Narrowed Extended XPath I (NEXI)

Andrew Trotman
Department of Computer Science
University of Otago
Dunedin, New Zealand
andrew@cs.otago.ac.nz

Börkur Sigurbjörnsson
Informatics Institute
University of Amsterdam
Amsterdam, The Netherlands
borkur@science.uva.nl

ABSTRACT
INEX has through the years provided two types of queries: Content-Only queries (CO) and Content-And-Structure queries (CAS). The CO language has not changed much, but the CAS language has been more problematic. For the CAS queries, the INEX 02 query language proved insufficient for specifying problems for INEX 03. This was addressed by using an extended version of XPath, which, in turn, proved too complex to use correctly. Recently, an INEX working group identified the minimal set of requirements for a suitable query language for future workshops. From this analysis a new IR query language NEXI is introduced for upcoming workshops.

1. INTRODUCTION
The INEX [4] query working-group recently identified the query language requirements for future workshops. While no changes were suggested for the CO queries, several amendments were suggested for the CAS queries. The most over-riding requirement was a language continuing to look like XPath [2], but not XPath. An alternative syntax was proposed at the workshop [6].

The working group identified many aspects of XPath to be dropped (e.g. functions), aspects to be severely limited (e.g. the only operator to be allowed in a tag path is the descendant operator). New features were also added (e.g. the about() filter). The shape of XPath was considered appropriate while the verbosity was considered inappropriate. The complete list of changes is outlined in the working group report [8]. Amendments were considered sufficient to warrant an XPath derivative language. NEXI is now introduced as that language. Extra to the working group list, the use of wildcards in search terms has been dropped.

The most significant diversion from XPath is semantics. Whereas in XPath the semantics are defined, in NEXI the retrieval engine must deduce the semantics from the query. This is the information retrieval problem – and to do otherwise is to make it a database language. For clarity, strict and loose interpretations of the syntax are included herein, however these should not be considered the only interpretations of the language.

A NEXI parser has been implemented in Flex [7] and Bison [3] (the GNU tools compatible with LEX and YACC). The parser is made available for public use (and is included in the appendices).The existing INEX queries (queries 1-126) have been translated into NEXI (where possible) and are also included.

2. QUERY TYPES
There are currently two query types in INEX, the content only (CO) query and the content and structure (CAS) query [5].

2.1 The Content Only (CO) query
This is the traditional information retrieval query containing words and phrases. No XML [1] element restrictions are allowed, and no target element is specified. This kind of query occurs when a user is unfamiliar with the tagging structure of the document collection, or does not know where the result will be found. To answer a CO query a retrieval engine must deduce the information need from the query, identify relevant elements (of relevant documents) in the corpus, and return those sorted most to least relevant.

Deduction of the information need from the query is to determine semantics from syntax. This is the information retrieval problem, the problem being examined at INEX. As such, the queries must be considered as "hints" as to how to find relevant documents. Some relevant documents may not satisfy a strict interpretation of the query. Equally, some documents that do satisfy a strict interpretation of the query may not be relevant.

2.2 The Content And Structure (CAS) query
Content and structure queries may contain either explicit or implicit structural requirements. Such a query might arise if a user is aware of the document structure. To answer a CAS query a retrieval engine must deduce the information need from the query, identify elements that match structural requirements, and return those sorted most to least relevant. CAS queries can be interpreted in two ways, either strictly (SCAS) or loosely (VCAS).

2.2.1 The SCAS interpretation
The target structure of the information need can be deduced exactly from the query. All target-path constraints must be upheld for a result to be relevant. If a user asks for <sec> tags to be returned, these must be returned. All other aspects of the query are interpreted from the IR perspective, i.e. loosely.

2.2.2 The VCAS interpretation
Specifying an information need is not an easy task, in particular for semi-structured data with a wide variety of tag-names. Although the user may think they have a clear idea of the structural properties of the collection, there are likely to be aspects to which they are unaware. Thus we introduce a vague interpretation where target-path requirements need not be fulfilled. Relevance of a result will be based on whether or not it satisfies the information need. It will not be judged based on strict conformance to the target-path of the query

3. THE INEX TOPIC FORMAT
This discussion of the INEX topic format is included for context. As the topic format is likely to change from year to year readers are advised to consult the latest edition of the guidelines for topic development for complete details.

3.1 Restrictions on Queries

For an individual query to be useful for evaluation purposes it must satisfy several requirements (the details of which are explained below):

- It must be interpretable loosely. To satisfy this requirement, every query must contain at least one about() clause requiring an IR interpretation (i.e. non-numerical). That clause must occur in the final filter. In //A[B] queries, this is B. In //A[B]//C[D], this is D.
- It must not be a simple mechanical process to resolve the path. To satisfy this requirement, every query must be in the form //A[B] or //A[B]//C[D]. The form //A[B]//C is not allowed at INEX as the resolution of //C from //A[B] is a simple mechanical process.
- It must have more than 5 known results. If this cannot be satisfied, abandon the query and choose another.
- It must be "middle" complex. Perform the search and examine the top 25 results. If there are less than 2 or more than 20 relevant results, the query is not middle-complex.
- Queries should reflect a real information need. Contrived queries are unlikely to be accepted.
- Queries should be diverse. If submitting more than one query, please make each different.

