v2.1

GCX Engine Version 2.1

G(arbage) C(collected) X(Query) Engine – User Manual –

Michael Schmidt[*] Gunnar Jehl[†]

May 2009

Saarland University Database Group

Freiburg University Database Group

The G(arbage) C(ollected) X(Query) engine is the first streaming XQuery engine that implements active garbage collection, a novel buffer management strategy in which both static and dynamic analysis are exploited. This technique actively purges main memory buffers at runtime based on the current status of query evaluation. This approach aims at both keeping main memory consumption low at runtime and speeding up query evaluation. For detailed information on active garbage collection in XQuery engines please visit the GCX project homepage at

http://dbis.informatik.uni-freiburg.de/index.php?project=GCX.

[*] mschmidt@informatik.uni-freiburg.de
[†] jehl@informatik.uni-freiburg.de

Contents

1 Installation

1.1 Requirements

The easiest way to get started with GCX is to download one of the pre-compiled binaries from sourceforge.net at

http://sourceforge.net/project/showfiles.php?group_id=258398.

For the latest release there are currently binaries for Linux, Mac OS and Windows (i386 architecture) available.

If your operating system is not yet supported, i.e. there is no pre-compiled binary for your operating system available, you need to compile the GCX engine from source. If you need to compile it manually you will find ready-to-use Makefiles for Linux and Windows (Makefile.Linux/Makefile.Windows) in the *src* folder.

Note: *If you want to compile the GCX engine from the sources using Mac OS you should use the Linux Makefile (Makefile.Linux).*

Before manual compilation you should make sure that the following required (additional) tools are installed.

- GNU Make (installation tested with version 3.81)
- GNU Bison (installation tested with version 2.1 and 2.3)
- GNU Flex (installation tested with version 2.5.4 and 2.5.33)
- GNU Sed (installation tested with version 4.1.5)

1.2 Using the Binaries

If you decide to use one of the binaries, no installation is needed. Simply download the binary bundle that corresponds to your operating system

- Linux: “*gcx_2-1_bin_linux.tar.gz*”
- Windows: “*gcx_2-1_bin_win32.zip*”
- Mac OS: “*gcx_2-1_bin_macos.tar.gz*”

and extract the file. This will – for all operating systems – create the following directory structure (where OS denotes your chosen operating system).

```
gcx_2-1_bin_OS................ The root folder.
 |
 |__bin........................ This folder contains the binary.
 |
 |__examples.................. This folder contains sample queries
     |                          for testing purpose.
     |
     |__sgml................... This folder contains sgml sample
     |                          queries.
     |
     |__tree................... This folder contains tree sample
     |                          queries.
     |
     |__xmark ................. This folder contains xmark sample
     |                          queries.
     |
     |__xmp.................... This folder contains xmp sample
                                queries.
```

The executable (Linux/Mac OS: *gcx* or Windows: *gcx.exe*, depending on the operating system) is found in the *bin* directory. To run GCX, simply open a *shell* (Linux/Mac OS) or a *command prompt window* (Windows), change into the *bin* directory, and run the executable. We refer the reader to section 3 for the usage and the complete list of available command-line arguments.

1.3 Compiling the Sources

1.3.1 Compiling under Linux/Mac OS

Note: *If you are using Mac OS you will also need to install – if not already present – the following required (additional) tools.*

- *GNU Bison from* *http://www.gnu.org/software/bison/bison.html*.
- *GNU Flex from* *http://flex.sourceforge.net/*.
- *GNU Sed from* *http://www.gnu.org/software/sed/sed.html*.

1. Download the archive "*gcx_2-1_src.tar.gz*" (you can also download the *.zip* archive "*gcx_2-1_src.zip*") from

   http://sourceforge.net/project/showfiles.php?group_id=258398.

2. Extract the archive by typing

   ```
   > tar -xzf gcx_2-1_src.tar.gz
   ```

   in a shell.

This will create the following directory structure.

- gcx_2-1_src .... The root folder.
  - bin .... This folder is empty and will - after compilation - contain the binary.
  - examples .... This folder contains sample queries for testing purpose.
    - sgml .... This folder contains *sgml* sample queries.
    - tree .... This folder contains *tree* sample queries.
    - xmark .... This folder contains *xmark* sample queries.
    - xmp .... This folder contains *xmp* sample queries.
  - src .... This folder contains all required sources for compilation.

