Querying Linguistic Trees

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
Telephone: +1-215-898-6046, Fax: +1-215-573-2091
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. These linguistic annotations are often syntactic or prosodic in nature, and have a hierarchical structure. Query languages are used to select particular structures of interest, or to project out large slices of a corpus for external analysis. Existing languages suffer from a variety of problems in the areas of expressiveness, efficiency, and naturalness for linguistic query. We describe the domain of linguistic trees and discuss the expressive requirements for a query language. Then we present a language that can express a wide range of queries over these trees, and show that the language is first-order complete over trees.

1. Introduction

Over the past decade, large databases of annotated text and speech – *linguistically annotated corpora* – have found increasing acceptance as primary sources of linguistic evidence. This development is adding new rigour to the empirical foundations of theoretical linguistics, which has previously relied on impressionistic grammaticality judgements as the primary source of data. However, acceptance of linguistic corpora has not been universal, partly because we still lack suitable tools for interrogating the data. As the data becomes richer, the problem only becomes more acute.

In response to this problem, a great variety of linguistic query languages have been proposed. Their primary application is for extracting information about linguistic structures from corpora. Most of these languages are designed for trees, and many have been applied to corpora such as the Penn Treebank (Marcus et al., 1994). Figure 1 gives an example of a linguistic tree. It is assumed to have originated in an external linguistic event which has been orthographically transcribed and syntactically annotated. Linguistic query languages navigate such trees in terms of their hierarchical and temporal structure. Despite considerable effort in developing and implementing these languages,

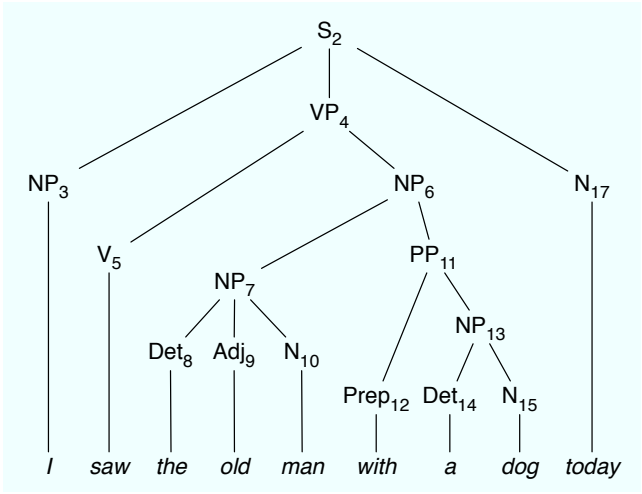


Figure 1. Tree Representation

relatively little is known about their formal expressiveness, or the computational resources required to process them as the size of the data grows.

In this paper we re-examine the requirements of linguistic tree query, discuss a language LPath that is designed to address them, and establish important formal properties of the language. The paper is organized as follows. Section 2 lays out the linguistic motivation for our approach, and presents expressive requirements that arise from examination of the literature on linguistic tree query. Section 3 presents the LPath language and Section 4 establishes its formal properties. The paper concludes with a discussion of the adequacy of LPath in the light of the requirements.

2. Linguistic Tree Query

This section reviews previous work on linguistic tree query, leading to a comprehensive list of requirements for any linguistic tree query language.

2.1. Linguistic Tree Query Languages

More than a dozen linguistic tree query languages have been developed,[1] many of which we have surveyed in an earlier paper (Lai and

[1] (Cassidy and Bird, 2000; Cassidy, 2002; König and Lezius, 2001; Heid et al., 2004; Hinrichs et al., 2000; Steiner and Kallmeyer, 2002; Randall, 2008)

Bird, 2004). In this section we discuss two exemplars which serve to highlight the key issues and lay the foundation for the ensuing discussion of requirements.

Finite structure query (fsq) is a tool for querying syntactic corpora that employs a language of first order logic (Kepser, 2003). Hierarchical and temporal constraints on trees are expressed using four binary relations: `(> x y)` "x is the mother of y"; `(>> x y)` "x dominates y (reflexive) "; `(>+ x y)` "x dominates y (non-reflexive)"; `(. x y)` "x immediately precedes y"; and `(.. x y)` "x precedes y". Precedence is defined temporally and is non-reflexive: a terminal node n_1 precedes a terminal node n_2 iff n_1 appears earlier in the sentence than n_2. More generally, a node n_1 precedes another node n_2 iff the rightmost terminal node under n_1 (or n_1 itself if it is a terminal) precedes the leftmost terminal node under n_2 (or n_2 itself if it is a terminal). Finally, a node n_1 *immediately* precedes a node n_2 iff there is no node n_3 such that n_1 precedes n_3 and n_3 precedes n_2. Note that immediate precedence is a many-to-many relationship.

The fsq language has additional binary relations to support orthographic and morphosyntactic labelling, and special non-tree edges linking nodes (e.g. for linking a pronoun to its antecedent full noun phrase). Complex formulas are built up in the usual way using boolean operators and quantifiers. The semantics of queries is just classical first-order model-theoretic semantics.

For example, query (1) finds trees containing a *VP* dominating an *NP*, where the right edge of the *VP* and *NP* are aligned (e.g. VP_4 and NP_{13} in Figure 1).

```
(E x (E y (& (cat y NP) (cat x VP) (>> x y) (!= x y)
        (A z (-> (. y z) (! (>> x z)))))))          (1)
```

The fsq language also permits axioms about trees. This allows us, for example, to require that trees be connected, rooted and acyclic, or to require that each node has exactly one syntactic category. Here is a query that ensures trees contain sentence nodes conforming to the context free grammar production `S -> NP VP`:

```
(A x (-> (cat x S) (E y (E z ( & (> x y) (> x z)
     (cat y NP) (cat z VP) (. y z)
     (A w (-> (> x w) (| (= x y) (= x z))))))))))  (2)
```

Such axioms may be useful during treebank development, or for linguists wanting to test if a treebank meets certain more stringent requirements, e.g. that no nodes exist with a branching degree of 1. However, in day-to-day practice, queries are existential: a linguist is

simply attempting to locate trees of interest, as in query (1). Even in the context of curation, a linguist does not provide axioms as in (2), but expresses existential queries – the negation of the axioms – in order to identify the exceptions. Since fsq is closed under negation, queries can be formulated to extract trees that constitute exceptions to such axioms. These trees can then be examined more closely by the treebank developers.

The value of this full first order approach is clear. However, we believe that the form of queries and results is not ideally suited to use by linguists. First, the syntax is cumbersome. For many queries, each introduction of a variable must be accompanied by quantifiers (although the addition of irreflexive operators has reduced the need for extra inequality statements). The prefix notation and LISP-like need for parentheses makes the language difficult to read and write correctly, even for experienced programmers.

A second concern is with the result of a query. By design, fsq returns the set of trees for which the query evaluates to true with respect to the model theoretic semantics. Yet linguists often need to identify particular nodes or subtrees having a specific property. Finding the matching tree is not enough: substantial extra effort may be required to identify the matching substructures when trees are large or queries are complex.

