Extending XPath to Support Linguistic Queries

Steven Bird*,†, Yi Chen*, Susan B. Davidson*, Haejoong Lee*, and Yifeng Zheng*
*University of Pennsylvania, †University of Melbourne
{sb,yicn,susan,haejoong,yifeng}@cis.upenn.edu

ABSTRACT
Linguistic research and language technology development employ large data repositories of ordered trees, known as “treebanks.” We define a path language for linguistic trees represented in XML called LPath, based on XPath, and provide a new labeling scheme for LPath query evaluation. We report a strategy for evaluating expressions of the language against treebank data. The language contains three expressive features which are important for linguistic query, namely immediate precedence, subtree scoping, and edge alignment. We motivate and illustrate these features with a variety of linguistic queries. This work provides a scalable and reusable model for linguistic tree queries, and relates it to well-understood semistructured and relational languages.

1. INTRODUCTION
Large repositories of text and speech data are routinely collected, curated, annotated, and analyzed as part of the task of developing and evaluating new language technologies. These technologies include information extraction, question answering, machine translation, and so forth. Linguistic data repositories may contain $10^6 - 10^8$ words, along with annotations at the levels of phonetics, prosody, orthography, syntax, dialog, and gesture. For instance, Penn Treebank [15] consists of over 1,000,000 words of manually parsed text from the Wall Street Journal. The Switchboard corpus contains 2,400 recorded and transcribed telephone conversations, some with phonetic, prosodic, syntactic and disfluency annotations [10]. In the general case, we begin with time-series data such as a text or a recording which represents a linguistic artifact; this “primary data” is usually considered to be immutable. Then we associate structured annotations with extents of this data, annotations which are usually related to some level of linguistic analysis or to a particular application domain. The relationship between linguistic data and linguistic annotations is shown schematically in Figure 1.

Annotations are often hierarchically organized. For instance, a segment of a sound file might be annotated with a phonetic transcription. The transcription could then be annotated orthographically, and then the orthographic representation could be annotated linguistically (e.g. as a named entity of a particular type). This hierarchical structure, and its connection to stream data, can be represented in XML as shown below:

```xml
<phoneme id="ph20" span="2045ms-2092ms">f</phoneme>
<phoneme id="ph21" span="2092ms-2132ms">l</phoneme>
<phoneme id="ph22" span="2132ms-2204ms">I</phoneme>
<phoneme id="ph23" span="2204ms-2230ms">f</phoneme>

...<word id="wd03" span="ph20-ph31">Philadelphia</word>
<name id="n1" span="wd03-wd04" type="location"/>
```

Despite the large amount of attention that has been paid to data collection and associated computational tools for linguistic annotation, there has been relatively little consideration of efficiency and scalability. Specialized languages for querying linguistic data repositories have been proposed [3, 16, 17], but they are ad hoc and have two critical shortcomings. First, they are tied to specific data file formats and are difficult to generalize and reuse. Second, expressiveness is emphasized to the detriment of efficiency, and in most cases little is known about the computational properties of the language, how data can be indexed, and how queries can be optimized.1 As the data repositories grow and find uses other than those for which they were originally created, reusability and scalability become critical considerations. In general, the design of a query language must balance expressiveness and efficiency. First, it should express, as naturally as possible, the queries that the user community needs. Second, it should be optimizable, supporting query rewriting, execution planning and index selection.

The goal of this work is to develop a query language for linguistic data which is sufficiently expressive and which can be implemented efficiently by exploiting the mature technology of relational databases. We propose a language LPath, extending the XPath 1.0 syntax,2 supporting immediate precedence, subtree scoping, and edge alignment. We also propose a labeling scheme which is effective for both vertical and horizontal navigations, and describe a query translation algorithm from LPath to SQL.

This paper is organized as follows. In §2 we introduce our working example, then present the LPath language in §3. In §4 we give the

---

1We refer the reader to [6, 11] for a discussion of linguistic tree query language requirements.

