[**document published 29 Dec 2014 on Uyghur Light Verbs website.** Page numbers added here. Original document in XML, delivered to CLAS on 10 November 2015. Printed for CCAPT on 11 November 2015.]

XQuery Databases in the IAIA and UyLVs projects

Paper delivered at DH 2013, Lawrence, Nebraska, July 2013

C. M. Sperberg-McQueen, Black Mesa Technologies LLC
Arienne M. Dwyer, University of Kansas
Rev. 15 November 2013

1 Overview

Increasing numbers of linguists have multimodal data of relatively rare languages, richly annotated for grammatical information. Querying these data, in the service of linguistic discovery, should be powerful and flexible. This paper explores both the promise of linguistic discovery, as well as the problems of open-ended querying and sibling searches: How to balance maximum accessiblity while attending to security and allowing open-ended queries? How to optimize searches for noncontiguous morphemes of a sentence? We first provide specific examples to contextualize the discussion in 3 below of document structure and queries.

2 Linguistic, cultural, and historical resources

The resources that our team (which has included seven University of Kansas students) [1] is creating are part of two projects, Interactive Inner Asia (IAIA) and Uyghur Light Verbs

(UyLVs). For documentary linguistics, and to document large geographic areas (in the IAIA case) and hundreds of years of a language's history (in the Uyghur case), the research questions, and the logistical and technical requirements are considerable. After brief project descriptions, we detail the major linguistic and non-linguistic issues facing projects such as these.

2.1 Interactive Inner Asia (IAIA)

What if we could analyze the grammar of an area rather than a single language? Northern (Amdo) Tibet is a cultural convergence zone where a dozen related and unrelated languages have come to resemble each other through intense contact over centuries. Many languages of this area are little understood, so the ability to compare forms accross languages allow us to more easily discover similarities between languages and more fundamentally to figure out the grammars of the least-documented languages. This project focuses on mining those texts to compare lexical items and grammatical forms across these languages.

We've created a database-backed website for the North Tibetan *Sprachbund*, or language area, including five unwritten endangered languages: a Turkic language known as Salar (ISO 639-3: slr); two Mongolic languages, Mangghuer a.k.a. Southeastern Monguor (mjg-se) and Baonan (peh); and two mixed languages, Wutun (wuh), a Sinitic/Tibetan creole, and Kangjia (kxs), a Mongolic-Sinitic mix. These languages are interacting with the regional dominant languages, Amdo Tibetan and Northwest Mandarin; data from these contact languages are also provided for comparison.

An earlier project supported by the Volkswagen Foundation DoBeS programme suppported data collection; several native

speakers from each language group determined what genres to collect and did much of the audio-video recordings and some base transcriptions in a range of genres. That project resulted in masses of naturalistic discourse in desperate need of regularization. The current project thus allows a uniform presentation of digital editions of those texts; and when the data has been collected by 25 people with a variety of orthographies and translation languages, creating annotated editions of those texts is no trivial undertaking.
Thus, the focus of the Interactive Inner Asia project is to regularize, annotate, and make queryable synchronic text data from five endangered language and two dominant languages of Northern Tibet. Since close contact has resulted in the languages sharing many features, a priority of the project is to allow queries across multiple languages, with the ulitmate goal of allowing discovery of a grammar of a region, rather than merely that of a single ethnolinguistic group.

2.2 Uyghur Light Verbs (UyLVs)

In contrast to the descriptive, synchronic, and multilingual IAIA project above, the UyLVs project focuses on diachronic change in one language, Uyghur, in order to answer questions of linguistic theory: Are light verbs a valid cross-linguistic category? What features are common to light verbs across languages?
Light verbs modulate the meaning of a main verb, providing aspectual or semantic nuance. Taking the high-frequency Uyghur verbs *al-* 'take, buy', *két-* 'leave, depart,' and *tur-* 'stand, stay' as examples: all can be used alone for a direct, neutral factual statement: *aldi* 'S/He bought (something)'; *kétti* 'S/He left'; *turdi* 'S/He stood.' But when for example *két-* or *tur-* is used as a light verb, then these main verbs lose their

lexical meaning, and indicate that the preceding action or state is instantive or durative, respectively: *él-ip ketti* 'S/He bought up everything'; *él-ip turdi* 'S/He keeps on buying.' Through these so-called light verbs, the speaker can express many kinds of nuance.

That such verbs of body motion are grammaticized into aspectual and actional markers is common cross-linguistically. These markers take several forms, e.g. serial verbs, verbal complements, or light verbs. Light verbs (e.g. *tur-*)combine with a main verb (e.g. *al-*) to form a single predicate. Languages with such light verbs include Persian and Hindi/Urdu, and several Turkic languages. Among the languages which have light verbs, what is unusual about Uyghur is the sheer number of verbal light verbs (about 20), allowing the expression of a wide range of semantic nuance. Again using the above verb *al-* 'take, buy,' two more examples of light verb combinations are *él-iwal-di* 'S/He bought it for his own benefit'; *élip berdi* 'S/He bought it for someone else's benefit.'

The current project focuses on these *verbal light verbs*; LVs also accompany nouns; for example the English LV *do* in *do lunch*. The verbal LV constructions developed from earlier serial verb constructions in Uyghur, with the structure V1-(y)(I)p V2, whereby the second verb (V2) was incorporated with V1 as a complex predicate, and the V2 was semantically bleached. In order to understand when the V2 in a serial verb construction came to develop these properties (therefore becoming a light verb), the Uyghur Light Verbs project has created a grammatically annotated diachronic corpus of the Uyghur language, 1530-present, with a focus on the 19th century, where the innovation appears to have occurred.