3. Step into the *src* directory by typing

   ```
   > cd ./gcx_2-1_src/src
   ```

   in a shell.

4. Optionally, but not necessarily, you may want to enable or disable one or more special features by uncommenting or adding `FLAGS` in the Makefile (Makefile.Linux). A complete list of all available compilation `FLAGS` can be found in subsection 1.4.

5. Now type

   ```
   > make -f Makefile.Linux
   ```

   to compile the sources. After compilation a binary file named *gcx* will be created in the *bin* directory.

6. You might also consider to add the *bin* directory to your `PATH` variable or creating a link to the *gcx* binary in */usr/bin*.

1.3.2 Compiling under Windows

In case you are using Windows we recommend the MinGW environment from

http://www.mingw.org

to install GCX. You will also need to install – if not already present – the following required (additional) tools.

- GnuWin32 Make from
  http://gnuwin32.sourceforge.net/packages/make.htm.
- GnuWin32 Bison from
  http://gnuwin32.sourceforge.net/packages/bison.htm.
- GnuWin32 Flex from
  http://gnuwin32.sourceforge.net/packages/flex.htm.
- GnuWin32 Sed from
  http://gnuwin32.sourceforge.net/packages/sed.htm.

1. Download the archive "*gcx_2-1_src.zip*" (you can also download the *.gz* archive "*gcx_2-1_src.tar.gz*") from

   http://sourceforge.net/project/showfiles.php?group_id=258398.

2. Extract the archive "*gcx_2-1_src.zip*".

This will create the following directory structure.

- gcx_2-1_src .... The root folder.
  - bin .... This folder is empty and will - after compilation - contain the binary.
  - examples .... This folder contains sample queries for testing purpose.
    - sgml .... This folder contains *sgml* sample queries.
    - tree .... This folder contains *tree* sample queries.
    - xmark .... This folder contains *xmark* sample queries.
    - xmp .... This folder contains *xmp* sample queries.
  - src .... This folder contains all required sources for compilation.

3. Step into the *src* directory by typing

   ```
   > cd ./gcx_2-1_src/src
   ```

   in a command prompt window.

4. Optionally, but not necessarily, you may want to enable or disable one or more special features by uncommenting or adding `FLAGS` in the Makefile (Makefile.Windows). A complete list of all available compilation `FLAGS` can be found in subsection 1.4.

5. Now type

   ```
   > make -f Makefile.Windows
   ```

   to compile the sources. After compilation a binary file named *gcx.exe* will be created in the *bin* directory.

6. You might also consider to add the *bin* directory to your `PATH` variable.

1.4 Compiling with/without Special Features

There are several `FLAGS` that enable or disable one or more (special) features. These `FLAGS` can be found in both Makefiles (Makefile.Linux/Makefile.Windows) and have the following effects.

- `-DROLE_REFCOUNT`: Use reference counting instead of role (multi-)sets; this implementation is faster, but not suited for debugging purposes, since role IDs are "invisible". It is strongly recommended to turn this compile option ON.
- `-DNO_OPTIMIZATIONS`: Disable (most of the) optimizations; this should be used only for debugging purposes or to get better insights into the engine's internal processing strategy.
- `-DREWRITE_VARSTEPS`: Rewrite varstep expressions into for-loops. On the one hand this option causes earlier signOff statement execution but on the other hand it (might) interfere with other optimizations and therefore can slow down query evaluation.
- `-DVALIDATION`: Enable XML document validation; please note that only those parts of the XML document are validated that are kept according to the projection strategy. For the remaining part only depth is kept track of (but closing tags are not matched against opening tags). You should ignore this option if you are sure that your XML documents are well-formed.

By default, both Makefiles (Makefile.Linux/Makefile.Windows) come with

```
FLAGS = -DROLE_REFCOUNT.
```

If you want to adjust `FLAGS` to your own needs this must be done before compilation of the sources.

To change `FLAGS` you can either uncomment one of the following lines in your Makefile (Makefile.Linux/Makefile.Windows)

```
# FLAGS = -DROLE_REFCOUNT -DREWRITE_VARSTEPS
# FLAGS = -DROLE_REFCOUNT -DNO_OPTIMIZATIONS
# FLAGS = -DROLE_REFCOUNT -DNO_OPTIMIZATIONS -DREWRITE_VARSTEPS
```

by removing the `#` before one of these line or just type your own `FLAGS` line, for example

```
FLAGS = -DROLE_REFCOUNT -DVALIDATION
```

if you want to use role (multi-)sets instead of reference counting and want to ensure that your XML document is well-formed.

