Syntactic tree queries in Prolog

Gerlof Bouma
Universität Potsdam, Department Linguistik
Campus Golm, Haus 24/35
Karl-Liebknecht-Straße 24–25
14476 Potsdam, Germany
gerlof.bouma@uni-potsdam.de

Abstract

In this paper, we argue for and demonstrate the use of Prolog as a tool to query annotated corpora. We present a case study based on the German TüBa-D/Z Treebank to show that flexible and efficient corpus querying can be started with a minimal amount of effort. We end this paper with a brief discussion of performance, that suggests that the approach is both fast enough and scalable.

1 Introduction

Corpus investigations that go beyond formulating queries and studying (graphical renderings of) the retrieved annotation very quickly begin to require a general purpose programming language to do things like manipulating and transforming annotation, categorizing results, performing non-trivial counting and even statistical analysis, as query tools only offer a fixed, restricted set of operations. The use of a general purpose programming language has drawbacks, too, however: one has to deal with interfacing with a database, non-deterministic search, definition of linguistically relevant relations and properties in terms of the lower level database relations, etcetera.

As a solution for this dilemma of trading flexibility and power against the ease with which one can query corpora, we propose to use Prolog. Prolog is well suited to query databases (Nilsson and Maluszynski, 1998). Unlike in other general purpose languages, the programmer is relieved of the burden of writing functions to non-deterministically search through the corpus or database.

In comparison to dedicated query languages and their processors, the fact that one can always extend the Prolog predicates that constitute the query language lifts many restrictions on the kinds of queries one can pose. A more specific point is that we can have fine grained control over the scope of negation and quantification in queries in Prolog, something that is sometimes lacking from dedicated languages (for discussion, see Lai and Bird (2004); for a prominent example, König et al. (2003); for an exception, Kepser (2003))

Lai and Bird (2004) formulated a number of queries to compare query languages for syntactically annotated corpora. In this paper, we demonstrate the ease with which a flexible and fast query environment can be constructed by implementing these queries and using them as a rudimentary benchmark for performance.

2 Representing the TüBa-D/Z corpus

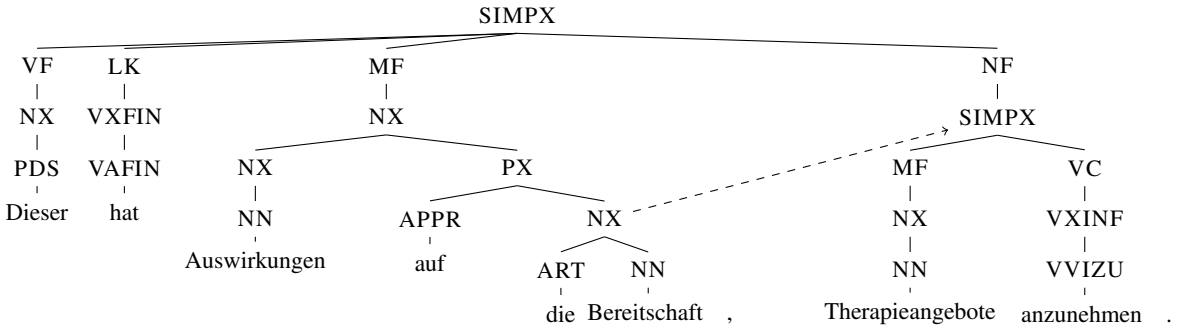
The TüBa-D/Z treebank of German newspaper articles (Telljohann et al., 2006, v5) comprises about 800k tokens in 45k sentences. We store the corpus as collection of directed acyclic graphs, with edges directed towards the roots of the syntactic trees (Brants, 1997).

```
% node/7 SentId NodeId MotherId
%                       Form Edge Cat Other
node(153, 4, 503, die, -, art, [morph=asf]).
node(153, 503, 508, '$phrase', hd, nx, []).
```

By using the sentence number as the first argument of `node/7` facts, we leverage first argument indexing to gain fast access to any node in the treebank. Provided we know the sentence number, we never need to consider more nodes than the largest tree in the corpus. Since all nodes that stand in a *syntactic* relation are within the same sentence, querying syntactic structure is generally fast. An example tree and its full representation is given in Figure 1. Note that in this paper, we only consider the primary nodes and edges, even though we are in no fundamental way restricted to querying only this annotation level.

A set of interface relations provide a first level of abstraction over this representation. Direct dom-

Proceedings of the Fourth Linguistic Annotation Workshop, ACL 2010, pages 212–216,
Uppsala, Sweden, 15-16 July 2010.



