

# Designing and Evaluating an XPath Dialect for Linguistic Queries

Steven Bird  
University of Pennsylvania &  
University of Melbourne  
sb@ldc.upenn.edu

Yi Chen  
Arizona State University  
yi@asu.edu

Susan B. Davidson  
University of Pennsylvania  
susan@cis.upenn.edu

Haejoong Lee  
University of Pennsylvania  
haejoong@cis.upenn.edu

Yifeng Zheng  
University of Pennsylvania  
yifeng@cis.upenn.edu

## Abstract

*Linguistic research and natural language processing employ large repositories of ordered trees. XML, a standard ordered tree model, and XPath, its associated language, are natural choices for linguistic data and queries. However, several important expressive features required for linguistic queries are missing or hard to express in XPath. In this paper, we motivate and illustrate these features with a variety of linguistic queries. Then we propose extensions to XPath to support linguistic queries, and design an efficient query engine based on a novel labeling scheme. Experiments demonstrate that our language is not only sufficiently expressive for linguistic trees but also efficient for practical usage.*

## 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 databases may contain up to a billion words, along with annotations at the levels of phonetics, prosody, orthography, syntax, dialog, and gesture. Of particular interest here are so-called treebanks. For instance, Penn Treebank [22] contains a million words of parsed text.

Unfortunately, different corpora use different data formats and rely on specialized search tools to extract data of interest. This lack of standards has become a critical problem for data sharing, on-line retrieval and distributed collaboration. Furthermore, as observed in [17], the relationship between these linguistic tools and existing database

query languages has not been well studied, making it difficult to apply standard database indexing and query optimization techniques. As data size grows and the analysis tasks become more complex, scalability has become a critical factor.

Linguistic data and its annotations is typically modeled as an ordered hierarchical structure. For example, an English sentence with its grammatical analysis annotation (syntax tree) is presented in Figure 1. Due to the reliance on an ordered tree model, a natural candidate for representing linguistic data is XML.

Despite increasing efforts to use XML for representing linguistic data, XML's associated standard query languages, XPath [10] and XQuery [3], are not widely used for querying the data. After discussions with linguists and annotators associated with the Treebank project, we found that this is due to three primary issues: expressibility, user friendliness and efficiency.

First, a language should naturally express the queries that the user community needs. Since linguistic data has both a sequential organization related to the primary data (for example, sentences) and a hierarchical organization related to the annotations, its query language must express tree navigations in both directions. XPath and XQuery support vertical navigations of a tree using the parent, ancestor, child and descendant axes, and certain horizontal navigations using the following and preceding sibling axes, and the following and preceding axes. However, other horizontal navigations which are important to linguistic queries, are lacking or can not be easily expressed in XPath.

To illustrate, consider the syntax tree in Figure 1. A common linguistic query for this tree would be: Find constituents which *immediately follow* a verb. The query asks for the constituent right after a node  $V_5$  in a syntactic analysis of the given sentence. For example, the sentence can be analyzed as "I  $V_5$   $NP_6$  today". Therefore  $NP_6$  is a node

that immediately follows node  $V_5$  according to this analysis. The sentence also can be analyzed as “I  $V_5$   $NP_7$   $PP_{11}$  today” which is a finer granularity of analysis since an NP is composed of an NP and a PP. Therefore node  $NP_7$  also immediately follows  $V_5$ . Similarly,  $Det_8$  also immediately follows  $V_5$ .

However, this type of tree navigation can not be easily expressed in XPath. Furthermore, it turns out that this type of horizontal navigation not only has practical applications in linguistic queries, but also has interesting theoretical consequences for tree models [21].

Second, user friendliness is an important consideration. During our discussions with linguists and annotators, we found that a path language without variable bindings is most convenient, thanks to its similarities with regular expressions which are used widely. Most of the additional features of XQuery, such as node construction, iteration, joins and type checking, are not usually required for linguistic tree queries.

Finally, a query language should be efficiently evaluated to be practically useful. XPath has been extensively studied in terms of expressivity [21], complexity [13], as well as optimization techniques [18, 11, 8], and it is widely used in various applications.

Therefore we are particularly interested in how to augment XPath to express linguistic queries and how to efficiently evaluate this more expressive query language.

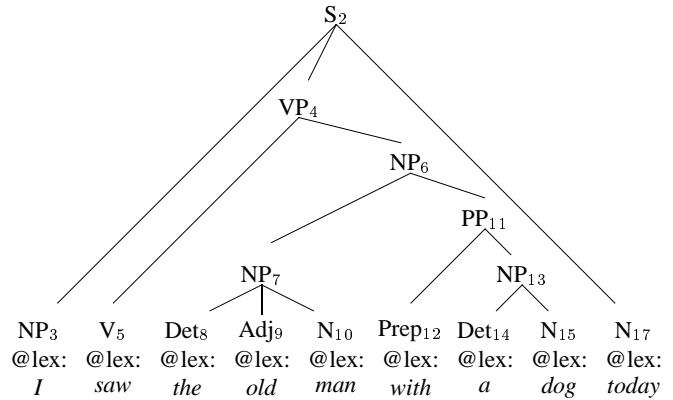
After studying the requirements of linguistic queries, we propose a linguistic query language *LPath*, which extends the XPath 1.0 syntax.<sup>1</sup> By adding certain horizontal navigation axes, we have both primitives and transitive closures for vertical and horizontal navigation, filling a gap in the XPath axis set. We also include subtree scoping and edge alignment which we will show are required by linguistic queries.

We then discuss how to efficiently evaluate LPath queries. Labeling schemes have proven to be a very effective technique for evaluating XPath queries [18, 8]. However, we have found that existing labeling schemes cannot support the new features in LPath. We propose a new labeling scheme which speeds up the existing as well as new axes in LPath. Based on this labeling scheme, we design and implement an efficient query engine to evaluate LPath queries.

