Finite Structure Query: A Tool for Querying Syntactically Annotated Corpora

Stephan Kepser
SFB 441 – University of Tübingen
Nauklerstr. 35, 72074 Tübingen, Germany
kepser@sfs.uni-tuebingen.de

Abstract

Finite structure query (fsq for short) is a tool for querying syntactically annotated corpora. fsq employs a query language of high expressive power, namely full first order logic. It can be used to query arbitrary finite structures, not just trees.

1 Introduction

In recent years large amounts of electronic texts have become available providing a new base for empirical studies in linguistics and offering a chance to linguists to compare their theories with large amounts of utterances from "the real world". While tagging with morphosyntactic categories has become a standard for almost all corpora, more and more of them are nowadays annotated with refined syntactic information. Examples are the Penn Treebank (Marcus et al., 1993) for American English annotated at the University of Pennsylvania, the French treebank (Abeillé and Clément, 1999) developed in Paris, the NEGRA Corpus (Brants et al., 1999) for German annotated at the University of Saarbrücken, and the Tübingen Treebanks (Hinrichs et al., 2000) for Japanese, German and English from the University of Tübingen. To make these rich syntactic annotations accessible for linguists the development of powerful query tools is an obvious need and has become an important task in computational linguistics.

In this paper, we present finite structure query (fsq), a query tool for syntactically annotated corpora. Special emphasis in the development of fsq is layed on providing users a very powerful query language. Therefore full first-order logic was chosen as query language which allows arbitrary quantifications over nodes in a tree. This choice is to the authors' knowledge unique, no other query tool offers such an expressive power. It is on the other hand not arbitrary, we will show that there are interesting linguistic queries that require such an expressive power. The tool is developed for treebanks in NEGRA export format (Brants, 1997), but can be adopted to other corpus formats as well. The linguistic examples that follow are from the Tübingen Treebanks.

1.1 The Tübingen Treebanks

The Tübingen Treebanks, annotated at the University of Tübingen, comprise a German, an English and a Japanese treebank consisting of spoken dialogs restricted to the domain of arranging business appointments. In this paper, we focus on the German treebank (TüBa-D) (Stegmann et al., 2000; Hinrichs et al., 2000) that contains approximately 38.000 trees.

The treebank is part-of-speech tagged using the Stuttgart-Tübingen tag set (STTS) developed by Schiller et al. (1995). One of the design decisions for the development of the treebank was the commitment to reusability. As a consequence, the choice of the syntactic annotation scheme should not reflect a particular syntactic theory but rather be as theory-neutral as possible. Therefore a surface-oriented scheme was adopted to structure German sentences that uses the notion of topological fields in a sense similar to that of Höhle (1985). The verbal elements have the categories LK (*linke Klammer*) and VC (*verbal complex*), roughly everything preceeding the LK forms the "Vorfeld" VF, everything between LK and VC

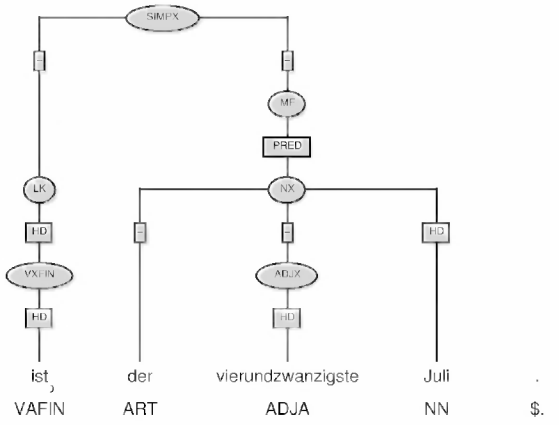

Figure 1: An example sentence from TüBa-D

forms the "Mittelfeld" `MF`, and the material following the `VC` forms the "Nachfeld" `NF`.

The corpus is annotated with syntactic categories as node labels, grammatical functions as edge labels and dependency relations. The syntactic categories follow traditional phrase structure and the theory of topological fields. An example of a tree can be found in Figure 1. To cope with the characteristics of spontaneous speech, the data structures in the Tübingen Treebanks are of a more general form than trees. For example, an entry may consist of several tree structures. It also may contain completely disconnected nodes. In contrast to NEGRA or the Penn Treebank, there are neither crossing branches nor empty categories.

