

# GoogleLing: The Web as a Linguistic Corpus

Joseph Smarr  
Symbolic Systems Program  
Stanford University  
jsmarr@stanford.edu

Tim Grow  
Computer Science Department  
Stanford University  
grow@stanford.edu

## Abstract

We describe software to transform any search engine or searchable corpus into a tool for linguistic research with a rich query syntax. We provide support for case sensitive searches, within-sentence and within-N-words match constraints, part-of-speech restrictions on words, and “smart” verb-ending inflection wildcards. The software generalizes the query for the underlying search engine, and then processes the resulting pages with a set of natural language processing tools to extract matching sentences. Preliminary evaluation suggests that this greatly enhances linguists’ ability to use the web as a linguistic corpus.

## 1 Introduction

As access to large repositories of digital text continues to improve, linguists are increasingly relying upon corpus searches to complement and extend personal judgments (Corley et al, 2001, Volk, 2002, Levin, 2002). Consulting “real-world” language uses is an important way to check intuitions about what sounds acceptable, find similarities and differences in the expression of related words, and assess the relative prominence of different expressions.

### 1.1 The potential of the web

The World Wide Web has the potential to be an invaluable corpus for linguistic research, due to its size, breadth, modernity, and universal availability. Search engines such as Google (2002) provide access to the content on the web, but are designed primarily for novice users trying to simply find pages on a given subject.

In order to retrieve specific enough examples of language use to answer linguistic queries, linguists require a richer

and more powerful query syntax than casual web surfers. As Volk (2002) explains, “search engines are not tuned to the needs of linguists” (p.7). Thus the potential of the web as a linguistic corpus has yet to be fully realized because the right tools do not exist.

### 1.2 Managed corpora

The need to search for text with increased precision has led linguists to manually construct corpora and annotate them with part-of-speech tags and/or grammatical structure. For example, the COBUILD corpus (Collins, 2002) allows linguists to search a 56-million-word corpus using wildcards, “near constraints” (e.g. “break within-2-words-of vase”), part-of-speech tags for words (e.g. “break/VERB”), and verb-ending expansions (e.g. “break@” expands to “break OR breaks OR breaking OR broke OR broken”). The British National Corpus (BNC, 2000) has a similar search interface.

Such query options are extremely useful to linguists, but the relatively small size of these managed corpora are an inherent limitation, as is their lack of public availability. Thus the demand to be able to search the web as a linguistic corpus remains compelling.

### 1.3 Our approach

We have taken a step towards addressing this demand by building software to wrap any existing search engine or searchable corpus with a rich query interface and a set of on-the-fly natural language processing (NLP) tools to restrict and refine the results returned.

Using GoogleLing (our initial implementation wraps the popular search engine Google though other search engines could be substituted as we explain below), linguists can search the web using queries

with comparable precision to COBUILD and other managed corpora. The query is first transformed and generalized for use with the underlying search engine. As pages are fetched, the text is extracted and broken up into sentences, which are processed by a set of NLP tools to assess whether they match the rich query. Sentences that match are displayed in the results page, with the query terms highlighted and a link to the source page provided.

## 2 Related Work

There are a number of other researchers attempting to build tools that help linguists use the web as a corpus.

### 2.1 WebCorp

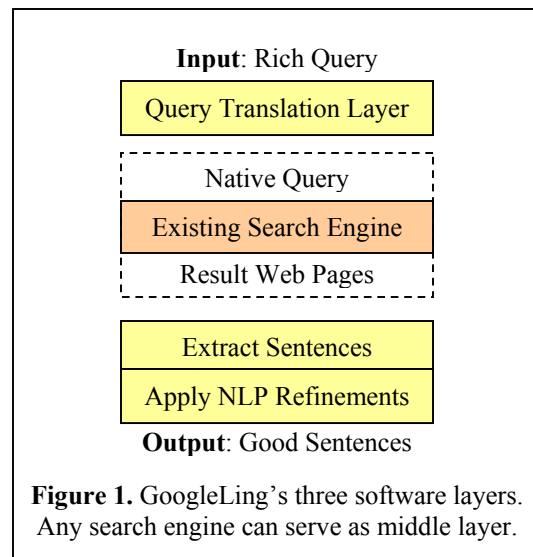
WebCorp (2002) is a project at the University of Liverpool that wraps Google, Altavista, and other search engines and displays results in “key-word-in-context” (KWIC) format where the query term is padded on either side by a few words of context from matching pages. WebCorp allows case sensitive search, wildcards (e.g. “run\*”) and alternatives (e.g. “r[u|a]n” matches “run OR ran”), but does not allow part-of-speech tags or verb-ending wildcards in the query. WebCorp has been heralded as “a major breakthrough” (Rundell, 2000).

