Mining Discourse Treebanks with XQuery

Xuchen Yao [*]
Johns Hopkins University
Computer Science
xuchen@cs.jhu.edu

Gosse Bouma
University of Groningen
Information Science
g.bouma@rug.nl

Abstract

We argue that a combination of XML and XQuery provides a generic and powerful method for representing and querying complex (multilevel) annotated corpora. XML is a widely used standard for coding and distributing annotated corpora, but the advantages of techniques for processing XML are not always realized in full. We show that XQuery, a completely generic query language for XML, has the expressive power required for advanced linguistic queries, its modular nature can help in providing corpus-specific functionality to users, and support of XQuery by XML database systems allows large corpora to be searched efficiently.

1 Introduction

Annotated corpora can contain information on many different aspects of linguistic structure, and as a consequence, tools to search annotated corpora differ quite substantially. Even for corpora with the same level of annotation, many different search tools exist. Lai and Bird [4], for instance, evaluate six different query languages (Tgrep2, TIGERSearch, Emu, CorpusSearch, NiteQL and LPath) on seven common search tasks for syntactic treebanks. All languages succeed in at least five of the seven tasks. The source code for each of the languages shows clearly, however, that these languages have differences in syntax and need special attention to work properly. As each corpus tends to support only a single query language, this means that users, especially those working with multiple corpora, must learn to work with a different query language each time they need to use a different corpus.

[*] We would like to thank the Erasmus Mundus European Masters Program in Language and Communication Technologies (EM-LCT) for support. This work was carried out while the first author was an EM-LCT student at the University of Groningen.

For corpora encoding different levels of annotation, the situation can be even more frustrating. The Penn Discourse TreeBank (Prasad et al. [9]), which we discuss in more detail below, provides annotation of discourse structure for data that is syntactically annotated in the Penn Treebank (Marcus et al. [7]). Although pointers from the discourse annotation to the syntactic annotation are given, it is not the case that both levels of annotation are available in a single format. Tools are provided which support searching the discourse annotation while imposing syntactic constraints, but the functionality of these tools is limited, and many naturally occurring questions require additional programming.

A final problem with many linguistic query languages is that they are exactly that: they allow the formulation of queries that return fragments of the corpus satisfying the constraints formulated in the query, but they allow little or no control over the output of the query. Many lexical acquisition tasks require pairs of items (such as pairs of verb (stems) and (the (stem of the) head of its) direct object) to be extracted. For the Penn Discourse Treebank, one might be interested in listing all discourse connectives, along with the syntactic category of their arguments. Such tasks require languages which not only support selection of fragments, but also support for selection of elements in (the context of) matching fragments, and some form of control over the resulting output.

The majority of modern corpora are made available in XML or can be converted to XML. Given the fact that very powerful languages for processing XML are available (most notably, Xpath for searching, and XSLT and XQuery for processing and querying), the question naturally arises to what extent such languages make corpus specific search tools superfluous, and to what extent these generic languages can overcome some of the shortcomings of linguistic query languages.

In this paper, we argue that XML and generic XML processing languages, XQuery in particular, allow a uniform method for representing complex linguistic data, and for searching and extracting data from complex corpora. We use the Penn Discourse Treebank (PDTB) as an example. We use an XMLized version of the PDTB and show that data discussed in recent research using the PDTB can be extracted from the corpus using XQuery. Finally, we discuss efficiency issues.

The original PDTB is encoded in plain text and shipped with PDTB API, a Java package, to accomplish common query tasks. Conversion of the PDTB to XML means that both the discourse information and the syntactic information of the corresponding constituents can be represented in a uniform way in a single XML file. Consequently, querying such files with XQuery becomes possible. This has several advantages:

- XQuery is capable of extracting both tree-based information (such as syntactic trees) and nontree-like information (such as discourse relations) or more generally, “non-tree navigation” (Lai and Bird [4]).

