-8"
```

Netgraph only accepts files encoded in UTF-8.

10.2.3 Identifiers, Attribute Names and Values

An identifier is one of the main elements of the FS file syntax. It is a string of arbitrary characters starting by the first character and ending before the first functional character. Functional characters can be parts of identifiers when they are escaped by a backslash (the backslash used for escaping a special character is not a part of the identifier).

Note: The length of identifiers is limited, the limit depends on the usage. In Netgraph, an attribute name is limited to 30 bytes, an attribute value it is limited to 5000 bytes.

```
<attribute-name> ::=
      <identifier>
<attribute-value> ::=
      <identifier>
<identifier> ::=
      <identifier-character>+
<identifier-character> ::=
      <normal-character> | <escaped-character>
<functional-character> ::=
      "\" | "=" | "," | "[" | "]" | "|" | "<" | ">" | "!"
<normal-character> ::=
      <any-character> - <functional-character>-"\n"
<escaped-character> ::=
      "\" (<any-character> - "\n")
```

10.2.4 Node Attributes Definition

The beginning of each file contains a header with definitions of the attributes which can appear in tree nodes. Each header line begins with the "@" character. A capital letter follows, denoting properties of the attribute, then a space and the attribute name. For example "@P m/lemma".

```
<definition-line> ::=
      ("@" <property> " " <attribute-name> "\n") |
      ("@L" " " <attribute-name> "|" <values> "\n")
<property> ::=
      "K" | "P" | "O" | "N" | "V" | "W" | "H"
<values> ::=
      <attribute-value> ("|" <values>)?
```

Properties

property	*description*
K	A key attribute. The word "key" does not really mean anything except "this has no specific properties".
P	A positional attribute. All other attributes require that their name is written before their value in the data (e.g. a/ord=7). Positional attributes do not. The name of a positional attribute is figured out of the relative position of its value with respect to the previous values (see details below in the paragraph "Node").

property	*description*
O	An obligatory attribute. Its value has to be non-empty for every node (the empty string is the default value for all attributes). Thus the value must appear in the data.
L	A list attribute. Such an attribute can only have a value from a predefined list, or be empty. The values cannot be repeated in the definition of the list.
H	A hiding attribute. Nodes that have value "true" in this attribute are hidden.
N	A numeric attribute (the value is a non-negative real number), specifying the order of the nodes in the tree from left to right. Maximally one such attribute per FS file can be defined.
W	Another numeric attribute. It denotes the order of words in the sentence. If it is not defined in the header, the attribute with the property @N (which is obligatory) is used.
V	A value attribute. The linear form of the sentence is assembled from values of this attribute, the values are ordered according to an attribute with the property @W. Maximally one such attribute per FS file can be defined.

More than one property can be defined for one attribute. The definition lines with all the properties need not follow each other in the file header. They must however fulfil the following constraints:

- Only one `@V` attribute per file can be defined.
- Only one `@W` attribute per file can be defined.
- Only one `@N` attribute per file can be defined.
- The `@N` property cannot be combined with other properties. Nevertheless, the `@N` attribute has automatically the properties `@P` and `@O` as well.
- An attribute cannot be both `@V` and `@L`.
- `@L` must be the last property defined for an attribute but it cannot be the only property of that attribute.

10.2.5 A Tree

Trees are described in the usual parentheses notation, i.e. after the description of a node, the parenthesized comma-separated list of its sons (or their subtrees) follows.

The order of the brothers is not significant, since the attribute with property `@N` is used for controlling the order of nodes.

```
<tree> ::=
     <node> ("(" <children> ")")?
<children> ::=
     <tree> ("," <children>)?
```

10.2.6 A Node

Besides the pure syntax, it is also necessary to check the relations between the element `<attributes>` and the definitions of the respective attributes in the header of the file. The constraints following from these relations are described below.

```
  <node> ::=
     <attribute-set> ("|" <node>)?
<attribute-set> ::=
     "[" <attributes>? "]"
<attributes> ::=
     <attribute> ("," <attributes>)?
<attribute> ::=
     (<attribute-name> "=")? <values>
<values> ::=
     <attribute-value> ("|" <values>)
```

The element `<attributes>` must fulfil the following constraints (based on the particular definition of attributes in the file header):

- The attribute name is required for non-positional attributes.
- If the attribute name is not present it is necessary to figure out the attribute the value belongs to. It is the first positional attribute whose definition in the header follows the definition of the last read attribute (positional or not).
- The identifier in the `<attribute-name>` element must equal to a name of an attribute defined in the header.
- No attribute can be read more than once.
- The identifier representing a value of a numeric attribute can contain only non-negative real numbers
- The value of a `@L` attribute must be one of the predefined values from the definition of the attribute.
- Values of all obligatory attributes (with property `@O`) have to be defined.

10.3 Appendix C: FS Query Format Description

The syntax of FS Query Format is almost identical to FS File Format (described in “Appendix B: FS File Format Description”). We therefore only show the different parts.

10.3.1 The FS Query Structure

The FS Query contains a header with node attribute definitions, a single tree or a logical combination and a sequence of trees. FS Queries are always encoded in UTF-8, therefore the encoding line is missing.

```
<fs-query> ::=
     <definition-line>+ "\n"+ <query-definition>
<query-definition> ::=
     <tree> | <multi-tree-query>
<multi-tree-query> ::=
     <logical-combination> ("\n" <tree>)+
<logical-combination> ::=
     "AND" | "OR"
```

The syntax of the header (<definition-line>) is identical to its definition in FS File Format.

In Netgraph, the user only creates <query-definition>. The header is generated automatically. All attributes in FS Query in Netgraph are positional and non is obligatory.

The syntax of the tree (<tree>) is the same as in FS File Format, with the exception of definition of node (<node>) and attribute value (<attribute-value>), see below.

10.3.2 A Node

The definition of a node (<node>) in FS Query Format differs from FS File Format only in allowing other relations than "=".

