Integrating Web Resources and Lexicons into a Natural Language Query System*

Boris Katz, Deniz Yuret, Jimmy Lin, Sue Felshin
Rebecca Schulman, Adnan Ilik, Ali Ibrahim, Philip Osafo-Kwaako
Massachusetts Institute of Technology
Artificial Intelligence Laboratory
545 Technology Square, Cambridge, MA 02139, USA
{boris, deniz, jimmylin, sfelshin, rebecka, adnan, aibrahim, osafo}@ai.mit.edu

Abstract

The START system responds to natural language queries with answers in text, pictures, and other media. START's sentence-level natural language parsing relies on a number of mechanisms to help it process the huge, diverse resources available on the World Wide Web. Blitz, a hybrid heuristic- and corpus-based natural language preprocessor, enables START to integrate a large and ever-changing lexicon of proper names, by using heuristic rules and precompiled tables of symbols to preprocess various highly regular and fixed expressions into lexical tokens. LaMeTH, a content-based system for extracting information from HTML documents, assists START by providing a uniform method of accessing information on the Web in real time. These mechanisms have considerably improved START's ability to analyze real-world sentences and answer queries through expansion of its lexicon and integration of Web resources.

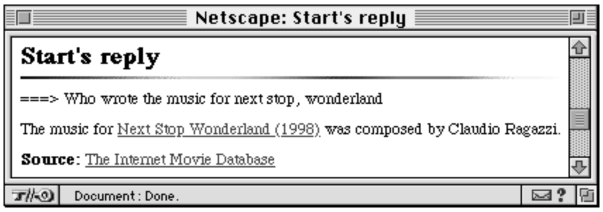
1. Introduction

With recent advances in computer and Internet technology, people have access to more information than ever before. As the amount of information grows, so does the problem of finding what one is looking for. We believe that the most natural form of communication and information access for humans is natural language (NL). We strive to create systems which can answer such queries.

The START system has answered over a million English questions on the World Wide Web since 1993.[1] START (SynTactic Analysis using Reversible Transformations) analyzes sentences as embedded *ternary expressions* and stores them in a knowledge base.[4] An information segment can be *annotated* with a computer-analyzable collection of NL sentences and phrases that describe it.[5] START analyzes these annotations and when a query matches a representation derived from an annotation, START responds with the annotated information fragment. START's annotation mechanism gives it the ability to store and retrieve information of any complexity and in any media.

In order to respond to a query, an NL system must understand the question, find the resource with the answer, and extract "just the right information" to present to the user. NL systems have traditionally accessed data in a uniform database and included lexicons containing all relevant words. Due to the explosion of the World Wide Web this approach is no longer adequate. Given the size, complexity, and diverse nature of information on the Web, three bottlenecks stand in our way to making use of the Web:
1. Special-purpose extra-linguistic constructions defeat or bog down morphological/lexical/syntactic analyzers.
2. Ever-changing myriads of proper names are not amenable to storage in a traditional static lexicon.
3. Incorporating knowledge distributed across the Web requires flexible systems for data access and combination.

This paper describes solutions we have implemented to address the above problems. For example, START answers the query "who wrote the music for next stop, wonderland"



even though word-for-word, the sentence is not parseable according to standard English grammar, and one of the words in it is not likely to be found in most lexicons.

To address the first two bottlenecks, we created Blitz, a hybrid heuristic- and corpus-based NL preprocessor that eases the burden on the parser by recognizing human names,

*This research was supported in part by DARPA under Air Force Research Laboratory grant F30602-98-1-0036.

[1] http://www.ai.mit.edu/projects/infolab

institutions, places, addresses, currencies, etc.

To address the third bottleneck, we created LaMeTH, a content-based system for extracting information from HTML documents that provides a uniform method for accessing real-time information on the Web.

Integrating these mechanisms with START has considerably improved START's ability to analyze real-world sentences and to answer hundreds of thousands of live queries submitted over the Web. Thus by blending linguistic-based and corpus-based methods of language processing, we gain the best of both worlds and achieve a far more powerful system than could be built with either method alone.