2.3 Some linguistic issues

These two projects provoke a long list of linguistic questions; these, of course, will have to be implemented as queries to the corpora. Despite differing in specific aims, the two projects share a common data structure; moreover, the broader linguistic and data preservation issues are the same. These include understanding how languages in contact interact; evaluating universals and language-specific aspects of language; and the documentation of little-understood languages for future generations of speakers and for academia.

The kinds of questions that the Uyghur LVs project needs to query the corpus for include: Via a diachronic corpus, can we pinpoint when Uyghur LVs developed? Are there universal principles of LVs? How do UyLVs compare to those in, say, Hindi and Persian? What are their syntactic and semantic properties? With what subjects can they co-occur? Does the complementizer *dep* always select for accusative-marked subjects?

For the IAIA project, which primarily examines contact between modern Inner Asian languages, the questions include e.g.: What percentage of lexis in the smaller languages originates in Tibetan and Chinese? Is the postposed numeral 'one' a calque from Tibetan? How is it realized in each of these languages? Inner Asian languages: Do all the Mongolic languages under study have the same tripartite Mongolic anteriority system?

2.4 Some non-linguistic issues

For ‘under-resourced’ languages, some non-linguistic issues arise. Those of us who create such linguistic resources have a

dual responsibility to share the results in forms useful to both the language communities and to academic researchers. (Of course, the first step is to avoid taking one's video and audio recordings to the grave.) After that, we face four main issues: First, in a field without mature tools and techniques, we are obliged to develop querying methods that run over limited grammatical annotation for languages whose grammar we in some cases do not yet understand well. Applying text mining techniques to little-resourced *unwritten* languages means that we have no off-the-shelf segmenters or POS taggers, nor can we whip up parsers for these languages yet. [2]
Second, in terms of workflow, we have multiple annotators with different skills and Internet access in far-flung locations. Such a collaboration results in versioning issues, as well as dictating the modification of software choices and workflow. Data normatization is also a significant focus, given that the projects have documents in 15 languages and 23 orthographies, including early handwritten Perso-Arabic, Tibetan, Chinese, and Cyrillic.
Third, our users are not only interested in linguistics, they also want to be able to discover cultural and historical information from these resources. The corpora and website must therefore be designed for as a generalist primary resource, rather than creating site which exclusively answers theoretical questions.
Fourth and finally, we'd like users to be able to devise queries we haven't thought of across the whole of both corpora, while attending to numerous security considerations (as detailed below) and still providing maximal access.

2.5 Existing data

Existing audiovisual data was transcribed in Transcriber XML, OpenOffice, or is untranscribed; existing handwritten manuscript data was keyboarded; published data was OCRd

or keyboarded. From those sources, we make simple XML, with at least three annotation tiers: an orthographic representation, a broad phonemic transcription in the International Phonetic Alphabet, and a sentence-level English gloss:

```
...
<s lang="uig" who="Mejit Axun" ref="2">
  <orth>Qašqarda azīz ōlijāłar bar.</orth>
  <ipa>qʰaʃqʰarda aziːz øːlijaːłar bar</ipa>
  <gloss lang="eng">In Kashgar there are
    the blessed and the holy,</gloss>
  <gloss lang="deu">In Qašqar gibt es
    Selige und Heilige,</gloss>
</s>
...
```

If sentence glosses in other languages are available (like the German gloss here (from [Menges 1933]), or more typically like the glosses we have in the regional languages, Chinese and Tibetan).

2.6 Annotation

The reader will immediately notice our hoc home-brewed vocabulary. The annotation consists of several tiers in existing data:

`orth` utterance in 'orthographic' form
`ipa` utterance in IPA
`gloss` sentence-level gloss (various languages)

Annotation adds three additional elements:

`seg` segmentation into morphemes
`pos` part-of-speech for each morpheme
`ilg` morpheme-level gloss

2.7 Tier-major form

For simplicity of data entry, we put each annotation tier (`seg`, `pos`, `ilg`) in an element of its own.

```
<s lang="uig" who="Mejit Axun" ref="2">
  <orth>Qašqarda azīz ōlijāłar bar.</orth>
```

```
  <ipa>qʰaʃqʰarda aziːz øːlijaːłar bar</ipa>
  <seg>qʰaʃqʰar-da aziːz øːlijaː-łar bar</seg>
  <pos>Ntop-LOC AJ N-PL EXIST</pos>
  <ilg>Qashqar-LOC  blessed holy-PL exist</ilg>
  <gloss lang="eng">In Kasqar there are
    the blessed and the holy,</gloss>
  <gloss lang="deu">In Qašqar gibt es
    Selige und Heilige,</gloss>
</s>
```

N.B. This risks misalignment.

2.8 Segment-major form

For simplicity of search, it's sometimes better to put each segment into an element of its own, thus guaranteeing a 1-to-1 alignment. (It would therefore be trivial to find e.g. the suffix *-da* as a locative vs. *da* as discourse particle.)

For example,

```
<s lang="uig" who="Mejit Axun" ref="2">
  <orth>Qašqarda azīz ōlijāłar bar.</orth>
  <ipa>qʰaʃqʰarda aziːz øːlijaːłar bar</ipa>
  <segments>
    <segment>
      <ipa>qʰaʃqʰar-</ipa>
      <pos>Ntop-</pos>
      <ilg>Kashgar-</ilg>
    </segment>
    <segment>
      <ipa>-da</ipa>
      <pos>-LOC</pos>
      <ilg>-LOC</ilg>
    </segment>
    <segment>
      <ipa>aziːz</ipa>
      <pos>AJ</pos>
      <ilg>blessed</ilg>
    </segment>
    <segment>
      <ipa>øːlijaː-</ipa>
      <pos>N-</pos>
      <ilg>holy-</ilg>
    </segment>
    <segment>
      <ipa>-łar</ipa>
      <pos>-PL</pos>
      <ilg>-PL</ilg>
    </segment>
```

```
    <segment>
      <ipa>bar</ipa>
      <pos>EXIST</pos>
      <ilg>exist</ilg>
    </segment>
  </segments>
  <gloss lang="eng">In Kashgar there are
    the blessed and the holy,</gloss>
  <gloss lang="deu">In Qašqar gibt es
    Selige und Heilige,</gloss>
</s>
```

But this one-to-one alignment of part of speech annotation is only the starting point for linguistic analysis, which requires query interfaces that are as powerful and flexible as possible.