- XQuery is capable of extracting information from the discourse annotation and syntactic annotation simultaneously (given a linking between the two, which is already provided by the original encoding), which is beyond the ability of the original PDTB APIs.
- As a functional programming language, XQuery code can be modular, reusable and extensible, and thus answers the call for “reusable, scalable software” in Lai and Bird [4]. A Javadoc style documentation mechanism[1] exists. Corpus-specific modules allow developers to hide much of the complexity of the underlying XML and can help users to access relevant parts of the annotation easily.
- XQuery is the *de facto* standard for querying XML databases. By storing the corpus in an XML database, fast and easy-to-manage retrieval becomes available, while XQuery can still be used to perform complex queries.

In section 2 the format of PDTB-XML is introduced. Queries which need to access both the syntactic and discourse annotation and the way in which these can be implemented in XQuery are introduced in section 3. XML databases and their efficiency on linguistic queries are tested in section 4. The final section concludes and addresses future development. Most of the algorithms described in this paper are implemented as XQuery APIs.[2]

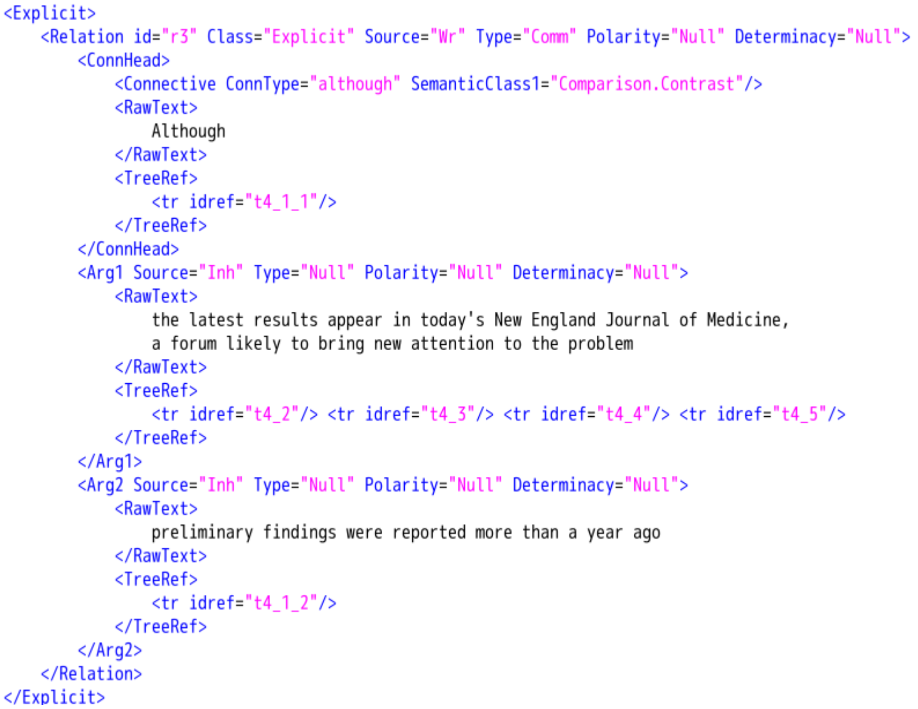
2 PDTB-XML

The XMLized Penn Discourse TreeBank (PDTB-XML, Yao et al. [12]) is an XML version of the PDTB (Prasad et al. [9]), created to support unrestricted access to both discourse and syntactic annotation. The original PDTB corpus uses a specific format[3] for its annotation. Each annotated article corresponds to three files, containing the original sentences, syntactic trees and discourse relations. A set of PDTB APIs is provided to support access to and visualisation of the annotation. Due to this architecture, the PDTB itself is not easily extensible or modifiable. By converting the annotation to XML files, the three separate annotation layers can be stored in a single XML file and annotations can be made more explicit and thus easier to understand and use by introducing elements and attributes. PDTB XML also inherits the merit of extensibility from XML.

