

LPath⁺: A First-Order Complete Language for Linguistic Tree Query

Catherine Lai and Steven Bird

Department of Computer Science and Software Engineering
University of Melbourne, Victoria 3010, AUSTRALIA

Department of Linguistics and Linguistic Data Consortium
University of Pennsylvania, Philadelphia PA 19104, USA

laic@ling.upenn.edu, sb@csse.unimelb.edu.au

Abstract

Large databases of linguistic annotations are used for testing linguistic hypotheses, and for training language processing models. Linguistic annotations are often syntactic or prosodic and typically have a tree structure. Our goal is to develop a language that can express a wide range of linguistic tree queries and has an efficient implementation. We argue that by adding some simple closures to the LPath language, we can meet this goal. We call this new addition to the XPath family LPath⁺. We place LPath and LPath⁺ in the hierarchy of XPath languages and conclude that LPath⁺ is first-order complete over trees. This means LPath⁺ is efficiently implementable in SQL.

1 Introduction

In recent years, a great variety of linguistic query languages have been proposed (X, 2004). Their primary use is in extracting information about linguistic structures from large annotated corpora. Most of these languages are designed for trees, and many have been applied to corpora such as the Penn Treebank (Marcus et al., 1993). Despite this considerable effort, relatively little is known about the formal expressiveness of these languages, or the computational resources required to process them as the size of the data grows.

XPath is a language for describing paths in trees, and it has proven to be very popular for the tree-structured document markup of the XML world. Path languages are well-suited to linguistic trees, both for identifying subtrees relative to the root and for representing binary relationships between tree nodes, as shown in Figure 1.

Recently, the LPath language has been proposed as a convenient path-based language for querying linguistic trees (X, 2006). This language augments the navigational axes of XPath with three new operators, greatly increasing query succinctness. Moreover, LPath can be translated into SQL

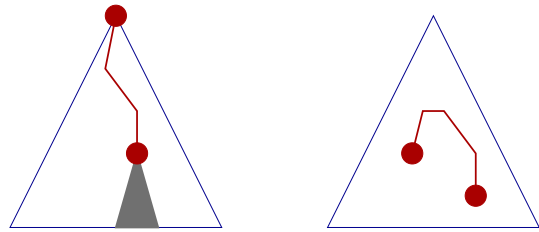


Figure 1: Paths in Trees

Core XPath	
$\not\sqsubset$	(Theorem 3)
LPath	
$\not\sqsubset$	(Theorem 7)
Conditional XPath	
\parallel	(Theorem 8)
LPath ⁺	

Figure 2: Expressiveness of the XPath Family

for efficient execution. In this paper we investigate the expressiveness of LPath with respect to *Core XPath* and to a first-order complete language called *Conditional XPath*. We also extend LPath to permit restricted path closures, and argue that this new language which we call LPath⁺ supports all the navigational and closure requirements of linguistic tree query while retaining LPath's efficient query evaluation. A summary of the relationships between these languages appears in Figure 2.

This paper is organized as follows. Section 2 reviews LPath, XPath, and Conditional XPath, and Section 3 examines the LPath operators to see which of them can be expressed in XPath or Conditional XPath. Section 4 presents Conditional LPath, or LPath⁺. We formally establish its expressiveness and query evaluation

```

locpath := locstep | /locpath | locpath '/' locpath
         | locpath '|' locpath
locstep := axis::test | axis::test[fexpr]...[fexpr]
fexpr   := locpath | fexpr 'and' fexpr
         | fexpr 'or' fexpr | 'not' fexpr
         | '(' fexpr ')'
axis    := ancestor | ancestor_or_self | self
         | parent | child
         | descendant | descendant_or_self
         | following | preceding
         | following_sibling | preceding_sibling
test    := p | _

```

Figure 3: Syntax of Core XPath (p is a node label; attribute syntax is omitted)

complexity, and discuss its merits as a linguistic tree query language.

2 XPath, Conditional XPath, and LPath

Marx (2004) presents a family of XPath languages that extend the navigational functionality of XPath 1.0. Core XPath (\mathcal{X}) was originally presented by Gottlob et al. (2003). This language is XPath 1.0 stripped of non-navigational components such as attributes and namespaces. The syntax of \mathcal{X} is shown in Figure 3.