Experiments demonstrate that LPath is not only sufficiently expressive for querying linguistic trees but also efficient for practical usage.

The contributions and organization of the paper are as follows. Section 2 describes a new application of semistructured data, linguistic treebanks. We analyze the data model

<sup>1</sup>We focus on the discussion of XPath 1.0 (abbreviated as XPath in the rest of the paper) without user-defined functions in this paper. As with XPath, LPath can have a function library.



*S: sentence; NP: noun phrase; VP: verb phrase; PP: prepositional phrase; Det: determiner; Adj: adjective; N: noun; Prep: preposition; V: verb. Nodes are assigned identifiers to facilitate the discussion.*

**Figure 1. A Syntax Tree T**

and query requirements and introduce a running example. Next, in Section 3 we propose an expressive and intuitive linguistic query language, LPath, by extending XPath. To evaluate LPath queries, a new labeling scheme which efficiently supports both horizontal and vertical tree navigations is introduced in Section 4. The LPath query evaluation system has been implemented and tested against several linguistic query engines as well as an XPath query engine. Experimental results, reported in Section 5, show that the proposed approach efficiently evaluates linguistic queries on various data and query sets. Furthermore, the additional expressiveness of LPath does not compromise its efficiency compared with XPath query evaluation. Finally, Section 6 discusses related work on linguistic query languages as well as XPath query evaluation. Section 7 concludes the paper and discusses future research directions. We also discuss its implications for XPath design and evaluation.

## 2 Data Model and Query Requirements

### 2.1 Linguistic Data

Linguistic data consists of linguistic artifacts (for example, texts or recordings), which are considered immutable, together with hierarchical annotations. A common data model for linguistic data is therefore an ordered labeled tree, in which the leaves or *terminals* are units of linguistic artifacts (e.g. utterances or words), and the annotations are the tree structure. Since the terminals of a linguistic tree are linearly ordered, an order is also induced on the non-terminals.

For example, Figure 1 shows the syntax tree of a sentence. Here the words in the sentence are represented by

| Query   | Result                       | LPath                           |
|---|------------------------------|---------------------------------|
| $Q_1$ Find a sentence containing the word <i>saw</i>  | $\{S_2\}$                    | <code>//S[//_[@lex=saw]]</code> |
| $Q_2$ Find noun phrases that is an immediate following sibling of a verb                            | $\{NP_6\}$                   | <code>//V=&gt;NP</code>         |
| $Q_3$ Find noun phrases that immediately follow a verb  | $\{NP_6, NP_7\}$             | <code>//V-&gt;NP</code>         |
| $Q_4$ Find nouns that follow a verb which in turn is a child of a verb phrase                       | $\{N_{10}, N_{15}, N_{17}\}$ | <code>//VP/V--&gt;N</code>      |
| $Q_5$ Within a verb phrase, find nouns that follow a verb which is a child of the given verb phrase | $\{N_{10}, N_{15}\}$         | <code>//VP{/V--&gt;N}</code>    |
| $Q_6$ Find noun phrases which are the rightmost child of a verb phrase                              | $\{NP_6\}$                   | <code>//VP{/NP\$}</code>        |
| $Q_7$ Find noun phrases which are the rightmost descendant of a verb phrase                         | $\{NP_6, NP_{13}\}$          | <code>//VP{/ /NP\$}</code>      |

**Figure 2. Example Linguistic Queries**

a sequence of terminals. The linguistic annotation is an ordered tree built over the terminals. Non-terminal nodes are annotations of sequences of terminal nodes or other non-terminals. For instance, the node  $NP_7$  is an annotation of annotations  $Det_8$ ,  $Adj_9$  and  $N_{10}$ , with the interpretation that “a determiner, an adjective and a noun together compose a noun phrase.”

## 2.2 Linguistic Queries

### 2.2.1 Tree Navigation

Large-scale empirical linguistics involves searching and collating tree data. Since linguistic data is two dimensional, it is frequently navigated in both the vertical and horizontal directions.

Vertical navigations on linguistic trees are the same as in XML tree navigation, e.g. parent, child, ancestor, and descendant. As an example, we may want to find all sentences containing the word *saw* as in query  $Q_1$  in Figure 2.

Horizontal navigations traverse the sequential organization of a linguistic tree. Some of them are supported by XPath axes such as following, preceding, following sibling and preceding sibling. For example, a linguist may want to find nouns that follow a verb which in turn is a child of a verb phrase, specified as  $Q_4$  in Figure 2.

Other forms of horizontal navigation used in linguistic queries, such as immediate following sibling, do not have a corresponding XPath axis but still can be represented by an XPath expression using the core function library. For example,  $Q_2$  in Figure 2, which finds noun phrases that are the immediate following sibling of a verb, can be expressed in XPath as `//V/following-sibling::_[position()=1][self::NP]`.<sup>2</sup> It first finds all the following-siblings of a  $V$  node, then uses the position function to filter out the first one; finally it checks the tag. Arguably, this is not a natural way of expressing the query.

<sup>2</sup>Instead of using `*` to denote a wildcard to match any tag name as defined in the XPath specification, we use `_` as wildcard and `*` to denote transitive closure in this paper.

|    |   |            |  |
|----|---|------------|--|
| S  | → | NP VP NP   | $I$ saw the old man $PP_{11}$ today                |
| VP | → | V NP NP    | $I$ $V_5$ $Det_8$ $Adj_9$ $N_{10}$ $PP_{11}$ today |
| NP | → | NP PP      | $I$ $V_5$ $NP_7$ $PP_{11}$ today                   |
| NP | → | Det Adj* N | $I$ $V_5$ $NP_6$ today                             |
| PP | → | Prep NP    | $I$ VP today                                       |

(a) CFG Productions                      (b) Some Proper Analyses

**Figure 3. CFG and Its Proper Analyses**

Furthermore, as shown by [16], two commonly used horizontal navigations in linguistic queries cannot be expressed in XPath: *immediate following* (see query  $Q_3$  in Figure 2), and its inverse *immediate preceding*.