2.9 The problems

We have explained the interest and the promise of this material. In the remainder of this paper we describe two problem complexes which arise in our efforts to fulfil that promise and which may be of general interest. The first problem is providing powerful but easily usable search interfaces; it arises in some form or other for any project which builds a complex information resource for research use. The second is perhaps peculiar to corpus-linguistic projects providing word by word annotation, without providing syntax trees: it is finding a way to make it convenient (and efficient) to search for linguistically interesting patterns among the words of a sentence (or, in XML terms: to search for occurrences of patterns among siblings of a common parent).[3]

3 Supporting open-ended query

Like many other projects, the IAIA and UyLVs projects can expect the users of the resources we are developing to fall into a number of different categories. Most numerous by far

will be casual users unfamiliar with the data, the annotations, or the markup used to represent them. From time to time, one user or another may work with the materials repeatedly over a short while; such short-term users will, we expect, become more familiar with the material and its annotation than a casual user ever will. A very small number of researchers will use the material intensively, perhaps over a longer period. Users of these different classes will, we expect, require different kinds of search interfaces for working with the data. The needs of project-internal usage will again be different, though in many ways they will closely resemble those of intensive research users.

For casual users, the best approach appears to be a very simple search interface, the simpler the better. The examples of Google and other internet-search utilities seem to illustrate clearly that for such users the advantage of a simple interface (e.g. one consisting of a single text widget in which to type search terms) far outweighs the disadvantage of being unable to be explicit about what kind of information one is looking for. Such interfaces now also have the advantage of wide familiarity.

So for casual users, we expect to provide a very simple interface in which the user types one or more search terms, without specifying whether the terms represent subject-language words or phrases, words or phrases in an English gloss, part-of-speech tags, document-level metadata, or something else. The system must simply do the best it can with the search terms provided, searching for those strings appearing anywhere in the corpus and returning some representation of the results.

Anyone who spends more than a few minutes with the corpora, however, is likely (we expect) to want to construct searches that are more complex and powerful, or at least

more specific about what they are looking for. So as a sort of ‘advanced’ or ‘power-user’ option, we expect to provide an interface for straightforward fielded search, and possibly (if we can keep the interface simple enough) for simple Boolean combinations of searches. Like the interface for casual users, the fielded-search interface should require no knowledge of the markup and annotation of the corpus, and will indeed not enable a user who understands the markup and annotation to apply that knowledge. Interfaces of this kind are simple enough to construct using conventional form-building technologies; we will use XForms, because it works better with XML than most other forms technologies [W3C 2009], [Raman 2004], [Dubinko 2003].

Intensive users, and project-internal users, on the other hand, require something more than simple fielded search. There is no natural upper bound to the complexity of questions that may arise in the course of research, and it is not hard to formulate queries interesting for a researcher which cannot be formulated conveniently in a simple forms interface. Research is open-ended; it would seem that a search interface intended to support research must similarly be open-ended.

As is natural, given that the material to be searched is a collection of XML documents, all searches against our project data, no matter which search interface the user actually employs, are performed by an XQuery engine.[4] One obvious way to provide an open-ended search interface would be to allow the user to formulate arbitrary queries using XQuery, and execute them on the fly. This will please users who know XQuery or XPath but may intimidate others. Presumably for analogous reasons, some corpus query software provides a query constructor as part of their graphical user interface;[5] unfortunately, construction of GUI tools requires resources our projects do not have, and such

tools tend to have a distressingly short lifespace. An XForms-based query constructor seems to be a plausible alternative, and we hope to explore the creation of such a tool when time and resources allow.

One widespreach approach is to provide some other query language, thought to be simpler or easier for corpus linguists or other intensive users of the materials to master [Burnard, n.d.], [Pajas 2010], [Schmidt 2010], [Zeldes 2012].

Unfortunately, if every corpus resource uses its own query language, there is essentially no interoperability among corpora, and users who master one language for one corpus cannot exploit that mastery in working with another corpus. XQuery does at least have the advantage of standardization; if intensive users of our materials must learn a new query language, they might do worse than to learn a language they can use elsewhere, and not only with our project data. For this reason, our current approach to the fact that linguists and other potential users are not born knowing how to formulate queries in XQuery is to teach participants in the project how to formulate queries using XPath and XQuery.

We may nevertheless explore the use of other query languages, for reasons described below.

4 Security issues in open-ended query

However, there is another problem with the idea of allowing the user to formulate arbitrary queries in XQuery: it creates a serious security vulnerability, analogous to the well known *SQL injection*.[6] The simplest way to support arbitrary queries is to provide a text widget for the user, pass the user's query string to the server, and evaluate it using the XQuery *eval* function:

```
declare variable $query as xs:string external;
```

```
xquery:eval($query)
```

The danger is that in the general case, we do not know what is in the query string `$query`. Our search interface for open-ended queries is intended for intensive use by researchers, but if an adversary uses that interface as a means of attack, there is every reason to expect that the query string may contain the XQuery expression "`delete node //*`" or some other destructive expression.