Next we turn to TGrep, the first special-purpose linguistic tree query language, now available in extended form as TGrep2 (Rohde, 2001). TGrep2 includes the operators provided by fsq but adds a sibling relation `$` and sibling versions of the precedence relations `$.` and `$..`, plus a plethora of other relations including: `>,` (first child), `>‘` (last child), `>:` (only child), transitive closures of all relations, e.g. `>>,` (first child of first child etc), inverses of all relations, e.g. `$,,` (preceding sibling), negations of all relations, e.g. `!<<` (not ancestor), and reflexive versions of all transitive relations, e.g. `<<=` (ancestor or equal). Notably, TGrep2 uses node labels where fsq has variables. For example, query (1) can be expressed very succinctly as follows:

```
VP >>‘ NP                                    (3)
```

TGrep2 queries can be chained: we can find trees containing a *VP* dominating both an *NP* and a *PP* (4), or trees containing a *VP* dominating an *NP* which in turn dominates a *PP* (5). We can also specify that a particular subtree should be returned instead of the whole tree, by adding an backquote before the node name, as shown in (6).

```
VP > NP > PP                                    (4)
VP > (NP > PP)                                  (5)
VP > 'NP > PP                                   (6)
```

TGrep2 includes variables that allow nodes to be referenced multiple times within a query. For example, the following query finds the first common ancestor of an *NP* and a *VP*, i.e. a node p having an arbitrary label dominating nodes n and v, such that p has no descendents dominating both n and v:

```
*=p << (NP=n .. (VP=v >> =p !>> (* << =n >> =p)))    (7)
```

Comparing TGrep2 and fsq, we see that TGrep2 is very concise. This conciseness is partly due to TGrep2's rich inventory of tree relations, and it would be a simple matter to extend fsq so that queries like (3) could be expressed directly. More important differences concern negation, quantification, and variables. TGrep2's negated relations are shorthand for negated expressions, thus the following TGrep2 and fsq expressions are equivalent:

```
A !< B                                                          (8)
A ![< B]                                                        (9)
(E x (& (cat x A) (! E y (& (cat y B) (< x y)))))              (10)
```

TGrep2 syntax does not permit negation outside an entire expression, nor does it allow explicit universal quantification of variables. Therefore the (implicit) outermost quantification is always existential, and axioms like (2) cannot be expressed. Variables in TGrep2 are not purely logical; they have global scope regardless of their level of nesting within a query. If a variable name occurs on both sides of a disjunction, constraints on the variable within earlier disjuncts are somehow propagated to later disjuncts. It is not clear how this works when the constraints on a shared variable involve more deeply nested shared variables. The expressiveness of TGrep2 is not known, and there is no explicit definition of the syntax and semantics, only software and a user manual.

Several themes arise from our discussion of these two linguistic tree query languages, and we believe they are relevant to the much larger set of languages we have examined. On the methodological level, the above two approaches couldn't be more different: fsq begins with a

well-understood formalism and instantiates it for this domain, while TGrep2 is an evolving software tool with no underlying formalism. The explicit variables and quantifiers of fsq give it more expressive power, but queries are more cumbersome and only return entire trees. In contrast, the path-like style of TGrep2 gives rise to concise and natural queries, and it effectively provides us with a partial set of requirements for practical linguistic tree query.

These observations lead us to frame the following question: can we retain the universality and formal rigour of the first order language along with a more convenient path-based syntax? Before addressing this question, we flesh out a detailed list of requirements on any general purpose linguistic tree query language.

2.2. Requirements for a linguistic tree query language

Based on our study of existing linguistic tree query languages, we believe any expressively adequate query language should meet the following requirements.

Hierarchy. In general, we want to be able to navigate from a node to its descendents, children, parent, and ancestors. For example, suppose we wish to find all sentences containing a given word, such as: *saw*. We can express this in terms of tree navigation: 'find S nodes having a terminal node *saw* as a descendent'. This condition is met by S_2 in Figure 1. As another example, we might want to identify any prepositional phrases contained inside a noun phrase: 'find *PP* nodes having an *NP* as a parent', i.e. PP_{11}.

Constituency. In general, we want to be able to navigate from a node to its (immediate) left and right siblings. For example, suppose we wish to locate noun phrases that have a prepositional phrase as a sibling: 'find *NP* nodes with an immediately following *PP* sibling node', i.e. NP_7. As another example, we might want to locate nouns that are qualified with a determiner: 'find N nodes with a preceding *Det* sibling node that may or may not be adjacent to the N node', i.e. N_{10}, N_{15}.

Temporal organisation. In general, we want to be able to navigate left and right to (adjacent) nodes in the interval structure defined by the tree. For example, suppose we wish to identify all words that immediately follow a given word, regardless of the syntactic structure built over those words. As another example, we might want to locate noun phrases having a verb immediately to their left: 'find *NP* nodes which appear immediately after a V node', i.e. NP_6, NP_7. Similarly,

we might want to find nouns that appear anywhere in the tree to the right of the first verb of a sentence: 'find N nodes that have a V node somewhere to their left', i.e. N_{10}, N_{15}, N_{17}.

Interactions. The hierarchical and temporal dimensions interact. In general, we want to be able to identify the leftmost (rightmost) child (descendents) of a node: 'find *NP* nodes that are rightmost within the scope of a *VP* ancestor node', i.e. NP_6, NP_{13}. As another example, we might want to restrict our earlier query concerning nouns and verbs to the scope of a given verb phrase: 'find N nodes within a *VP* phrase, to the right of its first V', i.e. N_{10}, N_{15}.

Boolean operations. In general, we want to be able to form complex queries out of simple queries using negation, disjunction, and conjunction. For example, suppose we wish to identify noun phrases that are not dominated by verb phrases: 'find all *NP* nodes whose parent is not a *VP*', i.e. NP_3, NP_7, NP_{13}. As another example, one might want to identify the phrasal constituents inside a verb phrase: 'find all *NP*, *PP* and *VP* nodes with a *VP* ancestor', i.e. NP_6, NP_7, PP_{11}, NP_{13}. We often want to place multiple constraints on a node: 'find all *NP* nodes having a *VP* parent and a *PP* child, i.e. NP_6.

Closures. In general, we want to be able to express closures of basic relations such as dominance, precedence and sibling precedence, including restrictions on the properties of nodes involved. For example, consider the trees produced in X-bar theory (Chomsky, 1981), in which the distance between the head of a phrase N (at terminal level) and its maximal projection N'' is unbounded. Navigating from the N node up to the N'' ancestor requires a closure over the parent relation which constrains the category to be N'. As another example, noun phrase chunking, a precursor to full parsing, involves processing a sequence of POS-tagged words in temporal order: 'find *Det* followed by an unbounded number of *Adj*s followed by one or more *N*s'.

