Proceedings of PACLIC 19, the 19th Asia-Pacific Conference on Language, Information and Computation

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

Annotated linguistic databases are widely used in linguistic research and in language technology development. These annotations are typically hierarchical, and represent the nested structure of syntactic and prosodic constituents. Recently, the LPath language has been proposed as a convenient path-based language for querying linguistic trees. We establish the formal expressiveness of LPath relative to the XPath family of languages. We also extend LPath to permit simple closures, resulting in a first-order complete language which we believe is sufficiently expressive for the majority of linguistic tree query needs.

1. Introduction

In recent years, a great variety of linguistic query languages have been proposed, most of them specialised for linguistic trees (Lai and Bird, 2004), and 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. One reason for this is that much of the work in this area has taken place in isolation from well-understood database query languages such as SQL and XPath (Clark and DeRose, 1999).

Recently, the LPath language has been proposed as a convenient path-based language for querying linguistic trees (Bird et al., 2006). It augments the navigational axes of XPath with three additional tree operators, and it can be translation into SQL 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 simple closures, and argue that this new language supports all the navigational and closure requirements of linguistic tree query.

This paper is organised 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 an extended language, Conditional LPath, or LPath$^+$, and discusses its merits as a linguistic tree query language.

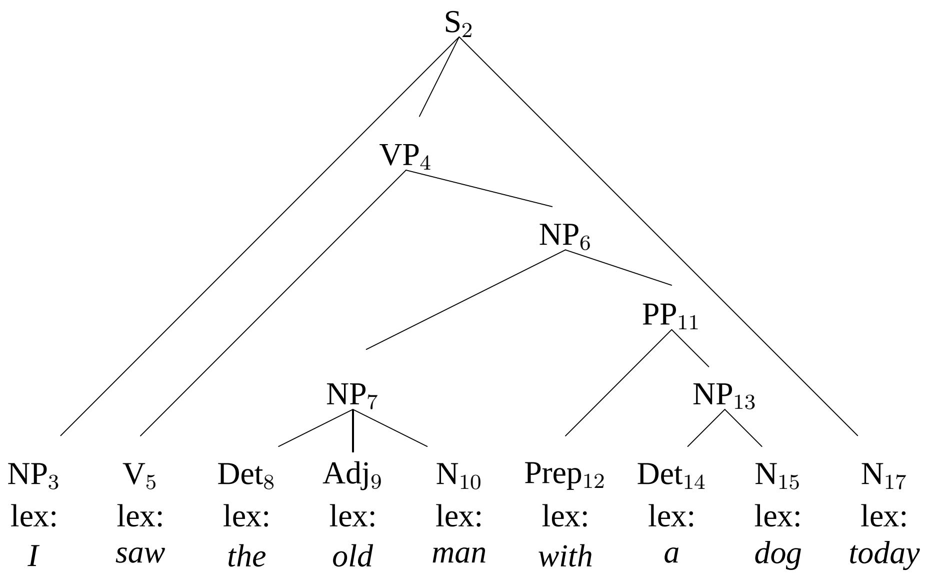

Figure 1: Tree Representation

2. Background: LPath and XPath

2.1. LPath

LPath was developed to be expressive enough for linguistic query but also to take advantage of relational database technology. As the name suggests, LPath is an extension of XPath. Bird et al. (2006) present three linguistically motivated syntactic additions. These are the *immediate following* axis (and its converse), tree edge alignment, and a scoping operator. They also present an efficient interpreter for LPath which converts LPath expressions into equivalent SQL expressions over annotation graphs (Bird and Liberman, 2001). Here are a selection of examples intended to illustrate the syntax and interpretation of LPath queries. When applied to the tree in Figure 1 they return the specified node sets.