2. Blitz

Linguistically motivated natural language parsers have been plagued by the vast complexity of language. Parsers which attempt to handle the richness of unrestricted language often grow to contain unmanageably large grammars. Other parsers which reduce language to a simple and tightly constrained linguistic model either cannot analyze syntactically odd structures or cannot decide between multiple interpretations. Certain NL constructions, such as numbers, dates, times, emails, URLs, and proper names, while quite simple in form, contribute significantly to the complexity of grammars. These constructions do not exhibit the typically richly hierarchical structure of language, and therefore are well-suited to heuristic-based non-linguistic analysis in the absence of surrounding context. These structures can be analyzed by a preprocessing module and converted to single tokens, vastly speeding and simplifying parsing.

Other NL constructions, particularly names, present a problem to linguistically motivated parsers because their structures are extraordinarily ambiguous and because they involve large numbers of ever-changing vocabulary items.

To solve these problems, we have created special purpose pattern recognizers to extract such constructions from free text and return results in a uniform structure.

These recognizers can be divided into two classes:

1. Heuristic recognizers are built to recognize open classes of constructions like addresses, dates and numbers. It is impossible to enlist all members of such classes. They have to be recognized based on partial cues such as special characters (e.g., the @ sign of an email address or the capitalization marking a generic proper noun) or small fixed classes of words (e.g., names of numbers or months).
2. Symbol-table recognizers, in contrast, attempt to find matches using a list of all members of a given class of symbols. Symbol tables permit quick recognition of non-dictionary words ("Reebok") and tokenization of phrases used as names ("Gone with the Wind"). We have collected large lists of people, place, and institution names, titles of books, movies, songs, and so forth.

These recognizers and their control module comprise Blitz, a hybrid heuristic- and symbol-table-based NL preprocessor. Blitz components are compartmentalized in layers, yielding a highly customizable modular system. Ultimately, all frames are passed back to START (or any NL system), endowing it with the ability to understand sentences that it otherwise would not be able to understand.

2.1. Methodology

These premises and philosophy underlie Blitz:

Minimal linguistic and lexical knowledge Blitz's heuristic component recognizes typographical properties such as case and certain closed classes of words, e.g., names of months, and employs simple rules for generating constructions, e.g., a month name and ordinal represent a date ("June 3rd"). Such rules don't result in much overgeneration because most special constructions take highly defined forms. Other components which recognize fixed tokens access lists of symbols, e.g., proper names, compiled from databases without reference to significant linguistic knowledge.

Supplementation, not Replacement Blitz was not designed as a standalone system, but rather as a component of an NL system which assists in parsing and understanding. The NL parser, equipped with greater syntactic and semantic knowledge, will consider each suggestion and attempt to incorporate it into its analysis of the sentence.

Compartmentalization Blitz components are isolated from each other in independent modules that can easily be interchanged and switched on or off. This architectural design allows Blitz to be specifically adapted to any application. This compartmentalization strategy leads to a system that is easily fine-tuned, maintained, and improved.

Comprehensiveness and accuracy Blitz's heuristics recognize a wide range of constructions which are syntactically impoverished and are limited to a relatively small number of forms; not only are they composed from closed category lexical items, but in addition, it is possible to enumerate the rules for forming them. Blitz can very accurately extract the information within each recognized token, such as the value of a written number.

Recall is (locally) more important than precision All suspected special constructions are detected by Blitz, even under the threat of overgeneration. This is acceptable because a true NL parser will restore precision by deciding the final treatment of all tokenized constructions, employing semantic and linguistic knowledge and aided by confidence values provided by Blitz. (See Section 2.6.)

2.2. Frames

Blitz communicates extracted tokens in the format of a "frame" which encodes the lexical information for the token, following this template: `(type "`*string*`" :span (`*begin end*`) :symbol` *symbol* `:`*attribute value* `...)`

type indicates the syntactic category of the token. *string* is the token as entered by the user and *span* gives the token's character position in the input sentence. *symbol* is the token as found in a database, if any; the string and symbol may differ if the Blitz has used its "fuzzy matching" to account for minor differences such as case. Finally, an arbitrary number of attribute/value pairs contain extracted information specific to the type. For example:

```
(number "4" :span (0 1) :value 4)
(propernoun "gone with the wind" :span (0
18) :symbol "Gone with the Wind" :database
"imdb-movie")
```

2.3. Heuristic layer

The heuristic layer of the Blitz system consists of several independent modules. This design facilitates the removal, addition, or improvement of any module without drastic changes to the system architecture:

Email Looks for @ sign and domain endings.