Conditional XPath (\mathcal{X}^{c*}) extends \mathcal{X} primarily by adding a conditional axis (cf. Palm’s propositional tense logic for trees (Palm, 1999)). This expresses conditional paths where every node satisfies a particular condition (filter expression). \mathcal{X}^{c*} replaces the definition of `axis` in Figure 3 as follows:

```

axis    := primaxis | '[' fexpr ']' primaxis |
         primaxis '[' fexpr ']' | axis '*'
primaxis := self | child | parent |
         immediate_following_sibling |
         immediate_preceding_sibling

```

Primary axes (`primaxis`) represent the smallest steps that can be taken in each direction from a node in a tree. Note, \mathcal{X} does not include the one-step sibling axis *immediate following sibling* or its converse. However, it does include its transitive closure, *following sibling*. In fact, the transitive closures of each primary axis are included in \mathcal{X} . We define `axis+` as the non-reflexive transitive closure of an axis as `/axis::_/(axis)*`.

The XPath language family has strong ties to a family of modal logics developed to describe syntactic structure. In fact, there is a one-to-one correspondence between the XPath hierarchy laid out by Marx (2004) and the modal logics described by Blackburn et al. (2003). A major result of the modal characterization has been that \mathcal{X}^{c*} is a first-order complete language over the signature $\tau = \{\text{descendant}, \text{following-sibling}\}$, also

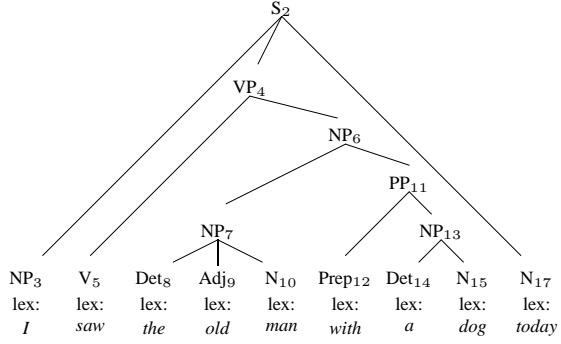


Figure 4: Tree Representation

known as FO^{tree} (Marx, 2005). That is, every \mathcal{X}^{c*} expression is equivalent to a FO^{tree} formula, $\phi(x, y)$, with exactly two free variables, and the converse also holds.

LPath consists of three linguistically motivated syntactic additions to XPath (X, 2006). These are the *immediate following* axis (and its converse), tree edge alignment, and a scoping operator. An efficient interpreter for LPath converts LPath expressions into equivalent SQL expressions over annotation graphs (Bird and Liberman, 2001). The following examples illustrate the syntax and interpretation of LPath queries. When applied to the tree in Figure 4 they return the specified node sets.

- `//S[//_[@lex=saw]]` $\{S_2\}$
Find a sentence containing the word *saw*.
- `//V->NP` $\{NP_6, NP_7\}$
Find noun phrases that are immediately following a verb.
- `//VP/V-->N` $\{N_{10}, N_{15}, N_{17}\}$
Find nouns that follow a verb which is a child of a verb phrase.
- `//VP{/V-->N}` $\{N_{10}, N_{15}\}$
Within a verb phrase, find nouns that follow a verb which is a child of the given verb phrase.
- `//VP{/NP$}` $\{NP_6\}$
Find noun phrases which are the rightmost child of a verb phrase.
- `//VP{/NP$}` $\{NP_6, NP_{13}\}$
Find noun phrases which are rightmost descendants of a verb phrase.
- `//VP[{//^V->NP->PP$}]` $\{VP_4\}$
Find verb phrases comprised of a verb, a noun phrase, and a prepositional phrase.

The syntax of LPath is given in Figure 5, while Figure 6 gives the interpretation of the axes. We briefly review the syntax of LPath, and refer the reader to (X, 2006) for full details.

Scoping: The subtree *scoping* operator is denoted by braces, $\{\}$. These braces represent queries constrained to the subtree that is rooted at the node immediately before the opening brace.

```

locpath  :=  abspath | abspath '{' locpath '}' |
             locpath '|' locpath
abspath  :=  | locstep abspath
locstep  :=  axis test | axis test '[' fexpr ']'
fexpr    :=  locpath | fexpr 'and' fexpr
             | fexpr 'or' fexpr
             | 'not' fexpr | '(' fexpr ')'
axis     :=  '\' | '\\ | '\\*' | '.'
             | '/' | '/' | '/'*
             | '->' | '<-' | '-->' | '<--'
             | '=>' | '<=' | '==>' | '<=='
test     :=  p | _ | '^'p | p'$'

```

Figure 5: LPath Syntax

The location path inside the scoping braces is evaluated as if this subtree were the whole of the input. For example, if a query $/. . . P$ finds some constituent of interest P , then the extended query $/. . . P\{. . . Q\}$ finds the some other item Q but only if it occurs inside the subtree rooted at P .

Alignment: Left and right tree edge alignment, \wedge and $\$$ respectively, together with the scoping operator allow us to constrain a node to be leftmost (rightmost) edge in a constituent. For example, this allows us to single out words which occur at major phrase boundaries.