Non-Navigational Requirements. In general, we want to be able to express constraints on node labels. For example, a part of speech tag *N.** would match any noun tag including `NN` (singular noun), `NNS` (plural noun), `NNP` (proper noun), `NNPS` (plural proper noun). Queries may need to access node attributes and node indexes. These are represented in the Penn Treebank using complex labels like `NP-SBJ-3`, `NP-PRD`, `PP-LOC`, and so forth. Finally, syntactic structures may not be trees at all, but more general dependency graphs, or multi-layer annotations including intersecting syntactic, prosodic and discourse

structures, contexts where query reduces to general graph matching. For the purposes of the current investigation we consider this last category of requirements to be out of the scope of this paper. However, it is clearly not outside the scope of the approach (LPath) we will present in the following sections.

2.3. Evaluating existing linguistic query languages

Examined in the light of these requirements, existing linguistic query languages vary greatly. The fsq language can express all of the above queries. For example, (11) is the translation of the above closure query (i.e. 'find *Det* followed by an unbounded number of *Adj*s followed by one or more *N*s'.).

```
(E x (E y ( & (cat x Det) (cat y N) (.. x y)
        & (A z (-> (& (.. x z) (.. z y)
        (! (E w (>> z w)))) (cat z Adj))))))    (11)
```

TGrep2 can express a subset of the above requirements: the outermost quantifier is required to be existential, and there are no closures other than the transitive operators built into the language. In the CorpusSearch language, boolean operators can only be applied to literals, not arbitrary subexpressions (Randall, 2008). More formal analysis of languages such as Tgrep2 and CorpusSearch is not possible when they are only defined in evolving software tools. The TIGERSearch language is well defined and like tgrep2 seems to reflect specific requirements of linguistic tree query. TIGERSearch includes many basic operators, e.g. `>3` to traverse to a child's child's child (and versions for arbitrary values and ranges `>n`, `>m,n`) (König and Lezius, 2001). It also includes left (right) corner operators `>@l`, `>@r` , navigating from a node to its earliest (latest) descendent. However it lacks anything corresponding to TGrep2's more general operators `>>,` (`>>‘`) to navigate to intervening initial (final) descendents. In TIGERSearch, all variables are existentially quantified. However, the language does not allow negation to scope over this implicit existential.[2] This is problematic because, as noted above, linguists need to be able to look for the absence of certain structures in a treebank. Looking across these and several other languages, it is unclear whether the specialised primitives are fundamental or just syntactic sugar, whether gaps in expressiveness are accidental or deliberate, and whether deliberate gaps are motivated by linguistic or computational factors.

[2] In fact, the language of TIGERSearch is the existential fragment of first order logic and so strictly less expressive than fsq.

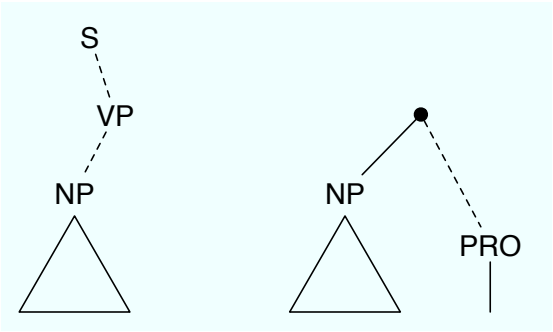
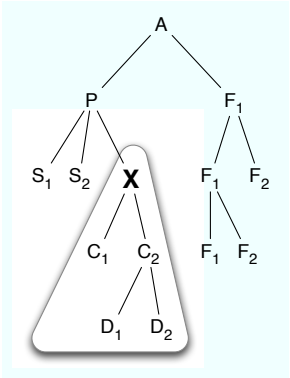


Figure 2. Paths in Trees: Identifying Subtrees and Antecedents

The problem of comparison is exacerbated by the fact that most implementations are tailored to the flat file representation used by a specific corpus, and cannot be compared directly. Comparing code is hampered by the fact that the code is not always available, and because query processing is intertwined with idiosyncratic indexing and storage. In general, these languages are not compiled into an existing general purpose language (such as SQL), which means that their relationship to such languages is not known. Moreover, standard indexing and optimisation techniques cannot be applied, and the implementations we have experimented with do not scale (Bird et al., 2006).

For these reasons, it is difficult to establish the formal expressiveness of existing linguistic tree query languages, or establish the asymptotic efficiency of their implementations. Instead, we return to the question of the convenience of the syntax. From our consideration of the actual queries used in the various languages mentioned above, we observe that descriptions of structure almost always involve paths (as also observed by Palm (1999)). Paths are routinely used to identify particular subtrees relative to the root and to describe binary relationships between tree nodes, as shown in Figure 2. Path languages cannot explicitly express cyclic queries (cf. (7)) without the addition of variables. However, the range of cyclic queries required for our task appears to be limited. As we will see in Section 4.1, cycles can be eliminated from positive queries. The next section presents a linguistic tree query language that allows a restricted set of cyclic queries to be easily represented.



relation	description	Y
`X\\Y`	ancestor	A,P
`X\Y`	parent	P
`X/Y`	child	C_*
`X//Y`	descendent	C_*, D_*
`X<=Y`	immed preceding sibling	S_2
`X<==Y`	preceding sibling	S_*
`X->Y`	immed following	F_1
`X-->Y`	following	F_2
`X/^Y`	leftmost child	C_1
`X//Y$`	rightmost descendent	C_2, D_2
`X//Z-->Y`	following a descendent	C_2, D_*, F_*
`X{//Z-->Y}`	following a descendent (scoped)	C_2, D_*
`X{//Z-->Y$}`	following a descendent (scoped, rightmost)	C_2, D_2

Figure 3. Required Tree Navigations and LPath Relations

3. LPath: A path language for linguistic tree query

XPath is a language for describing paths in trees, and is popular for the tree-structured document markup of the XML world (Clark and DeRose, 1999). It provides a well-understood starting point for investigation of modal-style languages for linguistic tree query. LPath and LPath$^+$ are linguistically motivated extensions to XPath (Bird et al., 2006; Lai, 2005). An interpreter converts LPath expressions into equivalent SQL expressions over annotation graphs stored in a relational database (Bird and Liberman, 2001; Bird et al., 2006). An open-source implementation is available as part of the Natural Language Toolkit (Bird et al., 2008), and a graphical interface is described by Bird and Lee (2007).

As with XPath, LPath permits navigation from a node labelled `X` to a child labelled `Y` with the expression `X/Y`. The irreflexive closure of this relation, to navigate from a node to its descendents, is `X//Y`. Here the similarity with XPath syntax ceases. Further navigations are summarised in Figure 3.