2We focus on the discussion of XPath 1.0 without functions in this paper. As with XPath, LPath can have a function library.
transformation to the relational model, before closing with a summary of the work and a discussion of future work in §5.

2. QUERYING LINGUISTIC TREES

Figure 2 shows the representation of a parsed sentence of English. Here the immutable, primary data is the sentence comprising the fringe of the tree, while the linguistic analysis is an ordered tree built over the primary data. Non-terminal nodes are annotations of sequences of one or more terminal nodes (i.e. the words) or of other non-terminals. For instance, the node NP9 (noun phrase) is an annotation of the Det9, Adj9 and N10 annotations. Linguistic trees are navigated vertically via the hierarchical relationships, and also horizontally via a precedence relation, which we turn to next.

The terminals of a linguistic tree are linearly ordered; this ordering induces an ordering on the non-terminals. For instance V5 immediately precedes NP0, NP3 and Det6. (A formal definition will be given later, in Algorithm 1.) Traditionally, this precedence relation has been understood with respect to the context-free grammar which licenses trees such as the one in Figure 2. We can apply grammar productions in reverse to a sentence in order to get sequences, or so-called “proper analyses” [7], as illustrated below:

<table>
<thead>
<tr>
<th>CFG Productions</th>
<th>Some Proper Analyses</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP (NP)</td>
<td>I saw the old N with NP today</td>
</tr>
<tr>
<td>VP → V NP (NP)</td>
<td>I V the Adj man PP today</td>
</tr>
<tr>
<td>NP → NP PP</td>
<td>NP saw NP with a telescope NP</td>
</tr>
<tr>
<td>NP → Det Adj N</td>
<td>I VP NP</td>
</tr>
<tr>
<td>PP → Prep NP</td>
<td>I saw NP today</td>
</tr>
</tbody>
</table>

Large-scale empirical linguistics involves searching and collating tree data. We have compiled a representative sample of linguistic tree queries below, and give their result for a query against the tree in Figure 2.

\[ Q_1 \text{ Find noun phrases that immediately follow a verb: } \{ NP_0, NP_3 \} \text{ (both nodes immediately follow } V_3) \]
\[ Q_2 \text{ Find nouns that follow a verb which is a child of a verb phrase: } \{ N_{10}, N_{15}, N_{17} \} \text{ (all three follow } V_5) \]
\[ Q_3 \text{ Within a verb phrase, find nouns that follow a verb which is } \]
\[ \text{ a child of a verb phrase: } \{ N_{10}, N_{15} \} \text{ (within VP}_4, N_{10} \text{ and } N_{15} \text{ follow } V_5) \]
\[ Q_4 \text{ Find noun phrases which do not have an adjective descendant: } \{ NP_3, NP_{13}, NP_{16} \} \]
\[ Q_5 \text{ Find noun phrases which are the rightmost child of a verb phrase: } \{ NP_6 \} \]
\[ Q_6 \text{ Find noun phrases which are the rightmost descendant of a verb phrase: } \{ NP_{10}, NP_{15} \} \text{ (both are descendants of VP}_4, \text{ and no other descendants of VP}_4 \text{ follow them) } \]
\[ Q_7 \text{ Find all verb phrases that are comprised of a verb, a noun phrase, and a prepositional phrase: } \{ VP_3 \} \text{ (VP}_4 \text{ is comprised of } V_5, NP_7 \text{ and PP}_{11} \)

A natural candidate for representing and querying linguistic data is XML. Its associated query language, XPath, has been much studied within the database community in terms of expressiveness [14, 9]. Various evaluation and optimization techniques have been proposed, leveraging relational databases [2, 5, 12].