```
<node> ::=
     <attribute-set> ("|" <node>)?
<attribute-set> ::=
     "[" <attributes>? "]"
<attributes> ::=
     <attribute> ("," <attributes>)?
<attribute> ::=
     (<attribute-name> <relation>)? <values>
<values> ::=
     <attribute-value> ("|" <values>)
<relation> ::=
     "=" | "!=" | "<" | "<=" | ">" | ">="
```

The same constraints as in FS File Format apply to the element `<attributes>`, with the exception of a numeric attribute, which can contain any value.

10.3.3 Attribute Values

The syntax of the attribute name (`<attribute-name>`) is identical to its definition in FS File Format, only the definition of attribute value differs in allowing regular expressions, arithmetic operations and references.

Note: The length of identifiers is limited, the limit depends on the usage. In Netgraph, an attribute name is limited to 30 bytes, an attribute value it is limited to 5000 bytes.

```
<attribute-value> ::=
      <regular-expression-value> | <value>
<regular-expression-value> ::=
      """ <perl-like-regular-expression> """
<value> ::=
      <one-value> (<operator> <value>)?
<one-value> ::=
      (<identifier-character> | <reference>)+
<identifier-character> ::=
      <normal-character> | <escaped-character>
<functional-character> ::=
      "\" | "=" | "," | "[" | "]" | "|" | "<" | ">" | "!"
<normal-character> ::=
      <any-character> - <functional-character>-"\n"
<escaped-character> ::=
      "\" (<any-character> - "\n")
<reference> ::=
      "{" <node-name> "." <attribute-name> ("." <position>)? "}"
<node-name> ::=
      <identifier-character>+
```

`<perl-like-regular-expression>` is a regular expression defined in Hazel (2007) with `<functional-attributes>` escaped with `"\"`. `<position>` is a positive natural number.

10.4 Appendix D: List of Attributes in PDT 2.0

This appendix contains a list of all attributes in PDT 2.0, available in Netgraph, along with their brief description.

Not all attributes from the lower layers are accessible at the hidden nodes from the tectogrammatical layer. For those that are, the names that are used at the hidden nodes are noted in parentheses.

10.4.1 The Word Layer

w/token (w/token at hidden nodes)

A word token as it appears in the source data, even with misprints. Words, numbers, punctuation marks all form individual tokens.

w/no_space_after (w/no_space_after at hidden nodes)

This attribute contains value "`1`" if there is no space between the actual token and the next token in the data (e.g. there is usually no space between the last word in the sentence and the full stop).

w/id

A unique identifier of the word token (the position in the data).

10.4.2 The Morphological Layer

For a detailed description, see Hana et al. (2005).

m/form (m/form at hidden nodes)

A word token copied from `w/token` with the original capitalization but with corrected misprints.

m/form_change

If the attribute `m/form` differs from `w/token`, this attribute describes the nature of the change. For example, for corrected misprints it contains the value `"spell"`.

m/id

A unique identifier of the morphological annotation of the `m/form`.

m/lemma (m/lemma at hidden nodes)

A base form of the `m/form`. For example, for nouns, `m/lemma` contains the noun in the nominative, singular and non-negative. Together with `m/tag`, it can be used to re-generate the original form of the token.

If several “different” words have the same base form, the lemmas are distinguished by a variant, often followed by a short description. If the variant is present, it is always expressed by a number and is separated from the base form by a dash (`"-"`). The comment may follow after `"_"`, `"^"` or `"~"` (and may even appear at lemmas without variants). For example, `stát-1_^(státní_útvar)` is a different lemma from `stát-2_^(něco_se_přihodilo)`, although the base form is the same. (In English: `stát-1_^(state_system)`, `stát-2_^(something_happened)`)

Note: Netgraph automatically searches for different variants and comments if only a base form is set in the query. This behaviour can be changed in the settings.

m/src.rf

The source of the morphological annotation. In PDT 2.0, it is always `"manual"`.

m/tag (m/tag at hidden nodes)

A positional morphological tag describing morphological categories of the form `(m/form)`. It is a string of 15 characters. Every position encodes one morphological category using one character (mostly upper case letters or numbers); if not specified, the position contains a dash (`"-"`):

position	*description*	*examples of values*
1	Part of speech	N – noun, A – adjective, V – verb, R – preposition
2	Detailed part of speech	# – sentence boundary, R – preposition, V – vocalized preposition
3	Gender	F – feminine, I – masculine inanimate, M – masculine animate, N – neuter
4	Number	D – dual, S – singular, P – plural, X – any
5	Case	1 – nominative, 2 – genitive, ..., 7 – instrumental, X – any
6	Possessor's gender	F – feminine, M – masculine animate, Z – non feminine
7	Possessor's number	S – singular, P – plural, X – any
8	Person	1 – 1st person, 2 – 2nd person, 3 – 3rd person, X – any

position	*description*	*examples of values*
9	Tense	F – future, P – present, R – past, H – past or present
10	Degree of comparison	1 – positive, 2 – comparative, 3 – superlative
11	Negation	A – affirmative, N – negated
12	Voice	A – active, P – passive
13	Reserve	–
14	Reserve	–
15	Variant, style	1,2 – variant, 5,6,7 – colloquial, 8 – abbreviation

10.4.3 The Analytical Layer

For a detailed description, see Hajič et al. (1999).

afun (a/afun at hidden nodes)

`Afun` is a principle attribute on the analytical layer. It contains an analytical function, in other words, a type of relation to the governing node. The following table, which is taken from Hajič et al. (2006), shows possible values of the attribute.

afun	*description*
Pred	Predicate, a node not depending on another node; depends on #
Sb	Subject
Obj	Object
Adv	Adverbial
Atv	Complement (so-called determining) technically hung on a non-verb. element
AtvV	Complement (so-called determining) hung on a verb, no 2nd gov. node
Atr	Attribute
Pnom	Nominal predicate, or nom. part of predicate with copula *be*
AuxV	Auxiliary vb. *be*
Coord	Coord. node
Apos	Apposition (main node)
AuxT	Reflex. tantum
AuxR	Ref., neither Obj nor AuxT, Pass. refl.