After changing `FLAGS` you need to *clean* and *rebuild* GCX by typing

```
> make -f Makefile.Linux clean all
```

or

```
> make -f Makefile.Windows clean all
```

depending on your operating system.

Warning: *Compiling GCX with different `FLAGS` such as `-DVALIDATION` for XML document well-formed validation or `-DNO_OPTIMIZATIONS` to disable (most of the) optimizations might significantly slow down query evaluation and is not a recommended compile option!*

2 Supported Fragment of XQuery 1.0

Currently GCX supports composition-free XQuery [1], i.e. without let-clauses, and allows to use the following syntactic constructs (whereas node comparisons in conditions are always string value comparisons).

- *comment* expressions "above" a query (*not* supported *inside* a query)
- arbitrary (well-formed) *XML elements* (with or without *PCDATA content*)
- *string constants* in output or conditions
- *numeric constants* in output or conditions
- *aggregate function* expressions in output or conditions supporting
    - **standard functions**: *fn:sum*, *fn:avg*, *fn:min*, *fn:max* and *fn:count*
    - **non-standard functions**: *fn:stddev_ samp*, *fn:stddev_ pop*, *fn:var_ samp*, *fn:var_ pop* and *fn:median*
- *rounding function* expressions in output or conditions supporting
    - **standard functions**: *fn:ceiling*, *fn:floor*, *fn:round* and *fn:round-half-to-even*
    - **non-standard functions**: *fn:abs*, *fn:cover* and *fn:truncate*
- arbitrarily deep-nested *sequences of expressions*
- nested *FWR* (for-where-return) expressions
- *if-then-else* expressions
- conditions support
    - **conjunctions**: *and*, *or*
    - **functions**: *fn:not*, *fn:exists*, *fn:empty*, *fn:true*, *fn:false*, *all aggregate function expressions* and *all rounding function expressions*
    - **relational operators**: $<$, $\leq$, $=$, $\geq$, $>$, $\neq$
- *variables* defined by FWR expressions (no *let-clause* support) in output or conditions (with or without *multi-step path* expressions)
- *multi-step path* expressions (arbitrarily length) with (optional) *fn:doc* function expression for specifying absolute paths using
    - **axis**: / (*child*::) or // (*descendant*::)
    - **node tests**: node(), text(), wildcard ($*$) or a tagname

The explicit grammar of the supported XQuery 1.0 fragment is provided in the following Figure 1.

