fsq User Manual

Stephan Kepser
CRC 441, University of Tübingen, Germany
kepser@sfs.uni-tuebingen.de

February 28, 2008

1 Introduction

This small user manual is supposed to complement the description of fsq that can be found in a paper by Kepser [2]. Whereas [2] gives a general introduction and overview over fsq, this manual expects a user to know what fsq is and focuses on how to use it.

Current version of fsq: 2.1 – February 2008.

fsq is developed by Stephan Kepser (search engine, user interface) and Hartmut Keck (treebank browser). It uses the graphics library yFiles by yWorks (http://yworks.com). We would like to thank yWorks for granting the use of their graphics library.

2 Installation

fsq requires Java (runtime environment or development kit) version 1.6. It may run with earlier versions (in particular 1.5 and 1.4), but this is not tested and not supported.

fsq consists of one Java library (jar files): fsq.jar. You may store the jar file at any place on your system that you find appropriate. But the jar file has to be in the class path in order to start fsq. We also recommend to extend the Java standard heap space upon calling Java (using the option -Xmx).

fsq is developed operating system independent. It is tested to run under Linux, MS Windows, and Mac OS X. It is likely to also run under Solaris 10.

3 Preprocessing

Before a treebank can be queried, it must be preprocessed. Preprocessing is the process of translating a data exchange format into a binary format that can be quickly queried. We support the following two data exchange formats:

- NEGRA export format [1]
  Both format 3 and format 4 of NEGRA are supported.

- Tiger XML format

The format is detected by file extension. The preprocessor is used on the command line or the graphical user interface. The syntax for use from the command line is
`java fsq.Init` *Treebankfile*
or more explicitely
`java -Xmx512M -cp /path/to/fsq.jar fsq.Init` *Treebankfile*
where *Treebankfile* is a file name of a treebank, potentially including a path. Example: If you want to initialise the treebank file `/opt/corpora/cd15.export` you type
`java fsq.Init /opt/corpora/cd15.export`

The preprocessor creates a file binarily encoding the treebank. Its name is *Treebankfile*`.cdat`. In order for the preprocessor to be able to do so, the user executing it must have write permission in the directory containing the treebank file. An existing `.cdat`-file will be overwritten *without* prior confirmation. The input file is read only and *never* written to.

During initialisation the preprocessor outputs the tree it is currently processing and the number of nodes of that tree to standard output. At the end, some statistical information is printed out. Here is an example:

```
Number of Trees: 217
Number of POS tags: 38
Number of morph tags: 1
Number of edge labels: 22
Number of secondary relations: 4
```

The preprocessing can also be started from the graphical user interface described in the next section.

4 Graphical User Interface

The aim of the graphical user interface (GUI) is to simplify the composition of queries and the choice of a treebank. The GUI is started by
`java -jar /path/to/fsq.jar`

It is recommended that the Java heap space is expanded when treebanks to be searched exceed the size of about 5.000 trees. Otherwise one may face Java “Out of memory” errors. A prototypical `fsq` start may look like this
`java -Xmx512M -jar /path/to/fsq.jar`

Upon startup the main window appears: (Figure 1)

To select a treebank the user can type in the treebank file (potentially with a full path, may or may nor end with the extension `.cdat`) into the text field to the right of the word “Treebank”. Alternatively, the user can click onto the word “Treebank”. Doing so opens a file chooser menu that lets one select the appropriate treebank.