Immediate following navigation can be understood with respect to the context-free grammar (CFG) which licenses the trees. For example, Figure 1 shows a derivation tree of the context-tree grammar with the production rules in Figure 3(a), where the parent child relationship corresponds to the derivation of rules. Applying grammar productions in reverse to a sentence, we can get a set of sequences, called *proper analyses* [9]. In other words, each proper analysis is a derivation of the root that can ultimately produce the sentence. Figure 3(b) shows some proper analyses of the sentence *I saw the old man with a dog today* with respect to the grammar in Figure 3(a).

We say that a node  $n$  *immediately follows* another node  $m$  in a linguistic tree if and only if  $n$  appears immediately after  $m$  in a proper analysis according to the productions of the grammar. From the sample proper analyses in Figure 3(b), we know that  $V_5$  is immediately followed by  $NP_6$ ,  $NP_7$  and  $Det_8$ , and therefore we can determine that  $NP_6$  and  $NP_7$  are the results of  $Q_3$ .

The transitive closure of immediate following is expressible in XPath: If a node  $n$  appears after  $m$  in a proper analysis, then the relationship between  $n$  and  $m$  is equivalent to the navigation defined by the *following* axis in XPath. For example, in Figure 1, node  $N_{10}$ ,  $N_{15}$  and  $N_{17}$  all follow  $V_5$ .

**Table 1. LPath Navigation Axes**

| Type       | LPath Axis                  | Abbreviation  | Closure | Core XPath Support |
|------------|-----------------------------|---------------|---------|--------------------|
| Vertical   | child                       | /             |         | ✓                  |
|            | descendant                  | /descendant:: | /+      | ✓                  |
|            | parent                      | \             |         | ✓                  |
|            | ancestor                    | /ancestor::   | \+      | ✓                  |
| Horizontal | immediate-following         | ->            |         | ×                  |
|            | following                   | -->           | ->+     | ✓                  |
|            | immediate-preceding         | <-            |         | ×                  |
|            | preceding                   | <--           | <-+     | ✓                  |
| Sibling    | immediate-following-sibling | =>            |         | ×                  |
|            | following-sibling           | ==>           | =>+     | ✓                  |
|            | immediate-preceding-sibling | <=            |         | ×                  |
|            | preceding-sibling           | <==           | <=+     | ✓                  |
| Other      | self                        | .             |         | ✓                  |
|            | attribute                   | @             |         | ✓                  |

Beside tree navigations, there are two commonly used features in linguistic queries which are difficult or impossible to express in XPath: subtree scoping and edge alignment.

### 2.2.2 Subtree Scoping

Linguistic tree navigation often needs to be scoped within a subtree. In contrast to  $Q_4$ ,  $Q_5$  searches for nouns which follow a verb within a verb phrase. For example, consider a verb  $V_5$  and three nouns which follow it,  $N_{10}$ ,  $N_{15}$ , and  $N_{17}$  in Figure 1. Since  $N_{17}$  is outside the verb phrase  $VP_4$ , it does not satisfy the query. [14] proposes a technique to convert a conjunctive query with an XPath axis that expresses scope to an XPath query. However, the size of the resulting query can be exponential in the size of the original query. By explicitly providing scope as a language primitive, we can implement it efficiently.

### 2.2.3 Edge Alignment

Linguists are often interested in nodes whose positions are the leftmost or rightmost within a particular subtree. The alignment of a child node with the leftmost or rightmost edge of its parent (as in  $Q_6$ ) can be expressed using the position function in XPath. For example,  $Q_6$  can be expressed as `//VP/_[last()][self::NP]`.

However, the alignment of a descendant with the leftmost or rightmost edge of a node as in  $Q_7$  cannot be expressed by an XPath query using the position function. A putative XPath equivalent for  $Q_7$  could be: `//VP/_[last()][self::NP]`. However, this XPath expression evaluates to  $\emptyset$  on the tree in Figure 1, while  $Q_7$  should evaluate to  $\{NP_6, NP_{13}\}$ . This is because edge alignment refers to the node order in an XML tree, while the XPath position function refers to the order in a sequence obtained

```

RLP ::= HP | HP '{ RLP }'
HP ::= | S HP
S ::= A '::' LA NodeTest RA Predicates*
LA ::= | '^'
RA ::= | '$'
A ::= '/' | '/descendant' | '.' | '\' | '\ancestor'
      | '<=' | '>' | '<==' | '==>'
      | '<-' | '->' | '<--' | '-->'

```

*RLP: RelativeLocationPath; HP: HeadPath; S: Step; A:AxisName; LA: Left Alignment; RA: Right Alignment.*

**Figure 4. The Grammar of LPath That Differs From XPath**

from intermediate results which does not necessarily represent the structural order in the original XML tree.

## 3 LPath: A Path Language For Linguistic Trees

In this section we present the LPath language for querying linguistic trees, which extends XPath with new primitive horizontal tree navigation axes, subtree scoping and edge alignment. These new features of LPath are shown in Figure 4, which highlights the difference between the LPath and XPath grammars. The rest of the LPath specification is the same as that in [10]. For space reason, it is omitted here.