$$
\begin{aligned}
\mathit{XQuery} &::= (\mathit{CommentExpr})?\ \mathit{XMLExpr}\\
\mathit{CommentExpr} &::= \textbf{(: [}\mathit{CommentExpr}\textbf{]*}\ \mathit{String}\ \textbf{[}\mathit{CommentExpr}\textbf{]* :)}\\
\mathit{XMLExpr} &::= \langle \mathit{QName}\rangle\ \mathit{NestedXMLExpr}\ \langle/\mathit{QName}\rangle\\
&\quad \mid \langle\mathit{QName}\rangle\langle/\mathit{QName}\rangle \mid \langle\mathit{QName}/\rangle\\
\mathit{NestedXMLExpr} &::= \textbf{\{}\mathit{QExpr}\textbf{\}} \mid \mathit{String} \mid \mathit{XMLExpr} \mid \mathit{NestedXMLExpr}\ \mathit{NestedXMLExpr}\\
\mathit{QExpr} &::= \mathit{ReturnQExpr} \mid \mathit{QExpr},\mathit{QExpr}\\
\mathit{ReturnQExpr} &::= \mathit{QExprSingle} \mid (\mathit{QExpr}) \mid ()\\
\mathit{QExprSingle} &::= \text{“}\mathit{String}\text{”} \mid \mathit{Numeric} \mid \mathit{FWRExpr} \mid \mathit{IfExpr} \mid \mathit{VarExpr}\\
&\quad \mid \mathit{AggregateFunct} \mid \mathit{RoundingFunct} \mid \mathit{NestedXMLExpr}\\
\mathit{FWRExpr} &::= \mathit{ForClause}\ [\textbf{where}\ \mathit{Condition}]?\ \textbf{return}\ \mathit{ReturnQExpr}\\
\mathit{ForClause} &::= \textbf{for \$}\mathit{VarName}\ \textbf{in}\ \mathit{VarExpr}\ [,\ \textbf{\$}\mathit{VarName}\ \textbf{in}\ \mathit{VarExpr}]^{*}\\
\mathit{IfExpr} &::= \textbf{if (}\mathit{Condition}\textbf{) then}\ \mathit{ReturnQExpr}\ \textbf{else}\ \mathit{ReturnQExpr}\\
\mathit{Condition} &::= \mathit{VarExpr} \mid \textbf{fn:true()} \mid \textbf{fn:false()} \mid \textbf{fn:exists(}\mathit{VarExpr}\textbf{)}\\
&\quad \mid \textbf{fn:empty(}\mathit{VarExpr}\textbf{)} \mid \textbf{fn:not(}\mathit{Condition}\textbf{)}\\
&\quad \mid \textbf{(}\mathit{Condition}\textbf{)} \mid \mathit{Condition}\ \textbf{and}\ \mathit{Condition}\\
&\quad \mid \mathit{Condition}\ \textbf{or}\ \mathit{Condition} \mid \mathit{Operand}\ \mathit{RelOp}\ \mathit{Operand}\\
\mathit{Operand} &::= \mathit{VarExpr} \mid \mathit{AggregateFunct} \mid \mathit{RoundingFunct} \mid \text{“}\mathit{String}\text{”} \mid \mathit{Numeric}\\
\mathit{RelOp} &::= \ < \ \mid\ <= \ \mid\ >= \ \mid\ > \ \mid\ = \ \mid !=\\
\mathit{AggregateFunct} &::= \textbf{fn:sum(}\mathit{VarExpr}\textbf{)} \mid \textbf{fn:avg(}\mathit{VarExpr}\textbf{)}\\
&\quad \mid \textbf{fn:min(}\mathit{VarExpr}\textbf{)} \mid \textbf{fn:max(}\mathit{VarExpr}\textbf{)}\\
&\quad \mid \textbf{fn:count(}\mathit{VarExpr}\textbf{)} \mid \textbf{fn:stddev\_samp(}\mathit{VarExpr}\textbf{)}\\
&\quad \mid \textbf{fn:stddev\_pop(}\mathit{VarExpr}\textbf{)} \mid \textbf{fn:var\_samp(}\mathit{VarExpr}\textbf{)}\\
&\quad \mid \textbf{fn:var\_pop(}\mathit{VarExpr}\textbf{)} \mid \textbf{fn:median(}\mathit{VarExpr}\textbf{)}\\
\mathit{RoundingFunct} &::= \textbf{fn:abs(}\mathit{AggregateFunct}\textbf{)} \mid \textbf{fn:ceiling(}\mathit{AggregateFunct}\textbf{)}\\
&\quad \mid \textbf{fn:cover(}\mathit{AggregateFunct}\textbf{)} \mid \textbf{fn:floor(}\mathit{AggregateFunct}\textbf{)}\\
&\quad \mid \textbf{fn:round(}\mathit{AggregateFunct}\textbf{)} \mid \textbf{fn:truncate(}\mathit{AggregateFunct}\textbf{)}\\
&\quad \mid \textbf{fn:round-half-to-even(}\mathit{AggregateFunct}\textbf{)}\\
\mathit{VarExpr} &::= \textbf{\$}\mathit{VarName} \mid \mathit{VarAxisExpr}\\
\mathit{VarAxisExpr} &::= \textbf{\$}\mathit{VarName}\ \mathit{PathExpr} \mid \mathit{PathExpr} \mid \mathit{DocPathExpr}\\
\mathit{DocPathExpr} &::= \textbf{fn:doc(}\text{“}\mathit{FileName}\text{”}\textbf{)} \mid \textbf{fn:doc(}\text{“}\mathit{FileName}\text{”}\textbf{)}\ /\ \mathit{PathStepExpr}\\
\mathit{PathExpr} &::= (/) \mid / \mid \mathit{PathStepExpr}\\
\mathit{PathStepExpr} &::= \mathit{Axis}\ \mathit{NodeTest} \mid \mathit{Axis}\ \mathit{NodeTest}\ /\ \mathit{PathStepExpr}\\
\mathit{Axis} &::= / \mid \textbf{/child::} \mid // \mid \textbf{/descendant::}\\
\mathit{NodeTest} &::= \textbf{node()} \mid \textbf{text()} \mid \textbf{*} \mid \mathit{QName}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{QName} &:= \text{tagname}\\
\mathit{String} &:= \text{string constant}\\
\mathit{Numeric} &:= \text{numeric constant}\\
\mathit{VarName} &:= \text{variable name (e.g. } \$x,\ \$y,\ \ldots)\\
\mathit{FileName} &:= \text{file name}
\end{aligned}
$$