The server might, it is true, scan the `$query` string and reject it if it contains the string "`delete`," but this is the first step down a very slippery slope, as the history of computer security shows. A scan for the string "`delete`" will work for the example shown above, but not for this equivalent attack:

```
concat('dele',
  'te n',
  'ode/',
  '/*')
```

A common obfuscation technique in current site cracking tools could be adapted to this case in the following form:

```
concat('&amp;#100;&amp;#101;&amp;#108;&amp;#x65;',
  '&amp;#116;&amp;#101;&amp;#x20;&amp;#x6E;',
  '&amp;#x6F;&amp;#x64;&amp;#x65;&amp;#x2F;',
  '&amp;#47;&amp;#x2A;')
```

In the face of such attacks, a set of ad-hoc searches for strings indicating dangerous operations appears to be doomed to failure. A better solution might be to define a 'safe' subset of XPath and/or XQuery and to check the user's input systematically to see that it falls into the subset. The subset is defined by a full grammar (which essentially consists of the standard grammar for XQuery or XPath, minus the constructs excluded as potentially dangerous), and a recognizer is written for that grammar, either by hand or using a parser generator. The user's query is then handled by being passed first to the recognizer and then evaluated (if the recognizer accepts the query) or rejected. The relevant logic can be expressed in XQuery thus:

```
import module namespace
       cqi :=
"http://iaia.ittc.ku.edu/2013/xquerytools";
declare variable $query as xs:string external;

if (cqi:nanny-says-ok($query)) then
   xquery:eval($query)
else
   <error>That query goes too far!</error>
```

In designing a safe subset of XQuery, we face some obvious design choices. Should the language be as weak as possible, including only what is thought absolutely necessary and excluding everything else? Or should it be as strong as possible, excluding what is thought dangerous and including everything else? A weak language may be easier to check and implement; it may also be less likely to include constructs which appear innocuous on the surface but prove dangerous. A strong language is less likely to impose unnecessary restrictions on the researcher, but also more likely to prove unsafe in the hands of a sufficiently determined and clever adversary.

In either case, the advice of experts on computer security (e.g. [Schneier 2000]) is explicit: the first step in defending against unwanted events is to identify clearly what events are unwanted and to consider how an adversary might bring them about. Projects will vary in their threat lists, but some threats will be common to many projects which aim to develop resources for research. Our current list of threats includes the following:

unwanted deletion of data

unauthorized alteration

excessive resource usage, denial-of-service attack

data download in excess of quota

access to server OS or file system (less a problem in itself than a step to other damaging exploits)

A full security analysis should also consider who possible

adversaries are and their likely avenues of attack. Potential adversaries include those hostile for some reason to a specific project but also casual vandals who deface sites for pleasure (and of course the ever-present purveyors of illicit pharmaceutical products).

Consideration of specific threats makes it easier to identify specific constructs in XPath and XQuery (or in any other query language) which ought to be off-limits for the Web query interface.[7] First and foremost, constructs which allow an attacker to delete or alter data in the database:

updating expressions (i.e. expressions which alter the data in the collection); the IAIA and UyLVs projects do update the collections via Web interfaces, but the update interfaces are separate and have their own security measures

Second, constructs which allow read or write access to the file system or other data streams. Write access to the file system or other locations seems obviously undesirable in itself. Read access to the file system is undesirable not because all material on the server file system is private (it's not) but because it may provide an adversary with information about the internals of the server and thus allow the identification of weaknesses suitable for attack.

access to file system via `doc('file:///...')`, etc.

serialization of results; this is undesirable because XQuery serialization essentially writes data to external data streams and thus may provide write-access to the server or to other locations. It must not be confused with the process of sending the results of the query back to the user via HTTP.

information leaks (`doc()` requests on other servers, HTTP requests, other interactions with the external world); these are usually felt to be insecure because they can

make it possible to expose information about the execution environment by means of apparently innocuous interactions with other servers. On the other hand, any information which could be leaked in this way could be just as easily presented to the user as part of a query result. So it is not entirely clear whether these constructs should be proscribed or not.

Finally, there are *cloaking devices*, constructs which are innocuous in themselves but which must be excluded in order to ensure that other dangerous constructs cannot evade detection.

the `eval()` function

calls to external functions

inclusion of external modules

In practice, however, we have found it unnecessary to define and enforce a safe subset of XQuery. The standard counter-measure against unauthorized alteration of the data is to evaluate queries under a database user ID with weak access. User access controls are not standardized as part of XQuery, but they have been a familiar feature of database management systems for decades, and problems of access control continue to be of interest for database researchers and implementors alike. This includes (fortunately for us) the implementors of XQuery. Most serious XQuery engines intended to be used as servers come with well elaborated mechanisms for user access control, and it is straightforward to define a user for the Web query interface who has READ access to the necessary databases, no READ access to other databases, and no WRITE access anywhere.[8]

There is little which projects of limited infrastructure can do to protect themselves reliably against denial-of-service attacks. That little, however, is probably worth doing: it involves using operating-system or server-level mechanisms to

set resource limits on the process which handles query requests. XQuery servers may have a configuration parameter for setting a maximum amount of time to spend on any single query;[9] queries that run longer are terminated and return no results. This is not a perfect protection against hostile denial-of-service attacks, but it does help protect against queries which inadvertently consume excessive resources.
Our current approach to security issues is simple: queries run under a read-only user ID, and queries time out after some number of seconds (currently ten). Only if these prove to be insufficient do we expect to define a safe subset of XQuery, write a parser for XQuery to translate queries into XML, and write an XQuery function to check XML-encoded queries for safety. These are all in themselves interesting tasks, but sufficient unto the day is the evil thereof.