1. `//S[//_[@lex=saw]` $\{S_2\}$
   Find a sentence containing the word *saw*.
2. `//V->NP` $\{NP_6, NP_7\}$
   Find noun phrases that are immediately following a verb.
3. `//VP/V-->N` $\{N_{10}, N_{15}, N_{17}\}$
   Find nouns that follow a verb which is a child of a verb phrase.
4. `//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.
5. `//VP{/NP$}` $\{NP_6\}$
   Find noun phrases which are the rightmost child of a verb phrase.
6. `//VP{//NP$}` $\{NP_6, NP_{13}\}$
   Find noun phrases which are rightmost descendants of a verb phrase.
7. `//VP[{//^V->NP->PP$}]` $\{VP_4\}$
   Find verb phrases comprised of a verb, a noun phrase, and a prepositional phrase.

```
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 2: LPath Syntax (`p` is a node label; attribute syntax is omitted)

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

Figure 3: LPath Axes and their Interpretation

The syntax of LPath is described in Figure 2. Figure 3 gives the translation of abbreviated axes. We briefly review the syntax of LPath here, and refer the reader to (Bird et al., 2006) for full details.

Scoping: The *scoping* operator is denoted by pairs of braces, `{}`. These braces represent queries constrained to a particular subtree rooted by the context node immediately before the opening brace. The location path inside the scoping braces is evaluated as if this subtree were the whole of the input. This allows us to write queries restricted to a particular constituent.

Alignment. Left and right tree edge alignment, ˆ and `$` respectively, together with the scoping operator allow us to constrain a node to be leftmost (rightmost) edge in a constituent. For example, the following query returns sentences (`S` labelled nodes) that begin with a noun phrase and end with a verb phrase: `//S{[//^NP-->VP$]}`.

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 context node. LPath also includes an *immediate following sibling* relation.

These extensions to XPath give LPath the ability to express a range of linguistic tree queries. However, LPath cannot express closures of any sort, and it is not clear where LPath lies on the hierarchy of XPath languages. Nor is it clear what extra expressiveness, if any, the LPath operators offer to these path-based languages. The following sections explore this question, and indicate how LPath can be extended to express closures, and how such an extension might be efficiently implemented.