### 2.2 KWICFinder

KWiCFinder (2002) is a similar project with the same basic query options, available as downloadable software for Windows. Like WebCorp it does not utilize any NLP to allow for linguistically rich queries.

### 2.3 Gsearch

Gsearch (Corley et al, 2001) uses a similar strategy as our approach to facilitate searching the web for grammatical constructions. The searcher provides a set of context-free grammar rules that define the constituents of interest, then web pages are downloaded and parsed using a fast bottom-up chart parser to find matches on the fly. The authors report that Gsearch has been used successfully by a wide variety of linguistic researchers.



## 3 Our Implementation

The software infrastructure for wrapping search engines to allow for rich queries consists of three separate layers (Figure 1).

### 3.1 Rich Query Language

The input to the system is a rich query language similar to that used for COBUILD and other managed corpora. The query consists of a set of terms that are implicitly ANDed together (i.e. they all must appear in the same sentence). Terms can be single words (e.g. *break*) or contiguous phrases (e.g. “white house”).

Each word can optionally be restricted to a particular part-of-speech (POS, e.g. *break/VERB*). We use the Penn Treebank (2002) POS tag set as well as the “macro tags” NOUN, VERB, ADJ, and ADV that match any of the more specific tags (e.g. VERB matches VB, VBD, VBG, VBN, VBP, and VBZ).

If an @ is placed at the end of a verb, it means that any inflected form of the verb is a valid match. For example, *break@* is equivalent to “*break OR breaks OR breaking OR broke OR broken*”. Any inflected form of the verb may be used with the @ (e.g. *break@* and *breaking@* are equivalent).

The query language also supports “near-constraints” such as *break w/2 vase*, which matches any sentence in which no

more than two words occur between “break” and “vase” (in either order). We do not currently support n-way near constraints (e.g. “A near B near C” is not allowed).

Searches are optionally case sensitive, a trivial feature to implement, but one not available with most search engines. This can be particularly useful for matching proper nouns that are also found as common words (e.g. the last name of Tim Grow).

Finally, results are determined per-sentence rather than per-document as with most search engines. Thus a single web page may contain multiple matching sentences or none at all. A query like `break vase` will only match sentences that contain both words, whereas with most search engines the words must simply appear somewhere on the page.

### 3.2 Layer 1: Query Translation Layer

The first software layer is a query translation layer that maps the rich query language available to the user into a more general query suitable for the underlying search engine. The set of results for the translated query must be a superset of the results for the rich query since after pages have been returned from the search engine, they can only be refined.

In the case of GoogleLing (wrapping Google will continue to be our worked example throughout), this means stripping out POS tags and near constraints and expanding verb-ending wildcards. The latter task is accomplished with a finite state transducer constructed with the Xerox Finite State Toolkit (Beesley & Karttunen, 2003).

We modified an existing morphological analyzer for English (that maps between surface and lexical forms) to allow only verb lexemes and to ignore all other morphological tags (e.g. person, number, tense, etc.) This transducer is then composed with its own inverse, so that any inflected form is mapped down to the root form and then back out to all other inflected forms. The compiled network is loaded and served by a daemon so individual requests are very quick (the original verb is run through the network and the set of outputs is

returned). Results are also cached to minimize redundantly expanding the same verb multiple times.

The final step of the query translation layer for GoogleLing is to mark all words with a + to prevent stopword removal (since function words are often extremely important to linguists) and to prefix the entire query with `allintext:` so that only the text of the web page is searched.

### 3.3 Layer 2: Pluggable Search Layer

Once the query has been translated, it is sent to the underlying search engine to retrieve a list of matching web pages. In our case we use the GoogleAPI to search Google and fetch the pages using their cached page service. The list of matching URLs are then downloaded and processed in sequence to find matching sentences.