[1] http://xqdoc.org/

[2] code.google.com/p/pdtb-xml/source/browse/trunk/xquery/pdtb.xq

[3] www.seas.upenn.edu/~pdtb/PDTBAPI/pdtb-annotation-manual.pdf



```
<Explicit>
    <Relation id="r3" Class="Explicit" Source="Wr" Type="Comm" Polarity="Null" Determinacy="Null">
        <ConnHead>
            <Connective ConnType="although" SemanticClass1="Comparison.Contrast"/>
            <RawText>
                Although
            </RawText>
            <TreeRef>
                <tr idref="t4_1_1"/>
            </TreeRef>
        </ConnHead>
        <Arg1 Source="Inh" Type="Null" Polarity="Null" Determinacy="Null">
            <RawText>
                the latest results appear in today's New England Journal of Medicine,
                a forum likely to bring new attention to the problem
            </RawText>
            <TreeRef>
                <tr idref="t4_2"/> <tr idref="t4_3"/> <tr idref="t4_4"/> <tr idref="t4_5"/>
            </TreeRef>
        </Arg1>
        <Arg2 Source="Inh" Type="Null" Polarity="Null" Determinacy="Null">
            <RawText>
                preliminary findings were reported more than a year ago
            </RawText>
            <TreeRef>
                <tr idref="t4_1_2"/>
            </TreeRef>
        </Arg2>
    </Relation>
</Explicit>
```

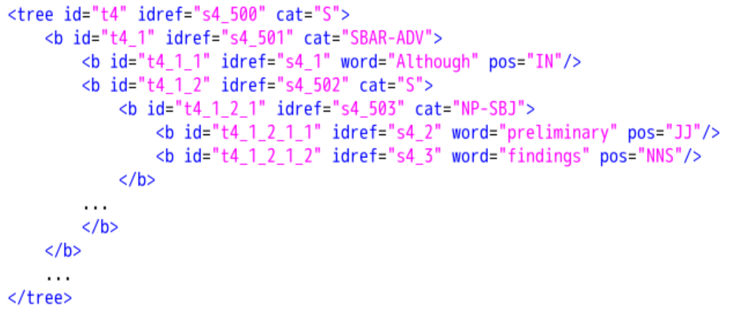
Figure 1: Example of an *Explicit* relation in XML.

There are five discourse relations in the PDTB (*Explicit, Implicit, AltLex, EntRel, NoRel*). Each relation has two arguments (*Arg1* and *Arg2*) and two optional supplements (*Sup1* and *Sup2*). Sentence (1) contains an *Explicit* discourse relation (the connective is underscored, *Arg1* is in italics and **Arg2** is in bold):

(1) Although **preliminary findings were reported more than a year ago**, *the latest results appear in today's New England Journal of Medicine, a forum likely to bring new attention to the problem.* (wsj_0003)

Figure 1 contains the annotation of (1) in XML. Instead of using fields, the role of each text fragment in the relation is made explicit by means of element names and attribute names. The link with the syntactic annotation is given in the `<TreeRef>` element associated with each `<ConnHead>, <Arg1>` and `<Arg2>`. A `<TreeRef>` contains one or more `<tr>` elements pointing to a node in a syntactic tree.

The syntactic annotation (i.e. the PTB) is encoded using the TIGER-XML format (Brants et al. [1]). Every sentence is syntactically represented by a *graph* containing *terminals* and *nonterminals*, where nonterminals have *edges* connecting to terminals. As this graph does not give a direct tree structure an extra *tree*



```
<tree id="t4" idref="s4_500" cat="S">
    <b id="t4_1" idref="s4_501" cat="SBAR-ADV">
        <b id="t4_1_1" idref="s4_1" word="Although" pos="IN"/>
        <b id="t4_1_2" idref="s4_502" cat="S">
            <b id="t4_1_2_1" idref="s4_503" cat="NP-SBJ">
                <b id="t4_1_2_1_1" idref="s4_2" word="preliminary" pos="JJ"/>
                <b id="t4_1_2_1_2" idref="s4_3" word="findings" pos="NNS"/>
            </b>
        ...
        </b>
    </b>
    ...
</tree>
```

Figure 2: Fragment of a syntactic tree

element has been added to build the cross references between the syntactic part and the discourse relation part, as illustrated in Figure 2.