The design goal of fsq has been to make this rich annotation queryable for linguists.

2 The Query Language of fsq

2.1 Syntax

In order to provide a high expressive power, full first-order logic is chosen as the query language for fsq. Atomic formulae express the syntactic annotation of a node in a tree or the relationships between nodes such as dominance or precedence. Hence, variables, words made of letters and digits, are supposed to range over nodes in a tree. The formula (or query) syntax is LISP-like. Let `x` and `y` be variables and let `T` be a token, `C` a category, `M` a morphological tag, and `F` a function from the annotation scheme. The atomic formulae are as follows:

- `(= x y)` "x equals y"
- `(> x y)` "x is the mother of y"
- `(>> x y)` "x dominates y"
- `(. x y)` "x immediately precedes y"
- `(.. x y)` "x precedes y"
- `(tok x T)` "x has token T"
- `(cat x C)` "x is of category C"
- `(mor x M)` "x has morphological tag M"
- `(fct x F)` "x is of function F"
- `(refl x y)` "there is a `refl`-secondary edge from x to y"

Trees in the treebank may have secondary edges. An example is the `refl` edge from the TüBa-D, which expresses a reference relation between infinitival verbs in a verb complex.

Complex formulae are constructed in a way normal for first-order logic. Let φ, $\varphi_1, \ldots, \varphi_n$, and ψ be formulae, and `x` a variable. Then the following are well-formed formulae:

- `(! φ)` (negation)
- `(& φ1 ... φn)` (conjunction)
- `(| φ1 ... φn)` (disjunction)
- `(-> φ ψ)` (implication)
- `(E x φ)` (existential quantification)
- `(A x φ)` (universal quantification)

A *query* is a closed formula, i.e., a formula where each variable is bound by some quantifier.

2.2 Semantics

One central idea of fsq is to regard a tree or tree-like structure as a finite first-order structure. The reason behind this decision is that a finite structure is the common denominator for existing treebanks and corpus formats. A closer look not only at the Tübingen Treebanks reveals that most of these corpora do not only contain *proper* trees in the mathematical sense. Some of these extensions of trees may be harmless. But some of them, like secondary relations, can, if used extensively, transcend the tree structure completely. A query approach that is designed to cope with all of these extensions therefore needs to opt for a general data structure. And the one data structure that fits

most easily with all treebank formats is that of a finite first-order structure.

One big advantage of the approach of regarding a "tree" as a finite first-order structure is that we are immediately given a very natural and widely understood semantics for queries, namely classical first-order logic semantics. The universe of a structure is a finite set of nodes. Relations like (immediate) dominance and precedence as well as secondary edges are interpreted as binary predicates over nodes. Relations like token, category, morphology, or function each introduce a finite family of unary predicates. Each such predicate stands for a particular token (or category or function) value like, e.g., the German word *Hannover*.

More formally, the signature for a treebank consists of the two binary relation symbols *Id, Ip* (for immediate dominance and immediate precedence), up to 8 binary relation symbols Sec_i for secondary relations, and finite families $(T_i)_{1\leq i\leq k}$, $(C_i)_{1\leq i\leq l}$, $(M_i)_{1\leq i\leq m}$, and $(F_i)_{1\leq i\leq n}$ of unary predicate symbols (for tokens, categories, morphological tags, and functions). A *tree* is a finite first-order interpretation of this signature, and a *treebank* is a finite sequence of trees.

Let $T = (N, (Id, Ip, Sec_1, \ldots, Sec_8, (T_i), (C_i), (M_i), (F_i)))$ be a tree, and X be a set of variables. A *variable assignment* $a : X \to N$ is a function that assigns a node from N to each variable. Let a be a variable assignment. The truth conditions of the atomic formulae are as follows: (Id^* (Ip^*) is the reflexive-transitive closure of Id (Ip).)