Since downloading web pages is by far the slowest part of the search process, we spawn several separate threads to queue results and download pages in parallel. We download at most 50 pages per query, with the repeatable option to “get more results” which downloads up to 100 more.

Asynchronous fetching improves performance by allowing the I/O intensive process of downloading pages and the CPU intensive process of manipulating the downloaded text to be run simultaneously. The tradeoff is that sometimes more pages are downloaded than required to find a given number of matching sentences. The enforced page cutoffs keep this cost to a minimum. Parallel fetching significantly improves performance when a high-bandwidth link to the Internet is available. From Stanford, GoogleLing can fetch ten pages in parallel in the same time it takes to fetch two pages in series.

### 3.4 Layer 3: Result Refinement Layer

Web pages returned by the search engine are first converted from HTML to text by collapsing whitespace (including newlines), stripping HTML tags (including the text inside code-block tags like `SCRIPT` and `HEAD`), and adding newlines around block tags like `P` and `DIV`. Common HTML

entities are also resolved (e.g. “; → “) and unknown entities are simply removed.

The text of each web page is then tokenized into words using the Penn Treebank-style tokenizer available in the JavaNLP code repository produced by the Stanford NLP group. The list of words is then segmented into sentences by looking for common sentence boundary tokens such as period, question-mark, exclamation-mark, or newline. The tokenizer intelligently handles non-sentence-boundary punctuation such as periods in numbers, abbreviations, and titles, so simply looking for these remaining boundary tokens is sufficiently accurate. The most common sentence boundary detection error is missing the final quotation mark of a sentence, but this is irrelevant for matching queries.

For efficiency, before any further processing is done to each sentence, it is first examined to ensure that it at least contains each word and phrase in the query (i.e. ignoring POS tags and near constraints). Unlike with Google, multi-word phrases are not allowed by our system to include any intermediate punctuation. For example, the phrase “Google it” will match across a sentence boundary in Google such as “Use Google. It is...” whereas this is not allowed in our system.

Matching sentences are then automatically tagged with the part-of-speech for each word using Toutanova’s (2000) Maximum Entropy POS tagger, also available in the JavaNLP repository. The tagged sentence is once again examined to ensure that each word and phrase appears, but this time the POS tag constraints and near-constraints are also enforced. Query words without a POS tag specified are assigned the ANY macro tag.

Near-constraints are enforced by taking the set of matching regions for the terms on both sides and keeping only those which have at most the given number of intermediate words (in either direction). This way the annotated sentence will only highlight the instances of terms and phrases that also satisfy the near-constraints.

Sentences that satisfy all of the rich query constraints are returned provided that a duplicate sentence has not already been returned. The search terminates when either a user-specified number of result sentences have been returned (e.g. 10) or after a given number of web pages have been processed (e.g. 50), in which case the results obtained so far are returned.

### 3.5 User interface/implementation

GoogleLing is available over the web and is served and controlled by JavaServer Pages running on the Tomcat application server. All the code is written in Java except for the verb-ending finite state network, which is written using the Xerox Finite State Toolkit.

The search interface is similar to most search engines: a text box for the query, a case-sensitive checkbox, a pull-down menu for the number of matching sentences to return, and a search button. Once a query has been dispatched, result sentences appear progressively as they are found to minimize perceived latency. The matching terms and phrases are highlighted in bold with a yellow background. URLs to the source page are displayed under each sentence. When the sentences have been returned (or the maximum number of pages have been downloaded), there is an option to “get more results” which continues to download pages.

## 4 Evaluation

We present a preliminary evaluation of the interface design, query-serving efficiency, and usefulness of results. Our test users were primarily Stanford PhD students in Linguistics. While quantitative results for the usefulness of this system are difficult to obtain and perhaps premature at this point, we are encouraged by this preliminary feedback from our intended audience.

### 4.1 User Interface

While most popular search engines have a single text box for the query, some linguistic corpora (e.g. Lexis-Nexis, 2002) have more complex user interfaces that separate search terms with pull-down menus for boolean connectors or near-constraints. Our users

said in general they prefer the single text box (used by BNC and COBUILD among other linguistic corpora) because of its flexibility, provided the query syntax is briefly explained on the web page. We have thus opted for the single box and provided sample queries and a query syntax explanation with search tips on the front page of the web site.