LPath navigation axes include all XPath axes and eight new axes: immediate-following (`->`) as formally defined in Definition 3.1, immediate-following-sibling (`=>`), immediate-preceding (`<-`), immediate-preceding-sibling (`<=`), following-or-self, preceding-or-self, following-sibling-or-self and preceding-sibling-or-self. We include the or-self axes so that the axis set contains both primary axes and their transitive closure (`*` and `+`). To be concise,

we omit the discussion of the or-self and namespace axes in the rest of the paper. A summary of LPath axes, their syntactic abbreviations, the relationships between them, as well as their relationship with Core XPath [13], a clean logic core of XPath language, is given in Table 1. Following XPath, we use ‘//’ as an abbreviation for `/descendant-or-self::node()`. Note that LPath has axes for both primitive and transitive closures of vertical and horizontal navigations, filling a gap in the XPath axis set.

**Definition 3.1:** In a tree  $T$ , a node  $n$  *immediately follows* a node  $m$  if and only if  $n$  follows  $m$  and there does not exist a node  $n'$ , such that  $n'$  follows  $m$  and  $n$  follows  $n'$ . ■

We introduce braces to express subtree scoping<sup>3</sup>. This forces all node navigations to be constrained to a subtree. When ‘{’ occurs after a query node  $n$ , all the axes between ‘{’ and ‘}’ are evaluated within the XML subtree rooted at the node matching  $n$ . For example,  $Q_4$  can be expressed as `//VP/V-->N`. In contrast,  $Q_5$  constrains the query with subtree scoping on node  $VP$  and can be expressed as: `//VP{/V-->N}`. Given the XML tree in Figure 1, although node  $N_{17}$  is a following node for  $V_5$  in the whole tree, it is outside the scope of  $VP_4$ ’s subtree and is therefore not part of the result for  $Q_5$ .

We introduce  $\wedge$  to force left edge alignment, and  $\$$  to force right edge alignment, motivated by the syntax of regular expression languages. For example,  $Q_6$  can be expressed as: `//VP{/NP\$}`. Often  $\wedge$  and  $\$$  are used together with subtree scoping to align nodes within a subtree instead of the whole tree.

LPath queries for all sample linguistic queries are shown in Figure 2 in the LPath column.

The following lemma is suggested by lemmas 5.2 and 5.3 in [16]:

**Lemma 3.1:** Scoping, immediate-following, immediate-following-sibling and their reverse axes can not be expressed by Core XPath. ■

## 4 LPath Query Evaluation

A good query language is both expressive and efficient. We have discussed the design of LPath and illustrated how to express linguistic queries using it. Next we will discuss how to efficiently evaluate LPath queries.

To capture hierarchical order in a linguistic tree and to enable efficient LPath axis and edge alignment processing, we propose a new interval-based labeling scheme. Using this labeling, we can detect the relationship between tree

<sup>3</sup>Note that here braces are used differently than the ones in attribute value templates in XSLT.

**Table 2. Axes and Label Comparisons**

| Vertical Navigation                   |  |
|---------------------------------------|--|
| child( $m, n$ )                       | $n.id = m.pid$                               |
| descendant( $m, n$ )                  | $m.l \geq n.l, m.r \leq n.r,$<br>$m.d > n.d$ |
| parent( $m, n$ )                      | $m.id = n.pid$                               |
| ancestor( $m, n$ )                    | $m.l \leq n.l, m.r \geq n.r,$<br>$m.d < n.d$ |
| Horizontal Navigation                 |  |
| immediate-following( $m, n$ )         | $m.l = n.r$                                  |
| following( $m, n$ )                   | $m.l \geq n.r$                               |
| immediate-preceding( $m, n$ )         | $m.r = n.l$                                  |
| preceding( $m, n$ )                   | $m.r \leq n.l$                               |
| Sibling Navigation                    |  |
| immediate-following-sibling( $m, n$ ) | $m.l = n.r, m.pid = n.pid$                   |
| following-sibling( $m, n$ )           | $m.l \geq n.r, m.pid = n.pid$                |
| immediate-preceding-sibling( $m, n$ ) | $m.r = n.l, m.pid = n.pid$                   |
| preceding-sibling( $m, n$ )           | $m.r \leq n.l, m.pid = n.pid$                |
| Others                                |  |
| attribute( $m, n$ )                   | $m.id = n.id, m.name$<br>begins with @       |

nodes with respect to all LPath axes simply by inspecting their labels.

The labeling scheme is based on the following observations for an ordered tree without unary branching (that is, every inner node has at least two children).

- **Containment:** A node  $m$  is a descendant of  $n$  if and only if every leaf descendant of  $m$  is a leaf descendant of  $n$ .
- **Adjacency:** A node  $m$  immediately follows  $n$  if and only if the leftmost leaf descendant of  $m$  immediately follows the rightmost leaf descendant of  $n$ .

To see the adjacency property, we notice that a node  $m$  immediately follows  $n$  if and only if  $m$  appears immediately after  $n$  in a derivation  $d$  of the root (proper analysis). If we replace  $m$  (and  $n$ ) with its derivation consisting of its leaf descendant sequence in  $d$ ,  $m$  immediately follows  $n$  if and only if the leftmost leaf descendant of  $m$  appears immediately after the rightmost leaf descendant of  $n$  in  $d$ .

According to these two properties, the relationship between two nodes in an ordered tree without unary branching can be detected according to the relationship between their leaf descendants.

For a tree with unary branches, it is possible that node  $n$  and its descendant  $m$  have the same leftmost and rightmost leaf descendants, and therefore their ancestor-descendant relationship cannot be determined by the containment property solely. To distinguish  $m$  and  $n$ , we need to further consider the node depth. The depth information can also be used to distinguish the parent-child relationship from the ancestor-descendant relationship.