`(= x y)`	true, if $a(x) = a(y)$,
`(> x y)`	true, if $(a(x), a(y)) \in Id$,
`(>> x y)`	true, if $(a(x), a(y)) \in Id^*$,
`(. x y)`	true, if $(a(x), a(y)) \in Ip$,
`(.. x y)`	true, if $(a(x), a(y)) \in Ip^*$,
`(ref1 x y)`	true, if $(a(x), a(y)) \in Sec_j$ where Sec_j is the secondary relation representing `ref1`,
`(tok x T)`	true, if $a(x) \in T_j$ where T_j is the predicate representing token `T`,
`(cat x C)`	true, if $a(x) \in C_j$ where C_j is the predicate representing category `C`,
`(mor x M)`	true, if $a(x) \in M_j$ where M_j represents morphological tag `M`,
`(fct x F)`	true, if $a(x) \in F_j$ where F_j is the predicate representing function `F`.

Complex formulae are interpreted classically. As a result, a query or closed formula is true or false of a tree. Thus, a tree is an answer to a query, if it is a model for the query. And the result of a query on a treebank is the set of all trees which are models of the query.

2.3 Expressive Power

The primary idea behind fsq is to provide a query system with a powerful language and clear and well-established semantics. We therefore opted to use full first-order logic as a query language. In this section, we will show the expressive power of this query language with linguistically motivated examples.

When so-called fuzzy tree fragments (see (Wallis and Nelson, 2000)) were first introduced, they provided a significant progress over simple string matching languages. Nowadays they are standard, every sensitive query tool for syntactically annotated corpora offers this type of underspecification, and fsq is no exception to this rule. We will therefore bypass them completely. Due to space restrictions, we will neither show how to query unconnected sub-parts of a tree. The reader interested in this issue is referred to the article by Kallmeyer and Steiner (2003). Everything discussed there for the query tool VIQTORYA can be queried by fsq, too. Rather we focus on the use of quantification, something that fsq offers, but no other query tool does in a similar way.

Suppose we are looking for trees where a clause lacks the subject. We can pose the following query to do so:[1]

$$\exists x \mathrm{SIMPX}(x) \wedge \forall y((x >> y) \rightarrow \neg \mathrm{ON}(y))$$

The formula reads "*There is a clause node (node of category* SIMPX*) such that no node below it is a subject node (node of function* ON *(Object in the Nominative)).*" An example result is the tree in Figure 1. Another result from the same corpus would be "*aber gut, wir können ja mal fragen, was gegeben wird.*" (*All right, we can ask, what's on play.*) where there is no subject in the German subordinate clause. If one is interested in finding only those trees where the subject is lacking in a

[1] For better readability, we use classical first-order logic syntax and not fsq syntax.

subordinate clause, the above query has to be extended to

$$\exists x \exists y \ \mathrm{SIMPX}(x) \wedge \mathrm{SIMPX}(y) \wedge (x >> y) \wedge (x \neq y) \wedge (\forall z \neg((y >> z) \wedge \mathrm{ON}(z)))$$

"*There are two different clause nodes, one dominating the other, and no node below the lower clause node is a subject node.*" This is a query of quantifier depth 3 (number of deepest nestings of quantifiers). On second thought one can see that this query is still too simple to find all subordinate clauses without subject. It does not reflect the possibility to have a subordinate clause with subject as a subclause of a subordinate clause without subject. Here is a query that does: (Query 1)

$$\begin{aligned}\exists x \exists y \ & \mathrm{SIMPX}(x) \wedge \mathrm{SIMPX}(y) \wedge \\ & (x >> y) \wedge (x \neq y) \wedge \\ & (\forall z \ \mathrm{ON}(z) \rightarrow (\neg(y >> z) \vee \\ & \quad \exists w \ \mathrm{SIMPX}(w) \wedge (y >> w) \wedge \\ & \quad\quad (y \neq w) \wedge (w >> z)))\end{aligned}$$

"*There are two different clause nodes, one dominating the other, and every subject node is either not dominated by the lower clause node or there is a further clause node intervening.*" This query is even of quantifier depth 4.