3.2 Equivalence Tags

In the current INEX collection there are several tags used interchangeable (for historical paper-publishing reasons). Tags belonging to the following groups are considered equivalent and interchangeable in a query:

Paragraphs:

`ilrj, ip1, ip2, ip3, ip4, ip5, item-none, p, p1, p2, p3`

Sections:

`sec, ss1, ss2, ss3`

Lists:

`dl, l1, l2, l3, l4, l5, l6, l7, l8, l9, la, lb, lc, ld, le, list, numeric-list, numeric-rbrace, bullet-list`

Headings:

`h, h1, h1a, h2, h2a, h3, h4`

Due to tag equivalence, the query

```
//article//sec[about(.//p, Computer)]
```

and

```
//article//ss2[about(.//item-none, Computer)]
```

are identical.

3.3 Submission format

Topics are submitted in the INEX topic format detailed each year in the annual guidelines for topic development [5]. Detailed here is the 2003 format, which to date has not changed for subsequent workshops.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!ELEMENT inex_topic (title, description,
        narrative, keywords)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT narrative (#PCDATA)>
<!ELEMENT keywords (#PCDATA)>
<!ATTLIST inex_topic
        topic_id CDATA #REQUIRED
        query_type CDATA #REQUIRED>
```

<inex_topic topic_id=""> – Supplied by INEX once all topics have been collected. This and other attributes may be present in the final topics selected by INEX.

<inex_topic query_type=""> – either "CO" or "CAS". This attribute determines whether the topic is a content only (CO) or content and structure (CAS) topic. It consequently determines the query type used in the <title> tag.

<title> – a NEXI query (either CO or CAS, depending in the query_type attribute of the inex_topic tag). It should be noted the usual XML character encoding will be necessary, this includes substituting '<' with '<'. See sections 4 and 5 for details.

<description> – a short (one or two sentence) natural language translation of the title. Although this can be used by any track, it is also used by the Natural Language track as the query specification.

<narrative> – a detailed explanation of the information need including a description of what makes a result relevant. It should be possible for someone other than the author to read the narrative and a result and determine unambiguously if the result is relevant or not.

<keywords> – a comma separated list of terms and phrases used during the topic formulation.

It is important that the title, description, and narrative all describe the same information need.

3.4 Example of an INEX topic

```
<inex_topic query_type="CAS">
        <title>
                //article[.//yr = 2001 or .//yr =
                2002]//sec[about(.,summer
                holidays)]
        </title>
        <description>
                Summer holidays either of 2001 or
                of 2002.
        </description>
        <narrative>
                Return section elements, which are
                about summer holidays, where the
                sections is descendent of article
                element, and the article is from
                2001 or 2002.
        </narrative>
        <keywords>
                summer, holiday, 2001,2002
        </keywords>
</inex_topic>
```

3.5 Topic Titles

The topic title contains the information retrieval query expressed in NEXI. The syntax of such queries is precisely defined below and a parser written in FLEX and BISON is included in the appendices. It is the information retrieval problem to deduce the semantics from the information need, however no meaningful language can exist without semantics. This duality can only be resolved by strictly defining the semantics to be loose.

4. THE CONTENT ONLY (CO) QUERY

4.1 Searching for words and numbers

The smallest searchable unit in a CO query is the word:

```
word: NUMBER | ALPHANUMERIC

ALPHANUMERIC: {LETTER}{LETTERDIGITEXTRAS}*
NUMBER: "-"?{DIGIT}+

LETTER: [a-zA-Z]
DIGIT: [0-9]
LETTERDIGIT: [a-zA-Z0-9]
LETTERDIGITEXTRAS [a-zA-Z0-9'-]
```

Positive numbers, negative numbers and sequences of alphanumerics proceeded by an alphabetic character are all valid search words. Alphanumerics have already been used in query 41 so must be included. Hyphens are allowed after the first character of an alphanumeric (to avoid confusion with term restrictions, see section 4.3). The apostrophe can only occur after the first character of an alphanumeric.

Example: To search for the single word Apple, the CO query is

```
Apple
```

Loose interpretation: It is anticipated that using the word Apple will help locate relevant documents. I won't tell you if I mean "Macintosh Computer", "Granny Smith", or "Mr Apple" but find what I want anyway.

4.2 Searching for phrases

A phrase is a double quoted sequence of words:

```
phrase: '"' word_list '"'
word_list: word word | word_list word
```

A phrase must contain two or more words. A phrase containing only one word is erroneous and the quotes should be removed to make a single word query.

Example: To search for Charles Babbage, the CO query will be

```
"Charles Babbage"
```

Loose interpretation: Relevant documents are anticipated to contain these two words adjacent to each other, but need not. They may contain both words non-adjacent. For that matter they might not contain both words. A relevant document might not even contain either word.

4.3 Term restrictions

Terms can be preceded by either a plus (+) or minus (-) sign

```
term: term_restriction unrestricted_term
term_restriction: EMPTY | '+' | '-'
unrestricted_term: word | phrase
```

Loose interpretation: The '+' signifies the user expects the word will appear in a relevant element. The user will be surprised if a '-' word is found, but this will not prevent the document from being relevant. Words without a sign are specified because the user anticipates such terms will help the search engine to find relevant elements. As restrictions are only hints, it is entirely possible for the most relevant element to contain none of the query terms, or for that matter only the '-' terms.

4.4 CO queries

A CO query is a sequence of one or more searchable terms.

```
co : term | co term
```

Example:

```
+"face recognition" approach
```

Loose interpretation: "I expect the phrase 'face recognition' will appear in a relevant document, I also anticipate the word 'approach' will help you find the documents I want".

4.5 Bag of Words

Term ordering in IR queries is often assumed to be irrelevant. In the "bag of words" interpretation, a query is an unordered set of search terms (and phrases). The assumption does not hold true for some queries. For example,

```
computer history
```

and

```
history computer
```

express different information needs even though the "bag of words" is identical.

Additionally, if a term occurs multiple times, the occurrence count is lost when the term is added to the "bag of words". For some queries, multiple term occurrences are needed to adequately specify the information need. For example, the query

```
The The
```

should search for documents about the well known rock band of the same name, and cannot be specified without the use of the multiple occurring term. Further, some search engines "stop" common words not considered useful for searching (such as the, and, of, etc). This query requires the use of such a term.

Loose interpretation: There may or may not be an implied order to the terms in a query. If a term occurs multiple times this may or may not imply meaning. Stopping common words may or may not alter the meaning of the query.

4.6 The pitfalls of queries

The minus sign (-) maintains two meanings; it is used for both exclusionary terms and negative numbers. For the purpose of clarity, 12 and –12 are numbers. By inserting a space (represented as '␣' in this paragraph) between the – and the 12 (-␣12), the meaning is changed to exclusionary. "Don't search for the number –12" can be expressed as --12 or -␣-12. Equally, --␣12 is an error.

5. THE CONTENT AND STRUCTURE (CAS) QUERY

CAS queries can take three possible forms:

//A[B]	Return A tags about B
//A[B]//C	Return C descendants of A where A is about B (used in INEX'02)
//A[B]//C[D]	Return C descendants of A where A is about B and C is about D

A and C are paths whereas B and D are filters. The syntax is defined as:

```
cas: path cas_filter
       | path cas_filter path
       | path cas_filter path cas_filter

cas_filter: '[' filtered_clause ']'
```

Use of the form //A[B]//C is not useful for information retrieval evaluation purposes. Once the result of //A[B] has been determined, it is a mechanical process to extract the //C descendants. Use of this form was deprecated in INEX'03.

5.1 Path specification

Tag and attribute names follow the XML 1.1 [1] specification

```
XMLTAG: {XML_NAME}{XML_NAMECHAR}*

XML_NAMECHAR: [-_.:a-zA-Z0-9]
XML_NAME: [_:a-zA-Z]
```

Element nodes in the XML tree are identified as "//tag" and attribute nodes as "//@attribute". The wildcard "//*" is included to identify first or subsequent descendant (tag or attribute). Convoluted use of attributes and wildcards is discouraged.

```
node: named_node | any_node | tag_list_node

NODE_QUALIFIER: "//"

named_node: NODE_QUALIFIER tag
attribute_node: NODE_QUALIFIER '@' tag
any_node: NODE_QUALIFIER '*'
```

In cases where either tag A or tag B is required, it is written "//(A|B)".

```
tag_list: tag '|' tag | tag_list '|' tag
tag_list_node: NODE_QUALIFIER '(' tag_list ')'
```

A path through the XML tree is specified as a sequence of nodes. The only relationship between nodes in a path is descendant. There is no way to specify the child relationship or other XPath axes. Attributes cannot have descendant nodes so may only be specified at the end of a path.