Horizontal axes: The *immediate following* axis, $->$, is the natural one-step version of the *following* axis, $-->$. We can consider this axis as taking a step to constituents immediately right of the current node. LPath also includes an *immediate following sibling* relation (and its converse). These axes make it possible to refer to context independently of structure, and are available in other languages, such as Tgrep2 (Rohde, 2001).

These extensions allow LPath to express a range of linguistic tree queries compactly and concisely. Despite this, it is unclear whether LPath is expressive enough for linguistic tree query. For example, linguistic constraints are often easiest expressed as transitive closures (eg. maximal projections of heads). However, LPath cannot *explicitly* express path closures of any sort. Moreover, the affect of adding closures on LPath’s efficiency is unclear. Certainly tree query languages that can express the full range of transitive closures, such as monadic second-order logic (Kepser, 2004), face major problems on the tractability front.

Accordingly, we would like to understand where LPath lies on the expressiveness hierarchy of XPath languages. In particular, we would like to know if the LPath operators offer any extra

\backslash	parent	$/$	child
$\backslash\backslash$	ancestor	$//$	descendant
$\backslash\backslash*$	ancestor or self	$//*$	descendant or self
$->$	immediate following	$<-$	immediate preceding
$-->$	following	$<--$	preceding
$=>$	immediate following sibling	$<=$	immediate preceding sibling
$==>$	following sibling	$<==$	preceding sibling
$.$	self		

Figure 6: LPath Axes and their Interpretation

expressiveness to these path-based languages. The following sections explore these questions and indicate how LPath can be extended to express closures while maintaining evaluation efficiency.

Notation: The following sections take an incremental approach in investigating Core XPath and LPath extensions. This involves several languages constructed and related by restrictions on closures and the LPath operators defined above. Subscripts and superscripts denote the addition of a particular operator. $\mathcal{X}_{\{\}}^{c*}$ denotes Conditional XPath extended with the scoping operator (but not $->$ or its converse). $\mathcal{X}_{->\{\}\$}$ represents Core XPath with $->$, $=>$ and their converses, scoping and edge alignment, i.e. LPath (\mathcal{L}). \mathcal{L}^{c*} denotes LPath extended with the conditional axis, i.e. LPath⁺.

3 Expressiveness of LPath with respect to XPath and Conditional XPath

As previously mentioned, both \mathcal{X} and \mathcal{X}^{c*} have elegant foundations in propositional modal logic and its extensions. This foundation enables us to combine a wealth of theoretical results with an incremental approach to understanding the formal properties of LPath.

3.1 LPath Operators and Core XPath

To begin, it is easy to see that edge alignment can be expressed in \mathcal{X} , for we can define:

$\wedge A \equiv A[\text{not}<--_]$; $A\$ \equiv A[\text{not}<-->_]$.

Now, the scoping operator asserts that the dominance relation exists between the scoping node and those appearing within the scoping braces. The query $NP\{ //PP-->NP\backslash\backslash VP\}$ is illustrated in Figure 7 as a cyclic graph whose edges are labelled with the axes relating pairs of nodes. The scoping constraint corresponds to the extra dashed edges.

The difficulty implementing the scoping operator in path-based *variable-free* languages such as \mathcal{X} is that they have no memory of previous steps; it is not possible, in general, for a path to loop back to a particular node (i.e. we cannot decorate our nodes with indexes to write $NP_i //PP-->NP\backslash\backslash VP\$

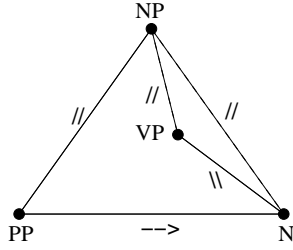


Figure 7: Scoping induced cycles:
 $\text{NP}\{ // \text{PP} \rightarrow \text{N} \setminus \text{VP} \}$

$\setminus \text{NP}_i$, to ensure that the first and last NPs were the same). Instead, to transform a ‘scoped’ expression into an \mathcal{X} expression we need to convert cyclic queries into a disjunction of acyclic ones. An algorithm that does this for the positive fragment of \mathcal{X} has been presented by Gottlob et al. (2004). Positive \mathcal{X} is the set of \mathcal{X} expressions that do not include negation in filter expressions. However, note that positive \mathcal{X} cannot express the edge alignment operators.