Another complicated query must be used if we want to find all trees in which the main clause lacks the subject, but subordinate clauses may have one. The query looks like this:

$$\begin{aligned}\exists x & \mathrm{SIMPX}(x) \wedge \\ & (\forall y \mathrm{SIMPX}(y) \rightarrow (x >> y \wedge x \neq y)) \wedge \\ & (\forall y((x >> y) \wedge \mathrm{ON}(y))) \rightarrow \\ & \quad \exists z \ (\mathrm{SIMPX}(z) \wedge (x >> z) \wedge \\ & \quad\quad (x \neq y) \wedge (z >> y)))\end{aligned}$$

"*There is a highest clause node such that for every subject node dominated by it there is a second clause node intervening.*"

Suppose now, we want to find trees with no Vorfeld. We can do this by querying:

$$(\exists x \mathrm{SIMPX}(x)) \wedge (\forall y \neg \mathrm{VF}(y))$$

"There is a clause node but no Vorfeld node." A look at the result trees shows that many of them, e.g., "*kein Problem, geht auf die Spesenrechnung*" (*no problem, will be put on the travel expenses bill*) have unconnected subparts. If we find that undesirable, we conjoin the above query with the simple $\exists x \forall y(x >> y)$ demanding a proper root node, and then get nicer results like "*machen wir den Juli*" (*Let's take July*).

Many more interesting examples for involved queries could be provided, but space does not permit to do so. When a linguist actually starts extracting instances of more complicated syntactic phenomena, he will quickly develop a desire to have access to arbitrary quantification in the query language. That's why we chose to provide this in the query language of fsq.

2.4 Complexity Issues

One of the fundamental insights of database theory is that there is a price to pay for providing a powerful query language, and this price is that the evaluation of a query can become quite expensive. On the theoretical side it is the so-called data complexity that is relevant for us. This notion, introduced by Vardi (1982), describes the complexity of evaluating a query in terms of the size of the database or treebank. It is common knowledge that the problem of evaluating a first-order sentence on a finite structure is in the complexity class LOGSPACE and therefore in PTIME in terms of the size of the structure (see, e.g., (Vardi, 1982)). But it is currently unknown, whether the problem fits into a proper subclass of PTIME in the polynomial hierarchy, i.e., whether there is a fixed exponent – independent of the structure and the query – such that the evaluation time is in the order of the size of the structure to the power of that exponent (see, e.g., the work by Stolboushkin and Taitslin (1994), which suggests a negative answer to the question).

The algorithm implemented in fsq, although simple, has a data complexity of PTIME. More precisely, if n is the number of nodes in a tree and k is the quantifier depth of the query, then the algorithm evaluates the query on the tree in time $O(n^k)$. As stated above, it is not known whether algorithms with a better complexity do exist at all.

Practically, this result is of course not unproblematic, and the question arises at what quantifier depth the evaluation time of a query becomes so long that it exceeds user sustainable response time. Our tests indicate that a strict quantifier bound can not really be given for fsq. Queries

of quantifier depth up to and including 3 pose no problem, they are almost instantaneously answered. Queries of depth 4 can take more time to answer: Some of them, like Query 1 in the previous section are answered in 2-3 seconds, others like Query 2 below require an evaluation time of half an hour. (All: wall clock time on a Pentium IV, 1.6GHz system.) Response times therefore depend on the individual query and the corpus.

A solution to the problem of longer response times can sometimes be found in a reformulation of the query. The same content can be logically formulated in different ways with formulas having quite different quantifier depth. Let us highlight this fact by an example. Suppose a user looks for the four words (tokens) *heute*, *Treffen*, *in*, and *Hannover* to appear together in a tree. This could be formulated as follows: (Query 2)

```
(E x (E y (E z (E w
   (& (tok x heute) (tok y Treffen)
      (tok z in) (tok w Hannover))))))
```

with quantifier depth 4. But it could also be formulated as

```
(& (E x (tok x heute)) (E x (tok x in))
   (E x (tok x Treffen))
   (E x (tok x Hannover)))
```

with quantifier depth just 1!

3 The Architecture of fsq

fsq consists of three main parts: A treebank initialiser, the query engine, and a graphical user interface to the query engine. All components of fsq are implemented in Java JDK 1.3. No propriatory or special libraries are used, so fsq can be run on any standard Java runtime environment (JRE 1.3) independent of the platform and operating system chosen.