Matching sentences are displayed one per line with query terms highlighted and the URL below. Note that this differs somewhat from the standard KWIC format in which the keyword is centered with a few words of context on each side. With multi-term queries, the “keyword” need not be a single word or even a contiguous set of words, so centering is not feasible. The linguists we talked to also preferred seeing the entire sentence, saying that a few words of context is often not enough to interpret the utterance.

Originally we didn’t display the URL for each sentence (we simply provided a “go to page” link) but user testing revealed that users evaluate the reliability of a web page by its URL. Specifically, several users wanted to avoid confusing professionally edited text with, as one user put it, “[a page by] some foreign guy that barely knows English”. Thus we include the URL below each sentence, in a smaller font and lighter color to minimize distraction.

Users also expressed frustration at having to do “find in page” to locate the sentence in the source page when they wanted additional context. Thus we changed our links to use Google’s cached page viewer, which allows us to highlight the query terms in the source pages as well as the sentences we extracted.

## 4.2 Query-serving efficiency

A major concern for this project was whether downloading web pages, processing the text, and collecting a suitable number of matching sentences would be too slow for practical use. WebCorp for example allows search results to be e-mailed so that the user doesn’t have to wait, and provides a disclaimer that results may appear slowly.

The time it takes to return results depends on how many web pages need to be downloaded and how much processing needs to be done to each page. Common words and shorter queries lead to fast searches, whereas complex queries with many constraints take more text to satisfy.

Another important consideration is how much the translation from the rich query to the native query generalizes the set of possible results. The difference between the queries `devour` and `devour/VERB` is likely to be small because most (if not all) uses of `devour` are verbal. In contrast, `code/VERB` is very slow because nearly also uses of `code` on the web (certainly the ones with the highest PageRank) are noun uses.

A baseline for performance is the time it takes to execute a Google search and download the first web page, since this is as fast as the entire process could ever hope to be. In our experiments this usually took about 10 seconds. To assess the range of processing times, we chose the query `google` as a fast example and `code/VERB` as one that would likely be slow.

It takes GoogleLing an average of 15 seconds to return 10 results for the query `google`. In the process, GoogleLing searches 334 sentences across 6 pages. For comparison, WebCorp takes an average of 12 seconds to return 10 matches for `google`. Thus we feel confident that the additional text processing we perform is not a significant source of latency (the slowest part is actually converting HTML to text).

Searching for `code/VERB` does not return any results after searching 50 pages. As described above, our system stops after processing 50 pages, but allows the user to “get more results” which searches an additional 100. Downloading and processing the first 50 pages for `code/VERB` involves processing 4307 sentences and takes an average of 59 seconds.

Finally, to test the expected latency for a complex, linguistically motivated query, we searched for `break@/VERB w/2 vase` (see Fillmore, 1970, among countless others). GoogleLing returned 10 results after

processing 2305 sentences in 28 web pages. Average processing time was 22 seconds. Users in general were quite positive about the speed of the system (perhaps spurred on by our nervousness about it), though as one user glibly put it, “faster is *always* better!”

#### 4.3 Overall utility for linguists

The linguists we talked to all expressed immediate enthusiasm at the prospect of having access to a tool like GoogleLing. Roger Levy expressed the apparently common sentiment that “I periodically hit a point in Googling that I say, if only I could search for POS sequences!”

Prof. Beth Levin noted that searching with POS tags is very important when researching verbs like “draw” that commonly show up as nouns and adjectives like “dry” that commonly show up as verbs. Levin estimates that having to sort through results with the wrong part-of-speech is one of the biggest hurdles to using Google directly for linguistic research. Sometimes one can help coerce Google into giving (transitive) verbal uses of a word by appending “the” or “it” but even this still produces many spurious results, it doesn’t work with all verbs, and it artificially limits the space of possible results.

Levin also considers the verb-ending wildcards to be extremely useful, since linguists are often interested in the full range of uses for a verb and searching for each inflected form separately (or even writing them all down with an OR) is usually too much trouble.

Another feature that received positive feedback was matching per-sentence rather than per-document. The common sentiment was that using Google alone, one can only specify too tight a context for two words (i.e. a contiguous phrase) or too loose a context (i.e. an entire document).