Lemma 1. *The scoping operator adds no expressiveness to Positive \mathcal{X} .*

Proof. Let L be an \mathcal{L} expression that uses the scoping operator. We simply draw the query graph of L adding *descendant* labelled edges between scoping and scoped nodes. Now, the algorithm of Gottlob et al. (2004) does the rest of the work. This provides us with a disjunction, D , of acyclic query trees equivalent to the original query. Each query tree in the disjunction has a node x that represents the target node set of the original expression L . Thus each query tree in D is equivalent to a finite set of filter expressions, $F = \{F_i\}$, based at x . Thus L is equivalent to a \mathcal{X} expression of the form $// _ [A]$ where $A \equiv \bigvee F_i$. \square

The result of applying this transformation on the \mathcal{L} expression $\text{NP}\{ // \text{PP} \rightarrow \text{N} \setminus \text{VP} \}$ is shown in Figure 8. The rather cumbersome equivalent \mathcal{X} expression is as follows.

$$\begin{aligned} // \text{N} [\setminus \setminus \text{VP} \setminus \setminus _ < = \setminus \setminus \text{NP} [// * \text{PP}] \\ \text{or } \setminus \setminus * _ < = _ [// * \text{PP}] \setminus \setminus \text{VP} \setminus \setminus \text{NP} \\ \text{or } \setminus \setminus \text{VP} < = _ [// * \text{PP}] \setminus \setminus \text{NP}] \end{aligned}$$

Unfortunately, this technique does not extend to \mathcal{X} expressions with negation. Negation within the scoping braces only holds inside the scoped subtree. In fact, the effects of negation and scoping on the *ancestor* axis give the following lemma.

Lemma 2. $\mathcal{X} \subsetneq \mathcal{X}_{\setminus}$

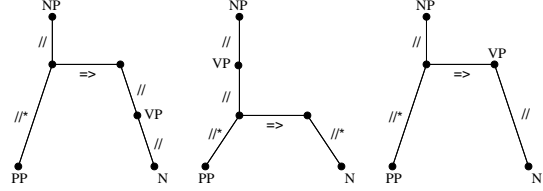


Figure 8: Acyclic Version of $\text{NP}\{ // \text{PP} \rightarrow \text{N} \setminus \text{VP} \}$

Proof. Consider the \mathcal{L} expression

$// \text{B} / \text{A} \{ // \text{A} [\text{not} (\setminus \setminus _ [\text{not} . \text{A}])] \}$. This finds A -labelled nodes such that there is a \setminus -path (upwards) of nodes whose labels conform to the regular expression A^+B . Now, Marx and de Rijke (2004) have shown that all \mathcal{X} queries can be expressed in first order logic over trees using at most two variables, extended with *child* and *immediate following sibling* relations. However, the regular expression above cannot be expressed in this signature in less than three variables (Marx, 2005). \square

A similar linguistic example is the \mathcal{L} query $// \text{NP} \{ // \text{VP} [\text{not} \setminus \setminus \text{PP}] \}$. This expression selects VPs that are dominated by NPs with no intervening PP. We can express this in \mathcal{X}^{c*} as $// \text{NP} (/ _ [\text{not} . \text{PP}]) * / \text{VP}$. However, as we have seen, this closure cannot be expressed in \mathcal{X} .

The other additions of \mathcal{L} to \mathcal{X} are the one-step horizontal axes. Now, we also know from Marx (2005) that $=>$ cannot be derived from $==>$ in less than three variables. Hence, the next theorem also follows from the two-variable property of \mathcal{X} .

Theorem 3. $\mathcal{X} \subsetneq \mathcal{L}$.

We can continue to add \mathcal{L} operators to \mathcal{X} as primitives to gain a further idea of the expressiveness they provide. Consider \mathcal{X}_{\setminus} . Since edge alignment can be expressed in \mathcal{X} , \mathcal{X}_{\setminus} is equivalent to \mathcal{L} without the scoping operator. Now, if scoping was redundant in \mathcal{X}_{\setminus} , then \mathcal{X}_{\setminus} would be expressively equivalent to \mathcal{L} . However, we can show that this is not the case.

Lemma 4. $\mathcal{X}_{\setminus} \subsetneq \mathcal{L}$.

Proof. The additional axes express sequential relations and so do not give \mathcal{X}_{\setminus} any more ability to express queries of the form $// \text{B} / \text{A} \{ // \text{A} [\text{not} (\setminus \setminus _ [\text{not} . \text{A}])] \}$. Thus, the scoping operator is not expressible in \mathcal{X}_{\setminus} . \square

In fact, the interaction of \mathcal{L} operators results in queries allowing implicit expression of

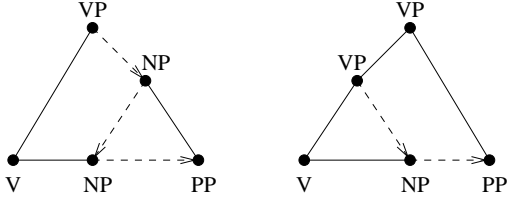


Figure 9: Scoping and immediate following. Dashed lines indicate the path of the query.

closures that are inexpressible in \mathcal{X} . First, consider the interaction of the scoping and edge alignment operators. As noted previously, this allows us to express subtree edge alignment. The query $//S\{[//^{\wedge}NP-->VP\$]\}$ is equivalent to constraining the NP (VP) to be a leftmost (rightmost) descendant of the S. To express this we need to state that every node on the $/$ -path between the S and NP has no left sibling. This is just the conditional axis, which is inexpressible in \mathcal{X} .