```
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 | parent | child |
                descendant | descendant_or_self | self |
                following | preceding |
                following_sibling | preceding_sibling |
test       :=   p | _
```

Figure 4: Syntax of Core XPath

Notation. The following sections take an incremental approach to investigating the expressiveness of 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. $\mathcal{X}^{+}_{\{\}}$ denotes Conditional XPath extended with the scoping operator (but not `->` or its converse). $\mathcal{X}_{\texttt{->\{\}\$}}$ represents Core XPath with `->`, `=>` and their converses, scoping and edge alignment, that is, LPath or $\mathcal{L}$. $\mathcal{L}^{+}$ denotes LPath extended with the conditional axis.

2.2. XPath and Conditional XPath

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 can be seen as XPath 1.0 stripped of non-navigational components such as attributes and namespaces. The syntax of $\mathcal{X}$ is shown in Figure 4.

Conditional XPath ($\mathcal{X}^{+}$) extends $\mathcal{X}$ primarily by adding a conditional axis. This expresses conditional paths where every node in that path satisfies a particular condition represented as a filter expression. $\mathcal{X}^{+}$ replaces the definition of `axis` in Figure 4 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 nodes 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 closures, *following sibling*. In fact, the transitive closures of each primary axis are included in $\mathcal{X}$ so we will use these axis names for non-conditional closures. For example, `child*` is equivalent to `descendant_or_self`. We also define `axis+` as the non-reflexive transitive closure of an axis as `/axis::_/(axis)*`. Again, we will denote `child+` as `descendant`.

$\mathcal{X}^{+}$ is a first-order complete language. However, it is not always easy translate first-order formulas into path expressions. For example, there is often a need to distinguish first and last constituents in a phrase or to find the next constituent. Moreover, linguistic phenomena are often

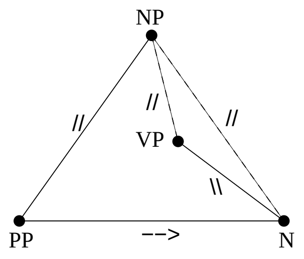

Figure 5: Scoping induced cycles: `NP{//PP-->N\\VP}`

restricted to particular types of constituents and their contents. In order to express scoping we need expressions that have some way of remembering the subtree they should be in. However, path-based, variable-free languages like $\mathcal{X}^+$ have no explicit memory and so can only do this implicitly.

3. The Relationship between LPath, XPath and Conditional XPath

How does the expressiveness of LPath, $\mathcal{L}$, compare with that of $\mathcal{X}$ and $\mathcal{X}^+$, existing well-understood languages? Perhaps LPath is just a syntactic variant of one of them, in which case we could build an interpreter to convert $\mathcal{L}$ expressions to $\mathcal{X}$ or $\mathcal{X}^+$ expressions. We take up this question in the next two subsections.

3.1. LPath Operators and Core XPath

To begin, it is easy to see that edge alignment can be expressed in $\mathcal{X}$. We note the following equivalences: `^A` $\equiv$ `A[not <-- _ ]` and `A$` $\equiv$ `A[not --> _ ]`.

The scoping operator can be thought of as the assertion that a dominance relation between the scoping node and those appearing within the scoping braces. This is illustrated in Figure 5. Here, the query `NP{//PP-->NP\\VP}` is drawn as a cyclic graph where 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 languages such as $\mathcal{X}$ is that they have no memory of previous steps. In general, it is not possible to return to a particular node unless every node in the tree is uniquely labelled. This is clearly not the case for linguistic trees. In order to to transform a ‘scoped’ expression into a $\mathcal{X}$ expression we need to convert cyclic queries into a disjunction of acyclic ones. An algorithm that does exactly 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) results in a disjunction, D of acyclic query trees that is equivalent to the original (cyclic) query. Each query tree in the disjunction has a node x that represents the target node set of of the original expression L. Thus each query tree in D is equivalent to a finite set

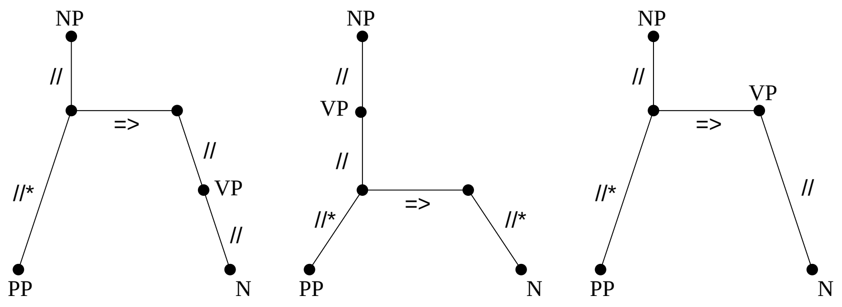

Figure 6: Acyclic version of `NP{//PP-->N \\VP}`

---

of filter expressions, $F = \{F_i\}$, based at x. Thus L is equivalent to a $\mathcal{X}$ expression of the form `//_[`A`]` where $A \equiv \bigvee F_i$. □

The result of applying this transformation on the $\mathcal{L}$ expression `NP{//PP-->N\\VP}` is shown in Figure 6. The equivalent $\mathcal{X}$ expression is as follows.