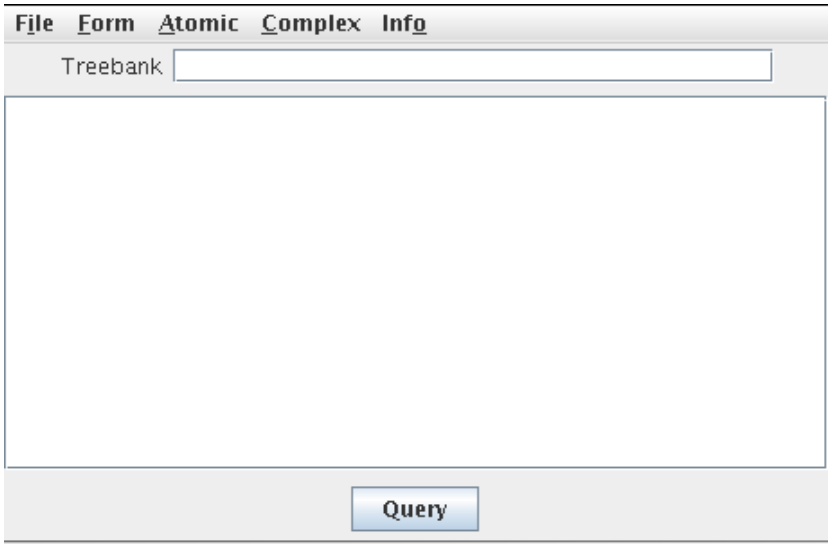
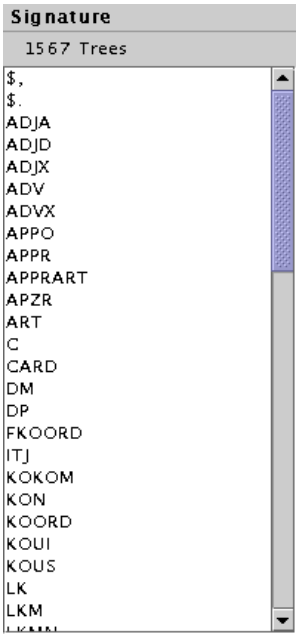


Figure 1: GUI upon startup

After a treebank is chosen its signature can be inspected by choosing the item *Treebank* from the *Info* menu. This action will open a new window with the signature information. The user can inspect the different parts of the signature by choosing the desired item from the *Signature* menu offering *Categories*, *Morphological Tags*, *Functions (edge labels)* and *Secondary relations*. An example of such a window can be seen on the right side.



The centre of the main window of the GUI consists of a list of formulae. It works as a kind of note pad for the composition of formulae and queries. The exact syntax of formulae is described in Section 7. The aim of the GUI is to relieve the user from manually typing these formulae. The GUI supports the composition of formulae in a bottom-up fashion. The user starts by creating the basic subformulae using operations from the *Atomic* menu and then combines these subformulae using operations from the *Complex* menu. Figure 1 shows an example of how the GUI looks like after some of these operations.

The *Atomic* menu offers operations for the creation of all types of atomic formulae. The typical procedure is as follows. The user decides on the type of atomic formula to compose and selects the

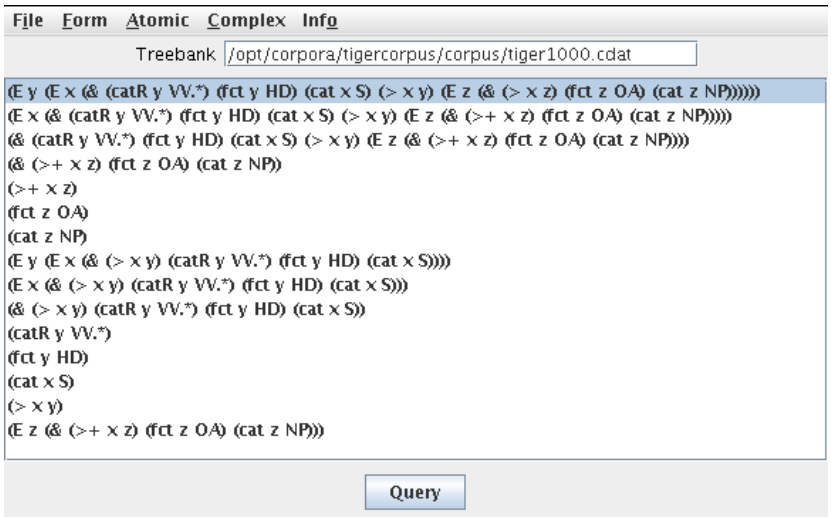


Figure 2: Example of GUI

according menu item. Then he is prompted for the relevant information such as variable names or tokens or categories. Thereafter the syntactically correct formula is added as the first formula on the formula list. For example, for dominance, the user is asked for the name of the dominating node variable, say `x`, and for the dominated node variable, say `y`. Then the formula `(>> x y)` is added to the formula list.