```
path: node_sequence | node_sequence attribute_node
node_sequence: node | node_sequence node
```

Strict interpretation: "//A" is any A tag in the tree. "//A//B", any B descendant of an A tag in the tree. "//@C" is the C attribute of any tag. "//A//@C" is any C attribute anywhere in the tree beneath an A tag in the tree.

For any descendant of A use "//A//*". Any descendant of the root, "//*", is also any tag in the tree. "//*//*//*" is any tag at least three levels deep in the tree. "//*//A" is an A that is not the root of the tree, while "//*//A//*" means any descendant of A so long as A is not the root.

The path "//(A|B)" means any A tag in the tree or any B tag in the tree. "//(A|B)//(C|D)" is any C or D descendant of either an A or B tag. This includes "//A//C", "//A//D", "//B//C" and "//B//D". Convoluted use of this syntax is discouraged.

The path $//T_1...//T_n$ is an ordered sequence of nodes in the tree starting with T_1 and terminating at T_n such that for all $p \in n$, T_{p+1} is a descendant of T_p.

Loose interpretation: There is likely to be relevant information in the document in places not specified in a user query. The path specifications should therefore be considered hints as to where to look.

5.1.1 A Note on Attributes

No real query using attributes on the INEX collection is believed to exist. Query authors are discouraged from using attributes simply because they can.

5.2 Path filters

At present paths can be filtered either with search strings, or numerically. In future versions, filtering based on proper nouns (e.g. Author Names), and other data types is anticipated.

5.2.1 String filtering

Documents can be filtered to only those that satisfy a given textural (CO) query in the given path (or relative to the given path).

```
about_clause : ABOUT '(' relative_path ',' co ')'
relative_path: '.' | '.' path

ABOUT: "about"
```

Relative paths are specified relative to a context path. At B in //A[B] the context path is //A. At B in //A[B]//C[D] the context path is //A. At D in //A[B]//C[D] the context path is //A//C. The relative path "." is interpreted as "the context path". The relative path ".//p" is interpreted as "a p descendant of the context path".

Example:

```
//article[about(.//p, "information retrieval")]
```

Strict interpretation: "What ever you do, you must return article tags. Now, as a suggestion, look for //article//p elements about information retrieval."

Loose interpretation: "What I want is most likely a whole article that mentions information retrieval in a p tag. Relevant results are not limited to this, but I'm pretty sure it'll help you find what I want."

5.2.2 Arithmetic filtering

Documents can also be filtered to only those that satisfy a numeric query. As with string filtering, this is specified with a relative path.

```
arithmetic_clause: relative_path
       arithmetic_operator NUMBER

arithmetic_operator: '>' | '<' | '=' | '>=' | '<='
```

Example:

```
//article[.//pdt//yr = 2003]
```

Strict interpretation: Retrieve article elements from documents that loosely "contain the value 2003 in an //article//pdt//yr element".

Loose interpretation: A loose interpretation could be to look at a year range (2002, 2003, and 2004). This might be useful if, for example, a workshop held in December 2003, published the formal proceedings in 2004. Alternatively, a paper published electronically in December 2002 might finally appear in print in January 2004 leading to confusion over the publication date.

The above example could also be described using string filtering

```
//article[about(.//pdt//yr, 2003)]
```

however, the arithmetic syntax is preferred.

Both positive and negative numbers are supported by CO and CAS queries. The ambiguity arising from the multiple meaning of the minus (-) was discussed in section 4.6.

5.2.3 Boolean Operators

Path filters can be joined with Boolean operators AND and OR. They can also be bracketed.

```
filter: about_clause | arithmetic_clause

filtered_clause: filter
       | filtered_clause AND filtered_clause
       | filtered_clause OR filtered_clause
       | '(' filtered_clause ')'

AND: "AND" | "and"
OR: "OR" | "or"
```

Examples:

```
//article[about(., apple) and about(., computer)]

//article[about(., apple) or about(., computer)]
```

Strict interpretation: The first example will return article elements from documents about apple and about computer, the second about apple or about computer (remember: these are only hints). This introduces a subtle difference in query meaning between the two queries:

```
//article[about(.//sec, apple computer)]
```

and

```
//article[about(.//sec, apple) and
       about(.//sec, computer)]
```

The first query asks for articles that have a section discussing 'apple computer'. The second asks for articles that have a section discussing 'apple' and a section discussing 'computer' (even if they are not the same section). In the first query, the topics must co-occur. In the second they may co-occur.

Loose interpretation: AND is interpreted as ANDish, OR as ORish. The query contains the Boolean operators strictly as hints

on how to resolve the information need. CO, SCAS and VCAS all interpret Boolean operators loosely.

5.2.4 Examples

Examples of some CAS queries are given here along with strict interpretations. Loose interpretation of each is the same "I'm sure this'll help find what I want".

```
//sec[about(., mobile electronic payment system)]
```

Return sec tags where the sec tag mentions mobile electronic payment systems.

```
//*[about(., singular value decomposition)]
```

Return elements about singular value decomposition. The retrieval engine must deduce the most appropriate element to return.

```
//article[.//fm//yr >= 1998]//sec[about(.//p,
"virtual reality")]
```

Return sec tags of documents about virtual reality and published on or after 1998.

