

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 $10^6 - 10^9$ 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:

```
<phoneme id="ph20" span="2045ms-2092ms">f</phoneme>
<phoneme id="ph21" span="2092ms-2132ms">I</phoneme>
<phoneme id="ph22" span="2132ms-2204ms">l</phoneme>
...
<word id="wd03" span="ph20-ph31">Philadelphia</word>
<nameex id="n12" span="wd03-wd03" type="location"/>
```

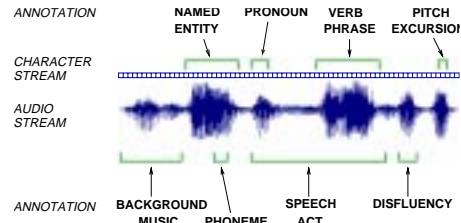


Figure 1: Linguistic Annotation: Structured Coding of Extents of Time-Series Data (e.g. Character Data, Audio Data)

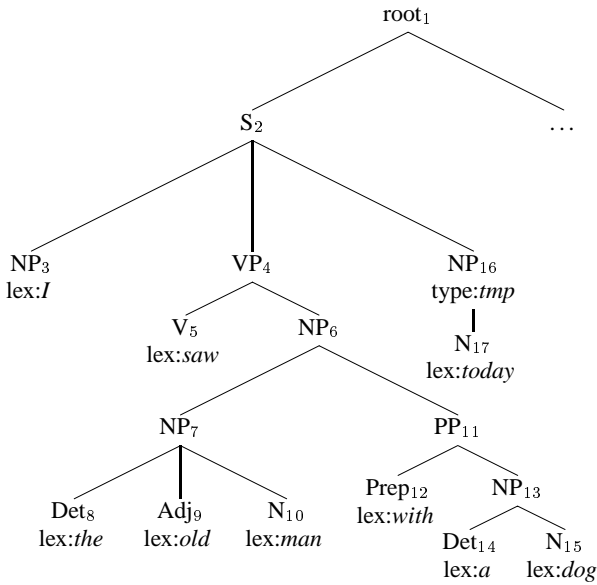
Despite the large amount of attention that has been paid to data collection and associated computational tools for linguistic annotation, there has been relatively little consideration of efficiency and scalability. Specialized languages for querying linguistic data repositories have been proposed [3, 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.



S: sentence; NP: noun phrase; VP: verb phrase; PP: prepositional phrase; Det: determiner; Adj: adjective; N: noun; Prep: preposition; V: verb; tmp: temporal (NB. Nodes are indexed to facilitate the discussion.)

Figure 2: Tree representation

translation 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 NP_7 (noun phrase) is an annotation of the Det_8 , Adj_9 and N_{10} 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 V_5 immediately precedes NP_6 , NP_7 and Det_8 . (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:

CFG Productions	Some Proper Analyses
$S \rightarrow NP VP (NP)$	<i>I saw the old N with NP today</i>
$VP \rightarrow V NP (NP)$	<i>I V the Adj man PP today</i>
$NP \rightarrow NP PP$	<i>NP saw NP with a telescope NP</i>
$NP \rightarrow Det Adj^* N$	<i>I VP NP</i>
$PP \rightarrow Prep NP$	<i>I saw NP today</i>

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 Find noun phrases that immediately follow a verb: $\{NP_6, NP_7\}$ (both nodes immediately follow V_5)
- Q_2 Find nouns that follow a verb which is a child of a verb phrase: $\{N_{10}, N_{15}, N_{17}\}$ (all three follow V_5)
- Q_3 Within a verb phrase, find nouns that follow a verb which is a child of a verb phrase: $\{N_{10}, N_{15}\}$ (within VP_4 , N_{10} and N_{15} follow V_5)
- Q_4 Find noun phrases which do not have an adjective descendant: $\{NP_3, NP_{13}, NP_{16}\}$
- Q_5 Find noun phrases which are the rightmost child of a verb phrase: $\{NP_6\}$
- Q_6 Find noun phrases which are the rightmost descendant of a verb phrase: $\{NP_6, NP_{13}\}$ (both are descendants of VP_4 , and no other descendants of VP_4 follow them)
- Q_7 Find all verb phrases that are comprised of a verb, a noun phrase, and a prepositional phrase: $\{VP_4\}$ (VP_4 is comprised of V_5 , NP_7 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 can be expressed as `//VP/V/following::N`, Q_4 can be expressed as `//NP[not(boolean(//Adj))]` and Q_5 as `//VP/_[last()][self::NP]`.³ However, there are several problems with XPath with respect to the other sample queries. For the remainder of this section we show why 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 1.⁴ For a given node n , the node set s reachable by the immediate-following axis is comprised of the nodes on the leftmost 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

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

⁴We 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←m/_[position()=first()]`.