To test the sibling relationship between nodes  $n$  and  $m$ , we need to check whether they share the same parent. To expedite sibling navigations, which are frequent in linguistic queries, we include  $id$  and  $pid$  in a node label, where  $id$  and  $pid$  are the unique identifier of a node and its parent, respectively.

We distinguish element nodes from attribute nodes using  $name$  which records either an attribute name starting with '@' or a tag.

Now we formalize the labeling scheme<sup>4</sup>.

**Definition 4.1:** We assign each node a tuple  $\langle left, right, depth, id, pid, name \rangle$ , shortened as  $\langle l, r, d, id, pid, name \rangle$ , in the following fashion:

1. Let  $n$  be the leftmost leaf element. Assign  $n.l = 1$ .
2. Let  $n$  be a leaf element. Assign  $n.r = n.l + 1$ .
3. Let  $m$  and  $n$  be consecutive leaf elements where  $m$  is on the left. Then assign  $m.r = n.l$ .
4. Let  $n$  be an inner element which has a sequence of leaf descendants in order:  $m_1, \dots, m_k$ . Then assign  $n.l = m_1.l$  and  $n.r = m_k.r$ .
5. For each element  $n$ , let  $n.d$  be the depth of  $n$ , where the root has a depth of 1.
6. For each element  $n$ , assign a nonzero  $id$  as its unique identifier ( $= f(l, r, d)$  where  $f$  is a Skolem function).
7. For each element  $n$ , assign  $n.pid$  to be  $n$ 's parent's unique identifier; if  $n$  is the root, assign  $n.pid = 0$ .
8. For each attribute  $a$  associated with an element  $n$ , assign the same  $\langle l, r, d, id, pid \rangle$  as  $n$  to  $a$ .
9. For each element  $n$ , let  $n.name$  be the tag of  $n$ . For an attribute  $a$ , let  $a.name$  be the attribute name of  $a$ . ■

The node labels can be constructed in a single depth-first traversal of a linguistic tree.

Table 2 shows how to determine the LPath axis relationship of any two nodes by inspecting their labels.<sup>5</sup>

We store linguistic tree nodes along with their labels in a relational database and translate an LPath query to an SQL query. According to Table 2, each LPath axis can be translated to an SQL join. The query translation module is similar as the XPath-to-SQL translation discussed in the literature [11, 18] and is omitted here.

**Example 4.1:** Figure 5 shows part of the relation including label information for the sample annotation tree in Figure 1, where the  $id$  attribute in the table  $T$  corresponds to the node ids in Figure 1. Consider node  $NP_6$ : it has label  $l=3, r=9, d=3$ . We detect that node  $S_2$  with label  $l=1, r=10, d=1$  is an ancestor of  $NP_6$  since  $S_2.l \leq NP_6.l, S_2.r \geq NP_6.r$ , and  $S_2.d < NP_6.d$  according to Table 2. Furthermore, node  $V_5$

<sup>4</sup>This definition can easily be extended to multiple trees by introducing tree identifiers.

<sup>5</sup>Extensions to reflexive versions of the axes are easy and are omitted here. For example, descendant-or-self( $m, n$ )= $m.l \geq n.l, m.r \leq n.r, m.d \geq n.d$

| <i>left</i> | <i>right</i> | <i>depth</i> | <i>id</i> | <i>pid</i> | <i>name</i> | <i>value</i> |
|-------------|--------------|--------------|-----------|------------|-------------|--------------|
| 1           | 10           | 1            | 2         | 1          | S           |              |
| 1           | 2            | 2            | 3         | 2          | NP          |              |
| 1           | 2            | 2            | 3         | 2          | @lex        | I            |
| 2           | 9            | 2            | 4         | 2          | VP          |              |
| 2           | 3            | 3            | 5         | 4          | V           |              |
| 2           | 3            | 3            | 5         | 4          | @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          |
|             |              |              | ...       |            |             |              |
|             |              |              | ...       |            |             |              |

Figure 5. Relational Representation of  $T$

with label  $l=2, r=3, d=3$  immediately precedes  $NP_6$  since  $NP_6.l = V_5.r$ . ■

## 5 Experimental Results

We have implemented the LPath query engine in C++ [1]. The labeled form of linguistic trees are stored in a commercial relational database, with schema  $\{ tid, left, right, depth, id, pid, name, value \}$ . The attribute  $tid$  is used to distinguish different trees, and  $value$  records data values. The relation is clustered by  $\{ name, tid, left, right, depth, id, pid \}$ . Indexes  $\{ tid, value, id \}$ ,  $\{ value, tid, id \}$  and  $\{ tid, id, left, right, depth, pid \}$  are also built to improve performance. We use Yacc to generate a parser to translate an LPath query to an SQL query, which is fed to the relational database to get the result.

The system was tested on a commercial relational database. Techniques proposed in the literature [15] can also be explored to speed up query processing of XML data in relational databases.

### 5.1 Experimental Setup

The experiments were performed on a 2GHz Pentium 4 machine, with 512M memory and one 7200rpm hard disk. All experiments were repeated 7 times independently, and the average query evaluation time was reported, disregarding the maximum and minimum values.