```
//article[(.//fm//yr = 2000 OR .//fm//yr = 1999)
AND about(., "intelligent transportation
system")]//sec[about(., automation +vehicle)]
```

Return sec elements about vehicle automation from documents published in 1999 or 2000 that are about intelligent transportation systems.

6. CONCLUSIONS

The INEX query working-group at the INEX workshop outlined a set of requirements necessary for a query language to be used for future workshops. The language was to be similar in form to XPath, while at the same time being both severely reduced, and expanded. The language, NEXI, is defined herein and satisfies these needs.

A parser written in Flex and Bison and is included. The existing INEX topics have been translated into NEXI and checked against the parser. Only those queries using features deprecated by the working-group could not be translated – in these cases a near translation is included.

7. ACKNOWLEDGEMENTS

Richard A. O'Keefe read several drafts and commented on many aspects of this language.

8. REFERENCES

[1] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., Yergeau, F., & Cowan, J. (2003). Extensible markup language (XML) 1.1 W3C proposed recommendation. The World Wide Web Consortium. Available: http://www.w3.org/TR/2003/PR-xml11-20031105/ [2003.

[2] Clark, J., & DeRose, S. (1999). XML path language (xpath) 1.0, W3C recommendation. The World Wide Web Consortium. Available: http://www.w3.org/TR/xpath [2004.

[3] Donnelly, C., & Stallman, R. (1995). Bison - the yacc-compatible parser generator. Available: http://www.gnu.org/directory/bison.html.

[4] Fuhr, N., Gövert, N., Kazai, G., & Lalmas, M. (2002). INEX: Initiative for the evaluation of XML retrieval. In *Proceedings of the ACM SIGIR 2000 Workshop on XML and Information Retrieval*.

[5] Kazai, G., Lalmas, M., & Malik, S. (2003). INEX'03 guidelines for topic development.

[6] O'Keefe, R. A., & Trotman, A. (2003). The simplest query language that could possibly work. In *Proceedings of the 2nd workshop of the initiative for the evaluation of XML retrieval (INEX)*.

[7] Paxson, V. (1995). Flex, version 2.5, a fast scanner generator. Available: http://www.gnu.org/directory/flex.html.

[8] Sigurbjörnsson, B., & Trotman, A. (2003). Queries: INEX 2003 working group report. In *Proceedings of the 2nd workshop of the initiative for the evaluation of XML retrieval (INEX)*.

A1. MAKEFILE

```
#
#  Makefile
#  --------
#  Andrew Trotman
#  University of Otago 2004
#
#  Script to build the NEXI parser
#
tokenizer : parser.tab.c lex.yy.c
   gcc lex.yy.c parser.tab.c -lm -o tokenizer

lex.yy.c : tokenizer.l parser.tab.h
   flex tokenizer.l

parser.tab.c : parser.y
   bison parser.y -d

clean :
   rm tokenizer parser.tab.h parser.tab.c lex.yy.c
```

A2. FLEX SCRIPT

```
%{
/*
   TOKENIZER.L
   -----------
   Andrew Trotman
   University of Otago 2004

   FLEX script to tokenize INEX NEXI queries and
check for syntax errors
```

```
*/
#include <stdio.h>
#include "parser.tab.h"
int c;
extern int yylval;
extern int line_number;
extern int char_number;

%}

LETTER [a-zA-Z]
DIGIT [0-9]
LETTERDIGIT [a-zA-Z0-9]
LETTERDIGITEXTRAS [a-zA-Z0-9'\-]
XML_NAMECHAR [a-zA-Z0-9_:.\-]
XML_NAME [a-zA-Z:_]

%%

" " { char_number++; }

"\r" { char_number++; }

"\n"    {
   line_number++;
   char_number = 1;
   return yytext[0];
   }

"about" {
   char_number += 5;
   yylval = yytext[0];
   return ABOUT;
   }

"AND"   {
   char_number += 3;
   yylval = yytext[0];
   return AND;
   }

"and"   {
   char_number += 3;
   yylval = yytext[0];
   return AND;
   }

"OR"    {
   char_number += 2;
   yylval = yytext[0];
   return OR;
   }

"or"    {
   char_number += 2;
   yylval = yytext[0];
   return OR;
   }

">" {
   char_number++;
   yylval = yytext[0];
   return GREATER;
   }

"<" {
   char_number++;
   yylval = yytext[0];
   return LESS;
   }

"=" {
   char_number++;
   yylval = yytext[0];
   return EQUAL;
   }

{LETTER}{LETTERDIGITEXTRAS}* {
   char_number += strlen(yytext);
   yylval = yytext[0];
   return ALPHANUMERIC;
   }

"-"?{DIGIT}+   {
   char_number += strlen(yytext);
   yylval = yytext[0];
   return NUMBER;
   }