LPath provides three substantive extensions to XPath: the *immediate following* axis (and its converse), a scoping operator, and tree edge alignment. First, the *immediate following* axis, `->`, is the natural one-step version of the XPath *following* axis, `-->`. We can consider this axis as taking a step to constituents immediately right of the current node. These axes make it possible to refer to temporal context irrespective of higher-level syntactic structure, as already exemplified in the earlier discussion of fsq and TGrep2. Second, a *scoping* operator, denoted by

braces `{}`, constrains navigations to the subtree that is rooted at a given node. The query inside the scoping braces is evaluated locally on the subtree, and cannot escape to the outside context of the enclosing tree. For example, `/...P{...Q}` finds some node `Q` only if it occurs inside the subtree rooted at `P`. Finally, left and right tree edge alignment, denoted by `^` and `$` respectively, combine with the scoping operator and permit queries to constrain a node to be leftmost (rightmost) within a constituent (cf TGrep `>>,` and `>>‘`). The alignment operators are just syntactic sugar:

$$\texttt{\^{}A} \equiv \texttt{A[not <-- _]} \tag{12}$$

$$\texttt{A\$} \equiv \texttt{A[not --> _]} \tag{13}$$

LPath$^+$ extends LPath by adding atomic closures to the language, e.g. `(/N)*` matches arbitrary length paths to descendents via nodes labelled *N*. Note, closures must be applied to a single axis although this may include filter expressions, e.g. `(/NP[not /S])*/S` is licit but `(/NP/S)*/S` is not.

We will further illustrate the features of LPath and LPath$^+$ with the help of a series of examples. Query (1) found an *NP* that is right-aligned with an enclosing *VP*, and we express this in LPath as follows: `//VP{//NP$}`. Query (2) was an axiom to require that all *S* nodes are licensed by the phrase structure rule `S -> NP VP`. The LPath query to find any exceptions to this rule is expressed as follows:

```
//S{[not /^NP => VP$]}
```
(14)

Query (14) combines scoping and alignment. All nodes reachable inside the braces in `//S{...}` are descendents of *S*. By definition, `VP$` is equivalent to `VP[not --> _]`, meaning that the *VP* descendent of *S* has no succeeding nodes under *S*. Thus the right edges of *VP* and *S* are aligned; equivalently, we can say they have the same rightmost terminal node.

Observe that LPath queries combining scoping and alignment are not generally expressible in XPath. The query `//VP{//NP$}` involves a path of arbitrary length through rightmost child nodes. To express this in XPath we need to state that every node on the `/`-path between the *VP* and *NP* has no right sibling. As we will see in Section 4, such constraints require a special ‘conditional’ axis which is inexpressible in XPath. A similar argument applies to the *immediate following* axis. First, observe that the *following* axis, `-->`, can be defined as:

$$\texttt{-->}t[F] \equiv \texttt{\textbackslash\textbackslash\_==>\_//}t[F] \tag{15}$$

If we were to define `->` in the same way we would need to be able to traverse from a node to an ancestor subject to the constraint that each

node on the way up has no right sibling, and to traverse from a node to a descendent subject to the constraint that each node on the way down has no left sibling. As before, such constraints cannot be expressed in XPath.

Using LPath$^+$ it is possible to conveniently express a useful range of closures which are either inexpressible or overly cumbersome in other linguistic tree query languages. For example, consider sequential closures in the 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 LPath$^+$ 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, while the scoping and alignment operators allow us to fully specify the contents of the (lexical) constituent selected as the scoping node.

More hierarchical closures can also be expressed. For example, to find *NP* nodes that conform to the grammar fragment, `NP` → `Adj NP`; `NP` → `N`, we can write:

```
//NP[(/NP$[<=^Adj])*/N]                    (16)
```

LPath$^+$ can express the first common ancestor query (cf TGrep2 query (7)) as follows:

```
NP (\_[not //VP])* \_[//VP]              (17)
```

Regular expressions over paths, such as `(/NP/S)*`, cannot be expressed in LPath$^+$. Such patterns of alternating non-terminals arise from mutually-recursive productions in the grammar that licenses the treebank. So long as the linguist is not trying to validate the treebank against the grammar, queries along these alternating paths can be adequately approximated using an atomic closure involving disjunction, e.g. `(/_[.NP or .S])*`.

In summary, LPath$^+$ is capable of expressing a large range of linguistic tree queries, including all the basic subtree matching queries identified in our discussion of requirements in Section 2.2. The LPath$^+$ axis set accounts for hierarchical, sequential and sibling orderings. Thanks to the inclusion of `->` and `=>`, all of LPath$^+$'s unbounded axes have corresponding one-step versions. As such, there do not appear to be any (unconditional) relations missing from the LPath$^+$ axis set, and LPath$^+$ appears to have the complete set of primitive axes necessary for linguistic tree query. In the next section we investigate the formal expressiveness of LPath and LPath$^+$.

4. Formal Results Concerning LPath and LPath$^+$

As already mentioned, one of our goals is to provide an *efficient* linguistic tree query tool. LPath and LPath$^+$ seem to meet our linguistic requirements, but we also need to establish the formal expressiveness of these languages in order to determine the type of technology needed to implement them. As we will see, our path-based approach to linguistic tree query can be implemented using efficient and well-understood technology, namely SQL and relational databases. This result falls out of the characterisation of LPath and LPath$^+$ with respect to an existing family of languages.

Marx (2005a) presents a family of XPath languages that extend the navigational functionality of XPath 1.0. *Core XPath*, or simply $\mathcal{X}$, is defined as the XPath 1.0 language stripped of non-navigational components such as attributes and namespaces (Gottlob et al., 2003).[3] *Conditional XPath*, or $\mathcal{X}^{c*}$, extends $\mathcal{X}$ primarily by adding a conditional axis.[4] This axis describes paths in which every node meets some condition, expressed as a *filter expression*. In linguistic queries, this condition tends to appear as a negation. For example, suppose we want to find the least deeply embedded relative clause inside an IP. For this, we must find CPs dominated by an IP such that there is no other IP on the path between those two nodes. [5]

```
IP(/_[not IP])*/CP
```
(18)

Marx has shown that $\mathcal{X}^{c*}$ is a first-order complete language over the signature, FO$^{\text{tree}}$:

$$\tau = \{\text{descendent}, \text{following-sibling}\} \tag{19}$$

So, every $\mathcal{X}^{c*}$ expression is equivalent to a FO$^{\text{tree}}$ formula, $\varphi(x, y)$, with exactly two free variables, and vice versa (Marx, 2005b). This makes it expressively equivalent to the linguistic query language fsq discussed previously.

We would like to determine where LPath lies on the expressiveness hierarchy of XPath languages. In particular, we would like to know

[3] Note, XPath 1.0 does not actually include a *immediately following sibling* axis although it does includes its unconditional closure, *following sibling*. However this axis is usually include in logical extensions as in Marx (2005a).

[4] cf. Palm's propositional tense logic for trees (Palm, 1999).

[5] XPath filter expressions describe conditions on a node in a path. However they do not affect the evaluation of subsequent parts of the path. In 18 the expression between square brackets is structural filter on the (unnamed) nodes within the closure.