```
//N[\\VP\\_<=\\NP[//*PP] or \\*_<=_[//*PP]\\VP\\NP or
\\VP<=_[//*PP]\\NP]
```

Unfortunately, this technique does not extend to $\mathcal{X}$ expressions with negation. Negation within the scoping braces only holds inside the scoped subtree. This is not a problem for negated paths that do not involve the $ancestor$ axis because such paths cannot escape the scoped subtree. However, the effects of negation and scoping on the $ancestor$ axis give the following lemma.

Lemma 2. *$\mathcal{X}_{\{\}}$ is strictly more expressive than $\mathcal{X}$.*

Proof. Consider the $\mathcal{L}$ expression `//B/A{//A[not (\\_[not .A])]}`. This finds A-labelled nodes such that there is a \-path 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*. However, the regular expression above cannot be expressed in this signature in less than three variables (Marx, 2005). □

A similar linguistic example is the $\mathcal{L}$ query `//NP{//VP[not \\PP]}`. This expression selects VPs that are dominated by NPs with no intervening PP. We can express this in $\mathcal{X}^+$ as `//NP(/_[not PP])*/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. The next lemma follows from Marx (2005).

Lemma 3. *The* immediate following sibling *and* immediate following *axes, and their converses, cannot be expressed in $\mathcal{X}$.*

Thus, $\mathcal{X}_{\text{->\{\}\$}}$ is strictly more expressive than $\mathcal{X}$, and so $\mathcal{L}$ is also strictly more expressive than $\mathcal{X}$.

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

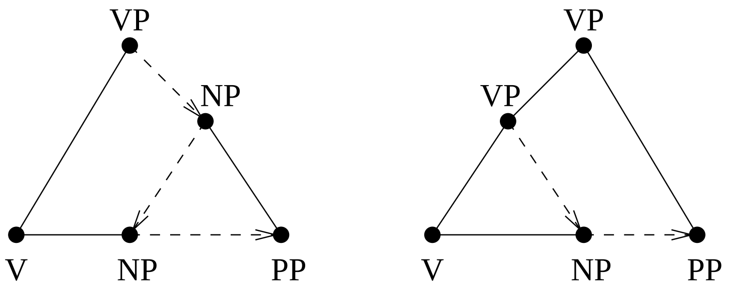

Figure 7: Scoping and immediate following. Dashed lines indicate the path of the query.

Lemma 4. *$\mathcal{L}$ is strictly more expressive than $\mathcal{X}_{\text{->}}$.*

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

In fact, the interaction of $\mathcal{L}$ operators results in queries that require other 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{[//^NP-->VP$]}` is equivalent to constraining the `NP` (`VP`) to be a leftmost (rightmost) descendant of the `S`. This requires us to be able 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 7. Consider a situation where we wish to find verb phrases (VP) containing a noun phrase (NP) immediately followed by a prepositional phrase (PP). That is `//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. If order to constrain `->` to be within a subtree, we need to phrase this constraint using other axes. (Note that the *following* axis, `-->`, can be alternatively defined as: `-->`$t[F] \equiv$ `\\*_==>_//*`$t[F]$.)

Now, the only chance that we may leave the scope 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 possible positions such an ancestor can take.

Although `->` has a similar form to its closure `-->` it requires further constraints that are inexpressible in $\mathcal{X}$. Specifically, we need to to be able to identify ancestors that are rightmost and descendants that are leftmost. This is much the same as subtree edge alignment. As in the previous example, these constraints cannot be expressed in $\mathcal{X}$. Importantly, this means that the only way we can represent the immediate following relation is with the primitive. Without some sort of memory device there is no way to force this primitive to stay within a scope. In a first-order formulas such a memory device would come in the form of extra variables.

Putting all this together gives a clear picture of the expressiveness is 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 indicate that $\mathcal{L}$ requires some of 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 ($\mathcal{X}^+$) is that the immediate following relation is now expressible: `->` $\equiv$ `([not(=> _)]\)* => (/[not(<= _)])*`

Since $\mathcal{X}$ is contained in $\mathcal{X}^+$, the definitions of edge alignment operators carry over from $\mathcal{X}$. Once again, the scoping operator requires more work. However, the ability to express this follows immediately from the first-order completeness of $\mathcal{X}^+$ queries (Marx, 2005). Consider now $\mathcal{X}^+$ with the scoping operator added to its syntax, $\mathcal{X}^+_{\{\}}$.

Lemma 5. *$\mathcal{X}^+$ is as expressive as $\mathcal{X}^+_{\{\}}$.*

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

Thus all the $\mathcal{L}$ operators are expressible in $\mathcal{X}^+$. Moreover, the first-order completeness of $\mathcal{X}^+$ means that the interactions between $\mathcal{L}$ operators can add no extra expressiveness. However, there is no Kleene star in $\mathcal{L}$ so the reverse case is clearly not true. This gives us the following theorem.

Theorem 6. *$\mathcal{L}$ is strictly contained in $\mathcal{X}^+$.*

That is, the expressiveness of LPath ($\mathcal{L}$) lies strictly between Core XPath ($\mathcal{X}$) and Conditional XPath ($\mathcal{X}^+$). 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. Conditional LPath