"//"    {
   char_number += 2;
   yylval = yytext[0];
   return NODE_QUALIFIER;
   }

{XML_NAME}{XML_NAMECHAR}*     {
   char_number += strlen(yytext);
   yylval = yytext[0];
   return XMLTAG;
   }

.  {
   char_number++;
   return yytext[0];
   }

%%
```

```
/*
   YYWRAP()
   --------
*/
int yywrap(void)
{
return 1;
}
```

A3. BISON SCRIPT

```
%{
/*
   PARSER.Y
   --------
   Andrew Trotman
   University of Otago 2004

   BISON script to tokenize INEX NEXI queries and check for syntax errors
*/

#define YYDEBUG 1
#include <math.h>
#include <stdio.h>
#include <ctype.h>

int line_number = 1;
int char_number = 1;
extern char *yytext;

void yyerror(char *err) /* Called by yyparse on error */
{
printf ("Line %d (char %d): %s at '%s'\n", line_number, char_number, err, yytext);
}

/*
   NOTES:
       INEX topics 10, 14, 19, 20 are not strict translations as they cannot be expressed (multiple specified target elements)
       INEX topic 13 is not a strict translation due to instance (au[1]) usage
*/

%}

%token NUMBER ALPHANUMERIC XMLTAG
%token ABOUT NODE_QUALIFIER
%token AND OR
%token GREATER LESS EQUAL

%left AND OR

%%/* Grammar rules and actions follow */

input: /* empty */ | input line;

line: '\n'
   | co '\n'  { printf("CO Passed\n"); }
   | cas '\n' { printf("CAS Passed\n"); };

/*
   in a CAS query:
       the initial can be the terminal "//*" to specify "a descendant of"
       the final part can be an unrestricted target path (for compatibility with INEX 2002)
*/
cas: path cas_filter | path cas_filter path | path cas_filter path cas_filter;

cas_filter: '[' filtered_clause ']';

filtered_clause : filter
       | filtered_clause AND filtered_clause
       | filtered_clause OR filtered_clause
       | '(' filtered_clause ')';

filter: about_clause | arithmetic_clause;

about_clause : ABOUT '(' relative_path ',' co ')';

arithmetic_clause: relative_path arithmetic_operator NUMBER;

arithmetic_operator: GREATER | LESS | EQUAL | greater_equal | less_equal;

greater_equal: GREATER EQUAL;

less_equal: LESS EQUAL;

/*
   child has been eliminated and replaced with descendant.  In the unlikley event child is ever needed, it can (most likley) be specified as those descendants enough to make the specification unambigious.

now, a PATH is either:
   "//" for root
   "//A" for tag A
   "//A//B" for tag B within tag A
   "//*" for any tag
   "//A//*" for any descendant of A
   "//@A" for attribute A
   "//A//@B" for attribute B descendant of node A
*/
path: node_sequence | node_sequence attribute_node;

relative_path: '.' | '.' path;
```

```
node_sequence: node | node_sequence node;

any_node: NODE_QUALIFIER '*';

attribute_node: NODE_QUALIFIER '@' tag;

named_node: NODE_QUALIFIER tag;

tag_list: tag '|' tag | tag_list '|' tag;

tag_list_node: NODE_QUALIFIER '(' tag_list ')';

node: named_node | any_node | tag_list_node;

tag: alphanumeric | XMLTAG;

/*
   CO topics are sequences of numbers, terms and phrases with optional specifiers mandatory (+) and unwanted (-)
note:
   "12" is a number
   "-12" is number
   "- 12" is don't search for number 12
   "--12" | "- -12" is don't search for number -12
   "-- 12" is an error
   "content-based" is an error
*/
co : term | co term;

term: term_restriction unrestricted_term;

term_restriction: /* empty */ | '+' | '-';

unrestricted_term: word | phrase;

/*
   A phrase is a sequence of two or more words surounded by double quotes
*/
phrase: '"' word_list '"';

word_list: word word | word_list word;

/*
   a word is a sequence:
        of alphabetics
        of digits
        of digits preceeded by a negative (-) sign (a negative number)
        alphanumerics starting with an alpha (for both ip1 tags and Y2K queries)
   As the operators are also valid search terms, a word is
   operator or a sequence of alphabetic characters
*/

word: NUMBER | alphanumeric;

alphanumeric : ALPHANUMERIC | ABOUT | AND | OR;