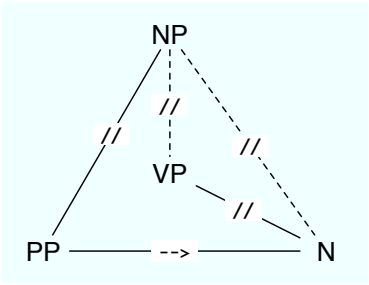


Figure 4. Scoping induced cycles: `NP{//PP-->N\\VP}`

if the LPath operators offer any extra 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. In the rest of this section we establish the expressiveness of LPath relative to $\mathcal{X}$ and $\mathcal{X}^{c*}$.

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). $\mathcal{X}_{\texttt{->\{\}\$}}$ 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^{+}. We also introduce the notation `axis+` – the non-reflexive transitive closure of an axis – as syntactic sugar for `/axis::_/(axis)*`.

4.1. LPath Operators and Core XPath

This section demonstrates that LPath is strictly more expressive than Core XPath ($\mathcal{X}$). To begin, we have already seen that left and right edge alignment can be expressed in $\mathcal{X}$, by virtue of their definitions (12), (13). However, the scoping operator is not expressible in $\mathcal{X}$. To see this, consider the query `NP{//PP-->N\\VP}` illustrated in Figure 4, where the scoping constraint corresponds to the dashed edges in that figure. The scoping operator asserts a dominance relation between the scoping node and those appearing within the scoping braces. The difficulty implementing this in path-based *variable-free* languages, like $\mathcal{X}$, is that there is no memory of previous steps in a path. It is not possible to force a path to loop back to a particular node (i.e. we cannot decorate our

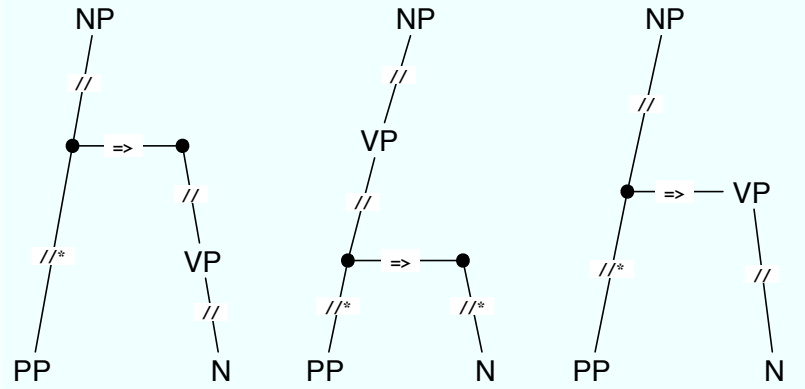


Figure 5. Acyclic Version of `NP{//PP-->N\\VP}`, equivalent to a 3-way disjunction

nodes with indexes to write $\texttt{NP}_i\texttt{//PP-->N\\VP\\NP}_i$, to ensure that the first and last *NP*s are the same). 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 this for the positive fragment of $\mathcal{X}$ was presented by Gottlob et al. (2004). Positive $\mathcal{X}$ is the set of $\mathcal{X}$ expressions that do not include negation in filter expressions. This transformation is demonstrated for a particular query in Figure 5, and the corresponding disjunctive $\mathcal{X}$ expression is shown below:

```
//N[\VP\\_<=_\\NP[//*PP]
or \\*_<=_[//*PP]\\VP\\NP
  or \\VP<=_[//*PP]\\NP]]
```
(20)

At this point it is instructive to note how this can be used to express other queries. For example, this sort of decycling can also be used to express least common ancestor type queries like (7). We need to reorient the query as a path. After doing so we can express the least common ancestor of a *VP* followed by an *NP* as follows:

```
//_[/_[(VP or //VP) and =>_ [(NP or //NP)]]]
```
(21)

This query uses the fact that a node must be the least common ancestor of a *VP* followed by an *NP* if the *VP* and *NP* are descendent of two different children of that node (in the specified order).

However, this technique does not extend to $\mathcal{X}$ expressions with negation. Besides overt negation, $\mathcal{L}$ contains implicit negation in its edge alignment operators. Thus, we cannot use this algorithm to decycle all $\mathcal{L}$ queries and reduce them to $\mathcal{X}$ queries. Instead, the extra expressiveness allowed by the scoping operator is established in the following lemma.

LEMMA 1. $\mathcal{X} \subsetneq \mathcal{X}_{\{\}}$

Proof. Consider the $\mathcal{X}_{\{\}}$ expression in (22).

```
//B/A{//A[not (\\_[not .A])]}
```
(22)

This finds A-labelled nodes such that there is a \-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 are expressible in first order logic over trees, extended with *child* and *immediate following sibling* relations, using at most two variables. The regular expression above cannot be expressed in this signature using fewer than three variables (Marx, 2005b), and so it immediately follows that $\mathcal{X} \neq \mathcal{X}_{\{\}}$.

The $\mathcal{X}_{\{\}}$ expression above has a simple linguistic version that we have seen before: find IPs dominating a CP with no intervening IP. This shows that the scoping operator allows us to express some conditional axis expressions on hierarchical paths. Since $\mathcal{X}_{\{\}}$ is strictly contained in $\mathcal{L}$, we can now state the following relation between $\mathcal{X}$ and $\mathcal{L}$.

COROLLARY 2. $\mathcal{X} \subsetneq \mathcal{L}$. [6]

At this point we want to know if the extra expressiveness provided by the scoping operator can be reduced to the other operators introduced in $\mathcal{L}$. This is clearly not the case. Consider $\mathcal{X}_{\text{-}\!>}$ which is equivalent to $\mathcal{L}$ without the scoping operator. The additional axes of $\mathcal{L}$ express sequential relations and so do not give $\mathcal{X}_{\text{-}\!>}$ any more ability to express $\mathcal{L}$ queries that only involve hierarchical relationships. Considering (22) and Figure 6, we see that the additional sequential relations of $\mathcal{X}_{\text{-}\!>}$ are powerless to express (22). Thus, the scoping operator cannot be expressed in $\mathcal{X}_{\text{-}\!>}$.

It is clear that the scoping and the *immediate following* axes are more than syntactic sugar in the context of $\mathcal{X}$. In fact, the interaction between all three $\mathcal{L}$ operators as well as negation admit some of the expressiveness of the conditional axes of $\mathcal{X}^{c*}$. The next section looks

[6] The other additions of $\mathcal{L}$ to $\mathcal{X}$ are the one-step horizontal axes. It follows directly from Marx (2005b) that `=>` cannot be derived from `==>` in $\mathcal{X}$. However, we are mostly interested in the effect of the other operators.

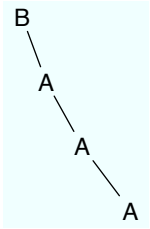


Figure 6. A tree that matches query (22) but has no sequential dimension.

at the affect of these operators in the setting of Conditional XPath. This allows us to find an upper bound on the expressiveness of $\mathcal{L}$. 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.