3.1 Initialising a Treebank

The input format of treebanks fsq can cope with is the NEGRA export format (Brants, 1997). This format is designed for data exchange and to be readable by humans. It is not very well suited as a format to be queried, because relations between nodes are implicitely instead of explicitely represented. Therefore we need an initialisation step that transforms the NEGRA format into one that can be queried a lot faster. The query format is a compact binary representation of the relations of the trees. Each tree has its own representation, independent of the others. The binary relations (dominance, precedence, etc.) are represented as a twodimensional quadratic array indexed by the nodes of the particular tree. Each cell of the array states for all relations if the two nodes addressing the cell stand in the relations or do not. The unary relations (token, category, morphology, and function) are represented by a onedimensional array indexed again by the nodes of the tree. Each cell codes the category and function of the addressing node. Needless to say that the initialisation step has to be performed only once per treebank. On a 1.5MB treebank of TüBa-D, it typically takes 2-4 minutes (wall clock time, Pentium IV, 1.6GHz), so it's rather fast for a precompilation step.

3.2 The Query Engine

The query engine parses a submitted query and successively evaluates it on each finite structure. Due to the clarity of the query syntax, the parsing step is simple and quickly performed. A query is a first order formula and therefore inductively defined. It is evaluated on a tree by a structural recursion on its form. During this recursion, a variable assignment, which is initially empty, is constructed stepwise. The evaluation of an existential quantification is achieved by extending the current variable assignment by supplying a value for the existentially quantified variable and then recursing on the body of the quantification with the extended variable assignment. For completeness reasons it is clear that each node of the structure under evaluation has to be at some time the value of the quantified variable. This is achieved by a loop. The first recursion returning true renders the formula true. If the body is evaluated to false under any assignment of a node to the quantified variable, the formula is false. Universally quantified formulae are evaluated similarly. The evaluation of a boolean connective is a straightforward recursion. In the case of a conjunction, e.g., each conjunct is evaluated separately with the current variable assignment. Only if each conjunct evaluates to true, the evaluation value of the conjunction is true, too. Of course, we stop the evalu-

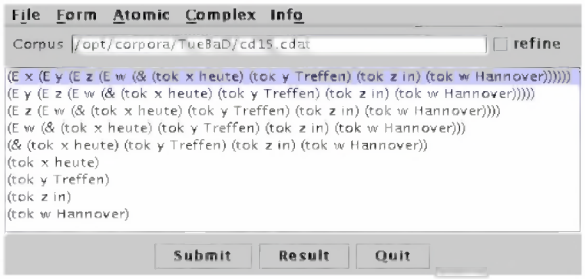

Figure 2: Graphical user interface of fsq

ation of the conjuncts the moment we receive the first conjunct being evaluated to false and return false as value of the conjunction. For atomic formulae, the evaluation is a mere look-up to check if the nodes of the structure as given by the variable assignment do stand in the relation that is expressed by the formula. All in all, the evaluation process follows the definition of the semantics rather closely. Each tree is checked separately, and those trees for which the query evaluates to true are returned as answers to the query.

3.3 The Graphical User Interface

The graphical user interface is designed to assist the user in formulating a query by supporting him to compose formulae in a bottom-up fashion.

The centre of the interface (see Figure 2) consists of a list of formulae that the user can manipulate. To pose a query, the user selects one of these formulae and clicks the *Submit* button. Thereafter, the result set in the form of the numbers of those trees for which the query is true can be browsed by pressing the *Result* button. It is intended that the user feeds this result set into tools like Annotate (Plaehn and Brants, 2000) that allow a detailed inspection of the results.

When composing a query most users think in a bottom-up fashion focusing first on the atomic constituents. This approach is systematically supported by the user interface in the following way. The *Atomic* menu lets the user compose atomic formulae. He picks the relation of his choice, say, e.g., the dominance relation. He is successively asked for names of the variables one dominating the other, say, e.g., `x` and `y`. Thereafter, the syntactically correct formula `(>> x y)` is added to the list of formulae as the topmost element. The other atomic formulae can be constructed in a similar fashion.

In order to get more complex formulae, the user can choose operations from the *Complex* menu. It contains menu options for the boolean connectives and quantifiers. To compose, e.g., a conjunction, the user first chooses the formulae he wishes to conjoin by clicking on them in the list of formulae. Thereafter he just picks the *Conjunction* menu item and the conjunction of the formulae he chose is added to the list of formulae as the topmost element. In case of an existential or universal quantification, the user selects a formula from the list, say, e.g., `(>> x y)` and, e.g., the *Existential Quantification* menu item. He will be asked for the name of the variable to quantify over, say `x`, and the existentially quantified formula, `(E x (>> x y))`, is added to the list of formulae as the topmost element.