5 The sibling search problem

The second problem to be discussed here is the difficulty of using XQuery for doing the kind of linguistically motivated searches described above. We are not producing syntactic parse trees; the annotated texts produced by the IAIA and UyLVs projects are essentially flat sequences of sentences, and each sentence is a uniform sequence of annotated morphemes. In consequence, most searches for linguistically interesting patterns take the form of searches for sequences of sibling `seg` elements which match some pattern. Searches for patterns among siblings pose two kinds of problems for XQuery (as also for many other query languages proposed in the past for XML or SGML). First is a notational (one might almost say psychological) difficulty; second is a purely technical issue of search efficiency and speed.
The notational difficulty is simple. XPath (and thus also

XQuery) is quite good at describing containment relations among elements: the parent, child, ancestor, and descendant axes of XPath make it easy to search for all elements of type `x` immediately contained in a `y` element but not (at any level) contained by or containing a `z` element.[10] A glance at a typical XSLT stylesheet or XQuery module will show that such containment-related qualifications of queries are ubiquitous in XML processing, and constantly employed by users of XPath.

But the very large majority of queries we expect to handle against the IAIA and UyLVs corpora involve precedence relations, not dominance relations. Search qualifications involve preceding and following words or segments, not containing or contained elements. Sample queries include:

Find nouns not immediately followed by postpositions.

Find sentences containing light verbs; for each such sentence, find the leftmost noun not immediately followed by a postposition.

Find *yi-* immediately followed by *-ge*.

Find POS `NU-` immediately followed by `-CL`.

(English) Find a determiner followed by two or more adjectives and a noun.

At first glance XPath appears to be much less effective at such sibling-related qualification of queries; it seems to have a pronounced bias in favor of vertical (dominance) relations over horizontal (precedence) relations. Certainly sibling qualifications are much rarer in typical queries and stylesheets than qualifications based on ancestry or descendants.) Deeper consideration suggests, however, that this apparent vertical bias is an illusion. The vertical `ancestor` and `descendant` axes are matched feature for feature by the horizontal `preceding` and `following` axes, not to mention the `preceding-sibling` and `following-sibling` axes.

So it is by no means impossible to write XPath expressions which capture useful patterns among sibling elements. We can find `x` when followed by `y` ("`x[following-sibling::y]`", or equivalently "`y / preceding-sibling::x`"), and `x` *immediately* followed by `y` ("`x [following-sibling::*[1] / self::y]`", and to express more elaborate patterns it is only necessary to write more and more elaborate expressions. The queries listed above can all be expressed in XPath, given the 'segment-major' form described above:

Find nouns not immediately followed by postpositions.

```
//pos[. = ("N", "Nmeas", "Nor", "Npr", "Ntop")]
/ .. / following-sibling::segment
[not(pos = ("P", "POST", "POST.CON"))]
```

Find sentences containing light verbs; for each such sentence, find the leftmost noun not immediately followed by a postposition.

```
//s[ segment/pos = ("LVV", "LVN"")]
 /child::segment
 [pos = ("N", "Nmeas", "Nor", "Npr", "Ntop")]
 [following-sibling::segment[1]
   [not(pos = ("P", "POST", "POST.CON"))]]
 [1]
```

Find *yi-* immediately followed by *-ge*.

```
//segment[orth = "yi-"]
[following-sibling::segment[1]/orth = '-ge']
```

Find POS `NU-` immediately followed by `-CL`.

```
//segment[pos = "NU"]
[following-sibling::segment[1]/pos = 'CL']
```

(English) Find a determiner followed by two or more adjectives and a noun. For simplicity, we assume that determiners, adjectives, and nouns are tagged `DET`, `JJ`, and `N`, respectively (in actual corpora, each of these terms will correspond to several part-of-speech codes). Finding a determiner followed by exactly two adjectives and a noun is straightforward:

```
//segment[pos='DET']
  [./following-sibling::segment[1][pos='JJ']
    /following-sibling::segment[1][pos='JJ']
    /following-sibling::segment[1][pos='N']]
```

Allowing more than two adjectives, but not allowing anything else

to intervene, is less straightforward:

```
    //segment[pos='DET']
[./following-sibling::segment[1][pos='JJ']
  /following-sibling::segment[1][pos='JJ']
   [some $j in self::segment
    satisfies
    some $n in ./following-sibling::segment[pos='N']
    satisfies
    every $seg in
        $j/following-sibling::segment
        intersect
        $n/preceding-sibling::segment
    satisfies $seg/pos='JJ']]
```

The appearance of bias comes from two sources. The syntax provides special short forms for the vertical axes, so `x` is short for `child::x`, `x//y` is short for `child::x / descendant-or-self::* / child::y`, and `..` is short for `parent::*`. There are no similar short-forms for the horizontal axes.

Also, the vertical axes come in both single-step and multi-step versions (the ancestor axis is the transitive closure of the single-step parent axis, and the descendant axis consists of everything that can be reached in one or more steps with the child axis). The horizontal axes come only in multi-step forms. This helps keep the number of different axes from exploding, but it has its price. It is easy, in XPath, to identify the parent of the current node if and only if it is a `chapter` element, by writing "`parent::chapter`". To identify the next element after the current element (its immediately following sibling) if and only if it is a chapter, one must write `following-sibling::*[1] / self::chapter`.[11]

The examples given may serve to demonstrate that the apparent vertical bias of XPath is at one level purely imaginary: the expressive power of the horizontal axes is every bit as great as that of the vertical axes. But they also serve to demonstrate that in another sense the bias is quite real: the sheer verbosity of the expressions given makes these queries harder to write than would be desirable. It is primarily

for this reason that we wish to explore the possibility of making some other query syntax available to users of our corpora; as time allows we expect to study a variety of query languages for SGML, XML, and language corpora, in search of a good combination of power and succinctness. Obvious candidate languages for study include the corpus linguistic query systems mentioned earlier ([Burnard, n.d.], [Pajas 2010], [Schmidt 2010], [Zeldes 2012]), the query language developed for the SGML hypertext system DynaText and its sister product DynaWeb ([[EBT 1996]], [[Silicon Graphics 1995]]), ARRAS ([Smith 1985]), the 'extended pointer notation' defined in an earlier version of the TEI *Guidelines* ([ACH/ACL/ALLC 1994]), and of course extended versions of XPath and of CSS Selectors ([W3C 2011]). A forms-based interface which assists in the construction of queries is also a possibility.