4.2. LPath operators and Conditional XPath

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

$$\texttt{->} \equiv \texttt{([not(=> _)]\)*} \texttt{ => } \texttt{(/[not(<= _)])*} \tag{23}$$

Thus, the immediate following relation is just syntactic sugar in $\mathcal{X}^{c*}$. Edge alignment operators carry straight over from $\mathcal{X}$ which is strictly contained in $\mathcal{X}^{c*}$. The derivability of the scoping operator follows immediately from the first-order completeness of $\mathcal{X}^{c*}$ (Marx, 2005b). To see this more clearly, consider now $\mathcal{X}^{c*}$ with the scoping operator added to its syntax, $\mathcal{X}^{c*}_{\{\}}$.

LEMMA 3. $\mathcal{X}^{c*} = \mathcal{X}^{c*}_{\{\}}$.

Proof. As Marx (2005b) has shown, we can convert any $\mathcal{X}^{c*}$ formula or $\mathcal{X}^{c*}_{\{\}}$ expression into a first-order formula $\phi(x, y)$. To represent subtree scoping we just need to assert a dominance relation between the scoping node and any node that would fall between the scoping braces. That is, if z is the variable representing the scoping node and $w_0, \ldots, w_k$ are variables representing nodes in the scoped location path, we conjoin the clause descendent(z, w_i) to the $\mathcal{X}^{c*}$ translation. This does not change the number of free variables so this has an equivalent $\mathcal{X}^{c*}$ expression.

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

in $\mathcal{X}^{c*}$ add no more expressiveness. We can now also see that $\mathcal{L}$ is strictly less expressive than $\mathcal{X}^{c*}$. The closures expressible in $\mathcal{L}$ are due to the scoping operator and only operate on the hierarchical structure. So, the horizontal conditional closures available in $\mathcal{X}^{c*}$ are not available in $\mathcal{L}$. We can show this in a few steps, as follows.

LEMMA 4. *The filter expressions of $\mathcal{X}_{\text{->}}$ are definable by first-order formulae $\varphi(x)$ in one free variable and at most two variables in signature $\tau_l = \{$/, //, ->, ==>, $P_i\}$ where P_i is a countable set of unary predicates.*

Proof. (Sketch) It is easy to translate $\mathcal{X}_{\text{->}}$ 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).

THEOREM 5. $\mathcal{L} \subsetneq \mathcal{X}^{c*}$

Proof. Consider the following two formulae.

$$\begin{aligned}\psi \quad &\equiv \text{following}(x,y) \wedge B(x) \wedge A(y) \\ &\wedge \forall z((\text{following}(x,z) \wedge \text{following}(z,y) \wedge \text{leaf}(z) \\ &\rightarrow \exists w((z = w \wedge A(w)) \\ &\vee(\text{ancestor}(z,w) \wedge A(w) \wedge \neg\, \text{ancestor}(w,x))))). \qquad (24)\\ \varphi \quad &\equiv \exists v(root(v) \wedge \forall u(u \neq v \rightarrow (\text{child}(u,v) \wedge \text{leaf}(u)))). \qquad (25)\end{aligned}$$

That is, ψ picks out BA^+ paths on the -> axis, while φ selects trees of depth exactly two. Now, suppose we wish to construct a query that picks out BA^+ paths on the -> axis in exactly trees of depth two. This can be expressed as follows:

$$\text{imf}^2_{BA^+}(x,y) \equiv \psi \wedge \varphi. \qquad (26)$$

Recall that $\mathcal{X}^{c*}$ is first-order complete so the formula above has an equivalent $\mathcal{X}^{c*}$ expression. However, this is outside of $\mathcal{X}_{\text{->}}$ due to Lemma 4. Now we can show that we cannot achieve the extra expressiveness with the scoping operator.

The only non-trivial subtree that the scoping operator can select in this sort of tree is the whole tree. We have also seen that the scoping operator amounts to adding descendent relationships between the scoping node and the nodes referenced between scoping braces. However, any extra descendent relationships are redundant in this situation because of the depth constraint (e.g. φ).

So, any correct query equivalent to $\text{imf}^2_{BA^+}(x,y)$ is equivalent to a query without the scoping operator. That is, if it is expressible in $\mathcal{L}$ it

must also be expressible in $\mathcal{X}_{\text{->}}$. However, we still need three variables to express this query. It now follows from Lemma 4 that this cannot be expressed in $\mathcal{L}$. This proves the theorem. [7]

We conclude that 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.

4.3. The Expressiveness of LPath$^+$

We have just seen that LPath ($\mathcal{L}$) is less expressive than Conditional XPath ($\mathcal{X}^{c*}$). In this section we discuss Conditional LPath, $\mathcal{L}^{c*}$, and its relationship to $\mathcal{X}^{c*}$. Since we are simply adding the LPath operators to $\mathcal{X}^{c*}$, by definition, $\mathcal{L}^{c*} \supseteq \mathcal{X}^{c*}$. Here, we consider whether $\mathcal{L}^{c*} \subsetneq \mathcal{X}^{c*}$ or $\mathcal{L}^{c*} = \mathcal{X}^{c*}$.

The proof of Theorem 5 shows how the conditional *immediate following* axis can be expressed in first-order logic over the signature of $\mathcal{X}^{c*}$. The formula ψ (24), from the previous section, can easily be modified to express a conditional `->` axis in general. This is the only new primitive axis in $\mathcal{L}^{c*}$. Accordingly, all $\mathcal{L}^{c*}$ expressions without scoping braces can be expressed in first-order logic. As we saw in Lemma 3, we can trivially include the scoping operator. Thus we have the following equivalence:

THEOREM 6. *$\mathcal{L}^{c*} = \mathcal{X}^{c*}$; consequently $\mathcal{L}^{c*}$ is expressively complete for first-order definable paths.*

That is, for every FO^{tree} formula $\phi(x, y)$ (cf. Section 4) 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, 2005b) (Theorem 2). Using Marx's notation we can write an expression equivalent to `//B(->A)+` as follows:

$$(?B/\,\text{following}?A) \cap \overline{\phi/\,\text{following}}$$

where

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

[7] In fact, the example in the proof above basically allows to prove the theorem using the more general result in Tiede (2008) (Proposition 20). The proof of that proposition itself calls on Schlingloff (1992) and Kamp (1968).

This gives us the upper bound on the expressiveness of this language. The hierarchy of expressiveness for LPath and LPath$^+$ is as follows.

$$\text{Core XPath} \subsetneq \text{LPath} \subsetneq \text{LPath}^+ \equiv \text{Conditional XPath} \tag{27}$$

Along with the proof of the $\mathcal{X}^{c*}$ closure under complementation, Marx (2005b) 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 benefit of understanding this translation, and understanding the LPath/XPath expressiveness hierarchy in general, is that it provides a clear range of options for efficient implementation (c.f. Afanasiev (2003)). The modal basis of $\mathcal{X}^{c*}$ lends evaluation tractability. Thanks to a result by 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.