4.1. The Expressiveness of Conditional LPath

The obvious question now is whether $\mathcal{L}$ with conditional axis, $\mathcal{L}^+$, is any more expressive than $\mathcal{X}^+$. The main point of difference between the two languages is the status of the the *immediate following* (`->`) axis. This is elevated to rank of primitive axis in $\mathcal{L}$. Unfortunately, treating `->` as a primitive axis does not necessarily give it the same properties as the one-step axes of $\mathcal{X}^+$. Consider the relations R_i where $i \in \{$`=>`, `<=`, `/`, `\`$\}$. If $(x, y) \in R_i^*$ there is a unique i-path between x and y. This is not the case for the immediate following axis. The `->` is a many-to-many mapping and its converse, the *immediate preceding* relation `<-`, is as well. Consider the $\mathcal{L}^+$ expression `B(->A)+`. We might express this in as:

$$\mathrm{following}(x, y) \wedge B(x) \wedge A(y) \wedge \forall z(\mathrm{following}(x, z) \wedge \mathrm{following}(z, y) \rightarrow A(z))$$

The possibility of multiple `->`-paths between x and y makes this formula too strong a statement. The original $\mathcal{L}^+$ expression only requires the *existence* of an `->`-path between nodes x and y

where x has label B and the other nodes are labelled A. The formula above requires all `->`-paths to have this property.

However, we can derive a formula that is equivalent to the set defined by `B(->A)+` as follows. Let x and y be nodes such that $\text{following}(x, y)$, where x and y are labelled B and A respectively. Let v be the first common ancestor of x and y and denote the subtree rooted at v as T_v. We also need the following to hold. For each leaf l between x and y there is at least one node z labelled A on the each `\`-path from a l to v. This set of z nodes gives us the required `->`-path. A first-order formula that expresses is as follows:

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

We can easily let A and B represent location paths instead of labels in the the formula above. So this formula can easily be modified to deal with the conditional `->` axis in general. This means that all $\mathcal{L}^+$ expressions without scoping braces can be expressed first-order logic. As in Lemma 5, we can trivially add the scoping operator. $\mathcal{L}^+$ clearly contains $\mathcal{X}^+$ so we have the following equivalence:

Theorem 7. *$\mathcal{L}^+$ has the same expressiveness as $\mathcal{X}^+$. As a result, $\mathcal{L}^+$ path sets are first-order complete.*

In fact, we can find an equivalent $\mathcal{X}^+$ expression for the conditional immediate following axis using the fact that $\mathcal{X}^+$ is closed under intersection and complementation (Marx, 2005). First note that the formula $\text{imf}_{BA^+}(x, y)$, hence the $\mathcal{L}^+$ query `//B(->A)+`, is equivalent to the following:

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

We can write this using intersections and complements of $\mathcal{X}^+$ path well-formed formulas. Let

$$\phi(x, y) \equiv (?B/ancestor/(child?\neg A)^+/?leaf) \cap following.$$

Now we can write an expression equivalent to `//B(->A)+`

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

Along with the proof, Marx (2005, Theorem 2) provides a method for finding the complement of any $\mathcal{X}^+$ path set. Thus, we now have a concrete method for translating $\mathcal{L}^+$ expressions into $\mathcal{X}^+$.

4.2. Conditional LPath as Linguistic Tree Query Language

$\mathcal{L}^+$ is capable of expressing a large range of linguistic tree queries, including all the basic subtree matching queries identified in our requirements analysis (Lai and Bird, 2004).

The only other current linguistic treebank query language with this level of expressiveness is fsq (Kepser, 2003). However, fsq only allows boolean queries. Moreover, $\mathcal{L}^+$'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 path-based variable-free approach over the variables and predicates of classical first-order logic.

The major advantage of the classical approach of fsq is that variables can be used to identify specific nodes throughout a query. The scoping operator accounts for cases where there is a need to identify the root of a particular subtree, the scoping node, at every step in a path expression. However, although $\mathcal{L}^+$ is first-order complete, it is not always clear how a first-order formula can be converted into a (variable-free) $\mathcal{L}^+$ 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:

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

However, even with the *immediate following* axis and the scoping operator it is not very obvious how this can be expressed in $\mathcal{L}^+$. Note that the following query is incorrect,