The second difficulty associated with sibling search in XQuery is technical. Many XQuery engines build indices which allow them to handle searches on the vertical axes fairly efficiently[12] but few if any index for searches on the horizontal axes.

We have had some success in this area, experimenting with user-level indices (by which we mean secondary documents created by the user, which have the property that some questions about the original corpus material can be answered by examination of the secondary documents). Our experiments thus far have involved some simple variants on Patricia tries ([Gonnet et al. 1992], [Gonnet / Baeza-Yates 1991]), mostly involving POS tags. The basic idea is simple: we make the POS tags into element names instead of content (to exploit the fact that most XQuery engines build good indices of element names), and we rewrite the preceding-sibling : following-sibling relation as a parent : child relation

(to exploit the engine's indexes for vertical axes). So a segment tagged N- immediately followed by one tagged -PL:

```
<segment>... <pos>N-</pos> ...</segment>
<segment id="x">... <pos>-PL</pos> ...</segment>
```

becomes an element named N with a child named PL:

```
<N>
  ...
  <PL><ptr target="x"/></PL>
  ...
</N>
```

As can be seen, the child here carries a pointer (here represented as an `target` attribute) pointing to the element in the base document represented by the child in the index. This allows queries to find a set of locations by looking in the index, and then to return pointers to the passages in the base document (which are of course what the user is presumed interested in).

As originally specified, Patricia tries index semi-infinite strings running from the index point to the end of the document. We have experimented with several variations on this: Patricia tries in the strict sense (indexing the sequence of segments from any segment to the end of the document), fixed-depth tries (essentially indexes of n-grams for some fixed *n*, not really Patricia tries at all, just conventional tries), and tries which index sequences of segments up to the end of the containing sentence.

This representation allows some of the verbose queries given earlier to be reformulated somewhat more concisely:[13]

Find nouns not immediately followed by postpositions. `/*/(N | Nmeas | Nor | Npr | Ntop) /(* except (P | POST | POST.CON))`

Find POS NU- immediately followed by -CL. `/NU/CL`

(English) Find a determiner followed by two or more adjectives and a noun. Again, we give two queries. To find a determiner followed by exactly two adjectives and a noun: `/DET/JJ/JJ/N`Allowing more than two

adjectives, but not allowing anything else to intervene, continues to require the use of existential and universal quantification: `/DET/JJ/JJ`

```
[some $j in self::JJ
 satisfies
 some $n in .//N
 satisfies
 every $seg in
     $j//*
     intersect
     $n/ancestor::*
 satisfies $seg[name()='JJ']]
```

The Patricia-trie representation also seems to be easier for the XQuery engine to process. A simple example may serve to illustrate the point. On a corpus of similar size (the Corpus of London Teenage English, which includes about 100,000 words of running text, with word-by-word part-of-speech tagging), a search for determiners followed by four adjectives and a noun can be formulated thus:

```
for $w in
         //w[@pos=('AT', 'AT1', 'DD', 'DD1', 'DD2',
                  'DDQ', 'DDQGE', 'DDQV')]
            [following-sibling::w[1]
              [@pos=('JJ', 'JJR', 'JJT', 'JK',
                     'MC', 'MCMC', 'MC1', 'MD')]
            [following-sibling::w[1]
              [@pos=('JJ', 'JJR', 'JJT', 'JK',
                     'MC', 'MCMC', 'MC1', 'MD')]
            [following-sibling::w[1]
              [@pos=('JJ', 'JJR', 'JJT', 'JK',
                     'MC', 'MCMC', 'MC1', 'MD')]
            [following-sibling::w[1]
              [@pos=('JJ', 'JJR', 'JJT', 'JK',
                     'MC', 'MCMC', 'MC1', 'MD')]
              [following-sibling::w[1]
                [@pos=('MC1', 'MC2', 'ND1',
                       'NN', 'NN1', 'NN2',
                       'NNJ', 'NNJ2', 'NNL1', 'NNL2',
                       'NNT1', 'NNT2',
                       'NNU', 'NNU1', 'NNU2',
                       'NP', 'NP1', 'NP2',
                       'NPD1', 'NPD2', 'NPM1',
'NPM2' )]]]
                       ]]]
```

```
let $s := $w/parent::*
return <hit s="{$s/@n}">{
  normalize-space(string($s))
}</hit>
```

On our test machine, this query executes in between 1500 and 1700 milliseconds.[14] The analogous query against the Patricia trie is:

```
let $index-points0 := pat-trie
  /(AT|AT1|DD|DD1|DD2|DDQ|DDQGE|DDQV)
  /(JJ|JJR|JJT|JK|MC|MCMC|MC1|MD)
  /(JJ|JJR|JJT|JK|MC|MCMC|MC1|MD)
  /(JJ|JJR|JJT|JK|MC|MCMC|MC1|MD)
  /(JJ|JJR|JJT|JK|MC|MCMC|MC1|MD)
  /(MC1|MC2|ND1|NN|NN1|NN2|NNJ|NNJ2
    |NNL1|NNL2|NNT1|NNT2|NNU|NNU1|NNU2
    |NP|NP1|NP2|NPD1|NPD2|NPM1|NPM2)

return $index-points//ptr
```

This query executes in 17-19 ms, for a speedup of slightly less than 100.