URL Looks for prefixes such as "`http://`", "`mailto://`".

Numbers Looks for numerals, numbers, and combinations.

```
(number "three hundred sixty-fourth" :span (0 25)
:value 364 :notation ordinal)
(number "42.2 million" :span (0 11) value 42.2+e7
:notation natural)
```

Because a single number may be written with the conjunction "and", it is difficult to separate cases of two actual numbers from one single number constructed with "and". True disambiguation may be impossible without additional insights offered by context and grammar.

Proper Names Looks for capitalized words optionally separated by a small, fixed set of connectors. See Section 2.4.

Time `(time "7:12 pm" :span (0 6) :hour 7 :minute 12 :time pm)`

Date `(date "Friday, May 13, 1998" :span (0 24) :day Friday :month May :date 13 :year 1998)`

Address `(address "77 Mass. Avenue" :span (0 15) :number 77 :location "Mass. Avenue")`

Quantity `(quantity "$23.5 billion" :span (0 13) :value 2.35e+10 :unit $)`

2.4. Proper names

In truth, the extraction and disambiguation of proper names[2] is extremely difficult to accomplish in the absence of context. Names by their very nature are deeply intertwined with the basic lexical and semantic fabric of the sentence; hence it is difficult, if not impossible, to understand and extract such information successfully without processing the entire sentence with a full parser. The following five sentences demonstrate a small sample of such ambiguities.

(1) The New York Times is a newspaper.

(2) In The New York Times today there was an article about artificial intelligence.

(3) For Better or Worse is a popular comic strip.

(4) The copy of the New York Times John read was missing an entire section.

(5) Is Mary Joe Frank's daughter?

"The New York Times" is the full name of the popular newspaper, but it is impossible to derive such information except with a priori knowledge. The beginning of every sentence is capitalized; therefore heuristics cannot determine whether or not that word is part of a name. This is also the problem encountered in sentence (2), where a preposition might be mistaken for part of the actual name. In this case, disambiguation is difficult unless there exists a large list of common words that should be excluded from any name, which might include all prepositions. However, even that scheme is far from foolproof, because prepositions can legitimately begin a name, as in sentence (3). Sentence (4) further demonstrates compounded ambiguity when two names are adjacent to each other, unbroken by any punctuation. Finally, there are truly ambiguous sentences, such as (5), where it may be that Mary is the daughter of Joe Frank, or that Mary Joe is Frank's daughter.

These ambiguities frustrate or confound heuristics and linguistic parsers, but knowledge of the world in the form of symbol tables nearly solves this problem; see Section 2.5. We might wonder, then, what the value is in a heuristic module. First, it allows the system to answer more intelligently in the negative; e.g., we can reply to "Tell me about Xloij Plkjw" with "I don't know anything about Xloij Plkjw" rather than "I don't understand you." Second, a symbol table may recognize a token without recognizing its internal structure, yet a heuristic can parse the token; e.g., we can reply to "Tell me about Pat Jones" with "Pat Jones is a staff member. Pat's office number is 5023." Third, a heuristic module can reinforce the result of a symbol table module, tipping the balance in favor of a particular interpretation of the input, e.g., "I bought an apple" (the fruit) vs. "I bought an Apple" (the computer).

The Blitz name module looks for sequences of adjacent capitalized words that may potentially be separated by a very small list of connecting words such as "and," "the," and

[2] "Proper noun" is sometimes used strictly to refer to a single noun used as a name. We use the term "proper name" to make it clear that we are discussing full noun phrases which function as names. These phrases may be made of any number of constituent words, whether nouns, e.g., "Queen Victoria", and/or other, e.g., "For Whom the Bell Tolls". With the exception of sometimes-optionally-lowercase connectors such as articles and prepositions, constituent words have fixed case—traditionally capitalized, although there are cases such as all-uppercase acronyms ("NASA"), bicapitalized words ("MacGregor"), and odd exceptions ("e.e. cummings").