It is clear that XPath can express some of the sample queries given earlier: \[ Q_2 \text{ can be expressed as } //VP/V/following::N, \]
\[ Q_4 \text{ can be expressed as } //NP[not(boolean(/Adj))] \text{ and } Q_5 \]
\[ \text{ as } //VP/_[last()]\\text{[self::NP]} ^ {3} \text{. However, there are several problems with XPath with respect to the other sample queries. For the remainder of this section we show how linguistic tree queries cannot be expressed naturally or, in some cases, expressed at all, using XPath.} \]

Immediate Precedence. It is possible to define an immediate precedence relation as a new XPath navigation axis. We call this axis immediate-following and show how it is computed in Algorithm 1. \(^3\) For a given node \( n \), the node set \( s \) reachable by the immediate-following axis is comprised of the nodes on the left-most path in the subtree rooted at \( m \), where \( m \) is the immediate-following sibling of the nearest ancestor-or-self of \( n \). The XPath following axis is the transitive closure of the LPath immediate-following axis. From the definition of immediate following, we can see that it cannot be expressed in XPath.

XPath supports vertical navigation through parent/child, ancestor/descendant relationships, however the horizontal axis only supports “following/preceding sibling” and “following/preceding”. Neither of these is adequate for expressing the “immediately follows” of query \( Q_1 \).

Subtree Scoping. In many cases, linguistic tree queries must circumscribe tree navigations so that they remain inside the subtree

\(^3\)Instead of using \( s \) to denote a wildcard to match any tag name as defined in XPath specification, we use \( _\_ \) as wildcard, since \( s \) is used to denote transitive closure in this paper.

\(^3\)We focus on the extension of LPath to XPath without functions here. Lines 10-15 can be simplified using XPath’s position function as follows: \( m \leftarrow m[\text{position()} \leftarrow \text{first}(1)] \).
rooted at a specified node. Query $Q_2$ exemplifies this requirement with respect to $Q_3$ which is not constrained in this way. Observe that $N_{17}$ is part of the solution to $Q_2$, but not of the solution for $Q_3$ since $N_{17}$ is outside the scope of $VP_4$. This subtree scoping cannot be expressed in XPath.

**Edge Alignment.** Queries $Q_3$ and $Q_6$ illustrate edge alignment. The alignment of a child node with the left or right edge of its parent can be expressed using XPath, as we have already seen above for $Q_2$. However, XPath cannot describe more deeply nested alignments, as required for $Q_6$. A putative XPath equivalent is: 

$$\text{//VP//}_{{\text{NP}}}$$. Consider the XML tree in Figure 2: although $N_{17}$ is a following node for $V_5$ in the whole tree, it is not part of the result for $Q_3$.

**Edge Alignment.** Linguistic queries need to refer to nodes at the left or right edge of the subtree rooted at a specified node (e.g. $Q_6$). To support queries involving edge alignment, we introduce syntactic sugar $\text{\texttt{\textbackslash}}$ to force left-alignment, and $\text{\texttt{\textbackslash}}$ to force right-alignment. These operators are defined as follows: $A = A[not<=_]$; $AS = A[not->_]$. Accordingly, $Q_6$ can be expressed as: $\text{//VP//}_{\text{NP}}$.

### 3.1 LPath Grammar

The grammar of the query language is presented in Figure 3. A path expression $P$ is an absolute path optionally followed by a scoped path. The absolute path expressions $AP$ are composed of steps $S$. A step consists of an axis $A$, a tag test $T$, and an optional restriction (or predicate) $R$. The axis $A$ represents the navigations we can perform between nodes. The tag test $T$ can be a string equality test, or a wildcard $\_\_\_\_$ which matches any tag. $R$ is the restrictions introduced by $[]$ to filter a node set. The restriction is a logical expression composed of one or more sub-expressions, connected by “and”, “or” and “not”.

Unlike the child and descendant axes which retrieve qualifying nodes within the subtrees of context nodes, several axes – parent, ancestor, immediate following/preceding and following/preceding – take node navigation out of the current subtree to the global data tree. Thus in query $Q_3$, if we specify the relationship between $Adj$ and $N$ using the following axis $=>$, all $N$ nodes that appear after $Adj$ nodes qualify.
3.2 LPath Examples

Now that we have discussed the syntax of the proposed language, let us consider how it can be used to represent the sample linguistic queries from §1.