Second, consider the scoping operator and the *immediate following* axis. This axis allows the current context node to move outside of the scoped subtree. This is demonstrated in Figure 9. Consider a situation where we wish to find verb phrases (VP) containing a noun phrase (NP) immediately followed by a prepositional phrase (PP), i.e. $//VP\{//NP->PP\}$. From the point of view of the NP node, there is no way to tell if an immediate following PP is dominated by the same VP originally being tested. In order to constrain $->$ to be within a subtree, we need to phrase this constraint using other axes.

For example, the *following* axis, $-->$, can be alternatively defined as:

$$-->t[F] \equiv \backslash\backslash_==>_//t[F].$$

Now, the only chance that we may leave the scope with $-->$ is if the ‘ancestor’ part of the expression takes us above the scoping node. As long as we constrain how far up the ancestor is chosen, we are assured of staying within the scope. The cycle-removing algorithm of Gottlob et al. (2004) enumerates the positions such an ancestor can occupy.

The similar version of $->$ requires an ability to identify ancestors that are rightmost and descendants that are leftmost. As in the previous example, these constraints cannot be expressed in \mathcal{X} . Without some sort of memory device there is no way to force this primitive to stay within a scope. Note, in first-order formulas such a

memory device would come in the form of extra variables. Importantly, this means that the only way we can represent the immediate following relation is with the primitive.

Putting all this together gives a clear picture of the expressiveness required to implement \mathcal{L} operators using members of the XPath family of languages. It is clear the scoping and the immediate following axes are more than syntactic sugar in the context of \mathcal{X} . The interaction between all three \mathcal{L} operators as well as negation admit some of the expressiveness of the conditional axes. The next section looks at the affect of these operators in the setting of Conditional XPath.

3.2 LPath operators and Conditional XPath

The first thing to notice in moving to Conditional XPath (\mathcal{X}^{c*}) is that the immediate following relation is now expressible:

$$-> \equiv ([\text{not}(=)_]) \backslash \backslash * => ([\text{not}(<=_)]) *$$

Since \mathcal{X} is contained in \mathcal{X}^{c*} , the definitions of edge alignment operators carry over from \mathcal{X} . The scoping operator follows immediately from the first-order completeness of \mathcal{X}^{c*} (Marx, 2005). Consider now \mathcal{X}^{c*} with the scoping operator added to its syntax, $\mathcal{X}_{\{\}}^{c*}$.

Lemma 5. $\mathcal{X}^{c*} = \mathcal{X}_{\{\}}^{c*}$.

Proof. Any $\mathcal{X}_{\{\}}^{c*}$ with scoping braces deleted is just a \mathcal{X}^{c*} expression. Therefore we can convert any $\mathcal{X}_{\{\}}^{c*}$ expression ignoring any scoping braces into a first-order formula $\phi(x, y)$. Let z be the variable representing the scoping node and let w_0, \dots, w_k be variables representing nodes in the scoped location path. For each w_i we conjoin the clause $\text{descendant}(z, w_i)$ in the appropriate variable scope. Since this does not change the number of free variables this has an equivalent \mathcal{X}^{c*} expression. \square

Thus all \mathcal{L} operators are expressible in \mathcal{X}^{c*} . Moreover, the first-order completeness of \mathcal{X}^{c*} means that the interactions between \mathcal{L} operators add no more expressiveness. However, there is no Kleene star in \mathcal{L} . In fact, the closures expressible in \mathcal{L} are due to the scoping operator. However, this does not give us any horizontal conditional closures available in \mathcal{X}^{c*} . We can show this in a few steps, as follows.