The PDTB-XML re-organizes the annotation format of the PDTB without loss of information, but with the advantage of a uniform, integrated, representation, and the possibility of future extensions. In the next section, we show how this uniform XML representation supports search and extraction tasks which need to refer to both discourse and syntax.

3 Search and Extraction with XQuery

3.1 XQuery and XPath

XQuery (Walmsley [11]) is a W3C recommendation[4] for querying XML databases. It uses the XPath standard[5] for locating elements in an XML document. A simple example of an XQuery script, which returns all relations with an *Explicit* connective of type *although*, is given below.

```
for $rel in //Relation[@Class="Explicit" and ConnHead/
    Connective[@ConnType="although"]]
return $rel
```

The `for` loop iteratively loops over all `<Relation>` elements somewhere inside the document. The @ symbol refers to attributes of an element and restricts the relations we are interested in to those of `Class` "`Explicit`". Furthermore, we require that the `<Relation>` must contain a `<ConnHead>` element which contains a `<Connective>` element whose `ConnType` attribute has the value "`although`".

[4] http://www.w3.org/TR/xquery/

[5] http://www.w3.org/TR/xpath/

3.2 Navigation in Trees

A variety of programs was developed to query syntactic trees ((Lai and Bird [4]; Lai and Bird [3]). Although the query languages are different, they are conceptually very similar. All languages contain operators or connectives for selecting mother, sibling, ancestor, and descendant nodes in a tree relative to a given node.

One common ground between syntactic trees and XML is tree-based structure. XPath, the XML navigation language incorporated in XQuery, has extensive support for selecting XML elements that are siblings, ancestors, or descendants relative to some given XML element. Table 1 shows how some of the operators of Tgrep2 (Rohde [10]) can be expressed using XPath and XQuery.

Even more functionality can be obtained by using the possibility in XQuery to add user defined functions. For instance, to select only elements that are a *leftmost descendant of* a given element, we can add the function `left-desc` to the module `pdtb`:

```
declare function pdtb:left-desc($desc, $top)
{ if ($top = $desc) then true()
  else if ($top/* ) then  pdtb:left-desc($desc, $top/*[1])
  else false()
} ;
```

Note that `left-desc` is a boolean function that checks whether `$desc` is a leftmost descendant of `$top`. The function is recursive in that it returns `true` if `$desc = $top` and else calls `left-desc` with the leftmost daughter of `$top` as second argument. If `$top` hs no daughters, the function returns `false`. A slight variant of this function returns the *set of leftmost daughters* of a given node.

In the PDTB-XML Query API, most of the tree patterns in (Rohde [10]) are implemented in less than 100 lines of code. This offers users functionality equivalent to that of other tree query languages with minimal development effort. The efficiency problem will be addressed in Section 4.

3.3 A Case Study: Range Relations

A number of discourse researches deal with positional relations between two arguments. Lee et al. [5] investigate the occurrences of shared discourse structures with a special focus on subordinate clauses. The relevance of their study is based on the necessity of describing not a single tree discourse hierarchy, but broader structures with complex dependencies. The authors identify four non-tree-like dependencies: shared argument, properly contained argument, pure crossing, and partially overlapping arguments. Lee et al. [6] investigate to what extent discourse arguments introduced by a subordinating conjunction (*while, because, after, since, ...*) can be

Axis	XQuery example	Tgrep2	Meaning
child::	$b:=$a/child::*	A < B	A immediately dominates B
descendant::	$b:=$a/descendant::*	A<<B	A dominates B
following::	$b:=$a/following::*	A .. B	A precedes B
following-sibling::	$b:=$a/following-sibling::*	A $.. B	A is a sister of B and precedes B
parent::	$b:=$a/parent::*	A > B	A is immediately dominated by B
ancestor::	$b:=$a/ancestor::*	A>>B	A is dominated by B
preceding::	$b:=$a/preceding::	A ,, B	A follows B
preceding-sibling::	$b:=$a/preceding-sibling::*	A $,, B	A is a sister of B and follows B