afun	*description*
AuxP	Primary prepos., parts of a secondary p.
AuxC	Conjunction (subord.)
AuxO	Redundant or emotional item, 'coreferential' pronoun
AuxZ	Emphasizing word
AuxX	Comma (not serving as a coordinating conj.)
AuxG	Other graphic symbols, not terminal
AuxY	Adverbs, particles not classed elsewhere
AuxS	Root of the tree (#)
AuxK	Terminal punctuation of a sentence
ExD	A technical value for a deleted item; also for the main element of a sentence without predicate (Externally-Dependent)
AtrAtr	An attribute of any of several preceding (syntactic) nouns
AtrAdv	Structural ambiguity between adverbial and adnominal (hung on a name/noun) dependency without a semantic difference
AdvAtr	Dtto with reverse preference
AtrObj	Structural ambiguity between object and adnominal dependency without a semantic difference
ObjAtr	Dtto with reverse preference

eparents (a/eparents at hidden nodes)

The attribute `eparents` contains identifiers (values of the attribute `id`) of effective linguistic parents of the node. If there are more than one effective parent, alternative values are used.

eparents_diff (a/eparents_diff at hidden nodes)

The attribute `eparents_diff` contains identifiers (values of the attribute `id`) of effective linguistic parents of the node only if the effective parents differ from the technical parent of the node in the tree.

id (a/id at hidden nodes)

A unique identifier of the node in the corpus. At the root, it is a unique identifier of the analytical tree.

is_member (a/is_member at hidden nodes)

The attribute `is_member` is set to `"1"` if the node is a member of a coordination or an apposition.

is_parenthesis_root

If set to `"1"`, this attribute denotes a root of a parentheses (an inserted word or clause).

ord (a/ord at hidden nodes)

The attribute `ord` controls the order of nodes in the analytical tree from left to right. It may contain non-negative real numbers. It also controls the order of words in the sentence (the sentence is assembled from values of attribute `w/token`).

s.rf

The attribute `s.rf` is only used at the root of the tree. It contains a unique identifier of the sentence in the corpus.

- (a/parent at hidden nodes)

The attribute `a/parent` is only available at the hidden nodes in the tectogrammatical trees. It contains an identifier (value of the attribute `a/id`) of the analytical parent of the node (a technical parent of the node in the analytical tree).

- (a/ref_type at hidden nodes)

The attribute `a/ref_type` is only available at the hidden nodes in the tectogrammatical trees. It classifies the hidden node in relation to a given tectogrammatical node. Value `"lex"` means that this hidden node contributes most to the lexical meaning of its tectogrammatical counterpart. Each tectogrammatical node can have at most one hidden son with value `"lex"`. All other hidden sons have value `"aux"`, meaning that these analytical nodes have less lexical meaning and are rather auxiliary. The only exception is the only hidden son of the technical root of each tectogrammatical tree; the value of attribute `a/ref_type` of this hidden node is set to `"tree"`.

10.4.4 The Tectogrammatical Layer

For a detailed description, see Mikulová et al. (2006).

atree.rf

The attribute `atree.rf` only appears at the root of a tectogrammatical tree. It links the tectogrammatical layer with the analytical layer through a reference to an analytical tree. It contains a value of the attribute `id` of the root of the analytical tree, prefixed with `"a#"`.

compl.rf

The attribute `compl.rf` is used to record second dependency of predicative complements. It contains an identifier (value of attribute `id`) of a node of the tectogrammatical tree which the particular node also depends on (apart from the dependency expressed by an edge).

coref_gram.rf

The attribute `coref_gram.rf` is used to record the grammatical coreference. It contains an identifier of a node of (usually the same) tectogrammatical tree that the particular node grammatically corefers to.

coref_special

The attribute `coref_special` marks special types of the textual coreference in which the coreferred element is not represented by a node or a subtree of a tectogrammatical tree. Value `segm` indicates that the coreferred element is a segment, a larger section of a text. Value `exoph` indicates an exophoric reference, i.e. coreference in which the coreferred element is represented by a extratextual situation which is not further specified.

coref_text.rf

Like `coref_gram.rf`, but concerns the textual coreference.

deepord

The attribute `deepord` reflects the deep structure word order and controls the order of nodes in the tectogrammatical tree from left to right. It may contain non-negative real numbers.

eparents

The attribute `eparents` contains identifiers (values of the attribute `id`) of effective linguistic parents of the node. If there are more than one effective parent, alternative values are used.

eparents_diff

The attribute `eparents_diff` contains identifiers (values of the attribute `id`) of effective linguistic parents of the node only if the effective parents differ from the technical parent of the node in the tree.

functor

A principle attribute on the tectogrammatical layer. Functors represent semantic values of syntactic dependency relations; they express the functions of individual modifications in the sentence. There are too many possible values to be listed here. Let us only present (from our point of view) the most important functors (most of them have been used in the examples in this book). For details on all functors, see Mikulová et al. (2006).

functor	*description*
ACT	argument - Actor
ADDR	argument - Addressee
AIM	adjunct expressing purpose
APPS	the root node of an appositional structure
BEN	adjunct expressing that sth is happening for the benefit (or disadvantage) of sb/sth
CAUS	adjunct expressing the cause (of sth)
COMPL	adjunct - predicative complement
COND	adjunct expressing a condition (for sth else to happen)
CONJ	paratactic structure root node - simple coordination/conjunction
CPHR	the nominal part of a complex predicate
DIR1	directional adjunct - answering the question "odkud (=where from?)"
DIR2	directional adjunct - answering the question "kudy (=which way?)"
DIR3	directional adjunct - answering the question "kam (=where to?)"
DISJ	paratactic structure root node - disjunctive relation
DPHR	the dependent part of an idiomatic expression
EFF	argument - Effect
LOC	locative adjunct - answering the question "kde (=where?)"