Two common tasks are searching for verbs and their arguments (e.g. break with vase or spray with wall) and searching for multi-word expressions (e.g. take advantage of). In both cases the sentence scope is the most appropriate for matching the terms. For example, the query `take@/VERB`

advantage of returns results like “Home’s design **takes** full **advantage** of its oceanside location” that it would be difficult to find with direct phrase queries.

#### 4.4 Limitations of current system

While initial user tests were overall quite positive, a number of limitations were also identified. In each case we describe the problem and consider what would be required to address it.

It is still difficult to find words of a given part-of-speech when another part-of-speech is more common. For example, finding verbal uses of `code` is difficult because most uses of the word are nouns, and that’s what Google returns. While technically the verbal uses are buried somewhere in the results, it takes prohibitively long to search for them.

Even using a query like `code@/VERB` returns very few results because most of what comes back has noun uses of `code`. One can of course still search for `coding` or `coded`, but then the only advantage over Google is pulling out the individual sentences. This is a fairly fundamental problem with any post-processing system, and would be difficult to fully solve without indexing words with their POS (which is obviously infeasible in the case of Google).

A related problem is that the sentences returned do not necessarily constitute a “balanced” or “representative” sample of all the uses on the web. In particular, if the first few pages contain many references to the same word, they are all returned. The seriousness of this problem thus depends to a certain extent on the density and commonness of the query term(s). The broader problem of uniform sampling on the web has received considerable attention recently (Bar-Yossef et al, 2000, Henzinger et al, 2000). Perhaps this research will yield a general tool for uniform web page sampling, which our software could be wrapped around.

A minor problem with our verb ending expander is that it only works for verbs in its fixed lexicon. There is no “guesser” to apply general rules of morphology in the

case of unknown words. Thus for example “Google@” cannot be expanded to “Googles”, “Googled”, or “Googling”, even though the morphology is fully regular. Several finite state morphological analyzers include guessers that should be suitable for use here, but the one we had access to didn’t map down to the base form (it only tagged the inflected form) so we were unable to use it for verb ending expansion.

A final limitation is our inability to allow for general wildcards like *run\** (which would match any word starting with *run*). Since Google does not support wildcards, there is no sensible way in the query translation layer to generalize the query without eliminating the word altogether. In cases where there were enough supporting query terms, this might be feasible but in a query like *run\** one would end up with an empty query to send to Google. The verb-ending wildcard is a viable alternative in some cases, but there are others where wildcards would still be quite useful.

## 5 Future Work

User testing suggested a number of additions and enhancements to the rich query syntax that would further increase the usefulness of the system. We also propose a possible solution to the problem of finding words with uncommon parts-of-speech.

### 5.1 Word- and phrase-level wildcards

As mentioned above, intra-word wildcards like *run\** are difficult to implement with the current system unless the underlying search engine also supports wildcards. Our users reported that an even more useful feature would be an entire-word wildcard (such as *break \*/NOUN*) so that they could search for verbs with particular argument patterns, noun-noun compounds, and similar phenomena. While there is still the problem of removing too much from the query during translation, a whole-word wildcard would be trivial to implement (just look for an intermediate word in the results) and given sufficient context in the query might yield results quickly enough to be useful.

Taking this idea one step further, it was expressed by several linguists that if one could “chunk” the sentences to allow for “phrasal wildcards” like *NP* or *PP*, one would have an even better tool for finding particular argument structures. Existing text chunkers (e.g. Ramshaw & Marcus, 1995) could likely be used in the same way that we now use the POS tagger to enable such queried to be serviced.

### 5.2 Semantic role constraints

In addition to looking for words with a particular grammatical function (e.g. part of speech), linguists are interested in words that play a particular semantic function (such as the agent, instrument, or location of an event). Recently there have been attempts to create corpora annotated with a rich set of semantic roles (e.g. FrameNet).

While automatically assigning semantic roles is considerably harder than automatically assigning parts-of-speech, technically speaking it is no more difficult to allow queries like *hammer/INSTRUMENT* and to run a “semantic role tagger” than to do the same with POS tags. Given the extensible framework of our system, it should not be hard to integrate such additional forms of automated linguistic analysis as they become available.

### 5.3 Collocations for guiding search

When faced with the problem of finding a word with an uncommon part-of-speech (e.g. *code/VERB*), a solution that is often effective is to imagine words that would only appear next to the target word when used in its desired part-of-speech. For example, searching for “code the” or “code it” or “to code” is more likely to return verbal uses of *code* than simply searching for *code* by itself. Similarly, searching for *hit/NOUN* is difficult but looking for “smash hit” helps a lot.