"of." All combinations of the entire token are then enumerated, in anticipation of the ambiguities mentioned above; e.g., New York Times would lead to "New York Times," "New York," "York Times," "New," "York," and "Times." Since the name module detects all combinations of capitalized words, it may return a large number of frames. "Confidence values," discussed in detail in Section 2.6, below, are used to choose among frames.

2.5. Symbol table layers

Because heuristics and linguistic parsing are only effective in extracting constructions according to their form, it is necessary to incorporate other knowledge sources for the recognition of names which have no set form. The easiest way to accomplish this is through lists of symbols for individual categories; e.g., lists of all famous people, Fortune 500 companies, movie titles, etc. The rich resources available on the World Wide Web make it possible to create such long symbol lists with relative ease.

In addition to simple heuristics, Blitz consults multiple common proper names databases. When a sentence is preprocessed, it is checked against the database for matches. The matches are also packaged in frames and ultimately returned to the natural language system.

Compartmentalization is also relevant in the context of symbol tables. Due to the potentially huge number of symbols in each database and the number of databases, it is imperative to isolate knowledge sources from each other to ensure scalability and flexibility. For example, the movies database should be separate from the database of Fortune 500 Companies. This modularization of data assists in the management of complexity, making the modification of individual databases easier.

There are several advantages to storing symbols in database format. The first is that large amounts of data can easily be added or changed, allowing great flexibility in preprocessing applications. More importantly, however, storing additional information about symbols is possible with this scheme. For instance, the symbol "Gone with the Wind," stored as a movie title, could also contain information about the director, date and cast of the movie. Such information can be passed on to a natural language engine. And for databases on the World Wide Web, URL information can be included, so that such symbols may be hyperlinked. (Another program, LaMeTH (see Section 3) enables convenient, real-time access to such data.)

2.6. Confidence and conflict resolution

Blitz overgenerates symbols because it works without regard to context, because some input is inherently ambiguous, and because identical symbols can be detected by more than one means (e.g., by both the name module and a symbol table of proper names). Blitz errs on the side of false positives when detecting symbols, leaving the NL parser ultimate responsibility for ruling out unwanted symbols. Nevertheless, Blitz evaluates the likelihood and accuracy of frames, insofar as it can, to assist the parser, returning its calculations as "confidence" values within frames.

In some cases, a heuristic module or symbol table can decide confidence without reference to other modules. For example, the name module always assigns lower confidence to a name at the beginning of the input, since the first word might be capitalized purely because it starts the sentence. Also, confidence values are adjusted appropriately if the entire input is in uniform uppercase or lowercase.

In other cases, Blitz can adjust the confidence of a frame based on the presence of another frame. For example, given "We went to a concert on May 1st," Blitz can lower the confidence on "May" as a month because a month name is highly unlikely to occur next to a possible date ordinal without being part of the larger date. On the other hand, given "Profits were high this year for Dewey, Cheatham, and Howe," Blitz has no way to assign higher probability to either the three-frame or one-frame interpretation (unless the symbol table contains the symbol).

2.7. Combining frames

Blitz can combine frames and reduce overgeneration, and in some cases can assign higher confidence when combining frames. For example, given "who wrote Gone With The Wind," Blitz can combine the heuristically derived proper name frame with the symbol table frames, reducing the total number of frames.

2.8. Using confidence values

A natural language parser will need some minor interface code in order to integrate information supplied by Blitz. The parser will likely want to combine its own lexical and syntactic knowledge with Blitz's confidence values in order to decide on the proper interpretation of the input. Thus it may prefer Blitz's interpretations in some cases:

* I saw [det The] [pronoun Who] in concert.
  I saw [NP (propernoun "The Who")] in concert.
 but not in others:
  [aux May] I go now?
* [NP (date "May")] I go now?

In addition, the parser can specify a confidence threshold to prevent Blitz from returning low-confidence frames. This greatly reduces the problem of overgeneration.