MANN	adjunct expressing the manner (of doing sth)
MEANS	adjunct expressing a means (of doing sth)
ORIG	argument - Origo
PAT	argument - Patient
PREC	atomic expression referring to the preceding context
PRED	effective root node of an independent verbal clause (which is not parenthetical)

functor	*description*
RHEM	atomic expression - rhematizer
RSTR	adnominal adjunct modifying its governing noun
TSIN	temporal adjunct - answering the question "od kdy? (=since when?)"
TTILL	temporal adjunct - answering the question "do kdy? (=until when?)"
TWHEN	temporal adjunct - answering the question "kdy? (=when?)"

Grammatemes (attributes gram/*)

Grammatemes are tectogrammatical correlates of morphological categories. All grammatemes start with the prefix `gram/`. All 16 grammatemes are listed in the following table, along with a very short and sometimes simplified description. For further information, see Mikulová et al. (2006).

grammateme	*description*
gram/aspect	a tectogrammatical correlate of the morpho-lexical category of aspect
gram/degcmp	a tectogrammatical correlate of the (adjectival/adverbial) category of degree
gram/deontmod	expresses the fact that the event is understood as necessary, possible, permitted etc.
gram/dispmod	signals whether the clause expresses the so called dispositional modality
gram/gender	a tectogrammatical correlate of the morphological category of gender
gram/indeftype	a semantic feature in which the pronoun / adverb / numeral in question differs from the t-lemma it is represented by
gram/iterativeness	marks multiple/iterated events
gram/negation	expresses whether a given semantic noun / adjective / adverb occurs in its negated or non-negated form in the surface structure of the sentence
gram/number	a tectogrammatical correlate of the morphological category of number
gram/numertype	a semantic feature in which the given numeral is distinct from the corresponding cardinal numeral

grammateme	*description*
gram/person	a tectogrammatical correlate of the morphological category of person
gram/politeness	signals a polite usage of pronouns
gram/resultative	marks the so called possessive passive
gram/sempos	a semantic part of speech
gram/tense	a tectogrammatical correlate of the morphological category of tense
gram/verbmod	a tectogrammatical correlate of the morphological category of (verbal) mood

id

A unique identifier of the tectogrammatical node in the corpus. At the root, it is a unique identifier of the tectogrammatical tree.

is_dsp_root

The attribute `is_dsp_root` indicates (with value `"1"` or `"0"`) whether a node is a root of a direct speech.

is_generated

The attribute is_generated indicates (with values `"1"` or `"0"`) whether a node represents a word on the surface layer.

is_member

The attribute `is_member` is set to `"1"` if the node is a member of a coordination or an apposition.

is_name_of_person

The attribute `is_name_of_person` is set to `"1"` at all nodes representing expressions that are constituents of proper names of people.

is_parenthesis

The attribute `is_parenthesis` is set to `"1"` at all nodes that are a part of a parentheses (an inserted word or clause).

is_state

The attribute `is_state` is set to `"1"` at all modifications expressing the meaning of “being in a state” or “getting into a state”.

nodetype

The attribute `nodetype` distinguishes eight types of tectogrammatical nodes: the technical root node (the value `"root"`), the atomic node (`"atom"`), the paratactic structure root node (`"coap"`), the list structure root node (`"list"`), the node representing a foreign-language expression (`"fphr"`), the node representing the dependent part of an idiomatic expression (`"dphr"`), the complex node (`"complex"`), and the quasi-complex node (`"qcomplex"`).

quot/set_id

For each text in quotation marks, a unique identifier is selected. For all nodes representing the relevant text, the identifier is stored in the attribute `quot/set_id`.

quot/type

The attribute `quot/type` specifies the type of usage of a quotation mark. There are five possible values: citation (`"citation"`), direct speech (`"dsp"`), metalinguistic expression (`"meta"`), proper noun identifier (`"title"`), other usage (`"other"`).

sentence

The attribute `sentence` only appears at the root of the tectogrammatical tree. It contains the linear form of the whole sentence.

sentmod

The attribute `sentmod` contains information about the sentential modality. It is assigned to a node on the basis of its position in the tree. Possible values are: indicative mood (`"enunc"`), exclamation (`"excl"`), optative (desiderative) mood (`"desid"`), imperative mood (`"imper"`), and interrogative mood (`"inter"`).

subfunctor

The attribute `subfunctor` describes a semantic variation within a particular functor. Possible values of attribute `subfunctor` depend on the particular functor.

t_lemma

For nodes representing lexical units present at the surface layer of the sentence, the value of the attribute `t_lemma` is the basic form of the lexical unit. For newly established nodes, an artificial value (one of almost 30 possible) is assigned to the attribute `t_lemma`.

tfa

The attribute `tfa` represents the contextual boundness of the node. Possible values are: the contrastive contextually bound expression (`"c"`), the contextually non-bound expression (`"f"`), the non-contrastive contextually bound expression (`"t"`).

val_frame.rf

The attribute `val_frame.rf` contains an identifier of a valency frame corresponding to the given meaning of the given word.

hide

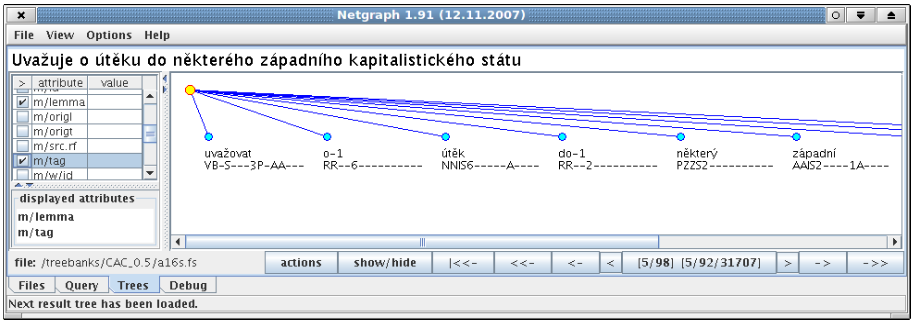
The attribute `hide` distinguishes the hidden nodes. Nodes with value `"true"` are hidden and are not considered a part of the tectogrammatical tree.