Q1 Find noun phrases that are immediately following a verb.
\[ /V -> NP \]

Q2 Find nouns that follow a verb which is a child of a verb phrase.
\[ /VP /V -> NP \]

Q3 Within a verb phrase, find nouns that follow a verb which is a child of a verb phrase.
\[ /VP /VP /V -> NP \]

Q4 Find noun phrases which do not have an adjective descendant.
\[ //NP /not (/Adj) ]\]

Q5 Find noun phrases which are the rightmost child of a verb phrase.
\[ //VP /NP \]

Q6 Find noun phrases which are rightmost descendant of a verb phrase.
\[ //VP /NP \]

Q7 Find verb phrases comprised of a verb, a noun phrase, and a prepositional phrase.
\[ //VP /NP /PP ]\]


4. EVALUATION

As we discussed in section 1, the two key features of a good language are expressibility and efficiency. We have justified the expressiveness of the proposed language according the linguistic query requirements in section 3, now we will justify its efficiency.

We begin by introducing a labeling scheme which efficiently represents both vertical and horizontal axis navigation. Based on the labeling scheme, we then transform XML data into relations, and translate XPath queries into SQL which can be efficiently evaluated over those relations.

**Interval Labeling.** A simple interval-based labeling scheme supports evaluation of all LPath axes, i.e. we can detect all relationships between tree nodes by inspecting these labels. This labeling can be constructed in a single pass over the XML representation of a tree, or equivalently using a depth-first traversal of a tree. This labeling is defined for a single tree, but it can easily be extended to multiple trees by introducing tree identifiers.

**Definition 4.1:** The labeling scheme assigns each node a tuple \(<left, right, depth, id, pid, name, value>\), shortened as \(<l, r, d, id, pid, name, value>\), in the following fashion:

1. Let \(n\) be the leftmost leaf node. Then assign \(n.l = 1\).
2. Let \(n\) be a leaf node. Then assign \(n.r = n.l + 1\).
3. Let \(m\) and \(n\) be consecutive leaf nodes where \(m\) is on the left. Then assign \(m.r = n.l\).
4. Let \(n\) be a non-terminal node which dominates leaf nodes \(a_1, \ldots, a_k\). Then assign \(n.l = a_1.l\) and \(n.r = a_k.r\).
5. For each node \(n\), let \(n.d\) be the distance of \(n\) from the root (i.e. the depth of \(n\)), where the root has a depth of 0.
6. For each node \(n\), assign a nonzero \(id\) as its unique identifier \((= f(l, r, d)\) where \(f\) is a Skolem function). 7. For each node \(n\), assign \(n.pid\) to be \(n\)'s parent node identifier; if \(n\) is the root node, assign \(n.pid = 0\).
8. For each node \(n\) it's \(name\) is either the tag name or attribute name; \(value\) stores attribute values or nullable text value.

**Data Storage.** The data loader handles events generated by a SAX parser reading the XML document of linguistic data. For each element node, it generates a tuple \(<left, right, depth, id, pid, name, value>\). Part of the relation generated for the sample annotation tree in Figure 2 is shown in Figure 4.

**Query Translation.** The query translation algorithm converts an LPath query into SQL statements based on the labeling scheme. First we convert each axis into a join over tuples. We detect axis relationships between any pair of nodes simply by checking their labels. This check can be implemented as a join \(T \bowtie C\), where \(C\) is the required constraint shown in Table 2.

\[ T \]

\[ \begin{array}{cccccc}
\text{left} & \text{right} & \text{depth} & \text{id} & \text{pid} & \text{name} & \text{value} \\
1 & 0 & 1 & 0 & \text{root} & \\
1 & 10 & 1 & 2 & 1 & S \\
1 & 2 & 2 & 3 & 2 & NP \\
1 & 2 & 2 & 3 & 3 & \@lex I \\
2 & 2 & 2 & 4 & 2 & VP \\
2 & 3 & 3 & 5 & 4 & V \\
2 & 3 & 3 & 5 & 5 & \@lex saw \\
3 & 9 & 3 & 6 & 4 & NP \\
3 & 6 & 4 & 7 & 6 & NP \\
3 & 4 & 5 & 8 & 7 & Det \\
3 & 4 & 5 & 8 & 7 & \@lex the \\
\end{array} \]