```
//_[{//NP->VP} and not(//_{//NP->VP})].
```

This is because each `NP` (or `VP`) may refer to completely different nodes. We can express this query by using the $\mathcal{X}^+$ 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.

As an example of how sequential closures are expressed consider the following query: *Find words consisting of consonant-vowel-consonant sequences*.[1] Let words, consonants and vowels be represented by the labels `W`, `C`, and `V` respectively. We can express this query in $\mathcal{L}^+$ as follows: `//W{[/^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. Moreover, the scoping operator provides a convenient way of specifying subtree edge alignment. This allows us to specify completely what a node dominates.

We can express more hierarchical closures too, for example, to find `NP` nodes that conform to the simple grammar fragment, `NP` $\rightarrow$ `Adj NP`; `NP` $\rightarrow$ `N` as: `//NP[({/^Adj=>NP$})*/N]`

The addition of the *immediate following* and *immediate following sibling* axes completes the set of $\mathcal{X}$ axes for navigating trees. In $\mathcal{L}^+$, each axis has a corresponding one-step axis. The $\mathcal{L}^+$ axis set accounts for both hierarchical, sequential and sibling orderings on unranked ordered trees. As such, there do not appear to be any such (unconditional) relations lacking in the $\mathcal{L}^+$ axis set. Thus, $\mathcal{L}^+$ appears to have the complete set of axes necessary for linguistic tree query.

[1]This is slightly harder version of the query in Cassidy (2002).

5. Conclusion

LPath was proposed as a new query language which augmented the navigational axes of XPath with three additional tree operators. The analysis of LPath operators shows that they are more than just syntactic sugar. In fact, LPath takes up a new rung on the expressiveness hierarchy strictly between Core and Conditional XPath. Conditional LPath, LPath extended with the conditional axis, has the same expressiveness as Conditional XPath. This provides evidence that the closures required for linguistic query can be restricted to conditional paths, and supports the argument that first-order logic provides enough expressiveness for our linguistic tree query needs.

6. Acknowledgements

This research has been supported by an Australian Postgraduate Award (Lai) and by the US National Science Foundation project 0317826 *Querying Linguistic Databases* (Bird). We are grateful to Marcus Kracht and Maarten Marx for helpful discussions of this work. A substantially expanded version of this work may be found in the first author's masters thesis (Lai, 2005).

References

Bird, S., Chen, Y., Davidson, S. B., Lee, H., and Zheng, Y. (2006). Designing and evaluating an XPath dialect for linguistic queries. In *22nd International Conference on Data Engineering*.

Bird, S. and Liberman, M. (2001). A formal framework for linguistic annotation. *Speech Communication*, 33:23–60.

Cassidy, S. (2002). XQuery as an annotation query language: a use case analysis. In *Proceedings of LREC 2002, Las Palmas, Spain, May*.

Clark, J. and DeRose, S. (1999). *XML Path language (XPath)*. W3C. `http://www.w3.org/TR/xpath`.

Gottlob, G., Koch, C., and Pichler, R. (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. ACM.

Gottlob, G., Koch, C., and Schulz, K. U. (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. ACM.

Kepser, S. (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.

Lai, C. (2005). A formal framework for linguistic tree query. Master's thesis, Department of Computer Science and Software Engineering, University of Melbourne.

Lai, C. and Bird, S. (2004). Querying and updating treebanks: A critical survey and requirements analysis. In *Proceedings of the Australasian Language Technology Workshop*, pages 139–146. `http://eprints.unimelb.edu.au/archive/00000774/`.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (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`.

Marx, M. (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. Springer.

Marx, M. (2005). First order paths in ordered trees. In Eiter, T. and Libkin, L., 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.

Marx, M. and de Rijke, M. (2004). Semantic characterization of navigational XPath. In *Proceedings of TDM'04 Workshop on XML Databases and Information Retrieval*, Twente, The Netherlands.

Palm, A. (1999). Propositional tense logic for trees. In *Proceedings of the Sixth Meeting on Mathematics of Language: MOL6*, University of Central Florida, Orlando, Florida.