Lemma 6. *The filter expressions of $\mathcal{X}_{\rightarrow}$ are definable by first-order formulae $\varphi(x)$ in one free variable and at most two variables in signature $\tau_l = \{/, //, \rightarrow, \Rightarrow, P_i\}$ where P_i is a countable set of unary predicates.*

Proof. (Sketch) It is easy to translate $\mathcal{X}_{\rightarrow}$ filter expressions into a propositional modal logic over τ_l . The mapping into the two-variable first-order logic over the signature τ_l follows easily from the standard translation from modal logic (Blackburn et al., 2001). \square

Theorem 7. $\mathcal{L} \subsetneq \mathcal{X}^{c*}$

Proof. Consider the following FO query over the signature of \mathcal{X}^{c*} .

$$\begin{aligned} \text{imf}_{BA^+}(x, y) \equiv & \\ \text{following}(x, y) \wedge B(x) \wedge A(y) \wedge & \\ \forall z((\text{following}(x, z) \wedge \text{following}(z, y) \wedge \text{leaf}(z) & \\ \rightarrow \exists w((z = w \wedge A(w)) & \\ \vee(\text{ancestor}(z, w) \wedge A(w) \wedge \neg \text{ancestor}(w, x)))))) & \end{aligned}$$

Here, we require that there be some subtree with a cut whose labels conform to the regular expression BA^+ . The leftmost node in the cut must be a B labelled node bound to variable x . Similarly, the rightmost node in the cut must be an A labelled node bound to variable y . That is, this defines the conditional axis over the *immediate following* axis of \mathcal{L} . Recall that \mathcal{X}^{c*} is first-order complete so the formula above has an equivalent \mathcal{X}^{c*} expression. Such a query is orthogonal to the scoping operator. It follows from Lemma 6 that this cannot be expressed in \mathcal{L} . \square

That is, the expressiveness of LPath (\mathcal{L}) lies strictly between Core XPath (\mathcal{X}) and Conditional XPath (\mathcal{X}^{c*}). Thus \mathcal{L} is a new member of the XPath family of languages, and not a notational variant of one of the existing languages.

4 LPath⁺

4.1 The Expressiveness of LPath⁺

The obvious question now is whether \mathcal{L} with conditional axis, \mathcal{L}^{c*} , is any more expressive than \mathcal{X}^{c*} . The main point of difference between the two languages is the status of the *immediate following* (\rightarrow) axis. However, the proof of Theorem 7 shows how the conditional *immediate following* axis can be expressed in first-order logic over the signature of \mathcal{X}^{c*} .

imf_{BA^+} can easily be modified to deal with the conditional \rightarrow axis in general. This means that

all \mathcal{L}^{c*} expressions without scoping braces can be expressed first-order logic. As in Lemma 5, we can trivially add the scoping operator. \mathcal{L}^{c*} clearly contains \mathcal{X}^{c*} so we have the following equivalence:

Theorem 8. $\mathcal{L}^{c*} = \mathcal{X}^{c*}$; consequently \mathcal{L}^{c*} is expressively complete for first-order definable paths.

That is, for every FO^{tree} formula $\phi(x, y)$ (cf. Section 2) there exists an equivalent \mathcal{L}^{c*} expression and vice-versa. In fact, we can find an equivalent \mathcal{X}^{c*} expression for the conditional immediate following axis using the fact that \mathcal{X}^{c*} is closed under intersection and complementation (Marx, 2005). Using Marx's notation we can write an expression equivalent to $//B(\rightarrow A)^+$ as follows:

$$\text{imf}_{BA^+}(x, y) \equiv (?B / \text{following}?A) \cap \overline{\phi / \text{following}}$$

where

$$\begin{aligned} \phi(x, y) \equiv & (?B / \text{ancestor} / (\text{child}? \neg A)^+ / ? \text{leaf}) \\ & \cap \text{following} \end{aligned}$$

Along with the proof, Marx (2005) provides a method for finding the complement of any \mathcal{X}^{c*} path set. Thus, we now have a concrete method for translating \mathcal{L}^{c*} expressions into \mathcal{X}^{c*} . The modal basis of \mathcal{X}^{c*} lends evaluation tractability. From Alechina and Immerman (2000) we can see that \mathcal{L}^{c*} queries can be evaluated in time linear both the size of the data and the query.

4.2 LPath⁺ as Linguistic Tree Query Language

\mathcal{L}^{c*} is capable of expressing a large range of linguistic tree queries, including all the basic subtree matching queries identified in the requirements analysis in X (2004). The \mathcal{L}^{c*} axis set accounts for hierarchical, sequential and sibling orderings on unranked ordered trees. The addition of \rightarrow and \Rightarrow mean that all of \mathcal{L}^{c*} 's unbounded axes also have one-step versions. As such, there do not appear to be any such (unconditional) relations lacking in the \mathcal{L}^{c*} axis set. Thus, \mathcal{L}^{c*} appears to have the complete set of primitive axes necessary for linguistic tree query.