The *Form* menu offers additional operations on formulae. The user can edit a formula by hand, or enter a new one by hand. He can duplicate or delete a formula from the list. Or he can swap the order of two formulae on the list.

The *File* menu offers the ability to save the current list of formulae for further use or to load a list of formulae saved as a file.

The user also has to choose a corpus, not just compose a query. This step is supported by a file chooser that starts when clicking the word *Corpus* and offers only corpora precompiled for fsq.

The refinement checkbox to the right of the chosen corpus field helps the user in narrowing in search results. If checked, the next query will not be executed on the whole corpus chosen but rather on the *previous search result* for that corpus. Thus the user can first pose an approximating query, inspect the result and then formulate a more fine grained query to be executed only on the result of the first query to get a smaller answer set. Of course these steps can be iterated.

4 Related Work

In recent times, a number of query tools for syntactically annotated corpora have been proposed. Amongst the most important ones are CorpusSearch (Randall, 2000), ICECUP III (Wallis and Nelson, 2000), NetGraph (Mírovský et

al., 2002), TGrep2 (Rohde, 2001), TIGERsearch (König and Lezius, 2000), and VIQTORYA (Kallmeyer and Steiner, 2003). We compare them here to fsq focusing our attention on the expressive power of the query language, on the underlying data structures, on the existence of a user interface, and on corpus formats they can cope with. Since unary and binary predicates for node labels, tokens, dominance, and precedence are in some form or other provided by all query languages, we do not mention them.

CorpusSearch was developed for querying the Penn-Helsinki Parsed Corpus of Middle English. Its query language offers only a restricted form of negation and disjunction, and no quantification. Node variables are implicitely existentially quantified. The underlying data structure must be strictly trees. No graphical user interface is provided. CorpusSearch can be used to query all corpora annotated in the Penn Treebank style.

ICECUP III was developed for querying the ICE-GB, the British component of the International Corpus of English. Its query language contains only a restricted form of negation, no disjunction and no quantification. Node variables are implicitely existentially quantified. The underlying data structure must be strictly trees. ICECUP has a nice graphical user interface. It is designed to query the ICE-GB, it is unclear whether it can be used on other corpora than ICE.

NetGraph was developed as a corpus workbench for the Prague Dependency Treebank. Its query component implements the positive existential fragment. The data structures underlying the queries are strict trees. NetGraph has a nice graphical user interface. It is designed for the Prague Dependency Treebank; whether it can be used on other corpora is unclear to the author.

TGrep2 was developed for querying corpora annotated in the Penn Treebank style. Its pattern matching query language is difficult to capture in logical terms. It goes beyond the existential fragment[2], but is not full first order. There is no overt quantification. The underlying data structures must be strictly trees. There is no graphical user interface provided.

TIGERsearch, originally developed to query a German newspaper treebank, is one of the most advanced tools. Its query language is the existential fragment, the underlying data structures can be more general than trees. TIGERsearch has a nice graphical user interface. Its greatest advantage is probably its usability on many different corpus formats including NEGRA and Penn Treebank. On a side note, TIGERsearch implements a strange notion of linear precedence which is probably counterintuitive and undesirable for linguists. For more details, see the discussion by Kallmeyer and Steiner (2003, p. 23).

VIQTORYA was developed for querying the Tübingen Treebanks. Its query language is the existential fragment, the underlying data structures can be more general than trees. VIQTORYA comes with a nice graphical user interface. It is applicable to those corpora in NEGRA export format where no trees have crossing branches or secondary edges.

None of the above query tools can compete with fsq on the grounds of expressive power of the query language and generality of the underlying data structures. For some, like TIGERsearch and VIQTORYA, one can see a division of labour. They are more appropriate for fast answers to purely existential queries, while fsq is the query tool of choice for queries that require an intensive use of quantification.

5 Conclusion and Future Work

In this paper, we presented fsq, a query tool for syntactically annotated corpora. fsq is designed to provide users a very expressive query system. Therefore its query language is full first-order logic, and its underlying data structures are finite first-order structures. These data structures include proper trees, but also all of those extensions one can find in existing corpora like discontinuous substructures, crossing branches or secondary edges. The query language allows to query these structures using full first-order quantification. Hence, fsq seems one of the most powerful query system for annotated corpora currently existing. fsq is publicly available, see `http://tcl.sfs.uni-tuebingen.de/fsq`.