"This has effects on the willingness to accept therapy."

```
node(153, 0, 500, 'Dieser', hd, pds, [morph=nsm]).
node(153, 1, 501, hat, hd, vafin, [morph='3sis']).
node(153, 2, 502, 'Auswirkungen', hd, nn, [morph=apf]).
node(153, 3, 508, auf, -, appr, [morph=a]).
node(153, 4, 503, die, -, art, [morph=asf]).
node(153, 5, 503, 'Bereitschaft', hd, nn, [morph=asf]).
node(153, 6, 0, (','), --, '$,', [morph= --]).
node(153, 7, 504, 'Therapieangebote', hd, nn, [morph=apn]).
node(153, 8, 505, anzunehmen, hd, vvizu, [morph= --]).
node(153, 9, 0, '.', --, $., [morph= --]).



secondary(153,503,512,refint).
```

```
node(153, 515, 0, '$phrase', --, simpx, []).
node(153, 506, 515, '$phrase', -, vf, []).
node(153, 500, 506, '$phrase', on, nx, []).
node(153, 507, 515, '$phrase', -, lk, []).
node(153, 501, 507, '$phrase', hd, vxfin, []).
node(153, 513, 515, '$phrase', -, mf, []).
node(153, 511, 513, '$phrase', oa, nx, []).
node(153, 502, 511, '$phrase', hd, nx, []).
node(153, 508, 511, '$phrase', -, px, []).
node(153, 503, 508, '$phrase', hd, nx, []).
node(153, 514, 515, '$phrase', -, nf, []).
node(153, 512, 514, '$phrase', mod, simpx, []).
node(153, 509, 512, '$phrase', -, mf, []).
node(153, 504, 509, '$phrase', oa, nx, []).
node(153, 510, 512, '$phrase', -, vc, []).
node(153, 505, 510, '$phrase', hd, vxinf, []).
```

Figure 1: A tree from Tüba-D/Z and its Prolog representation.

inance and other simple relations are defined directly in terms of this interface.

```
has_sentid(node(A_s,_,_,_,_,_,_),A_s).
has_nodeid(node(_,A_n,_,_,_,_,_),A_n).
has_mother(node(_,_,A_m,_,_,_,_),A_m).
  has_form(node(_,_,_,A_f,_,_,_),A_f).
has_poscat(node(_,_,_,_,_,A_p,_),A_p).

is_under(A,B):-
    has_mother(A,A_m,A_s),
    is_phrasal(B),
    has_nodeid(B,A_m,A_s).

are_sentmates(A,B):-
    has_sentid(A,A_s),
    has_sentid(B,A_s).

is_phrasal(A):-
    has_form(A,'$phrase').
```

None of these predicates consult the database. Actually looking up a graph involves calling the nodes describing it. So, `is_phrasal(A), A`, will return once for each phrasal node in the corpus. Transitive closures over the relations above define familiar tree navigation predicates like dominance (closure of `is_under/2`). In contrast with the simple relations, these closures *do* look up their arguments.

```
has_ancestor(A,B):-
    has_ancestor(A,B,_).

has_ancestor(A,B,AB_path):-
    are_sentmates(A,B),
    A,  is_under(A,A1),  A1,
    has_ancestor_rfl(A1,B,AB_path).

has_ancestor_rfl(A,A,[]).
has_ancestor_rfl(A,B,[A|AB_path]):-
    is_under(A,A1),  A1,
    has_ancestor_rfl(A1,B,AB_path).
```

At this point, linear precedence is still undefined for phrases. We define string position of a phrase as its span over the string, which we get by taking indices of the first and last words in its yield.

```
yields_dl(A,Bs):-
    is_phrasal(A)
      -> ( is_above(A,A1),
           findall(A1, A1, A1s),
           map(yields_dl,A1s,Bss),
           fold(append_dl,Bss,Bs)
         )
    ; % is_lexical(A)
      Bs = [A|Cs]\Cs.

spans(A,A_beg,A_end):-
    yields_dl(A,Bs\[]),
    map(has_nodeid,Bs,B_ns),
    fold(min,B_ns,A_beg),
    fold(max,B_ns,B_n_mx),
    A_end is B_n_mx+1
```

Thus, the span of the word *Auswirkungen* in the tree in Figure 1 is 2–3, and the span of the MF-phrase is 2–6. It makes sense to precalculate `spans/3`, as this is an expensive way of calculating linear order and we are likely to need this information frequently, for instance in predicates like:

```
precedes(A,B):-
    are_sentmates(A,B),
    spans(A,_,A_end),
    spans(B,B_beg,_),
    A_end =< B_beg.

directly_precedes(A,B):-
    are_sentmates(A,B),
    spans(A,_,A_end),
    spans(B,A_end,_).

are_right_aligned(A,B):-
    are_sentmates(A,B),
    spans(A,_,A_end),
    spans(B,_,A_end).
```

TIGERSearch implements an alternative definition of linear precedence, where two left-corners are compared (König et al., 2003). It would be straightforward to implement this alternative.