%%

/*
   MAIN ()
   -------
*/
int main(void)
{
//yydebug = 1;
yyparse();

return 0;
}
```

A4. INEX QUERIES 1-126

The pre-existing INEX queries have all been converted and checked against the parser. Topics 10, 14, 19 and 20 originally specified a set of target elements. This practice was banned for INEX'03 and is not supported here either. Topic 13 specifies a particular instance of an element as the target, again outlawed for INEX'03 and not supported here. Topic 44 used wildcards. As such, these 6 queries are not accurately translated.

1. `//article[about(.//(abs|kwd), description logics)]//fm//au`
2. `//ack[about(., research funded america)]`
3. `//*[about(.//kwd, information data visualization) and about(., large information hierarchies spaces multidimensional data databases)]`
4. `//*[about(.//(atl|abs|st), experience results problems) and about(., extreme programming)]`
5. `//article[about(.//bibl, QBIC) and about(., image retrieval)]//tig`
6. `//article[about(., Survey on Software Engineering) and about(.//sec, programming languages)]//tig[about(., software engineering survey programming survey programming tutorial software engineering tutorial)]`
7. `//article[about(., Content-based retrieval of video databases)]//sec`
8. `//article[about(.//fm//aff, ibm) and about(.//bdy//sec, certificates)]`
9. `//article[about(.//bdy//sec, nonmonotonic reasoning) and (.//hdr//yr = 1999 or .//hdr//yr = 2000) and about(.//tig//atl, -calendar) and about(., belief revision)]`
10. `//*[about(.//(atl|st|title), book review) and about(.//(st|p), machine learning adaptative algorithm probabilistic model neural network support vector machine kernel methods numerical computation)]`

11. //*[about(.//p, wireless) and about(.//(abs|kwd), security) and about(., security applications)]
12. //article[.//pdt//yr = 2001 or .//pdt//yr = 2002]//bdy//sec[about(., internet search engine)]
13. //article[about(.//fm//au//@sequence, additional) and about(.//fm//abs, review) and about(., AR VR virtual augmented reality system)]//fm//au
14. //*[about(.//fgc, Corba architecture) and about(.//p, Figure Corba Architecture)]
15. //article[.//fm//hdr//hdr2//pdt = 1996 or .//fm//hdr//hdr2//pdt = 1997]//bm//bib//bibl//bb[about(., hypercube mesh torus toroidal non-numerical database)]
16. //article[about(.//bm//bib//bibl//bb//atl, concurrency control)]//fm//tig//atl
17. //article[about(.//fm//au, -W -Bruce -Croft)]//bb[about(.//au, W Bruce Croft)]
18. //article[about(., Hypertext Information Retrieval) and about(.//bib//bibl//bb//atl, Hypertext Information Retrieval)]
19. //*[about(., singular value decomposition svd formula)]
20. //article[about(.//atl, Concurrency Control) and about(.//fm//hdr//hdr1//ti, data) and about(., Concurrency Control in real-time databases)]//sec
21. //*[about(.//(p|st|it|bb), recommender system recommender agent)]
22. //article[about(.//bb//au//snm, Mannila) and (about(.//bb//au//fnm, Heikki) or about(.//bb//au//fnm, H)) and about(., Mannila)]//fm//au
23. //article[(.//yr = 1995 or .//yr = 1996 or .//yr= 1997 or .//yr = 1998 or .//yr = 1999) and about(.//bdy, XML electronic commerce)]
24. //article[about(.//au, Smith Jones) and about(.//bdy, software engineering and process improvement)]
25. //article[about(.//fm//hdr//hdr1//ti, IEEE MultiMedia) and about(., QoS Quality of Service)]
26. //article[about(.//st, XML) and about(., data processing system)]//fm//tig//atl
27. //article[about(.//atl, 1999 Reviewers List) and about(.//ti, IEEE Transactions Visualization and Computer Graphics) and .//yr = 2000]//reviewer//name
28. //article[about(.//sec1//title, Special Feature) and about(.//ti, IEEE Micro)]//atl
29. //*[about(.//atl, image retrieval) and about(., image retrieval colour shape texture)]
30. //article[.//yr >= 1996 and about(., parallelism)]//au
31. computational biology
32. semantic web
33. software patents
34. Efficient database search structures and techniques
35. Parallel query optimization
36. Heat dissipation of microcomputer chips
37. Temporal database queries and query processing
38. multidimensional indices
39. Video on demand
40. Content-based retrieval
41. Y2K spending
42. Decryption of the Enigma code
43. approximate string matching algorithm
44. internet society communication netizen social sociology web usenet mail network culture
45. augmented reality and medicine
46. Firewalls in internet security
47. concurrency control semantic transaction management application performance benefit
48. active database rule specification
49. Query relaxation approximate and intelligent query answering
50. XML editors or parsers
51. Text Data Mining
52. History of Computing of USSR
53. information retrieval xml
54. knowledge building acquisition and sharing
55. Digital Divide city planning neighbourhood planning
56. open hypermedia systems and agents
57. public key cryptography RSA EC DSA algebraic number field