3 LaMeTH

START's goal is to be able to understand the structure and content of information such that START can index it automatically, and respond correctly and concisely to queries about it. Blitz assists START in understanding queries that contain unknown lexical items. However, this is not enough to complete the query answering process; an equally powerful system must be available to access relevant knowledge and produce a satisfactory answer for the user.

Traditional methods of storing and representing knowledge are no longer adequate in light of the explosion of information on the Web. Numerous Web databases provide a tremendous amount of information worth exploiting. However, data in these source databases is often not accessible directly by START; instead, information is returned from the remote server in the form of a complex HTML document. The desired knowledge is often buried in the midst of other irrelevant information, rendering its extraction difficult. Although it is possible to present the user with this entire HTML document as an answer, this is far from desirable because the majority of the document frequently bears no relevance to the user query.

One popular attempt to solve this problem is through use of relational databases. While information stored in this form is easy to access and these databases support relatively complex operations, the original problem of populating the database remains. Fully automatic parsing of HTML documents would require solving not only the problem of parsing and understanding human language, but also the problem of parsing visual document structure into logical document structure. Yet manual scanning of HTML text is difficult and tedious. Furthermore, it is often impractical to store Web-based information sources locally, because it essentially duplicates the remote database server; at times, the sheer size of the database limits local storage. Finally, storage of data in local databases erases the dynamic and active nature of the Web, where information is constantly updated and renewed. In order to mine the vast quantities of data on the Web, we need a fast and easy method of extracting specific information from HTML documents as needed.

The Web's population of numerous large "databases" of documents does have fairly regular format and structural content which is clear to a human reader. LaMeTH bridges the gap between our ability to automatically recognize knowledge and the complexity of HTML documents by making it easy for a human to write short scripts to describe the location of relevant knowledge in these untraditional "databases." After START understands a particular query, it calls LaMeTH to retrieve the requested piece of information, live and directly over the Web from a remote server, without having to store any information locally. For example, given a query such as "Who wrote the screenplay for Next Stop, Wonderland," START uses LaMeTH to fetch the "writing credits" attribute of the symbol "Next Stop, Wonderland." Thus LaMeTH serves as START's portal into knowledge reserves on the World Wide Web.

3.1. Creating LaMeTH scripts

It is difficult for humans to extract information using popular methods of HTML extraction which involve using regular expression-like scripts to search through HTML code or parsers which separate each tag and perform matching on them, because it requires detailed knowledge of HTML syntax, and because they do not allow extraction by "visual inspection," the most intuitive method for humans. The scripter can only view raw HTML as a linear, text-only representation of hierarchical material, understandable only through close reading, whereas displayed HTML can be quickly skimmed for textual or visual elements, according to the material the scripter is trying to locate, or the scripter's personal preferences in analyzing information.

The LaMeTH environment provides tools and methods for a human to expand START's knowledge in an intuitive manner based on content elements and layout: it provides an interface which assists the user in the extraction process by automatically annotating an HTML document. This simple interface creates a duplicate version of the input HTML document which contains annotations marking up relevant content elements. Each content element is assigned a linear reference number, which the scripter can use to index the element. This further simplifies the analysis of a Web document, making it as easy as visually locating the desired element and reading the adjacent number.[3]

3.2. The LaMeTH scripting language

The scripting language works as follows:

- LaMeTH recognizes the subset of HTML elements which are specifically relevant to the structure of a document in terms of data extraction. It recognizes headings, paragraphs, tables, lists, and text level elements such as emphasis. In addition to recognizing "logical" structural elements, LaMeTH recognizes "visual" elements such as tables, boldface and italics, and recognizes any series of text bigger than the surrounding text in size, such as a heading.
- Document elements are referenced in the scripting language by type. For example, if a document contains a heading followed by two paragraphs, one would index the second paragraph as "paragraph 2," not as "element 3."

[3]Other visually-based HTML extraction tools exist. For example, Netscape, Excel, and Word all have tools for visually creating extraction scripts. These tools are in many ways comparable to LaMeTH but are hard to integrate with other systems.