Algorithm 1 Immediate Following Axis of Node n

```

1:  $m \leftarrow n$   $\triangleright n$  is the input node
2: while  $m/\text{immediate-following-sibling}::\_ = \emptyset$  and
    $m/\text{parent}::\_ \neq \emptyset$  do
3:    $m \leftarrow m/\text{parent}::\_$ 
4: end while  $\triangleright m$  is the first ancestor having a following sibling
5:  $s \leftarrow \emptyset$ 
6: if  $m/\text{immediate-following-sibling}::\_ \neq \emptyset$  then
7:    $m \leftarrow m/\text{immediate-following-sibling}::\_$ 
8:    $s \leftarrow s \cup \{m\}$ 
9:   while  $m/\_ \neq \emptyset$  do
10:     $children \leftarrow m/\_$ 
11:    for all  $c \in children$  do
12:      if  $c/\text{preceding-sibling}::\_ = \emptyset$  then
13:         $m \leftarrow c$ 
14:      end if
15:    end for
16:     $s \leftarrow s \cup \{m\}$ 
17:  end while
18: end if
19: return  $s$   $\triangleright s$  is the set of nodes which immediately follow  $n$ 

```

rooted at a specified node. Query Q_3 exemplifies this requirement with respect to Q_2 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_5 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_5 . However, XPath cannot describe more deeply nested alignments, as required for Q_6 . A putative XPath equivalent is: $//VP//_ [\text{last}()][\text{self}::\text{NP}]$. However, this XPath expression evaluates to $\{NP_{13}\}$ on our running example, while Q_6 evaluates to $\{NP_6, NP_{13}\}$. The key difference is that \wedge and $\$$ are sensitive to node order in an XML subtree, while the XPath position function considers a node’s position in the sequence obtained from subquery evaluation which may be unrelated to its position in the original XML tree. In other words, the order restrictions imposed by edge-alignment are part of specifying a tree pattern match, and not expressible using the position function which considers a node sequence independently of the XML tree.

In light of these expressive shortcomings of XPath, we now develop some syntactic extensions which permit our queries to be expressed (§3), then show how the extended language can be evaluated (§4).

3. LPATH: A PATH LANGUAGE FOR LINGUISTIC TREES

Three extensions to XPath were motivated in the last section: immediate precedence, subtree scoping, and edge alignment.

Immediate Precedence. We introduce four new axes, namely immediate-following, its inverse immediate-preceding, and the sibling versions of these axes, namely immediate-following-sibling and immediate-preceding-sibling. These elementary horizontal navigations are intransitive, and none are supported by XPath (although their closures are supported). A summary of these LPath axes and their slash or arrow abbreviations is given in Table 1 (note that all closures use Kleene plus). For example, to express Q_1 to

Table 1: Linguistic Tree Navigation Primitives

Vertical Navigation	
/	child
//	descendant ($/^+$)
\	parent
\\	ancestor (\backslash^+)
Horizontal Navigation	
->	immediate-following
-->	following ($->^+$)
<-	immediate-preceding
<--	preceding ($<-^+$)
Sibling Navigation	
=>	immediate-following-sibling
==>	following-sibling ($=>^+$)
<=	immediate-preceding-sibling
<==	preceding-sibling ($<=^+$)

find all noun phrases that immediately follow a verb, we can write $//V->\text{NP}$.

Subtree Scoping. We introduce braces into the language to permit scopes to be expressed. These will force all navigation 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 XML node matching n . For example, consider Q_3 which is a scoped version of Q_2 . Q_2 can be expressed as $//VP/V-->\text{N}$, and we can add the scope restriction required for Q_3 as follows: $//VP\{ /V-->\text{N}\}$. Consider the XML tree in Figure 2: although N_{17} is a following node for V_5 in the whole tree, it is out of the scope of VP_4 , therefore 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 \wedge to force left-alignment, and $\$$ to force right-alignment. (These choices are motivated by the syntax of popular regular expression languages.) These operators are defined as follows: $\wedge A = A[\text{not}<--_]$; $A\$ = A[\text{not}<-->_]$. Accordingly, Q_6 can be expressed as: $//VP\{ / \text{NP}\}$.