Table 1: Tree-based navigation with XPath and XQuery

used as argument of a following discourse relation. That is, in examples such as (2), the discourse particle *however* connects the clause *All of the ... period* and the subordinated clause *while other ... results*. Lee et al. [6] found 349 instances of this configuration in the PDTB, about 4% of the relevant cases (in 12% of the cases only the matrix clause was selected as argument of a following sentence, and in 84% of the cases the complete preceding clause was selected).

(2) GM also had dismal results in the first 10 days of the month, <u>while</u> **other auto makers reported mixed results**. All of the Big Three suffered in the just-ended period, <u>however</u>. (wsj_1139)

Note that gathering the relevant data to study this phenomenon requires access to both discourse annotation and syntax. Here, we will demonstrate that we can do this using only the PDTB-XML and XQuery.

The XQuery script we used is given in Figure 3. It searches a corpus file for relations `$rel` which contain a discourse connective that is one of the frequent subordinating conjunctions in the corpus, as suggested by Lee et al. [6]. To find such relations, we use a regular expression (i.e. the `match` function) that searches the text of connectives for *although, however, after, etc.* Next, the variable `$shared` is a following discourse relation, which must meet the requirement that its first argument (`Arg1`) must be shared with the second argument (`Arg2`) of the discourse relation introduced by the subordinating conjunction. A complication of the PDTB annotation is that `$shared/Arg1` is always the concatenation of `$rel/ConnHead` (i.e. the conjunction word) and `$rel/Arg2`. Thus,

```
for $c in collection($dir)/corpus
for $rel in $c/Relations/*/Relation[ConnHead/RawText[
     matches(.,"(although|however|after|as|....)","i")]]
let $shared := $c/Relations/*/Relation[
                 pdtb:gorn2tree(Arg1/TreeRef) =
                      pdtb:gorn2tree($rel/Arg2/TreeRef)/.. ]
where $shared
return
  <shared> <first>{$rel}</first>
           <second>{$shared}</second>
  </shared>
```

Figure 3: XQuery script to find subordinate clauses linked to a discourse relation introduced by a following sentence (following Lee et al. [6]).

we cannot simply check for identity of the text of the two arguments. Instead, we check whether the syntactic tree that corresponds to `$shared/Arg1` is the mother of the tree that corresponds to `$rel/Arg2`. Syntactic trees are found by the corpus-specific function `gorn2tree`[6] which uses the `id/idref` mechanism linking discourse relations to PTB annotation. The XPath expression `/..` locates the mother of an XML element. The `where` statement checks whether a discourse relation introducing a shared argument indeed exists, and the `return` statement returns the results.[7]

We can do even better, however. Note that Lee et al. [6] restrict their search to the *"12 most common subordinating conjunctions in the* PDTB." The practical reason for this restriction is that it allows finding the relevant cases by string matching over the text of connectives. There seems to be no principled reason, however, for this restriction. In the PDTB-XML we can also require that `$rel/ConnHead` must introduce a subordinate clause. Thus, instead of using a regular expression, we can select the relevant `$rel` relations as follows:

```
$Arg2Tree[@cat="S" and starts-with(../@cat, "SBAR")]]
```

Here, we use `$Arg2Tree` as shorthand for the `$rel/Arg2` tree. If `$Arg2Tree` is of category S and is dominated by an SBAR (or one of the subcategories of SBAR used in the PTB), we assume `Arg2` is a subordinate clause.[8] Now, we also find cases where the subordinating conjunction is a less frequent, such as *so that* in the example below.

[6] Tree nodes are numbered using the method of Gorn [2].

[7] The actual script uses a more detailed `return` statement, which normalizes the results and returns only relevant parts of the two discourse relations.