#### 5.1.1 Systems

We compare the performance of the LPath query engine with two popular linguistic query language implementations, TGrep2 [25] and CorpusSearch [24]. Both TGrep2 and CorpusSearch are query engines for finding structures in a repository of linguistic trees and are used to query the Penn Treebank corpus. We also present the performance of

|               | WSJ     | SWB     |
|---------------|---------|---------|
| File Size     | 35983kB | 35880kB |
| Tree Nodes    | 3484899 | 3972148 |
| Unique Tags   | 1274    | 715     |
| Maximum Depth | 36      | 36      |

(a) Test Data Sets

|    | WSJ    |        | SWB    |        |
|----|--------|--------|--------|--------|
|    | Tag    | Freq   | Tag    | Freq   |
| 1  | NP     | 292430 | -DFL-  | 193708 |
| 2  | VP     | 180405 | VP     | 185259 |
| 3  | NN     | 163935 | NP-SBJ | 135867 |
| 4  | IN     | 121903 | .      | 135753 |
| 5  | NNP    | 114053 | ,      | 133528 |
| 6  | S      | 107570 | S      | 132336 |
| 7  | DT     | 101190 | NP     | 129804 |
| 8  | NP-SBJ | 95072  | PRP    | 114332 |
| 9  | -NONE- | 79247  | NN     | 76390  |
| 10 | JJ     | 75266  | RB     | 73477  |

(b) Top 10 Frequent Tags in Data Sets

|          | LPath Query                                 | Size of Result |        |
|----------|---|----------------|--------|
|          |   | WSJ            | SWB    |
| $Q_1$    | //S[//_[@lex=saw]]                          | 153            | 339    |
| $Q_2$    | //VB->NP                                    | 23618          | 16557  |
| $Q_3$    | //VP/VB-->NN                                | 63857          | 32386  |
| $Q_4$    | //VP{/VB-->NN}                              | 46116          | 25305  |
| $Q_5$    | //VP{/NP\$}                                 | 29923          | 22554  |
| $Q_6$    | //VP{/NP\$}                                 | 215104         | 112159 |
| $Q_7$    | //VP[{//^VB->NP->PP\$}]                     | 2831           | 1963   |
| $Q_8$    | //S[//NP/ADJP]                              | 7832           | 2900   |
| $Q_9$    | //NP[not(//JJ)]                             | 211392         | 109311 |
| $Q_{10}$ | //NP[->PP[//IN[@lex=of]]=>VP]               | 192            | 31     |
| $Q_{11}$ | //S[{//_[@lex=what]<br>->_[@lex=building]}] | 2              | 5      |
| $Q_{12}$ | //_[@lex=rapprochement]                     | 1              | 0      |
| $Q_{13}$ | //_[@lex=1929]                              | 14             | 0      |
| $Q_{14}$ | //ADVP-LOC-CLR                              | 60             | 0      |
| $Q_{15}$ | //WHPP                                      | 87             | 20     |
| $Q_{16}$ | //RRC/PP-TMP                                | 8              | 3      |
| $Q_{17}$ | //UCP-PRD/ADJP-PRD                          | 17             | 4      |
| $Q_{18}$ | //NP/NP/NP/NP/NP                            | 254            | 12     |
| $Q_{19}$ | //VP/VP/VP                                  | 8769           | 6093   |
| $Q_{20}$ | //PP=>SBAR                                  | 640            | 651    |
| $Q_{21}$ | //ADVP=>ADJP                                | 15             | 37     |
| $Q_{22}$ | //NP=>NP=>NP                                | 7              | 7      |
| $Q_{23}$ | //VP=>VP                                    | 20             | 72     |

(c) Test Query Sets

Figure 6. Test Data and Query Sets

an XPath engine using a popular XML labeling scheme for comparison [11].

### 5.1.2 Data Sets

Two data sets are tested: the Wall Street Journal corpus and the Switchboard corpus, both from Treebank-3 [19].

**Wall Street Journal (WSJ)** was created by the Penn Treebank Project [22], in which 2,499 stories were selected from a three year Wall Street Journal collection of 98,732 stories for syntactic annotation.

**Switchboard (SWB)** includes parsed text of 650 conversations from the Switchboard transcripts. Switchboard [12] is a collection of about 2,400 two-sided telephone conversations among 543 speakers from all areas of the United States.

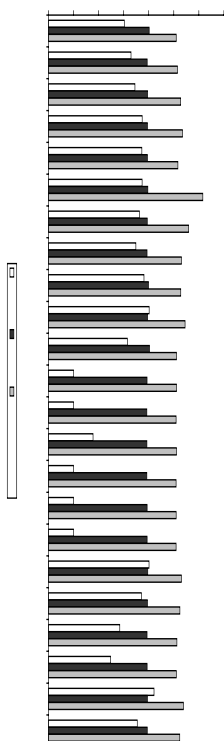
Characteristics of these data sets are presented in Figure 6(a), where *File Size* is the disk space required for the uncompressed ASCII representation of the linguistic trees. We list the ten most frequent tags appearing in each data set along with their frequencies in Figure 6(b).

### 5.1.3 Query Sets

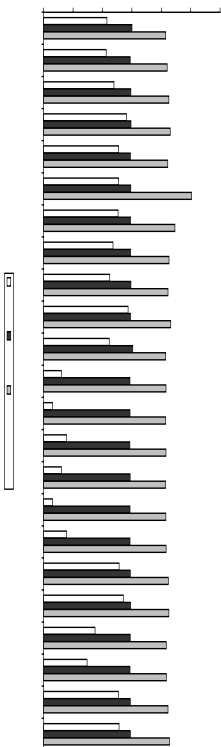
Since there is no benchmark for linguistic queries, we have asked linguists working on Treebank-3 to provide a set of queries with different characteristics as shown in Figure 6(c) and tested them on both data sets. It includes queries with value tests in predicates ( $Q_1$ ,  $Q_{10}$  to  $Q_{13}$ ), and queries containing sibling axis traversals ( $Q_{20}$  to  $Q_{23}$ ). We also select both queries with high selectivity ( $Q_{11}$ ,  $Q_{12}$  to  $Q_{17}$ ) and with low selectivity ( $Q_6$  and  $Q_9$ ). Some of the queries use slightly different tags from the ones presented in section 2 in order to match corresponding tags in data sets. 11 out of these 23 queries are expressible in XPath. In the experiment, we return the result size.