10.5 Appendix E: Other Usages of Netgraph

Netgraph query tool 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 as well as for constituent structure trees, provided the treebank is transformed to FS File Format, and also other kinds of usage are possible. We mention some (not all) of the usages in this appendix.

10.5.1 Morphological "Trees" of the Czech Academic Corpus 1.0

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, where all nodes depended on a technical root, as shown in the picture:



Uvažuje o útěku do některého západního kapitalistického státu

file: /treebanks/CAC_0.5/a16s.fs

Next result tree has been loaded.

In Czech: *Uvažuje o útěku do některého západního kapitalistického státu*
In English: *He thinks about flying to some west capitalistic country*

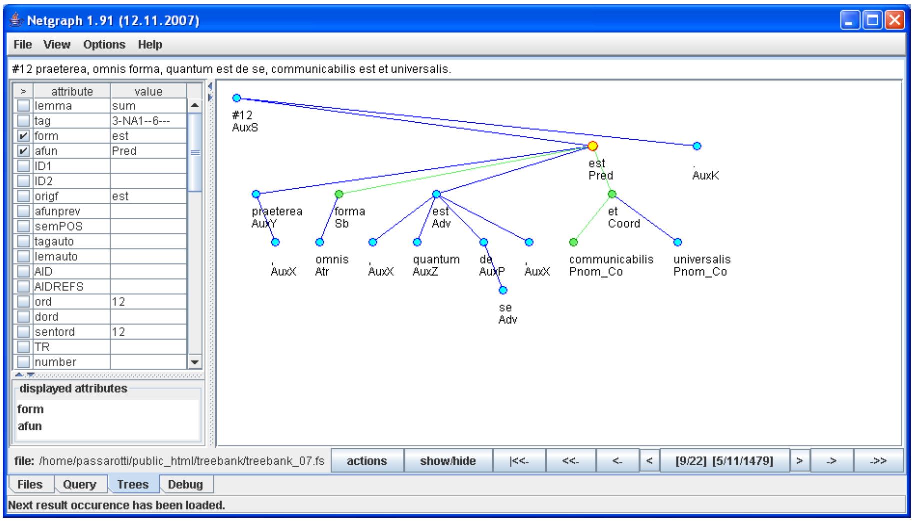
During the preparation of the second version of the Czech Academic Corpus (version 2.0), which is still going on, Netgraph has been used for searching for errors on the analytical layer. The annotation is almost identical to the analytical layer of PDT 2.0, therefore we do not include a picture.

10.5.2 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. [28] 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, as shown in the picture:



#12 praeterea, omnis forma, quantum est de se, communicabilis est et universalis.

attribute	value
☐ lemma	sum
☐ tag	3-NA1--6---
☑ form	est
☑ afun	Pred
☐ ID1	
☐ ID2	
☐ origf	est
☐ afunprev	
☐ semPOS	
☐ tagauto	
☐ lemauto	
☐ AID	
☐ AIDREFS	
☐ ord	12
☐ dord	
☐ sentord	12
☐ TR	
☐ number	

file: /home/passarotti/public_html/treebank/treebank_07.fs

Next result occurence has been loaded.

In Latin: *praeterea, omnis forma, quantum est de se, communicablilis est et universalis.*
In English: *In addition, every form is on its own communicable and universal.*

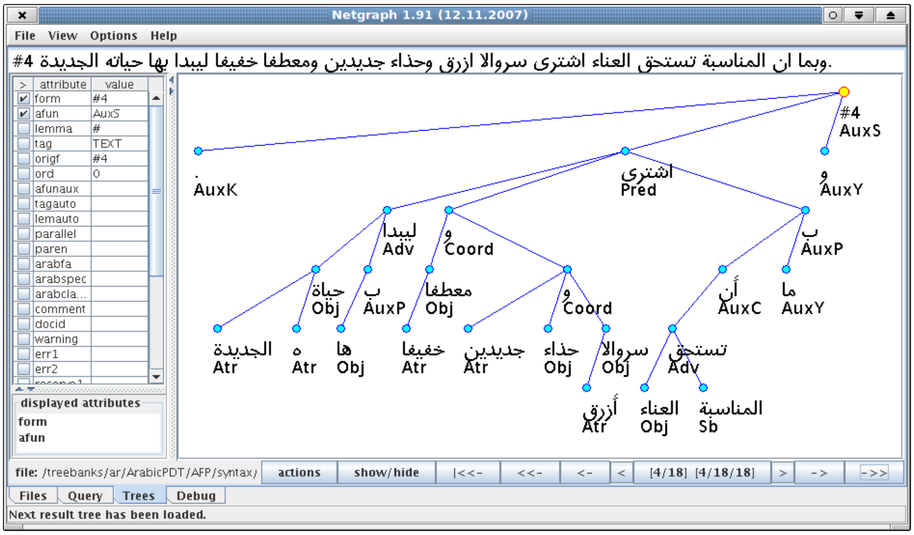
10.5.3 Arabic Trees

In the year 2003, Netgraph was installed in LDC (Linguistic Data Consortium) in Philadelphia, University of Pennsylvania[29], to be used with their Arabic treebank. In cooperation with LDC, the Prague Arabic Dependency Treebank (Smrž et al. 2005)

28 `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).

29 LDC – `http://www.ldc.upenn.edu`

was developed at ÚFAL (Institute of Formal and Applied Linguistics) at Charles University in Prague. 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, as demonstrated in the picture:



#4 وبما ان المناسبة تستحق العناء اشترى سروالا ازرق وحذاء جديدين ومعطفا خفيفا ليبدا بها حياته الجديدة.