The creation of complex formulae is supported by operations of the *Complex* menu. The prototypical way here is to first choose a formula or some formulae from the formula list and then pick an operation from the menu to be performed on the choice. The simplest case is that of negation. Just click on a formula from the formula list and choose *Negation* from the *Complex* menu. The negation of the chosen formula is added to the top of the formula list. For disjunction and negation choose any number of formulae from the list.[1] Then select *Conjunction* or *Disjunction* and the conjunction or disjunction of the formulae is added at the top of the list. For an implication, just pick two formulae and choose the *Implication* operation. For quantification choose a single formula and select the desired type of quantification (*Existential* or *Universal*). You are prompted for the variable name to quantify over and after this typed in the quantified formula is added at the top of the list.

When formulae are composed by operations from the *Atomic* and *Complex* menu only, they are guaranteed to always be syntactically correct.

To submit a query, choose a query from the formula list and press the *Submit* button. The query is sent to the query engine which evaluates the query on the chosen treebank. Upon the first hit a new window opens displaying the hits of the treebank. You can immediately start browsing and inspecting the hits while the search engine is still running. The set of hits is regularly updated when there are more hits found. The treebank browser is described in Section 5.

The *Form* menu contains operations for manually editing formulae. The *New* item opens a window to manually type in a new formula. The *Edit* item opens a window to manually edit a formula chosen from the formula list before selecting *Edit*. For copying, choose a formula and select the *Duplicate*

[1] To do so, click on a formula and then press and hold down the *Shift* or *Control* key while clicking on the next formula. With *Control* pressed you add just one more formula at a time. With *Shift* pressed you add the clicked formula and everything inbetween the clicked and the last chosen formula.

operation. It will add a copy of the chosen formula as the topmost element of the formula list. To delete a formula, click on in and select *Delete*. The *Swap* operation exchanges the position of two selected formulae in the formula list. It can be used to set up the right order of two formulae for forming their implication. When two formulae are selected for implication, it is always the higher one that will be the premise. The lower one gives the consequence. If by coincidence a user enters the intended premise before the intended consequence, the intended premise will show up below the intended consequence. If the user picks the two for forming the implication, he will get the opposite of the result he intended. The simplest way out of this problem is to exchange the positions of the two formulae using *Swap*. The last item in the *Form* menu, *Clear all*, allows the user to clear the whole list of formulae.

The *File* menu contains operations for saving and loading formula lists. If the user wishes to save his current formula list, he can select the *Save as* item. It opens a file chooser menu to enter a file name by which the formula list will be saved. If the user has already selected a file name, the *Save* item will save the current formula list under that name. The current file name is displayed below the *Save as* item. A formula list can be reloaded with the *Open* item. This operation deletes the current formula list and replaces it by the one in the specified file.

On a side remark, the formula list is actually just a list of lines of characters. This fact can be used to enhance such a list by comments, if desired. Just use the *New* operation from the *Form* menu to type in a line of comment. These comment lines will certainly be saved and loaded as any other formula lines.

The *File* menu offers a menu item *Open treebank*. This item opens the treebank browser described in the next section. The *File* menu also offers a menu item *Initialise treebank*. This item allows the preprocessing of a treebank as described in Section 3. One just chooses the treebank import file to be processed and the name of the binary coded treebank file. Please note that in difference to the command line version the GUI version of the preprocessor does not produce any progress information at all. All one can see is a final completion message similar to the one described for the command line version.

The *Info* menu contains besides the above mentioned *Treebank* item an item displaying an fsq version information.

To exit the GUI select the *Quit* item in the *File* menu.

Error Handling

If an error occurs during processing a query, a window opens displaying the error and some context information.

5 Treebank Browser

fsq comprises a treebank browser. It serves two purposes. Firsty it allows the user to browse a complete treebank. Secondly and more importantly it is used to display query results. The user interface is shown if Figure 3.