Figure 1: Supported Fragment in GCX of XQuery 1.0

Note: *The fn:doc() construct (cf. rule DocPathExpr) fixes the input document; all absolute paths in the query will be bound to this document and will override XML input stream specification from command-line (such as --xml). When fn:doc() is used multiple times, GCX expects all occurrences to contain the same document, i.e. input from multiples documents at a time is currently not supported.*

3 Command-Line Arguments

Usage:

```
gcx [STD EVAL MODE] or
gcx [EXT EVAL MODE] or
gcx [INFO MODE]
```

To do a quick start, which corresponds to usage `gcx [STD EVAL MODE]`, you can run a query from file "*query.xq*" against XML document "*doc.xml*" by typing

```
> ./gcx --query query.xq --xml doc.xml (using Linux/Mac OS)
> gcx.exe --query query.xq --xml doc.xml (using Windows)
```

in a shell (Linux/Mac OS) or a command prompt window (Windows).

3.1 Standard Evaluation Mode (usage: `gcx [STD EVAL MODE]`)

The *standard evaluation mode* provides the GCX engine with standard I(nput)/O(utput) stream options. This means that the (input) query and also the (input) XML document each comes from their own file. The usage of the *standard evaluation mode* is

```
gcx [STD EVAL MODE]
```

with

```
[STD EVAL MODE] ::= --query <query_file> [--xml <xml_file>][1] [OPTION]?
```

whereas the `--query` argument fixes the (input) query file and the `--xml` argument fixes the (input) XML document. The optional `[OPTION]` part is described in subsection 3.4.

3.2 Extended Evaluation Mode (usage: `gcx [EXT EVAL MODE]`)

The *extended evaluation mode* provides the GCX engine with extended I(nput)/O(utput) stream options. This means that it is possible that the (input) query and also the (input) XML document comes from different sources. These options also allow to redirect the evaluation result and/or the debug output.

[1] --xml <xml_file> required if document is not given in query through *fn:doc*()

The usage of the *extended evaluation mode* is

```
gcx [EXT EVAL MODE]
```

with

```
[EXT EVAL MODE] ::= [STREAM SPEC]+ [OPTION]?
```

With `[STREAM SPEC]` either

`--iqstream [INPUT TYPE] [PARAM]?`: input stream type of query

`--ixstream [INPUT TYPE] [PARAM]?`: input stream type of xml

`--oestream [OUTPUT TYPE] [PARAM]?`: output stream type of query result

`--odstream [OUTPUT TYPE] [PARAM]?`: output stream type of debug output

whereas `[INPUT TYPE]` is either

`file`: file input (DEFAULT)
→ when used with --iqstream provide parameter --query <query_file>
→ when used with --ixstream provide parameter --xml <xml_file>[2]

`null`: no input (support only for --ixstream for debugging purposes)

`stdin`: standard input (either for query or for xml document)

and whereas `[OUTPUT TYPE]` is either

`file`: file output
→ when used with --oestream provide parameter --eout <eval_output_file>
→ when used with --odstream provide parameter --dout <debug_output_file>

`null`: no output

`stdout`: standard output (DEFAULT)

Note: *If you want to enter/paste the XML document directly into the shell or into the command prompt window you need to signal end-of-file (EOF) of the XML document by typing 2 × `Strg + D` (if using Linux/Mac OS) or `Strg + Z` (if using Windows).*

In summary, the arguments `--iqstream` (*input query stream*), `--ixstream` (*input XML stream*), `--oestream` (*output evaluation stream*) and `--odstream` (*output debug stream*) specify by their option the type of stream you want to use.