attribute	value
✓ form	#4
✓ afun	AuxS
lemma	#
tag	TEXT
origf	#4
ord	0
afunaux	
tagauto	
lemauto	
parallel	
paren	
arabfa	
arabspec	
arabcla...	
comment	
docid	
warning	
err1	
err2	

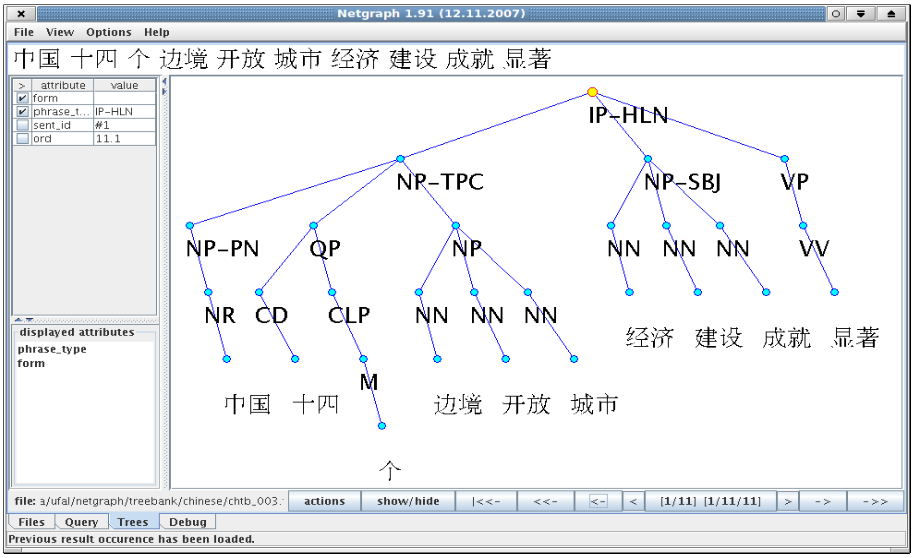
displayed attributes: form, afun

file: /treebanks/ar/ArabicPDT/AFP/syntax/ [4/18] [4/18/18]

Next result tree has been loaded.

10.5.4 Chinese Treebank

Netgraph has been also used for 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 was necessary. It is an example of usage of Netgraph with constituent-structure trees:



中国 十四 个 边境 开放 城市 经济 建设 成就 显著

attribute	value
✓ form	
✓ phrase_t...	IP-HLN
sent_id	#1
ord	11.1

displayed attributes: phrase_type, form

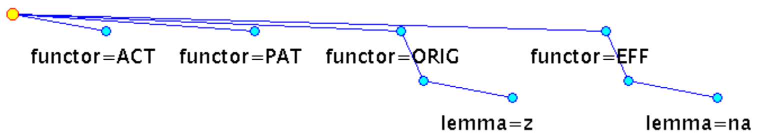
file: a/ufal/netgraph/treebank/chinese/chtb_003. [1/11] [1/11/11]

Previous result occurence has been loaded.

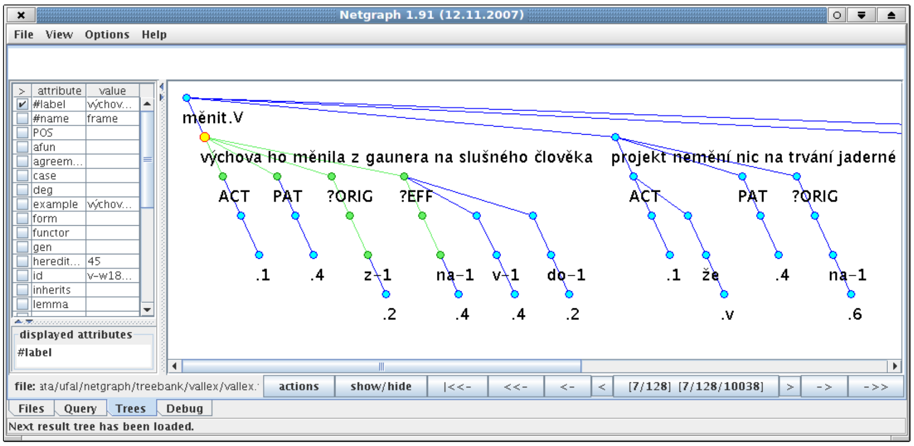
10.5.5 Vallex

Vallex is a valency lexicon of Czech (Lopatková et al. 2006). A recent usage of Netgraph for a 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 Origo and an Effect. The query also requires that on the surface, the Origo 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*$_1$ *ho*$_2$ *měnila* *z*$_3$ *gaunera*$_4$ *na*$_5$ *slušného člověka*$_6$
In English: *education*$_1$ *was changing him*$_2$ *from*$_3$ *a_scrounger*$_4$ *to*$_5$ *a decent man*$_6$

10.6 Appendix F: Installation and Usage of Netgraph – A Quick How-To

In this appendix, we show:

- how to quickly install the Netgraph client (optionally the Netgraph server too)
- how to connect to the public Netgraph server for PDT 2.0 (or to the local Netgraph server for PDT 2.0 sample data)
- how to enter a simple query and browse the result trees

10.6.1 Installation

These are only quick instructions how to install the client (optionally also the server) in order to access the public (or local) Netgraph server. For details, please consult the installation instructions on the Netgraph home web page: `http://quest.ms.mff.cuni.cz/netgraph`.

Java 2 Installation

Please note that Java Runtime Environment (JRE) from Sun Microsystems must be installed in order to run the Netgraph client. It is not a part of Netgraph installation programs – it must be installed separately. At least version 1.5 is needed. The newest version of JRE for various platforms/systems can be downloaded from `http://java.sun.com/javase/downloads/`. Please note that the Netgraph client may not work with other-parties versions of Java Runtime Environment. In case of troubles, please check which version of Java is started from the installed icon of the Netgraph client.