Figure 3: The grammar of LPath

Figure 4: Relational representation

\[ P = \text{Path expression}; \ AP = \text{Absolute Path expression}; \ A = \text{Step}; \ T = \text{Tag}; \ R = \text{Restriction (predication)} \]
Table 2: Axes and their Corresponding Join Constraints

<table>
<thead>
<tr>
<th>Vertical Navigation</th>
<th>Horizontal Navigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>child(m, n)</td>
<td>immediate-following(m, n)</td>
</tr>
<tr>
<td>descendant(m, n)</td>
<td>m.\textit{id} = m.\textit{pid}</td>
</tr>
<tr>
<td>parent(m, n)</td>
<td>m.\textit{id} = n.\textit{pid}</td>
</tr>
<tr>
<td>ancestor(m, n)</td>
<td>m.\textit{I} \leq n.\textit{I}, m.\textit{r} \leq n.\textit{r}, m.\textit{d} &gt; n.\textit{d}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sibling Navigation</th>
<th>Immediate-Preceding(m, n)</th>
</tr>
</thead>
<tbody>
<tr>
<td>immediate-following-sibling(m, n)</td>
<td>m.\textit{I} = m.\textit{r}, m.\textit{pid} = n.\textit{pid}</td>
</tr>
<tr>
<td>following-sibling(m, n)</td>
<td>m.\textit{I} \geq n.\textit{r}, m.\textit{pid} = n.\textit{pid}</td>
</tr>
<tr>
<td>immediate-preceding-sibling(m, n)</td>
<td>m.\textit{r} = n.\textit{I}</td>
</tr>
<tr>
<td>preceding-sibling(m, n)</td>
<td>m.\textit{r} \leq n.\textit{I}</td>
</tr>
</tbody>
</table>

The only remaining innovation of the language is the subtree scoping constraint, expressed using $\{\}$. For this we employ a stack; when we encounter a node $m$ followed by $\}$, we save $m$’s label on a stack, and we require that any node $n$ falling inside this scope be bounded by $m$, i.e., $m.\textit{I} \leq n.\textit{I}, n.\textit{r} \leq m.\textit{r}$, and $n.\textit{d} > m.\textit{d}$. Once the corresponding $\}$ is met, we pop the current environment from the stack.

Example 4.1: $//\text{NP}([//\text{Adj}]->\text{N})$ ($Q_3$) is translated to the following SQL query. The scope constraint requires that N node must be a descendant NP, and this is implemented by the conditions in italic.

```
SELECT l1.l3, r3.r3, d3.d3, l2.l2, r2.r2, d2.d2, l1.l1, r1.r1, d1.d1 FROM T,
(SELECT l2.l2, r2.r2, d2.d2, l1.l1, r1.r1, d1.d1 FROM T WHERE T.name = ‘NA’) T1
WHERE T.name = ‘NA’ AND T.11 >= l1 AND T.r <= r1 AND T.d > d1)
```

For translating from LPath with predicates to SQL, we use the techniques in [8]. When a $\{\}$ predicate is met, we add the keyword EXISTS to the WHERE clause. Logical operator AND (resp. OR) in LPath predicates are directly mapped to keyword AND (resp. OR) in SQL. The key feature of the LPath-to-SQL mapping is that we also initialize the processing environment for expressions in the predicates to be the current environment. Another difference compared to [8] is that, since “not” can appear in the predicates in an LPath expression, we translate it using NOT EXISTS in the SQL where clause.

As an optimization, rather than process $\wedge$ and $\vee$ directly according to their definitions, we can evaluate these constraints very efficiently as follows. Let $T'$ the relation at the top of the environment stack. $\wedge$ means that $T.\text{name} = A$ and $T.\text{Left} = T'.\text{Left}$ and $T.\text{Depth} > T'.\text{Depth}$. Similarly, $\vee$ means that $T.\text{name} = A$ and $T.\text{Right} = T'.\text{Right}$ and $T.\text{Depth} > T'.\text{Depth}$.