## 5.2 Query Processing Time

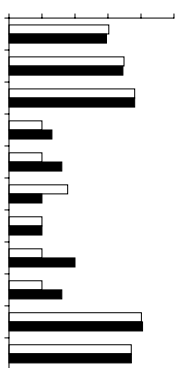
Figures 7 and 8 present the query execution time in log scale for the *WSJ* and *SWB* data sets, respectively, using the LPath query engine, TGrep and CorpusSearch. Since LPath to SQL query translation time is negligible, it is not included in the figure. For the *WSJ* dataset, the LPath query engine is the fastest except for queries  $Q_{10}$ ,  $Q_{18}$  and  $Q_{22}$ . In each of these three queries, low selectivity tags appear.  $Q_{10}$  contains the most frequent tag *NP*, the second most frequent



**Figure 7. Query Execution Time on Wall Street Journal Dataset**



**Figure 8. Query Execution Time on Switchboard Dataset**



**Figure 10. Execution Time of LPath and XPath Query Engine on Wall Street Journal Dataset**

### 5.4 Labeling Scheme

We compared the labeling scheme for LPath with a well-known labeling scheme designed for evaluating XPath queries [11], which is referred here as the XPath labeling scheme. This scheme uses textual positions of the start and end tags rather than *left* and *right* as used in the LPath labeling scheme. The XPath labeling scheme was proposed to efficiently evaluate the descendant axis and the child axis by testing label containment. We implement XPath-to-SQL query translation based on the strategy proposed in [11]. To compare the performance, we set other components of both labeling schemes to be the same.

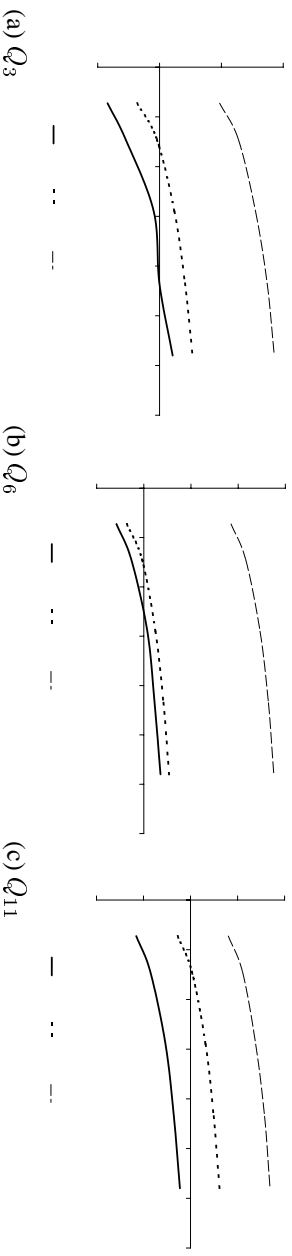
Figure 10 reports the query execution time on the *WSJ* dataset. The *SWB* dataset has similar results. 11 queries in the query set can be expressed using XPath and therefore

tag *VP*, and the fourth most frequent tag *IN*;  $Q_{18}$  contains *NP* 5 times in a row vertically; and  $Q_{22}$  contains *NP* 3 times horizontally. A relational database retrieves all the nodes whose name appears in a query as intermediate results and then joins them to compute the final query result. When low selectivity tags appear in a query, a lot of disk accesses are needed and the size of the intermediate results is big. On the other hand, when the tags appearing in a query have high selectivity, the LPath query engine works well by leveraging the indexes, e.g.  $Q_{16}$  and  $Q_{17}$ . Furthermore, the LPath query engine performs well for queries containing high-selectivity value predicates, as they effectively reduce the size of intermediate results for joins, e.g.  $Q_{11}$  and  $Q_{12}$ . For the *SWB* dataset, LPath query engine is the fastest for all queries since the frequency of tags in the *SWB* dataset is different. In particular, the tags appearing in the query set which are highly frequent in the *WSJ* dataset generally occur with much lower frequency in the *SWB* dataset.

### 5.3 Scalability of Query Processing Time

To test the scalability of these systems as the data size increases, we replicated the *WSJ* dataset between 0.5 and 4 times to get experimental datasets. Figure 9 reports the processing time on data of increasing sizes for representative sample queries of different types. The performance of other queries is similar and is omitted. As we can see, the LPath query engine scales well.





**Figure 9. Query Execution Time as WSJ Data Size Increases**

are supported by the XPath labeling scheme. As we can see, the performance of these two labeling schemes is almost the same. The proposed labeling scheme supports more queries without degrading the performance of XPath query evaluation.

## 6 Related Work

Several languages for querying linguistic data have been proposed [7, 23, 24, 4], but they are tied to specific data formats and are difficult to generalize and reuse. Moreover, little is known about whether query optimization techniques such as those developed in relational databases can be used.

TGrep2 is a *grep*-like tool to search linguistic trees [25]. Queries are expressed as nested expressions involving nodes and relationship between nodes. Query execution uses a binary file representation of the data, including an index on the words in the trees.

CorpusSearch is a language for syntax trees [24]. A query explicitly specifies a context node and a navigation within the context. It supports restricted disjunction and negation.

The Emu query language [7] is proposed to query speech annotations. This language supports node navigation and logical connectives. Though this query language has proved useful in phonetics research, it is not sufficient to express all queries for syntax trees, such as the child relationship ( $Q_1$ ) and negation.

[4] proposed a query language for annotation graphs, a data model proposed in [6]. The language focuses on expressing horizontal navigations and is not able to express all vertical navigations, for example, the parent-child relationship on trees with unary branching. A subset of the language is implemented using a relational engine [20].

A preliminary proposal of LPath is presented in [5].