Netgraph Client/Server Installation

For Linux and MS Windows, installation programs for the client and/or the server can be downloaded from the Netgraph home web page. Download and run the appropriate version of the installation program for your system:

- `Netgraph-1.95-PDT20Sample-Linux-x86-Install` - for Linux
- `Netgraph-1.95-PDT20Sample-Windows-Setup.exe` - for MS Windows

For other systems, please consult the installation instructions on the pages.

During the installation, the user can choose parts of the program to install. Either only the Netgraph client is installed (to access the public Netgraph server[30]), or the Netgraph server along with the Prague Dependency Treebank 2.0 sample data are installed, or both the client and the server (along with the data). For accessing the public

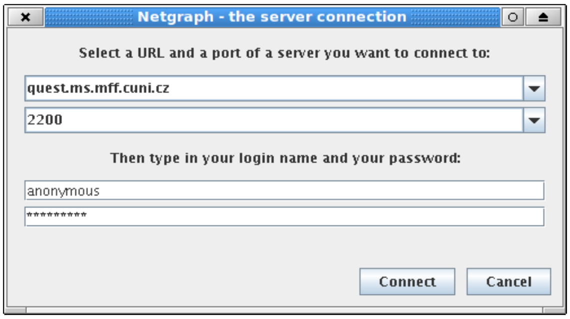
30 An internet connection is needed.

Netgraph server, choose only the installation of the client. At least the following icon should appear on your desktop, with the label "Netgraph client 1.95":

To access a locally installed Netgraph server and search in the Prague Dependency Treebank 2.0 sample data, choose the installation of the client and the server. Then, also the following icon appears on your desktop, with label "Netgraph server 1.95":

10.6.2 Connection to the Public Netgraph Server for PDT 2.0[31]

Start the Netgraph client (by clicking on the client icon). A dialog window appears:



Select a URL and a port of a server you want to connect to:

quest.ms.mff.cuni.cz

2200

Then type in your login name and your password:

anonymous

****

Fill-in the following connection and login information:

- server: `quest.ms.mff.cuni.cz`
- port: `2200` for the tectogrammatical trees (`2100` for the analytical trees)
- user (login name): `anonymous`
- password: `anonymous`

and click on the button "Connect" to establish a connection to the server.

10.6.3 Connection to the Local Netgraph Server for PDT 2.0 Sample Data

First, start the Netgraph server (by clicking on the server icon). A terminal window should appear with the following text:

```
The Netgraph server version 1.95 L (5.8.2008)
The server is trying to bind to the port: 2000 ... OK
The server has started and is waiting for connections.
```

31 You may also want to see a flash demonstration of the Netgraph client usage (on the Netgraph home page).

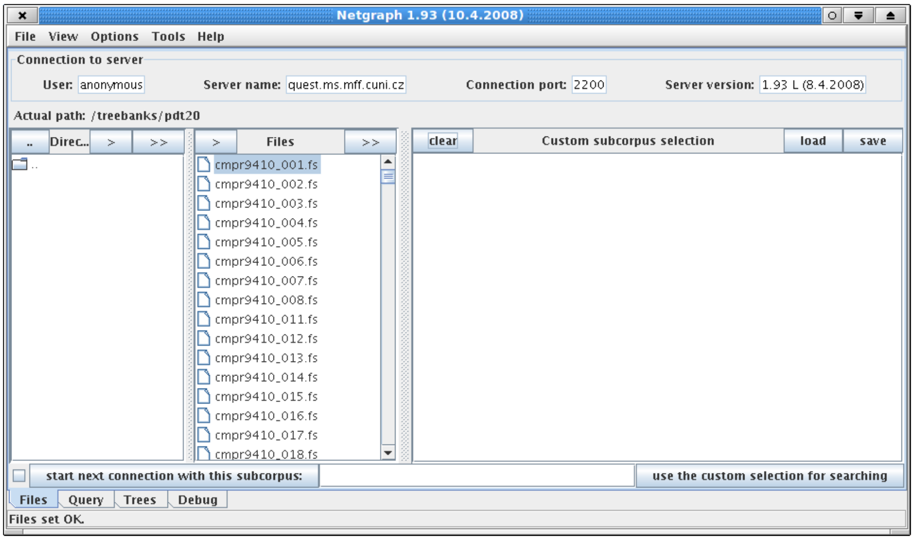
Then, start the Netgraph client (by clicking on the client icon). A dialog window just like for the public server appears. This time, fill-in this connection and login information:

- server: `localhost`
- port: `2000`
- user (login name): `anonymous`
- password: `anonymous`

and click on the button "Connect" to establish a connection to the server.

10.6.4 Selection of Files for Searching

After the connection to the server is established, the following window appears:


Netgraph 1.93 (10.4.2008)
File
View
Options
Tools
Help
Connection to server
..
Direc...
>
>>
Files
clear
Custom subcorpus selection
load
save
start next connection with this subcorpus:
use the custom selection for searching
Files
Query
Trees
Debug

User: anonymous   Server name: quest.ms.mff.cuni.cz   Connection port: 2200   Server version: 1.93 L (8.4.2008)

Actual path: /treebanks/pdt20

```
cmpr9410_001.fs
cmpr9410_002.fs
cmpr9410_003.fs
cmpr9410_004.fs
cmpr9410_005.fs
cmpr9410_006.fs
cmpr9410_007.fs
cmpr9410_008.fs
cmpr9410_011.fs
cmpr9410_012.fs
cmpr9410_013.fs
cmpr9410_014.fs
cmpr9410_015.fs
cmpr9410_016.fs
cmpr9410_017.fs
cmpr9410_018.fs
```

Files set OK.