[2] Arbitrary conjunctions, disjunctions, and negations of formulae, but *all* variables are interpreted as being *existentially* quantified.

The two main directions of extensions are the query language and the corpus input formats fsq can cope with. Although first-order logic is quite an expressive language, there are well-known restrictions. It may therefore be worthwhile to consider (fragments of) second-order logic as query language. Second order quantification allows to define new relations not contained in the signature such as transitive closure of a relation. Another extension of the query language can be a simplified syntax for simple queries without complex quantification to make the tool easier usable for linguists with less experience in formal logics.

The extension to other corpus input formats is of course an important step to increase usability of the tool. We therefore intend to extend fsq to cope with corpora in the Penn Treebank format and certain formats in XML. The main work for such an extension lies in extending the initialisation component.

Acknowledgements

The work presented here is part of the project A2 in the Special Research Programme (SFB) 441 "Linguistic Data Structures" at the University of Tübingen funded by a grant of the German Research Council (DFG SFB 441-02). We would like to thank Ilona Steiner for interesting and fruitful discussions.

References

Anne Abeillé and Lionel Clément. 1999. A tagged reference corpus for French. In *Proceedings of EACL-LINC*.

Thorsten Brants, Wojciech Skut, and Hans Uszkoreit. 1999. Syntactic annotation of a German newspaper corpus. In *Proceedings of the ATALA Treebank Workshop*, pages 69–76.

Thorsten Brants. 1997. The NEGRA export format. CLAUS Report 98, Universität des Saarlandes, Computerlinguistik, Saarbrücken, Germany.

Erhard Hinrichs, Julia Bartels, Yasuhiro Kawata, Valia Kordoni, and Heike Telljohann. 2000. The VERBMOBIL treebanks. In *Proceedings of KONVENS 2000*.

Tilman Höhle. 1985. Der Begriff 'Mittelfeld'. Anmerkungen über die Theorie der topologischen Felder. In A. Schöne, editor, *Kontroversen, alte und neue. Akten des 7. Internationalen Germanistenkongresses*, pages 329–340.

Laura Kallmeyer and Ilona Steiner. 2003. Querying treebanks of spontaneous speech with VIQTORYA. *Traitement Automatique des Langues*, 43(2).

Esther König and Wolfgang Lezius. 2000. A description language for syntactically annotated corpora. In *Proceedings of the COLING Conference*, pages 1056–1060.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Jiří Mírovský, Roman Ondruška, and Daniel Průša. 2002. Searching through Prague Dependency Treebank. In Erhard Hinrichs and Kiril Simov, editors, *Proceedings of Treebanks and Linguistic Theories*, pages 114–122.

Oliver Plaehn and Thorsten Brants. 2000. Annotate – an efficient interactive annotation tool. In *Sixth Conference on Applied Natural Language Processing (ANLP-2000)*.

Beth Randall. 2000. CorpusSearch user's manual. Technical report, University of Pennsylvania. `http://www.ling.upenn.edu/mideng/ppcme2dir/`.

Douglas Rohde. 2001. Tgrep2. Technical report, Carnegie Mellon University. `http://tedlab.mit.edu/~dr/Tgrep2/`.

Anne Schiller, Simone Teufel, and Christine Thielen. 1995. Guidelines für das Tagging deutscher Textcorpora mit STTS. Manuscript, Universities of Stuttgart and Tübingen.

Rosmary Stegmann, Heike Telljohann, and Erhard Hinrichs. 2000. Stylebook for the German treebank in VERBMOBIL. Technical Report 239, SfS, University of Tübingen.

Alexei Stolboushkin and Michael Taitslin. 1994. Is first order contained in an initial segment of PTIME? In Leszek Pacholski and Jerzy Tiuryn, editors, *Proceedings CSL*, volume LNCS 933, pages 242–248. Springer.

Moshe Vardi. 1982. The complexity of relational query languages. In *Proceedings of the 14th ACM Symposium on Theory of Computing*, pages 137–146.

Sean Wallis and Gerald Nelson. 2000. Exploiting fuzzy tree fragment queries in the investigation of parsed corpora. *Literary and Linguistic Computing*, 15(3):339–361.