• LaMeTH's scripting language flexibly allows the contents of an HTML document to be referenced either hierarchically or linearly. HTML is hierarchical by nature, and in referencing elements hierarchically, one refers to the set of indices which specify an element's nested location, e.g., the fifth table within the second table. In many cases, it may be more convenient to view the document by its linear text structure rather than its hierarchical HTML structure. In the linear method, hierarchical structure is ignored, and elements are numbered according to the order in which they begin. Thus, if a nested table appears at the beginning of another table, the nested table's index would be one higher than the outer table's index.

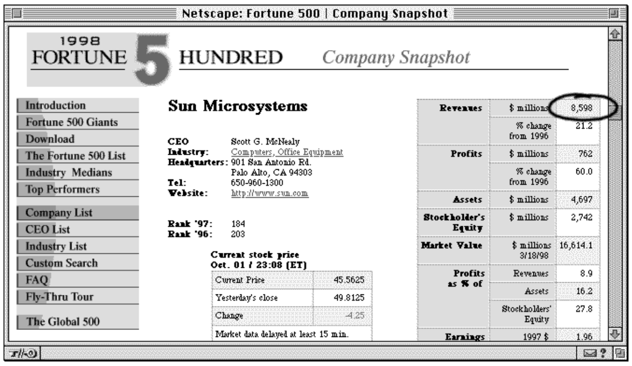
• LaMeTH performs detailed analysis on lists and tables. Every individual list heading and item, whether "ordered," "unordered," or "definition" style, can be referenced. Any specific table cell can be referenced by row and column index. Entire rows and columns can likewise be extracted, allowing advanced filtering and processing of tabular data. LaMeTH also handles irregularly shaped tables, and tables with cells which span multiple rows and columns.

• LaMeTH's scripting language uses LISP-like syntax. A basic scripting command extracts a single element from an HTML document. The basic commands can be combined to perform more complex extractions. Local variables allow one to keep track of nested references and to extract multiple items from the HTML source in a single reference.

• LaMeTH includes an ambiguity resolution algorithm in order to deal with inconsistencies across similar HTML documents. A location can be specified as a range or combination of particular locations, and LaMeTH will choose the correct location from the various possibilities based on secondary criteria such as text literals or other features. This allows a script to work across many Web pages whose HTML markup is similar but not identical.

3.3. Examples

Suppose we wanted to find the revenue of Sun Microsystems. The website of Fortune Magazine offers this information in its Fortune 500 companies section:

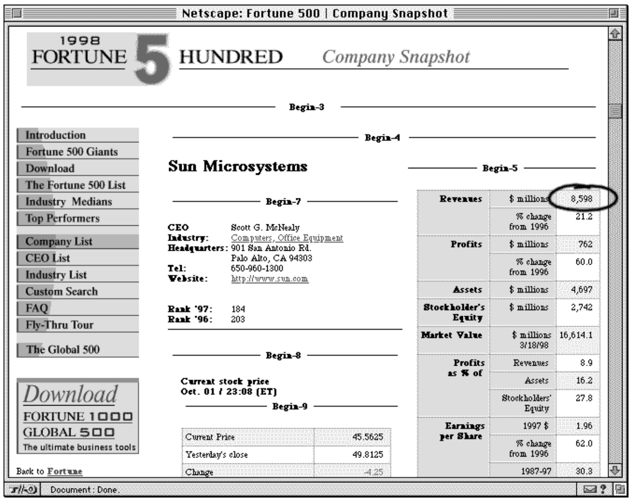


By visual inspection, it is easy to notice that the piece of information we want is presented in a table. However, only after detailed inspection will one discover the location of that same number within the HTML document:

```
(160 lines omitted)
...
<TABLE BORDER="1" CELLPADDING="3" CELLSPACING="0">
<TR VALIGN="top" ALIGN="right">
<TD ROWSPAN="2" BGCOLOR="#66FFCC"><FONT SIZE="-1">
<STRONG>Revenues</STRONG></FONT></TD>
<TD BGCOLOR="#66FFCC"><FONT SIZE="-1">
$ millions</FONT></TD>
<TD BGCOLOR="#EEEEEE"><FONT SIZE="-1">
8,598</FONT></TD></TR>
<TR VALIGN="top" ALIGN="right">
<TD BGCOLOR="#66FFCC"><FONT SIZE="-1">
...
(309 lines omitted)
```

Indeed, the HTML code of a nicely laid out Web page is often immensely complicated, bearing no resemblance to the visual layout; here, the revenues figure we want is buried within five hundred lines of hard-to-read HTML code. Locating this information requires not only a great deal of time, but also knowledge of HTML.