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.

```

P ::= AP | AP '{' P '}'
AP ::= S AP
S ::= A T | A T '[' R ']'
A ::= '/' | '/' '/' | '/' '/' | '/' | '\ ' | '\\ '
      | '<=' | '>' | '<==' | '==>'
      | '<-' | '->' | '<--' | '-->'
T ::= Qname | _ | '@' Qname C Qname
R ::= R 'or' R | R 'and' R | R 'not' R | '(' R ')'
      | P | P C ' "' Qname ' "'
C ::= '=' | '<=' | '>=' | '<>' | 'like'

```

P: Path expression; *AP*: Absolute Path expression; *S*: Step, *A*: Axis; *T*: Tag;
R: Restriction (predication)

Figure 3: The grammar of LPath

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.

- Q_1 Find noun phrases that are immediately following a verb.
`//V->NP`
- Q_2 Find nouns that follow a verb which is a child of a verb phrase.
`//VP/V-->N`
- Q_3 Within a verb phrase, find nouns that follow a verb which is a child of a verb phrase.
`//VP{V-->N}`. Compared to Q_2 , Q_3 restricts the following axis navigation within the scope of the noun phrase.
- Q_4 Find noun phrases which do not have an adjective descendant.
`//NP[not(//Adj)]`
- Q_5 Find noun phrases which are the rightmost child of a verb phrase.
`//VP{/NP$}`, where $\$$ is used to align the match to the rightmost child of a verb phrase.
- Q_6 Find noun phrases which are rightmost descendant of a verb phrase.
`//VP{/NP$}`
- Q_7 Find verb phrases comprised of a verb, a noun phrase, and a prepositional phrase.
`//VP[{/^V->NP->PP$}]`. Notice that “comprise” is a common notion in linguistic queries. To express it, we require the ability to scope, express left and right alignment and immediate following. As shown in the query, \wedge forces V to align to the leftmost of all descendants of VP , and $\$$ forces PP to align to the rightmost descendant.

4. EVALUATION

As 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.

T	left	right	depth	id	pid	name	value
1	.	.	0	1	0	root	
1	10	1	2	1	1	S	
1	2	2	3	2	2	NP	
1	2	2	3	3	3	@lex	I
2	9	2	4	2	2	VP	
2	3	3	5	4	4	V	
2	3	3	5	5	5	@lex	saw
3	9	3	6	4	4	NP	
3	6	4	7	6	6	NP	
3	4	5	8	7	7	Det	
3	4	5	8	7	7	@lex	the
.
.

Figure 4: Relational representation

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 $\langle \text{left}, \text{right}, \text{depth}, \text{id}, \text{pid}, \text{name}, \text{value} \rangle$, shortened as $\langle l, r, d, \text{id}, \text{pid}, \text{name}, \text{value} \rangle$, 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, \dots, 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 nonzero id as its unique identifier ($= f(l, r, d)$ where f is a Skolem function).
7. For each node n , assign $n.\text{pid}$ to be n ’s parent node identifier; if n is the root node, assign $n.\text{pid} = 0$.
8. For each node n , its 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 $\langle \text{left}, \text{right}, \text{depth}, \text{id}, \text{pid}, \text{name}, \text{value} \rangle$. 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 $T \bowtie_C T$, where C is the required constraint shown in Table 2.⁵

⁵Note that the condition $n.d < m.d$ for the descendant axis is required just for the case of unary branching. Reflexive versions of the axes are defined as follows: $\text{axis-or-self}(m, n) =_{\text{def}} \text{axis}(m, n) \vee m = n$.

Table 2: Axes and their Corresponding Join Constraints

Vertical Navigation	
$\text{child}(m, n)$	$n.id = m.pid$
$\text{descendant}(m, n)$	$m.l \geq n.l, m.r \leq n.r, m.d > n.d$
$\text{parent}(m, n)$	$m.id = n.pid$
$\text{ancestor}(m, n)$	$m.l \leq n.l, m.r \geq n.r, m.d < n.d$
Horizontal Navigation	
$\text{immediate-following}(m, n)$	$m.l = n.r$
$\text{following}(m, n)$	$m.l \geq n.r$
$\text{immediate-preceding}(m, n)$	$m.r = n.l$
$\text{preceding}(m, n)$	$m.r \leq n.l$
Sibling Navigation	
$\text{immediate-following-sibling}(m, n)$	$m.l = n.r, m.pid = n.pid$
$\text{following-sibling}(m, n)$	$m.l \geq n.r, m.pid = n.pid$
$\text{immediate-preceding-sibling}(m, n)$	$m.r = n.l, m.pid = n.pid$
$\text{preceding-sibling}(m, n)$	$m.r \leq n.l, m.pid = n.pid$

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 a $\{$, 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. $m.l \leq n.l, n.r \leq m.r$, and $n.d > m.d$. Once the corresponding $\}$ is met, we pop the current environment from the stack.