3 Application & Comparison

Lai and Bird (2004) compare the expressiveness of query languages by formulating queries that test different aspects of a query language, such as the ability to constrain linear order and dominance, to use negation and/or universal quantification, and to separate context from the returned subgraphs. The queries have thus been designed to highlight strengths and weaknesses of different query languages in querying linguistic structure. Six of these queries – with categories changed to match the Tüba-D/Z corpus – are given in Table 1 and expressed in TIGERSearch query syntax in Table 2. Since TIGERSearch does not allow for negation to outscope existential quantification of nodes, queries Q2 and Q5 are not expressible (also see Marek et al. (2008) for more discussion). In addition, Q7 has two interpretations, depending on whether one wants to return NPs once for each PP in the context or just once altogether. TIGERSearch does not allow us to differentiate between these two interpretations.

Q1 & Q2 The implementation of domination, `has_ancestor/2`, performs database lookup. We therefore call it last in `q1/1`. To ensure the correct scope of the negation, lookup of `A` in `q2/1` is explicit and *outside* the scope of negation-as-Prolog-failure `\+/1`, whereas `B` is looked up *inside* its scope.

```
q1(A):-
    has_cat(A,simpx),
    has_surf(B,'sah'),
    has_ancestor(B,A).

q2(A):-
    has_cat(A,simpx),
    has_surf(B,sah),
    A, \+ has_ancestor(B,A).
```

Q1 Find sentences that include the word *sah*.
Q2 Find sentences that do not include *sah*.
Q3 Find NPs whose rightmost child is an N.
Q4 Find NPs that contain an AdjP immediately followed by a noun that is immediately followed by a prepositional phrase.
Q5 Find the first common ancestor of sequences of an NP followed by a PP.
Q7 Find an NP dominated by a PP. Return the subtree dominated by that NP only.

Table 1: Query descriptions

```
Q1 [cat="SIMPX"] >* [word="sah"]
Q2 (not expressible)
Q3 #n1:[cat="NX"] > #n2:[pos="NN"]
   & #n1 >@r #n2
Q4 #nx:[cat="NX"] >* #ax:[cat="ADJX"]
   & #nx >* #n:[pos="NN"]
   & #nx >* #px:[cat="PX"]
   & #px >@l #pxl
   & #ax >@r #axr
   & #axr . #n
   & #n . #pxl
Q5 (not expressible)
Q7 [cat="PX"] >* #nx:[cat="NX"]
```

Table 2: TIGERSearch queries

Q3, Q4 The implementation of `spans/3` relies on given nodes, which means that database lookup is performed before checking linear order constraints, explicitly in `q3/1` and implicitly in `q4_a/1`. In addition, these constraints are expensive to check, so we make sure we postpone their evaluation as much as possible.

```
q3(A):-
    has_cat(A,nx),
    has_pos(B,nn),
    is_under(B,A),
    A, B, are_right_aligned(A,B).

q4_a(A):-
    has_cat(A,nx),
    has_cat(B,adjx),
    has_pos(C,nn),
    has_cat(D,px),
    has_ancestor(B,A),
    has_ancestor(C,A),
    has_ancestor(D,A),
    directly_precedes(B,C),
    directly_precedes(C,D).
```

If we precalculate `spans/3`, the alternative order of checking dominance and linear precedence constraints becomes viable, as in `q4_b/1`.

```
q4_b(A):-
    has_cat(A,nx,A_s),
    has_cat(B,adjx,A_s),
    has_pos(C,nn,A_s),
    has_cat(D,px,A_s),
    B,C,D,              % (cont. on next page)
```

```
    directly_precedes(B,C),
    directly_precedes(C,D),
    has_ancestor(B,A),
    has_ancestor(C,A),
    has_ancestor(D,A).
```

The procedural sides of Prolog make that these two alternatives are processed with considerable speed differences.

Q5 The *lowest common ancestor* part of Q5 can be implemented by constraining the paths between two nodes and their common ancestor:

```
q5(A):-
    has_cat(B,nx,A_s),
    has_cat(C,px,A_s),
    B, C,
    precedes(B,C),
    has_ancestor(B,A,BA_path),
    has_ancestor(C,A,CA_path),
    \+ ( last(BA_path,D), last(CA_path,D) ).
```

Q7 Precise control over the quantification of the two nodes in Q7 is achieved by using the built-in `once/1` predicate (∼existential quantification) and by choosing different moments of database lookup for the two nodes.

```
q7_a(A):-     % once for each np-pp pair
    has_cat(A,nx),
    has_cat(B,px),
    has_ancestor(A,B).

q7_b(A):-    % just once per np
    has_cat(A,nx),
    has_cat(B,px),
    A, once(has_ancestor(A,B)).
```