4.4. LPath$^+$ and Higher-Order Logics

The question remains whether first-order expressiveness is enough to describe linguistic structure. In particular, counting and full transitive closures over paths are not definable in our first-order language. They are, however, expressible in higher-order logics such as monadic second-order logic (MSO). MSO, in particular, has been extensively studied in Rogers (1994) in terms of model theoretic syntax. In that dissertation, Rogers demonstrates how a substantial set of principles of Government and Binding theory (GB) for English can be expressed in MSO.

Higher-order logics can 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). However, recently Maryns and Kepser (2008) have implemented MSO as a tree query language named MonaSearch.[8] In terms of syntax and semantics, MonaSearch can be seen as a direct extension of fsq (c.f. Section 2). Queries are converted into tree automata which are then run over a corpus using the MONA tree automaton toolkit (Henriksen et al., 1995). Unlike fsq, the processing time of logically equivalent queries is seen to be independent of the actual query formulation and query evaluation is linear in the size of the treebank.

Closures are also available in Propositional Dynamic Logic (PDL) (Harel et al., 2002) which also has an equivalent in the XPath family, *Regular XPath*. That is, XPath with full regular expressions on paths. In fact, there is a clear correspondence between the XPath family we

[8] `http://tcl.sfs.uni-tuebingen.de/MonaSearch/`

have considered in this paper and modal logics developed for model theoretic syntax (Blackburn et al., 1996; Palm, 1999; Kracht, 1997; Tiede, 2008). Various arguments have been made for the adequacy of different levels of expressiveness required and what more expressiveness buys. PDL is strictly less expressive than MSO (Kracht, 1997; Palm, 1999) and more expressive than FOL.

However, there are linguistic arguments that expressiveness beyond the first-order realm is unnecessary. For example, the ability to perform counting type queries is possible in MSO and PDL. That is, queries of the type `(->Adj->Adj->Adj)+` which looks for linear chunks in which the number of `Adj`s is a multiple of three. However, Berwick and Weinberg (1984) have argued against this sort of counting in natural language formalisms. Similarly, Hoeksema and Janda (1988) report that morphological infixation does not need to count syllables or feet, but refers to constituent boundaries with no need for counting past one. That is, we need to be able to distinguish the beginning (resp. end) of a constituent and the following (resp. preceding) constituent, but we do not need to be able to count higher temporal distances than this. In fact, the only query cited in Maryns and Kepser (2008) that cannot be handled by first-order logic is one that looks for S-labelled nodes with an even number of descendents. This suggest that the expressiveness offered by MSO and PDL is too much.

However, it is well known that there are natural language structures that are not expressible even in MSO. For example, the cross-serial dependencies which occur in Swiss German require context-sensitive structures (Shieber, 1985). Tiede and Kepser (2006) show that first order deterministic transitive closure logic (FO(DTC)) strictly extends MSO. As the name suggest, this logic simply adds the deterministic transitive closure operator to FO.[9] This proves expressive enough to capture the cross-serial dependencies mentioned above. This result is especially interesting given the discussion of closure requirements above. However, this logic is undecidable on finite ordered trees.[10] For the querying standpoint, Mönnich et al. (2001) have shown how such structures can be queried by 'lifting' the treebank grammar and the MSO query into an algebra where mildly context-sensitive structures can be coded. A filter grammar is then applied to obtain the query result.

It is clear that we need expressiveness up to FOL and in certain cases we need expressiveness beyond MSO. However, it is not clear

[9] That is, it deals with relations that are functions

[10] Also Kepser (2006) considers, from the model-theoretic syntax standpoint, the logic that results from adding only transitive closure over *binary* relations. Unlike FO(DTC), this logic is known to be less expressive than MSO.

that we need the intermediate expressiveness of PDL. The a gap in the expressiveness requirement certainly warrants further attention. Moreover, it would be interesting to see whether projection techniques could be used to bridge this gap from a path-based/modal stand point. Further development and refinement of logics in the model-theoretic syntax program will certainly shed more light onto the querying problem. However, given the linguistic arguments outlined above, it does not seem necessary or worthwhile to sacrifice the efficiency of $\mathcal{L}^{c*}$ for more expressiveness at this point. $\mathcal{L}^{c*}$ appears to have the right level of expressiveness for general purpose 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. As previously noted, the only other current linguistic treebank query language with this level of expressiveness is fsq (Kepser, 2003). Section 2 discussed reasons why the path-based approach we have taken here might be preferable. In particular, our example queries highlighted the comparative succinctness of path queries.

5. Conclusion

In recent years, over a dozen linguistic query languages have been developed. As shown in our earlier survey (Lai and Bird, 2004), these languages have many common features, but differ greatly 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 additional expressiveness. Similarly, the computational cost of any given syntactic feature is unknown.

LPath was proposed as a new linguistic query language which augmented the navigational axes of XPath with three additional tree operators. LPath is unique among linguistic query languages in being fully path-based, a characteristic which appears to be ideal for linguistically-motivated tree navigation. Moreover, LPath is unique among linguistic tree query languages 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 (Bird et al., 2006).

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 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 against large treebanks.

As we observed at the outset, an obstacle to the widespread adoption of linguistic corpora has been the lack of suitable tools for accessing interrogating the data. As the data has become richer this problem has only become more acute. Although we have addressed this problem, there are further obstacles in the area of linguistic adequacy. Does the corpus capture the linguistic phenomena being investigated? Do human annotators make reliable judgements about the correct way to mark up a particular linguistic construction? Still, now that we have a solution to the problem of access, solutions to the problems of coverage and quality follow directly. Armed with an effective way to query a corpus of linguistic trees, it becomes practical to check the accuracy of annotations and the suitability of the corpus for a particular study.

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 Yi Chen, Susan Davidson, Stephan Kepser, Marcus Kracht, Haejoong Lee, Maarten Marx, Uwe Mönnich, Balder ten Cate, Yifeng Zheng, and the anonymous reviewers for helpful feedback on the work reported here.

References

Afanasiev, L.: 2003, 'XML Query Evaluation via CTL Model Checking'. Master's thesis, ILLC Scientific Publications, MoL-2003-07.

Alechina, N. and N. Immerman: 2000, 'Reachability Logic: An Efficient Fragment of Transitive Closure Logic'. *Logic Journal of the IGPL* **8**(3), 325–337.

Berwick, R. C. and A. S. Weinberg: 1984, *The Grammatical Basis of Linguistic Performance : Language Use and Acquisition*, Vol. 11 of *Current studies in linguistics*. Cambridge, Mass: MIT Press.

Bird, S., Y. Chen, S. Davidson, H. Lee, and Y. Zheng: 2006, 'Designing and Evaluating an XPath Dialect for Linguistic Queries'. In: *22nd International Conference on Data Engineering (ICDE)*. pp. 52–61.