Ideally one could automate the process of finding useful neighboring words using an offline tagged corpus. One would find all the examples of the given word, then for each possible tag and neighboring word (before and after the target word), count the

number of times the neighboring word occurs with the desired tag for the target word, and compute  $P(\text{POS}|\text{neighbor})$ . This is essentially equivalent to constructing a Naïve-Bayes classifier using the neighboring words as features.

The neighboring words that are most indicative of the given part of speech could then be automatically added to the query in cases where the word alone was not enough. The hope is that this would make results where the target word had the desired part-of-speech more likely, while removing the burden on the user to guess likely neighboring words.

## 6 Conclusion

As more and more of our culture and progress as a society is captured on the web, it becomes increasingly important to have powerful tools for social scientists to find and study its content. Capturing and constructing curated corpora will never be a substitute for directly analyzing the always growing, always changing web. Thus we need tools that can take the web as is and analyze chunks of it as needed.

We have taken a step in this direction by creating a tool for searching the web as a linguistic corpus. NLP technology is now sufficiently accurate and efficient to create the illusion that one is searching a managed corpus like COBUILD or the BNC with a powerful query syntax. The web is the largest and most up-to-date collection of text available. It is our hope that as tools like GoogleLing continue to improve, linguists will be able to realize the full potential of the web as a tool for scientific inquiry.

## 7 Availability

GoogleLing is available to the public at <http://nlp.stanford.edu/googleling>.

## References

Bar-Yossef, Z., Berg, A., Chien, S., Fakcharoenphol, J., & Weitz, D. (2000). Approximating aggregate queries about web pages via random walks. In *Proceedings of the*

*26th International Conference on Very Large Data Bases (VLDB)*, pages 535-544.

Beesley, K. R., & Karttunen, L. (2003). *Finite State Morphology*. Stanford: CSLI Pubs.

British National Corpus. (2000). Simple Search of BNC-World. Available online at <http://sara.natcorp.ox.ac.uk/lookup.html>

Collins COBUILD. (2002). The Bank of English. Searchable sample available online at <http://titania.cobuild.collins.co.uk/form.html>

Corley, S., Corley, M., Keller, F., Crocker, M., & Trewin, S. (2001). Finding Syntactic Structure in Unparsed Corpora: The Gsearch Corpus Query System. *Computers and the Humanities*, 35, 81-94.

Fillmore, C. J. (1970). The Grammar of Hitting and Breaking. In Jacobs, R. A., & Rosenbaum, P. S. (eds.) *Readings in English Transformational Grammar*. Cambridge: Cambridge University Press.

Henzinger, M. R., Heydon, A., Mitzenmacher, M., & Najork, M.. On near-uniform URL sampling. (2000). In *Proc. Ninth International World Wide Web Conference*, Amsterdam.

KWiCFinder. (2002). <http://miniapolis.com/KWiCFinder/KWiCFinderHome.html>

Lexis-Nexis Academic Search Page (2002). [http://web.lexis-nexis.com/universe/form/academic/s\\_guidednews.html](http://web.lexis-nexis.com/universe/form/academic/s_guidednews.html)

Levin, B. (2002). Personal correspondence.

Penn Treebank. (2002). Tag set listed at [http://www.ling.upenn.edu/courses/ling001/penn\\_treebank\\_pos.html](http://www.ling.upenn.edu/courses/ling001/penn_treebank_pos.html)

Ramshaw, L.A. & Marcus, M.P. (1995). Text Chunking using Transformation-Based Learning. In *ACL Third Workshop on Very Large Corpora*, pp. 82-94, 1995.

Rundell, M. (2000). The biggest corpus of all. In *Humanising Language Teaching*, Issue 3.

Toutanova, K., & Manning, C. D. (2000). Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger. *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000)*, pp. 63-70. Hong Kong.

Volk, M. (2002). Using the web as a corpus for linguistic research. In Pajusalu, R., & Hennoste, T. (eds): *Tähendusepüüdja. Catcher of the Meaning. A festschrift for Professor Haldur Öim*. Department of General Linguistics 3, University of Tartu.

WebCorp. (2002). <http://www.webcorp.org.uk>