The centre of the window shows a tree from the treebank. You may zoom into or out of the tree. You may also select an area of the tree and zoom into it. The navigation block below allows to move

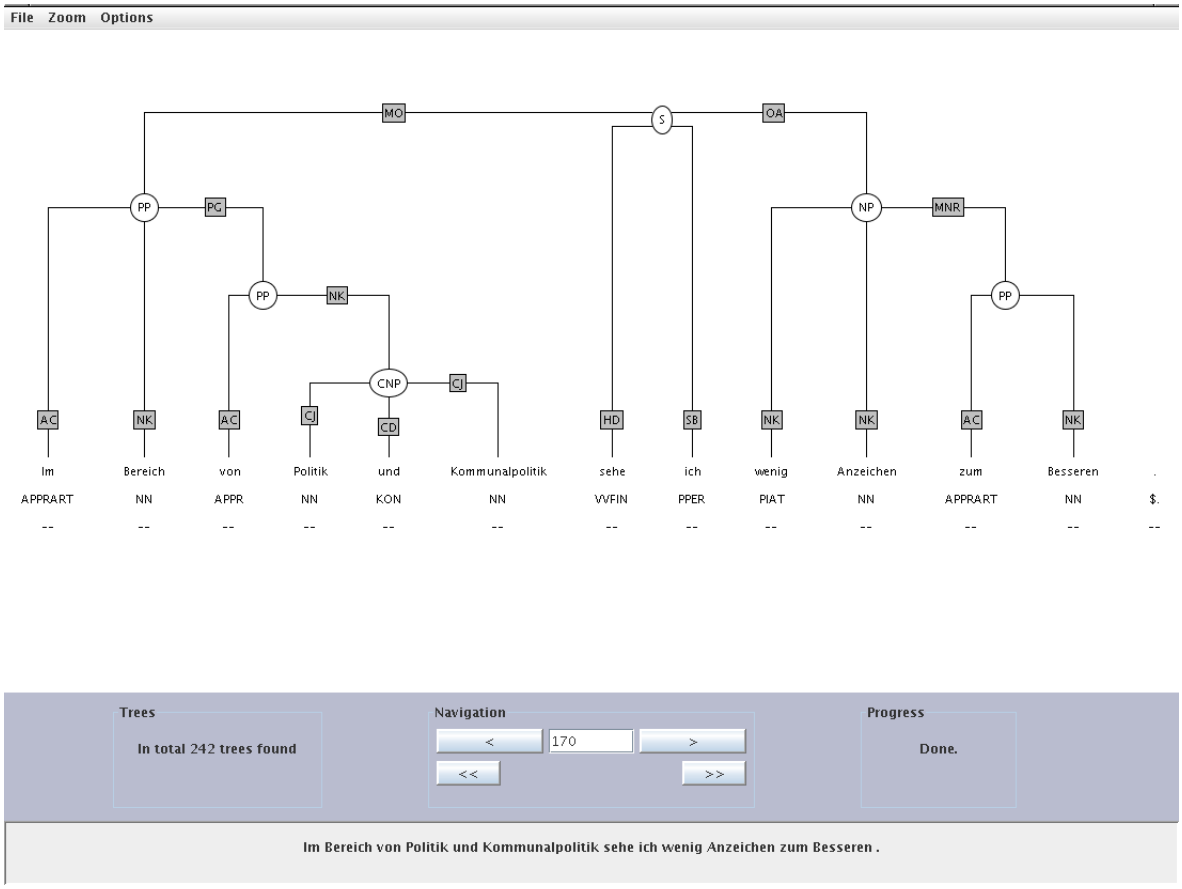


Figure 3: The treebank browser

through the treebank. The progress status information to the right tells you whether the query is still running in the background or not. The information block at the left side displays the number of trees found for the query, or the total number of trees in the treebank.

The *File* menu offers the options to export the currently displayed tree as a graphics image or to print this tree. One may also export the hits of the query as a file containing a list of tree ids. This maybe used as input for tools such as @nnotate.

To quit the treebank browser, choose the option *Close* from the *File* menu.