[8] We also defined a case where the tree corresponding to `$rel/Arg2` is of category S-NOM and dominated by a category PP-TMP (to cover the *after/before/since V-ing* cases.

(3) Computers have increasingly connected securities markets world-wide, so that **a buying or selling wave in one market is often passed around the globe**. So investors everywhere nervously eyed yesterday's opening in Tokyo, where the Nikkei average of 225 blue-chip stocks got off to a rocky start (wsj_2276)

The case-study in this section has concentrated on finding data that meets certain syntactic requirements (i.e. subordinate clauses), and that is used in a specific way in the discourse relations. Lee et al. [5] study cases where a discourse argument is shared, properly contained, crossing or overlapping with another discourse argument. Such studies can be carried out using the PDTB-XML and XQuery as well. All arguments in the PDTB-XML are linked to one or more syntactic constituents. All syntactic constituents have a yield which consists of words that have an `@id` attribute reflecting their position in the sentence. Given a set of pointers to syntactic constituents, we can easily obtain its *span* by collecting the `@id` in the yield of these constituents and selecting the smallest and largest member (using the sort function of XQuery and numerical comparisons). Given the *range* of a discourse argument, it is straightforward to define notions such as containment or overlap.

4 Performance Test

The PDTB-XML consists of files 2159 files (376MB in total). Running an XQuery script withn an XQuery processor such as Saxon[9] requires a scan of all the files and a tree-traversal of each file. For large amounts of data, this can lead to substantial memory consumption and long processing times. Indexing the PDTB-XML as a native XML database can improve performance. Here we chose to experiment with two open source XML database systems, eXist[10] (Meier [8]) and Oracle Berkeley DB XML.[11] A database system such as eXist brings two benefits to querying. Firstly, eXist has implemented an index-driven XQuery mechanism, where structural indices are used to identify relationships between nodes and thus in order to compute path expressions, nodes on the disk do not even need to be loaded into memory. This guarantees a high speed performance. Secondly, this index-based approach causes the axis (an axis defines a node-set relative to the current node, such as *parent*, *child*, etc.) to have only a minimal effect on performance. For instance, the XPath expression A//D is supposed to be even faster than A/B/C/D.

[9] `http://saxon.sourceforge.net/`

[10] `http://www.exist-db.org`

[11] `http://www.oracle.com/database/berkeley-db/xml/index.html`

The testing scenario is intended to investigate whether an XML database can facilitate a user's everyday usage. An open source XQuery processor, Saxon-HE 9.2, is compared against eXist (v1.4) and Berkeley DB XML (BDB in short, v2.5.13). The hardware setting is Intel Xeon X5355 @2.66GHz with 16G RAM. All of the systems execute exactly the same XQuery and the time is recorded. All output is diverted to a null device (/dev/null) to avoid a slow-down caused by flushing standard output.

We have selected seven query tasks covering all parts of the PDTB-XML (see Table 2). Tasks 1-3 are from Lai and Bird [4], and focus on the syntactic part of the PDTB-XML. Task 4 extracts only discourse relations. Task 5 first queries syntax and then queries corresponding discourse elements, while task 6 works the other way around. Task 7 is a complex example inspired by the case study in section 3. From the results in Table 2 the following observations can be drawn:

1. Indexing can greatly reduce query time, especially in simple tasks (such as task 1-4).

2. In complex tasks, computing references on-the-fly determines the total query time. Saxon outperforms eXist in task 6 and 7 and BDB in task 5 and 6.

3. BDB is not good at resolving coreferences, thus it was slow in task 5 and 6. The reason for this is unknown.

Our experiments lead to mixed results. For most simple queries, results can be retrieved within two minutes in a deployed database system. But as the complexity goes up, the execution time also increases. This increase is mostly due to the fact that relations between syntax and discourse must be computed during retrieval. Thus the advantages of indexing do not apply. Efficiency of querying complex and interlinked XML is still a problem that needs to be addressed in future work.