A clear advantage of \mathcal{L}^{c*} is its ability to explicitly express a useful range of closures which are unavailable in existing linguistic tree query languages. For example, consider sequential closures in the following query: *Find words consisting of consonant-vowel-consonant sequences.* Let

words, consonants and vowels be represented by the labels W, C, and V respectively. We can express this query in \mathcal{L}^{c*} as follows: $//W\{[/\wedge C(->C)*(->V)+(->C)+_ \$]\}$ Here, the $->$ axis allows us to capture the case where the consonants and vowels may not all be at the same depth, while the scoping and alignment operators allow us to fully specify the contents of the constituent selected as the scoping node.

More hierarchical closures can also be expressed. For example, to find NP nodes that conform to the simple grammar fragment, $NP \rightarrow Adj NP$; $NP \rightarrow N$, we can write: $//NP[(\{/\wedge Adj=>NP \$\})*/N]$.

Full regular expressions over paths cannot be expressed in \mathcal{L}^{c*} . However, such queries are unlikely to be required in linguistic analyses. Treebank annotations are usually licensed by grammars containing production rules (as in the previous example). Now, it is theoretically possible that mutual recursion may occur between two production rules, e.g. the rules $A \rightarrow \dots B \dots$ and $B \rightarrow \dots A \dots$. We can approximate this in \mathcal{L}^{c*} as $(/[_[A \text{ or } B]])*$. This sort of recursion is more concisely expressed as a regular path expression $(/A/B)*$. This seems to suggest that a higher level of expressiveness is motivated for reasons of convenience.

Counting and full transitive closures over paths are definable in higher-order logics used to describe linguistic structure, such as monadic second-order logic (Rogers, 1994). However, there are compelling linguistic arguments that expressiveness beyond the first-order realm is unnecessary. Berwick and Weinberg (1984) have argued against counting in natural language. Supporting evidence is provided by Hoeksema and Janda (1988), who find that morphological infixation refers to constituent boundaries but with no need for counting past one.

Moreover, higher order logics pose tractability problems in terms of query evaluation. For example, while MSO has linear data complexity, this result relies on a translation to tree automata which may result in a non-elementary blow-up in query size (Libkin, 1998). Thus, it is not at all clear how MSO can be implemented efficiently. Given the linguistic arguments outlined above, it does not seem necessary or worthwhile to sacrifice the efficiency of \mathcal{L}^{c*} for more expressiveness. Thus, \mathcal{L}^{c*} appears to have the right level of expressiveness

for linguistic tree query. By staying inside first-order logic, \mathcal{L}^{c*} also stays within reach of SQL and achieves an optimal trade-off between expressiveness and efficiency.

The only other current linguistic treebank query language with this level of expressiveness is fsq (Kepser, 2003). However, fsq only allows boolean queries. Moreover, \mathcal{L}^{c*} 's path-based syntax is much more intuitive and more closely aligned to actual descriptions of structure in the linguistics literature (Palm, 1999).

However, there is still a price to pay for choosing this approach over the variables and predicates of classical first-order logic. The major advantage of the latter is that variables can be used to identify specific nodes throughout a query. The scoping operator accounts for cases where we need to identify a subtree root throughout a path expression. Although \mathcal{L}^{c*} is first-order complete, it is not always clear how a first-order formula can be converted into a (variable-free) \mathcal{L}^{c*} expression. First-order completeness tells us that the following query is expressible: *Find the first common ancestor of noun phrases immediately followed by a verb phrase.* This can be expressed as follows:

$$\begin{aligned} \varphi(x) = & \exists y \exists z (\text{descendant}(x, y) \wedge \text{descendant}(x, z) \\ & \wedge NP(y) \wedge VP(z) \wedge \text{following}(y, z) \\ & \wedge \neg \exists w (\text{following}(y, w) \wedge \text{following}(w, z)) \\ & \wedge \neg \exists z' (\text{descendant}(x, z') \wedge \text{descendant}(z', y) \\ & \wedge \text{descendant}(z', z)). \end{aligned}$$

However, even with additional operators the equivalent \mathcal{L}^{c*} expression not obvious. We can express this query by using the \mathcal{X}^{c*} definition of the *immediate following* relation rather than the primitive $->$ axis.

```
//_[/[_[
(NP or (/_[not(=>_)])*/NP[not(=>_)])
and =>
(VP or (/_[not(<=_)])*/VP[not(<=_)])]]
```

However, this is a very different approach to representing this sort of query than suggested by the first-order formula above. These problems are inherent to path-based, variable-free languages.

5 Conclusion

In recent years, over a dozen linguistic query languages have been developed. As shown in our earlier survey (X, 2004), these languages have many features in common, but vary widely in syntax, in

supported linguistic relations, and in the kinds of quantification and negation they provide. Little is known about their formal expressiveness and so it is not clear which languages are notational variants, and which offer distinctive extra expressiveness. Similarly, the computational cost of any given syntactic feature is unknown.