As was noted earlier for the sample queries, this query against the Patricia trie is not quite equivalent to the original query, because it returns nodes from the index, not nodes from the base documents. A fully equivalent query, which uses the pointers to retrieve the relevant nodes from the base document, runs in about 100ms, for a 15- to 17-fold speedup vis-a-vis the original.[15]

```
 let $index-points0 := pat-trie
  /(AT|AT1|DD|DD1|DD2|DDQ|DDQGE|DDQV)
  /(JJ|JJR|JJT|JK|MC|MCMC|MC1|MD)
  /(JJ|JJR|JJT|JK|MC|MCMC|MC1|MD)
  /(JJ|JJR|JJT|JK|MC|MCMC|MC1|MD)
  /(JJ|JJR|JJT|JK|MC|MCMC|MC1|MD)
  /(MC1|MC2|ND1|NN|NN1|NN2|NNJ|NNJ2
    |NNL1|NNL2|NNT1|NNT2|NNU|NNU1|NNU2
    |NP|NP1|NP2|NPD1|NPD2|NPM1|NPM2)

let $hits := $index-points//ptr
for $hit in $hits
let $doc := $hit/@uri,
   $path := for $n in
tokenize(substring($hit/@path,2),'/')
            return xs:integer($n),
   $node := local:path(doc($doc), $path)
```

In further work, we hope to produce a function library to make it easier to handle searches of the form “An x, one or more y, then a z” (essentially regular patterns, analogous to regular expressions), and to find a way to allow users to refer conveniently to higher-level part-of-speech concepts like “noun” without having to remember and list all the codes used in the corpus for that high-level construct.

A Funding

We gratefully acknowledge support from the U.S. National Science Foundation's Documenting Endangered Languages program for the Interactive Inner Asia project (BCS1065524) and its Linguistics program for the Uyghur Light Verbs project (BCS1053152); IAIA data was collected in collaboration with speaker communities, with the support of the Volkswagen Foundation's Dokumentation Bedrohter Sprachen (DoBeS) programme (78 270).

B References

Association for Computers and the Humanities, Association for Computational Linguistics, and Association for Literary and Linguistic Computing. ‘Extended Pointers’, section 14.2 of *Guidelines for Electronic Text Encoding and Interchange (TEI P3)*, ed. C. M. Sperberg-McQueen and Lou Burnard. Chicago, Oxford: Text Encoding Initiative, 1994. On the web at http://quod.lib.umich.edu/cgi/t/tei/tei-idx?type=pointer&value=SAXR

Association for Computers and the Humanities, Association for Computational Linguistics, and Association for Literary and Linguistic Computing. ‘Extended Pointers’, section 14.2 of *Guidelines for Electronic Text Encoding and Interchange (TEI P4)*, ed. C. M. Sperberg-McQueen and Lou Burnard,

XML conversion by Syd Bauman, Lou Burnard, Steven DeRose, and Sebastian Rahtz. [n.p.]: Text Encoding Initiative, 2001-2004. On the Web at http://www.tei-c.org/Vault/P4/doc/html/SA.html#SAXR.
[Burnard, Lou]. *All about Xaira* Oxford: OUCS, n.d. On the Web at http://projects.oucs.ox.ac.uk/xaira/
Dubinko, Micah. 2003. *XForms Essentials*. Sebastopol, California [et al.]: O'Reilly, 2003.
Dwyer, Arienne M., and C. M. Sperberg-McQueen. *The Uyghur Light Verbs and Interactive Inner Asia Corpora*. [Lawrence, KS]: Interactive Inner Asia Project; Uyghur Light Verbs Project, 2013. On the Web at http://uyghur.ittc.ku.edu/2013/tsd/UyLVs-IAIA-corpora.xml.
EBT (Electronic Book Technologies). 'Searching'. Chapter 12 of *DynaText Publishing: Document Preparation* Providence: EBT, 1996.
Gonnet, G. H., and R. Baeza-Yates. 1991. *Handbook of algorithms and data structures: In Pascal and C*. Wokingham [et al.]: Addison-Wesley, 1991.
Gonnet, Gaston, Ricardo A. Baeza-Yates, and Tim Snider. 1992. 'New indices for text: PAT trees and PAT arrays'. Chapter 5 of *Information Retrieval: Data Structures & Algorithms*, ed. William B. Frakes and Ricardo Baeza-Yates. Englewood Cliffs, N.J.: Prentice-Hall, 1996, pp. 66-82.
Menges, Karl H.*Volkskundliche Texte aus Ost-Türkistan: aus dem Nachla von N.Th. Katanov. Aus den Sitzungsberichten der Preussischen Akademie der Wissenschaften Philologisch-Historische Klasse 1933 und 1936*. [Berlin]: Akademie der Wissenschaften.
Pajas, Petr. *The Prague Markup Language (Version 1.1)*. [Prague]: Institute of Formal and Applied Linguistics, Faculty of Mathematics and Physics, Charles University, 2006-2010. On the Web at http://ufal.mff.cuni.cz/jazz/PML/doc/.