[2] required if document is not given in query through *fn:doc*()

The additional – if needed – arguments `--query` (*query input stream information*), `--xml` (*XML input stream information*), `--eout` (*evaluation output stream information*) and `--dout` (*debug output stream information*) specify by their option further stream information, such as the file where output is written to or the file from which input comes. The optional `[OPTION]` part is described in subsection 3.4. The following examples show some possible usage.

```
> gcx --query query.xq --xml doc.xml
```

→ query input from file “query.xq” and xml input from file “doc.xml”

→ query result output to stdout

```
> gcx --iqstream stdin --xml doc.xml --odstream file --dout debug.out
--debug
```

→ query input from stdin and xml input from file “doc.xml”

→ debug output to file “debug.out” and query result output to stdout

```
> gcx --query query.xq --xml doc.xml --oestream file --eout result.xml
--odstream null --debug
```

→ query input from file “query.xq” and xml input from file “doc.xml”

→ discard debug output and query result output to file “result.xml”

3.3 Information Mode (usage: `gcx [INFO MODE]`)

The *information mode* provides additional information about the GCX engine. For this purpose the following command-line arguments are available.

`--fragmentxq`: Prints supported XQuery fragment (XQ) (see also Figure 1).

`--version`: Prints version number and compile flags used during compilation.

`--about`: Prints about (license and author) information.

`--help`: Prints general usage information.

3.4 Debug Options

The following command-line arguments for debugging purpose are available.

`--debug`: Prints detailed debug information.

`--streamdebug`: Prints the projected XML stream together with additional debug information (in particular with associated roles). Stream preprojection as a stand-alone tool is currently not implemented in a memory efficient way, i.e. in this mode the whole stream is loaded into the buffer before it is output.

`--streamnodebug`: Prints the projected XML stream without additional debug information. Stream preprojection as a stand-alone tool is currently not implemented in a memory efficient way, i.e. in this mode the whole stream is loaded into the buffer before it is output.

4 Developers Stuff

If you are interested in implementing features for GCX by your own, want to know how GCX itself is implemented or just want to know how GCX looks "behind the scenes" you can find additional resources such as an *UML-Class-Diagram* or the (doxygen[3] generated) *source code documentation* at

http://sourceforge.net/project/showfiles.php?group_id=258398.

5 Sample Queries

All GCX bundles contain either the sources or the binary and a *set of sample queries* with corresponding query-depending *XML documents*. These sample queries including the corresponding XML document can be found – sorted by XML categories – in subdirectories of the *examples* folder (see also the directory listing in subsection 1.2).

6 Contact

For feedback, such as questions, comments, bug reports, or feature requests please use one of the following GCX mailing list.

- http://lists.sourceforge.net/mailman/listinfo/gcx-engine-general
  Mailing list for general discussion about GCX (general questions, comments . . .).
- http://lists.sourceforge.net/mailman/listinfo/gcx-engine-support
  Mailing list to ask questions about using and building GCX.
- http://lists.sourceforge.net/mailman/listinfo/gcx-engine-bugs
  Mailing list for bug reports and discussion about bugs in GCX.
- http://lists.sourceforge.net/mailman/listinfo/gcx-engine-requests
  Mailing list to request new or desired features for future releases.

To get in direct communication with us, feel free to send an email to

Michael Schmidt (mschmidt@informatik.uni-freiburg.de)

or

Gunnar Jehl (jehl@informatik.uni-freiburg.de).

[3] www.doxygen.org

7 Project Members

- Michael Schmidt, Freiburg University, Contact Person
- Gunnar Jehl, Freiburg University, Contact Person
- Prof. Dr. Christoph Koch, Cornell University
- Prof. Dr. Georg Lausen, Freiburg University
- Stefanie Scherzinger, IBM Böblingen

References

[1] Christoph Koch: *On The Complexity Of Nonrecursive XQuery And Functional Query Languages On Complex Values.*
ACM Transactions On Database Syststem, 31(4):1215–1256, 2006.
11

[2] Michael Schmidt, Stefanie Scherzinger, Christoph Koch: *Combined Static And Dynamic Analysis For Effective Buffer Minimization In Streaming XQuery Evaluation.*
Master's thesis, Universität des Saarlandes – Lehrstuhl für Informationssysteme, 2006.
http://www.informatik.uni-freiburg.de/~mschmidt/docs/thesis_csada.pdf.

[3] Michael Schmidt, Stefanie Scherzinger, Christoph Koch: *Combined Static And Dynamic Analysis For Effective Buffer Minimization In Streaming XQuery Evaluation.*
In *ICDE*, pages 236–245, 2007.