LPath was proposed as a new query language which augmented the navigational axes of XPath with three additional tree operators. LPath is unique amongst linguistic query languages in being path-based, a characteristic which appears to be ideal for linguistically-motivated tree navigation. Moreover, LPath is unique in having an interpreter built on top of SQL, permitting query processing to leverage the existing indexing and optimizing technologies of relational database management systems.

We have analyzed each of the syntactic innovations of LPath and have shown that they are more than just syntactic sugar. In fact, LPath occupies a new position on the expressiveness hierarchy between Core XPath and Conditional XPath. We extended LPath with the addition of the conditional axis, resulting in a new language called Conditional LPath (or LPath⁺). We showed that LPath⁺ has exactly the same expressiveness as Conditional XPath.

This finding is significant, since it ensures that our path language, highly customised for the needs of linguistic query, incorporating scoping, alignment, horizontal navigation, and simple closures, does not exceed first-order expressiveness. We have shown that LPath⁺ is sufficiently expressive, and also that LPath⁺ queries can be mechanically translated to SQL for efficient execution.

References

- Natasha Alechina and Neil Immerman. 2000. Reachability logic: An efficient fragment of transitive closure logic. *Logic Journal of the IGPL*, 8(3):325–337.
- Robert C. Berwick and Amy S. Weinberg. 1984. *The Grammatical Basis of Linguistic Performance : Language Use and Acquisition*, volume 11 of *Current studies in linguistics*. MIT Press, Cambridge, Mass.
- Steven Bird and Mark Liberman. 2001. A formal framework for linguistic annotation. *Speech Communication*, 33:23–60. <http://arxiv.org/abs/cs/0010033>.
- Patrick Blackburn, Maarten de Rijke, and Yde Venema. 2001. *Modal logic*. Cambridge University Press, New York, NY, USA.
- Patrick Blackburn, Bertrand Gaiffe, and Maarten Marx. 2003. Variable free reasoning on finite trees. In *Proceedings of Mathematics of Language: MOL 8*, Bloomington, Indiana, USA, June 20–22.
- Georg Gottlob, Christoph Koch, and Reinhard Pichler. 2003. The complexity of XPath query evaluation. In *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS*, pages 179–190, San Diego, CA, USA, June 9–12. ACM.
- Georg Gottlob, Christoph Koch, and Klaus U. Schulz. 2004. Conjunctive queries over trees. In *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database System*, pages 189–200, Paris, France, June 14–16. ACM.
- Jack Hoeksema and Richard D. Janda. 1988. Implications of process-morphology for categorial grammar. In Richard T. Oehrle, Emmon Bach, and Deirdre Wheeler, editors, *Categorial Grammars and Natural Language Structures*. D. Reidel, Dordrecht.
- Stephan Kepser. 2003. Finite structure query: A tool for querying syntactically annotated corpora. In *EACL 2003: The 10th Conference of the European Chapter of the Association for Computational Linguistics*, pages 179–186.
- Stephan Kepser. 2004. Querying linguistic treebanks with monadic second-order logic in linear time. *Journal of Logic, Language and Information*, 13(4):457–470.
- Leonid Libkin. 1998. *Elements of Finite Model Theory*. Springer-Verlag.
- Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–30. <http://www.cis.upenn.edu/~treebank/home.html>.
- Maarten Marx and Maarten de Rijke. 2004. Semantic characterization of navigational XPath. In *Proceedings of TDM'04 Workshop on XML Databases and Information Retrieval*, Twente, The Netherlands, June.
- Maarten Marx. 2004. XPath with conditional axis relations. In *Advances in Database Technology - EDBT 2004, 9th International Conference on Extending Database Technology, Proceedings*, volume 2992 of *Lecture Notes in Computer Science*, pages 477–494, Heraklion, Crete, Greece, March 14–18. Springer.
- Maarten Marx. 2005. First order paths in ordered trees. In Thomas Eiter and Leonid Libkin, editors, *Database Theory - ICDT 2005, 10th International Conference, Edinburgh, UK, January 5-7, 2005, Proceedings*, volume 3363 of *Lecture Notes in Computer Science*, pages 114–128. Springer.
- Adi Palm. 1999. Propositional tense logic for trees. In *Proceedings of the Sixth Meeting on Mathematics of Language: MOL6*, University of Central Florida, Orlando, Florida.
- James Rogers. 1994. Studies in the logic of trees with applications to grammar formalisms. Technical Report 95-04, Department of Computer & Information Sciences, University of Delaware, Newark, Delaware, November.
- D. Rohde. 2001. Tgrep2 user manual. <http://citeseer.ist.psu.edu/569487.html>.