The *Options* menu allows to configure which information is displayed below the token level of a tree. These may be lemmata or morphological informations.

Note that you may have several open treebank windows, e.g., for different queries.

6 Command Line Interface

There exists a simple command line interface to the query component. It is used as follows:
`java fsq.query` *query treebank* [*result*]
where *query* is a query as defined in Section 7. Please note that you have to protect spaces in the

query against the shell. This is best done by surrounding the whole query with single or double quotes. *treebank* is the name of the treebank to be queried. It can contain a path, but it must not contain any file extension. The file extension `.cdat` is assumed. Example:
`java fsq.query "(E x (tok x Tanz))" /opt/corpora/cd15`

The optional argument *result* specifies the name of the result file. The result of the query is put into a file. This file consists of the tree ids (numbers) of those trees for which the query is true. All of these ids are in a single line, each two of them being separated by a space. This format is suitable as input for the @nnotate tool. If no result file name is specified, the default *treebank*`.res` is chosen. In this case, the user executing the query must have write permission in the directory of the treebank.

While the query engine is running the number of the tree on which the query is currently processed is printed out to show that the engine is still alive.

Error Handling

Errors occurring during querying from the command line do not lead to opening an error window. Rather they lead to immediate program termination. The type of error is returned via the system exit code. An exit code of 1 indicates a *system error*. This can be a file input/output error, but also a an array out of bounds error that occurs when the query contains a free variable. An exit code of 2 indicates a *parsing error*. The query could not be parsed. An exit code of 3 indicates a *calling error*. The comand line interface was called with the wrong arguments.

7 Formulae

A *variable* is a string of letters or numbers, delimited by white space or braces. Examples: `x`,`y`, `z`, `v12`, `V21`, `3`, ...

The syntax of *formulae* is LISP-like, i.e., each (sub-) formula is surrounded by braces `(`), and there is a strict prefix notation, the functional head always comes first.

Formulae are divided into atomic formulae and complex formulae. Atomic formulae are further divided into 4 groups. The first two groups comprise formulae for node labels (without or with regular expressions), the third group comprises relations between nodes and the fourth group contains a single formula for regular expression search of tokens.

(In the following, let `x` and `y` be variables, $\varphi, \varphi_1, \varphi_2, \varphi_3, \ldots$ and ψ formulae, and let `hd` be a secondary edge name from the treebank.)

Atomic node label formulae

- `(tok x T)`
  node `x` has token (terminal string) `T`.
- `(lem x L)`
  node `x` has lemma `L`.
- `(cat x C)`
  node `x` is of category (has POS tag) `C`.

- `(mor x M)`
  node `x` has morphological tag `M`.
- `(fct x F)`
  node `x` is of grammatical function (has edge label) `F`.

Atomic node label formulae with regular expressions

- `(tokR x T)`
  the token (terminal string) of `x` matches RE `T`.
- `(lemR x L)`
  the lemma of `x` matches RE `L`.
- `(catR x C)`
  the category of `x` matches RE `C`.
- `(morR x M)`
  the morphological tag of `x` matches RE `M`.
- `(fctR x F)`
  the function (edge label) of `x` matches RE `F`.

The regular expressions are those provided by Java. A documentation can be found at the following location: `http://java.sun.com/j2se/1.5.0/docs/api/java/util/regex/Pattern.html`.

Atomic node relations

- `(> x y)`
  node `x` is the mother of `y`.
- `(>> x y)`
  node `x` dominates `y`.
  `>>` is the reflexive transitive closure of `>`.
- `(>+ x y)`
  node `x` properly dominates `y`.
  `>+` is the transitive closure of `>`.
- `(.  x y)`
  node `x` immediately precedes `y`.
- `(..  x y)`
  node `x` precedes `y`.
- `(hd x y)`
  there is a `hd`-secondary edge from `x` to `y`.
- `(= x y)`
  equality of `x` and `y`.
- `(!= x y)`
  disequality of `x` and `y`.