58. Location management scheme
59. schema integration methods
60. Internet speed
61. //article[about(.,clustering +distributed) and about(.//sec,java)]
62. //article[about(.,security +biometrics) AND about(.//sec,"facial recognition")]
63. //article[about(.,"digital library") AND about(.//p, +authorization +"access control" +security)]
64. //article[about(., hollerith)]//sec[about(., DEHOMAG)]
65. //article[.//fm//yr > 1998 AND about(., "image retrieval")]
66. //article[.//fm//yr < 2000]//sec[about(.,"search engines")]
67. //article//fm[about(.//(tig|abs), +software +architecture) and about(., -distributed -Web)]
68. //article[about(., +Smalltalk) or about(., +Lisp) or about(.,+Erlang) or about(., +Java)]//bdy//sec[about(., +"garbage collection" +algorithm)]
69. //article//bdy//sec[about(.//st,"information retrieval")]
70. //article[about(.//fm//abs, "information retrieval" "digital libraries")]

71. `//article[about(.,formal methods verify correctness aviation systems)]//bdy//*[about(.,case study application model checking theorem proving)]`
72. `//article[about(.//fm//au//aff,United States of America)]//bdy//*[about(.,weather forecasting systems)]`
73. `//article[about(.//st,+comparison) and about(.//bib,"machine learning")]`
74. `//article[about(., video streaming applications)]//sec[about(., media stream synchronization) OR about(., stream delivery protocol)]`
75. `//article[about(., Petri net) AND about(.//sec, formal definition) AND about(.//sec, algorithm efficiency computation approximation)]`
76. `//article[(.//fm//yr = 2000 OR .//fm//yr = 1999) AND about(., "intelligent transportation system")]//sec[about(.,automation +vehicle)]`
77. `//article[about(.//sec,"reverse engineering")]//sec[about(., legal) OR about(.,legislation)]`
78. `//vt[about(.,"Information Retrieval" student)]`
79. `//article[about(.,XML) AND about(.,database)]`
80. `//article//bdy//sec[about(.,"clock synchronization" "distributed systems")]`
81. `//article[about(.//p,"multi concurrency control") AND about(.//p, algorithm) AND about(.//fm//atl, databases)]`
82. `//article[about(.,handwriting recognition) AND about(.//fm//au,kim)]`
83. `//article//fm//abs[about(., "data mining" "frequent itemset")]`
84. `//p[about(.,overview "distributed query processing" join)]`
85. `//article[.//fm//yr >= 1998 and .//fig//no > 9]//sec[about(.//p,VR "virtual reality" "virtual environment" cyberspace "augmented reality")]`
86. `//sec[about(.,mobile electronic payment system)]`
87. `//article[(.//fm//yr = 1998 OR .//fm//yr = 1999 OR .//fm//yr = 2000 OR .//fm//yr = 2001 OR .//fm//yr = 2002) AND about(., "support vector machines")]`
88. `//article[(.//fm//yr = 1998 OR .//fm//yr = 1999 OR .//fm//yr = 2000 OR .//fm//yr = 2001) AND about(., "web crawler")]`
89. `//article[about(.//bdy,clustering "vector quantization" +fuzzy +k-means +c-means -SOFM -SOM)]//bm//bb[about(.,"vector quantization" +fuzzy clustering +k-means +c-means) AND about(.//pdt,1999) AND about(.//au//snm, -kohonen)]`
90. `//article[about(.//sec,+trust authentication "electronic commerce" e-commerce e-business marketplace)]//abs[about(., trust authentication)]`
91. Internet traffic
92. "query tightening" "narrow the search" "incremental query answering"
93. "Charles Babbage" -institute -inst
94. "hyperlink analysis" +"topic distillation"
95. +"face recognition" approach
96. +"software cost estimation"
97. Converting Fortran source code
98. "Information Exchange" +XML "Information Integration"
99. perl features
100. +association +mining +rule +medical
101. +"t test" +information
102. distributed storage systems for grid computing
103. UML formal logic
104. Toy Story
105. +categorization "textual document" learning evaluation
106. Content protection schemes
107. "artificial intelligence" AI practical application industry "real world"
108. ontology ontologies overview "how to" practical example
109. "CPU cooling" "cooling fan design" "heatsink design" "heat dissipation" airflow casing
110. "stream delivery" "stream synchronization" audio video streaming applications
111. "natural language processing" -"programming language" -"modeling language" +"human language"
112. +"Cascading Style Sheets" -"Content Scrambling System"
113. "Markov models" "user behaviour"
114. +women "history of computing"
115. +"IP telephony" +challenges
116. "computer assisted art" "computer generated art"
117. Patricia Tries
118. "shared nothing" database
119. Optimizing joins in relational databases
120. information retrieval models
121. Real Time Operating Systems
122. Lossy Compression Algorithm
123. multidimensional index "nearest neighbour search"
124. application algorithm +clustering +k-means +c-means "vector quantization" "speech compression" "image compression" "video compression"
125. +wearable ubiquitous mobile computing devices
126. Open standards for digital video in distance learning