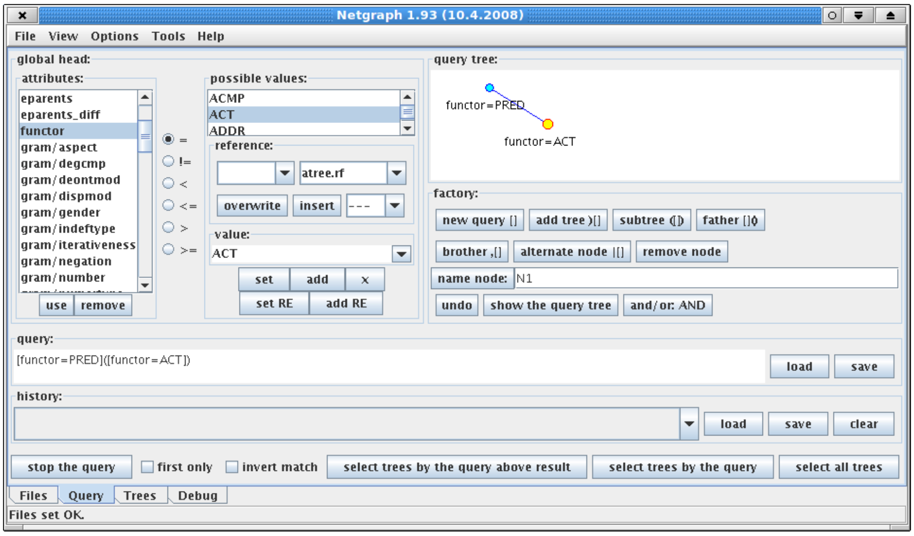
Files with trees for searching are listed and can be selected here. To select the whole treebank for searching, perform these steps (the first step is only applicable for accessing the locally installed Netgraph server):

- (Skip this first step if you are accessing the public Netgraph server.) Double-click on the directory "tectogrammatical" in the first column named "Directories". A list of files in the directory appears in the middle column.
- Click on the button ">>" in the middle column named "Files" to add all files with trees from the current directory to the selected files. The files appear in the right column named "Custom subcorpus selection".
- **Click on the button on the right bottom "use the custom selection for searching" to send the selection to the server.**[32]

32 This step is easily overlooked. Yet, it is essential and cannot be omitted.

10.6.5 Creation of a Simple Query

The second tab in the window named "Query" is automatically selected, as shown in the picture:



reference: atree.rf ---

value: ACT

name node: N1

query: [functor=PRED]([functor=ACT])

Files set OK.

The query is created here. On the left side, there is a list of available attributes. Possible values of the enumeration type of attributes are listed in the table "possible values". The graphical representation of the query is depicted in the right top corner in the panel "query tree". The textual representation of the query is in the text field "query". Both representations of the query are empty at first.

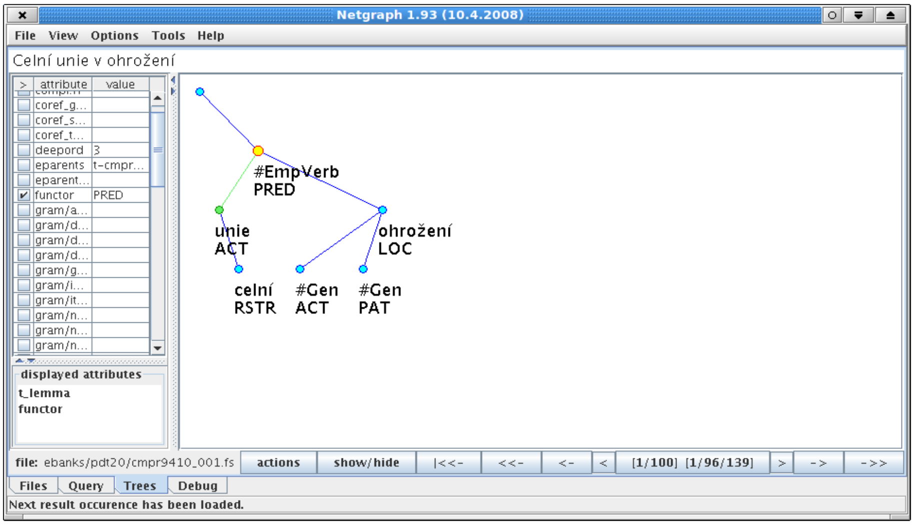
If you are connected to the server for the tectogrammatical trees, to create the query from the picture above, follow these steps[33]:

- click on the button "new query" in the panel "factory" on the right side; a node appears in the graphical representation of the query
- find the attribute "`functor`" in the list of attributes on the left side and double-click on it (alternatively, single-click on it and click on the button "use" at the bottom of the list); the name of the attribute appears in the textual version of the query (NOT yet in the graphical version)
- find the value "`PRED`" in the list of possible values and double-click on it (alternatively, single-click on the value and click on the button "set" below the list); an expression "`functor=PRED`" appears both in the graphical and in the textual representation of the query
- click on the button "subtree" in the panel "factory"; a son-node of the Predicate is created

33 On the other hand, it is possible to simply browse all trees from the selected files without setting a query, by clicking on the button "select all trees".

- double-click on the attribute "`functor`" in the list of attributes
- find the value "`ACT`" in the list of possible values and double-click on it; an expression "`functor=ACT`" appears both in the graphical and in the textual representation of the query at the son-node

The query should be created now just like in the picture above. Click on the button "select trees by the query" on the bottom to send the query to the server. The interface switches automatically to the tab "Trees", and the first result tree should appear (if you are searching in the locally installed sample data, a different tree appears):



A list of available attributes can be found on the left side. Choose some of the attributes to be displayed at the nodes in the trees, e.g. `t_lemma` and `functor`, to match the picture.

Buttons "<-" and "->" can be used for browsing the occurrences of the query in the result trees. Buttons "<<-" and "->>" skip multiple occurrences of the query in one tree, and buttons "<" and ">" can be used to browse the context trees.

The anonymous user has several restrictions:

- although potentially the whole corpus is searched, only first one hundred results are found
- result trees cannot be saved to the local disc
- the password cannot be changed

For the full access to the data without restrictions, a non-anonymous user account has to be created.[34]

To create another query, choose the "Query" tab from the list of tabs at the bottom of the window.

34 Contact the author of Netgraph to have a full account created: `mirovsky@ufal.mff.cuni.cz`.

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