Linear precedence is defined bottom up. I.e., it is firstly defined on the terminal nodes, because they have a clearly defined linear order. For internal nodes `x` and `y` it is defined that `x` precedes `y` iff the complete subtree dominated by `x` precedes the complete subtree dominated by `y`. In case of crossing branches this means that neither `x` precedes `y` nor vice verse.

RE string search

- `(sent R)`
  the frontier, i.e, concatenation of all terminal strings, separated by blanks, contains a substring that matches `R`.
  This formula is the only variable free atomic formula.

Complex Formulae

- `(! φ)`
  negation of φ.
- `(& φ1 ... φn)`
  conjunction of $\varphi_1 \ldots \varphi_n$.
- `(| φ1 ... φn)`
  disjunction of $\varphi_1 \ldots \varphi_n$.
  Disjunctions and conjunctions can have arbitrary width.
- `(-> φ ψ)`
  implication: φ implies ψ.
- `(A x φ)`
  universal quantification of `x`.
- `(E x φ)`
  existential quantification of `x`.

Queries

A *query* is a closed formula, i.e., a formula where each variable is either existentially or universally quantified. The query component cannot process open formulae. Only closed formulae are processed error-free. Submitting open formulae will lead to strange *Array out of bounds* error messages.

The semantics of formulae is classical first-order logic semantics (on finite structures).

8 Unsupported Features

fsq actually supports queries in monadic second-order logic, not just first-order logic. But users are strongly discouraged to pose MSO queries. The evaluation of even the simplest queries with set quantification may take more than a day when performed on a typical treebank with several tenthousand trees. Hence this feature is not supported by the user interface.

Still, the search engine can in principle handle these queries, and they can be typed in directly using the *New* item in the *Form* menu. Here are the additional formulae that can be handled. We follow the usual convention of using capital letters for set variables. There are two atomic formulae.

- `(mem x X)`
  individual `x` is a member of set `X`.
- `(sub X Y)`
  set `X` is a subset of set `Y`.

And there is MSO quantification.

- `(A2 X φ)`
  universal set quantification of `X`.
- `(E2 X φ)`
  existential set quantification of `X`.

9 Version History

Version 2.0 – January 2008

TigerXML as supported treebank input format.

Version 2.0 – January 2008

Major recoding, bug fixes, new jar structure.

Version 1.5 – 13th January 2006

Treebank browser for search results or complete treebank.

Version 1.4

Multithreading.

Version 1.3.1 – 18th November 2004

Regular expression search for categories, morphological tags, edge labels, and the frontier.

Version 1.2.2 – 5th March 2004

Bug fixes for batch mode: no progress bar pop-up window.

Version 1.2.1 – 16th Dec 2003

Search progress bar for GUI.

Version 1.2 – 3rd Dec 2003

Some extensions of the query language: Regular expression search on tokens; proper dominance; disequality.
Re-initialise treebanks: yes.

Version 1.1.1 – 27th Nov 2003

Important bug fixes for (linear) precedence.
Re-initialise treebanks: yes.

Version 1.1 - 9th April 2003

Major enhancements.

Version 1.0 - 3rd Feb 2003

First version out.

10 Known Problems

Out of Memory Error

During the initialisation or querying of a large treebank (more than 5.000 trees) one may encounter an `Exception in thread "main" java.lang.OutOfMemoryError`.
This is a problem of the Java Virtual Machine. Grant it more memory by using the `-Xmx` option. E.g.:
`java -Xmx512M -jar fsq.jar`
supposing your computer has more than 512MB Ram. It is recommended to grant the Java VM as much memory as possible.

References

[1] Thorsten Brants. The NEGRA Export Format. CLAUS Report 98, Universität des Saarlandes, Computerlinguistik, Saarbrücken, Germany, 1997.

[2] Stephan Kepser. Finite Structure Query: A Tool for Querying Syntactically Annotated Corpora. In Ann Copestake and Jan Hajič, editors, *Proceedings EACL*, pages 179–186, 2003.

[3] Oliver Plaehn and Thorsten Brants. Annotate – An Efficient Interactive Annotation Tool. In *Sixth Conference on Applied Natural Language Processing (ANLP-2000)*, 2000.