We refer the reader to [17] for a survey of tree query languages and a discussion of linguistic tree query language requirements.

One important class of XPath query engines is based on labeling schemes that encode a node by its positional information. These methods have been shown to be very efficient [18, 11, 8]. These labeling schemes enable us to determine the vertical navigation relationships and following/preceding relationship between two nodes efficiently by checking the containment relationship of their labels, but they do not address all the horizontal navigations as required in linguistic queries, such as immediate following/preceding.

## 7 Conclusions and Future work

We have proposed LPath as an expressive and efficient language for linguistic queries. LPath extends XPath by introducing horizontal navigation primitives, subtree scoping, and edge alignment. Once these horizontal axes are added, horizontal and vertical navigation primitives as well as their closure are fully supported (cf. Table 1).

To efficiently evaluate LPath queries, we have proposed a labeling scheme that supports both horizontal and vertical navigations. Based on the labeling scheme, an LPath query evaluation system is designed and implemented.

We believe this work has implications on XPath design and implementation beyond linguistics. First, we found that several important node navigations are not supported or not easily expressed by XPath, presumably because these navigations are not required in current applications. However, as XML is a standard data format representing a tree model, and XPath is its standard language, it is beneficial for XPath to support these navigations in order to support wider scientific applications. Second, from a theoretical perspective, by including these new axes in XPath, the XPath axis set will contain both primitive horizontal navigations as well as their closures, just as with the vertical navigation axes, leading to an elegant inventory of axes.

The evaluation of LPath queries leverages a novel labeling scheme which is also useful for XPath query processing. As shown in section 5, an LPath query engine has the same

performance as an XPath query engine, but supports more queries. It is an interesting alternative to existing XPath query evaluation techniques.

In the future, we would like to formally analyze the expressiveness of LPath compared with XPath and CXPath proposed in [21]. We are also investigating how to support path closures (e.g.  $(- \rightarrow \text{NP})^*$ ) and overlapping trees.

## 8 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 funded by NSF 0317826 *Querying Linguistic Databases*, NSF IIS 0415810 *Preserving Constraints in XML Data Exchange*, and NSF IIS 0513778 *Data Cooperatives: Rapid and Incremental Data Sharing with Applications to Bioinformatics*.

## References

- [1] <http://www ldc.upenn.edu/Projects/QLDB/>.
- [2] TIGER PROJECT. <http://www.ims.uni-stuttgart.de/projekte/TIGER/>.
- [3] XQuery 1.0: An XML query language, June 2001. <http://www.w3.org/XML/Query>.
- [4] S. Bird, P. Buneman, and W.-C. Tan. Towards a query language for annotation graphs. In *Proceedings of the Second International Conference on Language Resources and Evaluation*, 2000.
- [5] S. Bird, Y. Chen, S. B. Davidson, H. Lee, and Y. Zheng. Extending XPath to support linguistic queries. In *Proceedings of PLAN-X*, 2005.
- [6] S. Bird and M. Liberman. A formal framework for linguistic annotation. In *Speech Communication* 33, pages 23 –60, 2001.
- [7] S. Cassidy and J. Harrington. Multi-level annotation of speech: an overview of the emu speech database management system, 1999.
- [8] Y. Chen, S. Davidson, and Y. Zheng. BLAS: An Efficient XPath Processing System. In *Proceedings of SIGMOD*, 2004.
- [9] N. Chomsky. Formal properties of grammars. In *Handbook of Mathematical Psychology*, pages 323 – 428, 1963.
- [10] J. Clark and S. DeRose. XML Path language (XPath), November 1999. <http://www.w3.org/TR/xpath>.
- [11] D. DeHaan, D. Toman, M. Consens, and M. T. Ozsu. A comprehensive XQuery to SQL translation using dynamic interval encoding. In *Proceedings of SIGMOD*, 2001.
- [12] J. J. Godfrey and E. Holliman. SWITCHBOARD-1 Release 2, 1997. <http://wave ldc.upenn.edu>.
- [13] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. In *Proceedings of VLDB*, 2002.
- [14] G. Gottlob, C. Koch, and K. U. Schulz. Conjunctive Queries over Trees. In *Proceedings of PODS*, 2004.
- [15] T. Grust. Accelerating XPath location steps. In *Proceedings of SIGMOD*, 2002.
- [16] C. Lai. A Formal Framework for Linguistic Tree Query. Master’s thesis, Department of Computer Science and Software Engineering, University of Melbourne, Victoria, Australia., 2005.
- [17] C. Lai and S. Bird. Querying and Updating Treebanks: A Critical Survey and Requirements Analysis. In *Proceedings of the Australasian Language Technology Workshop*, 2004.
- [18] Q. Li and B. Moon. Indexing and querying XML data for regular path expressions. In *The VLDB Journal*, pages 361–370, 2001.
- [19] M. A. M. P. Marcus, B. Santorini and A. Taylor. Treebank-3, 1999. <http://wave ldc.upenn.edu>.
- [20] X. Ma, H. Lee, S. Bird, and K. Maeda. Models and Tools for Collaborative Annotation. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, 2002.
- [21] M. Marx. Conditional XPath, the first order complete XPath dialect. In *Proceedings of PODS*, 2004.
- [22] U. of Pennsylvania. The Penn Treebank Project, 1995. <http://www.cis.upenn.edu/treebank/home.html>.
- [23] R. Pito. TGrep manual. <http://mccawley.cogsci.uiuc.edu/corpora/tgrep.pdf>.
- [24] B. Randall. CorpusSearch, 2000. <http://www.cis.upenn.edu/brandal/CSSstuff/CManual/Contents.html>.
- [25] D. Rohde. TGrep2 manual. <http://tedlab.mit.edu/dr/Tgrep2/tgrep2.pdf>.