5 Conclusion

In this paper, we have investigated the merits of XQuery for processing corpora with complex linguistic annotation. We have concentrated on the PDTB, a corpus that combines annotation of discourse relations with links to the corresponding syntactic annotation (from the PTB). The PDTB-XML combines the two annotations in a single XML format, thus offering a uniform representation, with possibilities for future extensions. The general purpose XML query language XQuery offers the functionality to query the syntactic part of the PDTB-XML, and can be used to formulate queries that need to address both the syntactic and the discourse annotation. Finally, we have compared two native XML databases, eXist and Oracle Berkeley

	Query	Saxon	eXist	BDB
1	sentences that include the word *saw*	5m51s	19s	15s
2	NPs whose rightmost child is a noun	6m23s	55s	1m20s
3	VPs that contain a verb immediately followed by an NP immediately followed by a PP	6m43s	1m18s	1m20s
4	all *Explicit* relations whose connective type is *because*	2m9s	0.5s	0.1s
5	return connectives and corresponding POS tags of all *Explicit* relations	7m17s	2m27s	30m30s
6	all words with POS="CC" that function as connectives	7m3s	15m21s	21m33s
7	all arguments that are in a *range-contains* relation to another argument	32m26s	did not finish in 1h	7m13s

Table 2: Execution time of common queries

DB XML against a stand-alone XQuery processor, Saxon. The evaluation shows that by indexing the elements and attributes in the XML database the query time can be greatly reduced for simple tasks. Computing dependencies between XML elements on the fly, as is typically required for queries that address both syntax and discourse annotation, means that the advantages of indexing are lost to a large extent, and processing times go up sharply. Efficiency of such queries needs to be addressed in future research.

References

[1] Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. The tiger treebank. In *In Proceedings of the Workshop on Treebanks and Linguistic Theories*, pages 24–41, 2002.

[2] S. Gorn. Explicit definitions and linguistic dominoes. In J. Hart & S. Takasu, editor, *Systems and Computer Science*, pages 77–115. 1967.

[3] C. Lai and S. Bird. Querying linguistic trees. *Journal of Logic, Language and Information*, 19(1):53–73, 2010.

[4] Catherine Lai and Steven Bird. Querying and updating treebanks: A critical survey and requirements analysis. In *In Proceedings of the Australasian Language Technology Workshop*, pages 139–146, 2004.

[5] Alan Lee, Rashmi Prasad, Aravind Joshi, Nikhil Dinesh, and Bonnie Webber. Complexity of Dependencies in Discourse: Are Dependencies in Discourse More Complex than in Syntax? In *the 5th International Workshop on Treebanks and Linguistic Theories*, Prague, Czech Republic, December 2006.

[6] Alan Lee, Rashmi Prasad, Aravind Joshi, and Bonnie Webber. Departures from Tree Structures in Discourse: Shared Arguments in the Penn Discourse Treebank. In *Proceedings of the Constraints in Discourse III Workshop*, 2008.

[7] Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2):313–330, 1994.

[8] Wolfgang Meier. eXist: An Open Source Native XML Database. In Akmal Chaudhri, Mario Jeckle, Erhard Rahm, and Rainer Unland, editors, *Web, Web-Services, and Database Systems*, volume 2593 of *Lecture Notes in Computer Science*, pages 169–183. Springer Berlin / Heidelberg, 2003.

[9] Rashmi Prasad, Nikhil Dinesh, Alan Lee, Eleni Miltsakaki, Livio Robaldo, Aravind Joshi, and Bonnie Webber. The Penn Discourse TreeBank 2.0. In *Proceedings of the Sixth International Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, May 2008.

[10] D.L.T. Rohde. Tgrep2 user manual. *URL: http://tedlab.mit.edu/˜dr/Tgrep2*, 2004.

[11] Priscilla Walmsley. *XQuery*. O'Reilly, 2007.

[12] Xuchen Yao, Irina Borisova, and Mehwish Alam. PDTB XML: the XMLization of the Penn Discourse TreeBank 2.0. In *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC 10)*, Malta, 2010.