4 Performance

In Table 3, we list wall-clock times for execution of each of the queries. These serve to demonstrate the fact that our straightforward use of Prolog results in a system that is not only flexible and with short development times, but that is also fast enough to be usable. We have also included TIGERSearch execution times for the same queries to give an idea of the speed of querying with Prolog.[1]

Table 3 shows Prolog execution times fall well within useable ranges, provided we precalculate `span/3` facts for queries that rely heavily on linear order. The non-declarative side of Prolog is most clearly seen in the difference between Q4-a and Q4-b – the latter constraint ordering is more than twice as fast. Even with precalculated `span/3` facts, the whole corpus and query code uses less than 0.5Gbytes of RAM to run.

[1]Machine specifications: 1.6Ghz Intel Core 2 Duo, 2GBytes RAM. SWI-prolog (v5.6) on a 32-bit Linux. The TIGERSearch times were taken on the same machine. The TIGERSearch corpus was compiled with 'extended indexing'.

			Precalc. spans	
	# hits	T.Search	no	yes
Loading from source			30	50
Loading precompiled			3	4
Precalculating `spans/3`			90	
Q1	73	3	1	
Q2	65727		1	
Q3	152669	33	10	4
Q4-a	8185	200	60	50
Q4-b				21
Q5	312753		196	70
Q7-a	145737	6	8	
Q7-b	119649		6	

Table 3: Rounded up wall-clock times in seconds.

To give an impression of scalability, we can report Prolog queries on a 40M tokens, dependency parsed corpus (Bouma et al., 2010). The setup requires about 13Gbyte of RAM on a 64-bit machine. Loading a corpus takes under a minute when precompiled. Due to first-argument indexing, time per answer does not increase much. Handling of larger corpora remains a topic for future work.

5 Conclusions

On the basis of six queries designed to highlight strengths and weaknesses of query languages, we have demonstrated that querying syntactically annotated corpora using Prolog is straightforward, flexible and efficient. Due to space constraints, the example queries have been rather simple, and many of the more interesting aspects of using a general purpose programming language like Prolog for corpus querying have not been dealt with, such as querying structures between and above the sentence, result categorization, on-the-fly annotation transformation, and the combination of annotation layers. For examples of these and other use cases, we refer the reader to Witt (2005), Bouma (2008), Bouma et al. (2010), and Bouma (Ms). This paper's Prolog code and further conversion scripts will be available from the author's website.

Acknowledgements

This research was carried out in the context of the SFB 632 *Information Structure*, subproject D4: *Methoden zur interaktiven linguistischen Korpusanalyse von Informationsstruktur*.

References

Gerlof Bouma, Lilja Øvrelid, and Jonas Kuhn. 2010. Towards a large parallel corpus of clefts. In *Proceedings of LREC 2010*, Malta.

Gerlof Bouma. 2008. *Starting a Sentence in Dutch: A corpus study of subject- and object-fronting.* Ph.D. thesis, University of Groningen.

Gerlof Bouma. Ms. Querying linguistic corpora with Prolog. Manuscript, May 2010, University of Potsdam.

Thorsten Brants. 1997. The negra export format. Technical report, Saarland University, SFB378.

Stephan Kepser. 2003. Finite structure query - a tool for querying syntactically annotated corpora. In *Proceedings of EACL 2003*, pages 179–186.

Esther König, Wolfgang Lezius, and Holger Voormann. 2003. Tigersearch 2.1 user's manual. Technical report, IMS Stuttgart.

Catherine Lai and Steven Bird. 2004. Querying and updating treebanks: A critical survey and requirements analysis. In *Proceedings of the Australasion Language Technology Workshop*, Sydney.

Torsten Marek, Joakim Lundborg, and Martin Volk. 2008. Extending the tiger query language with universal quantification. In *KONVENS 2008: 9. Konferenz zur Verarbeitung natürlicher Sprache*, pages 5–17, Berlin.

Ulf Nilsson and Jan Maluszynski. 1998. *Logic, programming and Prolog*. John Wiley & Sons, 2nd edition.

Heike Telljohann, Erhard Hinrichs, Sandra Kübler, and Heike Zinsmeister. 2006. Stylebook for the tübingen treebank of written german (tüba-d/z). revised version. Technical report, Seminar für Sprachwissenschaft, Universität Tübingen.

Andreas Witt. 2005. Multiple hierarchies: New aspects of an old solution. In Stefani Dipper, Michael Götze, and Manfred Stede, editors, *Heterogeneity in Focus: Creating and Using Linguistic Databases*, Interdisciplinary Studies on Information Structure (ISIS) 2, pages 55–86. Universitätsverlag Potsdam, Potsdam.