5. CONCLUSIONS AND FUTURE WORK

We have addressed the problem of defining an expressive and efficient language for linguistic tree queries. Our language, LPath, extends the XPath syntax in three respects: immediate precedence, subtree scoping, and edge alignment. We review each of these in turn. First, several new axes are proposed for immediate precedence: immediate-following ($\rightarrow$), immediate-following-sibling ($\rightarrow$), immediate-preceding ($\leftarrow$), and immediate-preceding-sibling ($\leftarrow$). These “horizontal” axes are not supported by XPath, even though their closures are. Once added, there is a natural symmetry between horizontal and the vertical axes (cf. Table 1).

Second, a new kind of bracketing is proposed for subtree scoping, using $\{\}$ for example, $//\text{VP}([//\text{N}\rightarrow\text{N}])$ will only return those nodes labeled N which have a VP ancestor. Finally, new edge-alignment operators $\wedge$ and $\vee$ are introduced. When used in conjunction with $\{\}$, these force the specified node to be aligned to the left or right edge of the subtree. For example, $//\text{VP}([//\text{NP}\{\}]$ matches all noun phrases which are the rightmost descendant of a verb phrase.

For efficient evaluation of LPath queries, we have proposed a labeling scheme which supports both horizontal and vertical navigations. Additionally, this labeling scheme serves as relational storage for linguistic tree data. We have implemented a translator which converts LPath expressions to SQL queries.$^6$

We believe this work has implications beyond linguistics. XPath provides the ancestor and descendant axes, which are transitive closures of the parent and child axes. However, from the standpoint of this work, XPath is incomplete in that it defines transitive relations following, preceding, following-sibling, preceding-sibling, without defining their primitives. Furthermore, the evaluation of LPath queries employs a labeling scheme which may prove useful for general XPath query processing. The subtree scoping operator may be useful for general trees, not just ordered trees.

Additionally, the navigation requirements of linguistic trees presents an interesting challenge to work on semistructured data and querying.

The fringe of a linguistic tree is a collection of words – a sentence – constituting the immutable primary data upon which different linguistic theories construct their trees. These words form a total ordering, which in turn induces a partial ordering on the node set. Further research is required to apply semistructured data models and query languages to this domain.

Another area for further investigation is the expressiveness of the language. We would like to support simple kinds of path closures (e.g. ($\rightarrow\text{NP}^*)$); as well as querying “overlapping trees” arising from multiple linguistic annotations over the same text. Existing linguistic query languages need to be examined in depth, particularly for their use of variables and quantification, in case there are new expressive requirements. Standard benchmarks need to be established to provide objective measures of the scalability of different approaches to querying linguistic trees. Finally, we plan to extend LPath with update operations, permitting local rearrangements of linguistic trees, and facilitating the curation of treebanks.

6. ACKNOWLEDGMENTS

We would like to thank Val Tannen, Peter Buneman, and James Bailey for their valuable feedback on the work reported here. This research is sponsored by the National Science Foundation under Grant No. 0317826 Querying Linguistic Databases.

$^6$This implementation is available from http://www.ldc.upenn.edu/Projects/QLDB/
7. REFERENCES


Minerva Access is the Institutional Repository of The University of Melbourne

Author/s:
BIRD, STEVEN; Chen, Yi; Davidson, Susan; Lee, Haejoong; Zheng, Yifeng

Title:
Extending XPath to support linguistic queries

Date:
2005

Citation:

Publication Status:
Published

Persistent Link:
http://hdl.handle.net/11343/34055

File Description:
Extending XPath to support linguistic queries

Terms and Conditions:
Terms and Conditions: Copyright in works deposited in Minerva Access is retained by the copyright owner. The work may not be altered without permission from the copyright owner. Readers may only download, print and save electronic copies of whole works for their own personal non-commercial use. Any use that exceeds these limits requires permission from the copyright owner. Attribution is essential when quoting or paraphrasing from these works.