Example 4.1: $//NP\{ //Adj-->N\} (Q_3)$ 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 l l3, r r3, d d3, l2, r2, d2, l1, r1, d1 from T,
( select l l2, r r2, d d2, l1, r1, d1 from T,
( select l l1, r r1, d d1 from T where T.name = 'NP' ) T1
where T.name = 'Adj' and T.l >= l1 and
T.r <= r1 and T.d > d1 ) T2
where T.name = 'N' and T.l >= r2 and
T.l >= 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 $\$$ 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. $\wedge A$ means that $T.name = A$ and $T.left = T'.left$ and $T.depth > T'.depth$. Similarly, $A\$$ means that $T.name = A$ and $T.right = T'.right$ and $T.depth > T'.depth$.

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 ($->$), immediate-following-sibling ($=>$), immediate-preceding ($<-$), and immediate-preceding-sibling ($<=>$). 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, $//VP\{ /V-->N\}$ will only return those nodes labeled N which have a VP ancestor. Finally, new edge-alignment operators \wedge and $\$$ 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, $//VP\{ //NP\$ \}$ 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. $(->NP)^*$); 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*.

⁶This implementation is available from <http://www ldc.upenn.edu/Projects/QLDB/>

7. REFERENCES

- [1] 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.
- [2] N. Bruno, N. Koudas, and D. Srivastava. Holistic twig joins: Optimal XML pattern matching. In *Proceedings of SIGMOD*, 2002.
- [3] S. Cassidy and J. Harrington. Multi-level annotation of speech: an overview of the EMU speech database management system, 1999.
- [4] Y. Chen, S. Davidson, and Y. Zheng. Indexing keys in hierarchical data. Technical Report MS-CIS-01-30, University of Pennsylvania, Computer and Information Science Department, 2001.
- [5] Y. Chen, S. Davidson, and Y. Zheng. BLAS: An Efficient XPath Processing System. In *Proceedings of SIGMOD*, 2004.
- [6] S. Cassidy. XQuery as an Annotation Query Language: a Use Case Analysis In *Proceedings of the Third International Conference on Language Resources and Evaluation*, 2002.
- [7] N. Chomsky. Formal properties of grammars. In E. Galanter D. Luce, R. R. Bush (eds), *Handbook of Mathematical Psychology*, volume 2, pages 323–418. New York: Wiley and Sons, 1963.
- [8] D. DeHaan, D. Toman, M. P. Consens, and T. Ozsu. A comprehensive XQuery to SQL translation using dynamic interval encoding. In *Proceedings of SIGMOD*, 2003.
- [9] G. Gottlob, C. Koch, and R. Pichler. Efficient Algorithms for Processing XPath Queries. In *Proceedings of VLDB*, 2002.
- [10] D. Graff and S. Bird Many Uses, Many Annotations for Large Speech Corpora: Switchboard and TDT as Case Studies. *Proceedings of the Second International Conference on Language Resources and Evaluation*, 2000.
- [11] C.. Lai and S. Bird. Querying and Updating Treebanks: A Critical Survey and Requirements Analysis Unpublished manuscript, 2004.
- [12] Q. Li and B. Moon. Indexing and querying XML data for regular path expressions. In *The VLDB Journal*, pages 361–370, 2001.
- [13] 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.
- [14] M. Marx. Conditional XPath, the first order complete XPath dialect. In *Proceedings of PODS*, 2004.
- [15] U. of Pennsylvania. The Penn Treebank Project, 1995.
<http://www.cis.upenn.edu/~treebank/home.html>
- [16] R. Pito. Tgrep manual. <http://mccawley.cogsci.uiuc.edu/corpora/tgrep.pdf>
- [17] B. Randall. CorpusSearch, 2000.
<http://www.cis.upenn.edu/~brandall/CSstuff/CSManual/Contents.html>



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:

Bird, S., Chen, Y., Davidson, S., Lee, H., & Zheng, Y. (2005). Extending XPath to support linguistic queries. In, Proceedings, Programming Language Technologies for XML (PLANX), Long Beach, USA.

Publication Status:

Published

Persistent Link:

<http://hdl.handle.net/11343/34055>