Bird, S., E. Klein, and E. Loper: 2008, 'Natural Language Processing in Python'. http://nltk.org/book.html.

Bird, S. and H. Lee: 2007, 'Graphical query for linguistic treebanks'. In: *10th Conference of the Pacific Association for Computational Linguistics*. pp. 22–30.

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

Blackburn, P., M. de Rijke, and Y. Venema: 2001, *Modal logic*. New York, NY, USA: Cambridge University Press.

Blackburn, P., W. Meyer-Viol, and M. de Rijke: 1996, 'A Proof System for Finite Trees'. In: H. K. Büning (ed.): *Computer Science Logic*, Vol. 1092 of *Lecturn Notes in Computer Science*. Springer, pp. 86–105.

Cassidy, S.: 2002, 'XQuery as an Annotation Query Language: a Use Case Analysis'. In: *Proceedings of LREC 2002, Las Palmas, Spain, May*.

Cassidy, S. and S. Bird: 2000, 'Querying Databases of Annotated Speech'. In: *Database Technologies: Proceedings of the Eleventh Australasian Database Conference*. pp. 12–20.

Chomsky, N.: 1981, *Lectures on Government and Binding*. Dordecht: Foris.

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

Gottlob, G., C. Koch, and R. 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*. San Diego, CA, USA, pp. 179–190.

Gottlob, G., C. Koch, and K. U. Schulz: 2004, 'Conjunctive Queries over Trees.'. In: *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database System*. Paris, France, pp. 189–200.

Harel, D., D. Kozen, and J. Tiuryn: 2002, 'Dynamic Logic'. In: D. Gabbay and F.Guenthner (eds.): *Handbook of Philosophical Logic, Vol 4., 2nd Edition*. Dordrecht: Kluwer Academic Publishers, pp. 99–217.

Heid, U., H. Voormann, J.-T. Milde, U. Gut, K. Erk, and S. Pado: 2004, 'Querying both time-aligned and hierarchical corpora with NXT Search'. In: *Fourth Language Resources and Evaluation Conference, Lisbon, Portugal*.

Henriksen, J., J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm: 1995, 'Mona: Monadic Second-order logic in practice'. In: *Tools and Algorithms for the Construction and Analysis of Systems, First International Workshop, TACAS '95, LNCS 1019*.

Hinrichs, E. W., J. Bartels, Y. Kawata, and V. Kordoni: 2000, 'The VERBMOBIL Treebanks'. In: *KONVENS 2000 Sprachkommunikation, ITG-Fachbericht 161*. pp. 107–112.

Hoeksema, J. and R. D. Janda: 1988, 'Implications of Process-Morphology for Categorial Grammar'. In: R. T. Oehrle, E. Bach, and D. Wheeler (eds.): *Categorial Grammars and Natural Language Structures*. Dordrecht: D. Reidel.

Kamp, J.: 1968, 'Tense Logic and the Theory of Order'. Ph.D. thesis, University of California, Los Angeles.

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*. pp. 179–186.

Kepser, S.: 2006, 'Properties of Binary Transitive Closure Logic over Trees'. In: P. Monachesi, G. Penn, G. Satta, , and S. Wintner (eds.): *Formal Grammar 2006*. CSLI Publications, pp. 77–89.

König, E. and W. Lezius: 2001, 'The TIGER language - a description language for syntax graphs. Part 1: User's guidelines'. Technical report, University of Stuttgart, Stuttgart, Germany.

Kracht, M.: 1997, *Inessential Features*, Vol. 1328 of *Lecture Notes in Artificial Intelligence*, pp. 43–62. Springer.

Lai, C.: 2005, 'A Formal Framework for Linguistic Tree Query'. Master's thesis, Department of Computer Science and Software Engineering, University of Melbourne, Australia.

Lai, C. and S. Bird: 2004, 'Querying and Updating Treebanks: A Critical Survey and Requirements Analysis'. In: *Proceedings of the Australasian Language Technology Workshop*. pp. 139–146.

Libkin, L.: 1998, *Elements of Finite Model Theory*. Springer-Verlag.

Marcus, M. P., B. Santorini, and M. A. Marcinkiewicz: 1994, 'Building a Large Annotated Corpus of English: The Penn Treebank'. *Computational Linguistics* **19**(2), 313–330.

Marx, M.: 2005a, 'Conditional XPath'. *ACM Trans. Database Syst.* **30**(4), 929–959.

Marx, M.: 2005b, 'First Order Paths in Ordered Trees.'. In: T. Eiter and L. Libkin (eds.): *Database Theory - ICDT 2005, 10th International Conference, Edinburgh, UK, January 5-7, 2005, Proceedings*, Vol. 3363 of *Lecture Notes in Computer Science*. pp. 114–128.

Marx, M. and M. de Rijke: 2004, 'Semantic Characterization of Navigational XPath'. In: *Proceedings of TDM'04 Workshop on XML Databases and Information retrieval*. Twente, The Netherlands.

Maryns, H. and S. Kepser: 2008, 'MonaSearch–A Tool for Querying Linguistic Treebanks'. `http://tcl.sfs.uni-tuebingen.de/MonaSearch/`.

Mönnich, U., F. Morawietz, and S. Kepser: 2001, 'A Regular Query for Context-Sensitive Relations'. In: *IRCS Workshop Linguistic Databases 2001*. pp. 187–195.

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

Randall, B.: 2008, 'CorpusSearch 2 Users Guide'. `http://corpussearch.sourceforge.net/CS-manual/Contents.html`.

Rogers, J.: 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.

Rohde, D.: 2001, 'TGrep2 User Manual'. http://tedlab.mit.edu/ dr/Tgrep2/tgrep2.pdf.

Schlingloff, B.: 1992, 'On the Expressive Power of Modal Logics on Trees'. In: *Proceedings of the Second International Symposium on Logical Foundations of Computer Science, Springer LNCS 620*. pp. 441–451.

Shieber, S.: 1985, 'Evidence against the context-freeness of natural language'. *Linguistics and Philosophy* **8**(3), 333–343.

Steiner, I. and L. Kallmeyer: 2002, 'VIQTORYA – A Visual Query Tool for Syntactically Annotated Corpora'. In: *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC 2002)*. pp. 1704–1711.

Tiede, H.: 2008, 'Inessential Features, Ineliminable Features, and Modal Logics for Model Theoretic Syntax'. *Journal of Logic, Language and Information* **17**(2), 217–227.

Tiede, H. and S. Kepser: 2006, 'Monadic Second-Order Logic and Transitive Closure Logics over Trees'. *Electronic Notes in Theoretical Computer Science* **165**, 189–199.

Address for Offprints:
Catherine Lai
Department of Linguistics
619 Williams Hall
University of Pennsylvania
Philadelphia, PA 19104-6305
USA