Raman, T. V. 2004. *XForms: XML Powered Web Forms*. Boston [et al.]: Addison-Wesley, 2004.
Schneier, Bruce. 2000. *Secrets and lies: Digital security in a networked world*. New York [et al.]: Wiley Computer Publishing, 2000.
Schmidt, Thomas. *EXMARaLDA EXAKT Manual Version 1.0*. [Hamburg: n.p.], 2010. On the Web at http://www1.uni-hamburg.de/exmaralda/files/EXAKT_Manual.pdf
Silicon Graphics, Inc. *IRIS InSight DynaWeb User's Guide* [n.p.]: Silicon Graphics, 1995. On the Web at http://techpubs.sgi.com/library/dynaweb_docs/0620/SGI_EndUser/books/IIDWeb_UG/sgi_html/index.html
Smith, John B. *Arras User's Manual*. TR 85-036. Chapel Hill: University of North Carolina Department of Computer Science, 1985. On the Web at http://www.cs.unc.edu/techreports/85-036.pdf
Sperberg-McQueen, C. M. *PixCor 1.1* [Tag-set documentation for Project-internal corpus vocabulary] [Lawrence, Kansas]: Uyghur Light Verbs Project, Interactive Inner Asia Project, 2012. On the Web at http://uyghur.ittc.ku.edu/2012/tsd/pixcor.v1.1.tsd.xml
van der Vlist, Eric. 'XQuery injection: Easy to exploit, easy to prevent....'. Paper given at XML Prague 2011 and at Balisage: The Markup Conference 2011. Prague: XML Prague; Montreal: Balisage, 2011. In *Proceedings of Balisage: The Markup Conference 2011. Balisage Series on Markup Technologies*, vol. 7 (2011). doi:10.4242/BalisageVol7.Vlist02. On the Web at http://www.balisage.net/Proceedings/vol7/html/Vlist02/BalisageVol7-Vlist02.html; slides from a presentation of an earlier version of the material XML Prague 2011 at http://archive.xmlprague.cz/2011/presentations/eric-vdv-xquery-injection/xquery-injection.xhtml

World Wide Web Consortium (W3C). *XML Path Language (XPath) Version 1.0*, ed. James Clark and Steve DeRose. W3C Recommendation 16 November 1999 Published by the World Wide Web Consortium at http://www.w3.org/TR/xpath, November 1999.
World Wide Web Consortium (W3). 2009. *XForms 1.1*, ed. John M. Boyer. W3C Recommendation 20 October 2009. [Cambridge, Sophia-Antipolis, Tokyo]: W3C, 2009. On the Web at http://www.w3.org/TR/xforms/.
World Wide Web Consortium (W3C). *XML Path Language (XPath) 2.0* (Second Edition), ed. Anders Berglund et al. W3C Recommendation 14 December 2010 (Link errors corrected 3 January 2011) Published by the World Wide Web Consortium at http://www.w3.org/TR/xpath20, December 2010.
World Wide Web Consortium (W3C). *Selectors Level 3*, ed. Tantek elik et al. W3C Recommendation 29 September 2011. Published by the World Wide Web Consortium at http://www.w3.org/TR/css3-selectors/, September 2011.
Zeldes, Amir. *ANNIS: User Guide - Version 2.2.1*. Potsdam: SFB 632 Information Structure / D1 Linguistic Database, Humboldt Universitt zu Berlin und Universitt Potsdam, 18 April 2012. On the Web at https://launchpadlibrarian.net/102488706/ANNIS_User_Guide_2.2.1.pdf.

Notes

1 UyLVs: Gülnar Eziz, Travis Major, Rebecca Rosenkrans, Giulia Cabras; IAIA: Mfon Udoinyang, Jeremy Major, and Carolisa Watson.

2 For a number of reasons including workflow, these projects use XQuery, rather than off the shelf linguistic analysis tools commonly used by in situ ("field") linguists such as SIL's Toolbox/Fieldworks.

3 Not long ago, it would have been unthinkable for a project focused as these are to be able to list its top two or three technical challenges without including character-set issues and the processing of non-Latin scripts on that list. That we have the luxury of regarding other problems as our main technical challenges is due to long years of work on character-set issues by many people, prominently including representatives of the digital humanities community. To Joan Smith, Harry Gaylord, Syun Tutiya, David Birnbaum, Christian Wittern, Deborah Anderson, and others, we and all those concerned with the digital representation of human culture owe a large debt of gratitude.
4 We use the excellent BaseX query engine, an open-source XQuery implementation developed at the University of Constance. Other implementations of XQuery, both open-source and commercial, could be used without materially changing the structure of our site.
5 One example of this class of tool is the query constructor in the old SARA software distributed with the British National Corpus in the 1990s.
6 See [van der Vlist 2011] for a serious discussion of XQuery injection and its dangers.
7 We are grateful to Liam Quin, John Cowan, Eric van der Vlist, and Christian Grün for discussion of these issues. They are not responsible for errors in our analysis or discussion.
8 Details vary from system to system, of course, but BaseX, eXist, and MarkLogic all have access control mechanisms adequate to our projects' needs.
9 In BaseX this is the TIMEOUT parameter.
10 For example: Y / X [not .//Z and not ancestor::Z].
11 It is as if the expression denoting a chapter parent had to be written ancestor::*[1] / self::chapter.
12 This is not a necessary truth about XQuery

implementations: since implementation strategy is intentionally left unconstrained by specifications like that for XQuery, implementations are free to build whatever indices they choose. It is, however, a readily observable fact about existing XQuery implementations; it reflects in part, perhaps, the kinds of indexes computer scientists best understand how to build, and in part the kinds of queries implementors expect to see most frequently (which are thus expected to be the best targets for optimization). The vertical bias of XPath has perhaps unconsciously influenced both users and implementors of XPath and related languages.

13 The formulations given here are simplified by ignoring the need to map back from index locations to locations in the base documents. So the gains in concision are not quite as great as they might seem.

14 This paper is not about performance measurements, so the details of the machine and XQuery engine are not important; the machine was a MacBook Pro with a 2.8 GHz dual-core processor, and the XQuery engine was a then-current version of BaseX.

15 It seems likely that this result can be improved by using a less ad-hoc method of retrieving the original nodes. Many XQuery engines have non-standard functions for navigating direct to a particular node in the database; these are non-standard in part because they typically are tightly integrated with the database's internal organization.