After invocation of the LaMeTH markup interface to mark the boundaries of tables, we can now see exactly which table we want. This interface preserves the original content and layout of the HTML document, and merely adds annotation to label and distinguish each relevant content element—in this case, tables:



We can easily see that the revenues figure (8,598) we want is located in the first row, third column, and table number five. Using this information, a simple LaMeTH script can be written to retrieve this specific piece of information; that is, the value of the revenues attribute.

The execution of this script by the LaMeTH language interpreter will produce the desired answer.

4 Comparison with related systems

There already exist several systems which specialize in the extraction of proper nouns, names, and some other symbols, such as NetOwl by IsoQuest [7], Nominator by IBM and the University of Pennsylvania [9], Nymble by BBN [2], and others [1, 3, 6, 10, 8]. The focus of the Blitz system differs somewhat from these systems in that it aims towards natural language understanding, rather than towards somewhat less ambitious goals of automatic indexing, keyword extraction, and summary generation.

The systems cited above typically rely on heuristics and corpus-based training alone, rather than on precompiled lists of symbols. Although they add expense, due to the necessity of compiling and storing them, symbol tables also improve a system's ability to recognize tokens.

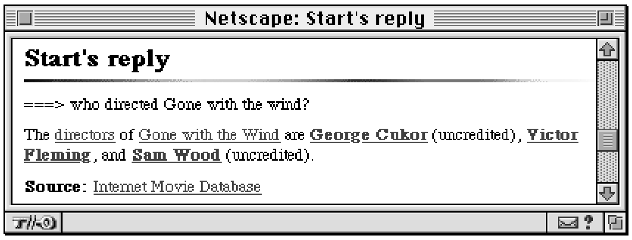
Moreover, in order to answer questions using a variety of resources on the Web, we need to identify both which resource to access and what specific information to extract based on the question. I.e., when someone asks "Who wrote X?", it is impossible to know what "database" (e.g., Web page) to access unless we know whether X is a book, movie, or song. Since we already need to store symbols in order to use LaMeTH to extract symbol attributes, we might as well make use of the symbols to improve Blitz's operation.

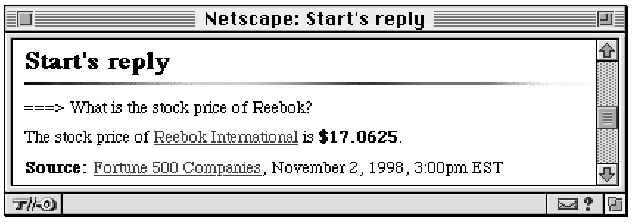
Although fully operational as standalone systems, the full potential of Blitz and LaMeTH can only be tapped through integration with a linguistically informed natural language system. Because they are specifically designed to be coupled with an NL system, they can afford, in a curious way, to be simultaneously both more and less powerful. That is, Blitz is more powerful in that it can afford to recognize less likely interpretation which only rarely turn out to be correct; the coupled NL system, using its own knowledge and tuning Blitz through modifying confidence values, compensates for Blitz's overenthusiasm. Blitz and LaMeTH are less powerful in that they need not choose or distinguish between multiple possible interpretations of detected or extracted expressions; they can leave this to the coupled natural language system. Thus combining Blitz and LaMeTH with START creates a system vastly superior in functionality to the individual components.

5 Conclusion

Today's readily available large databases and cheap computing power let us apply linguistically impoverished, computationally inexpensive mechanisms to huge amounts of data and fuse the result with strongly linguistically motivated techniques to produce a practical, effective natural language processing system which provides convenient access to all kinds of information.

Integrating Blitz with the START natural language system has improved START's ability to handle real-world sentences dramatically. LaMeTH helps find and return needed information. START is now able to interpret tokens and access live data on the Web in order to answer queries such as these:





Integrating mechanisms such as Blitz and LaMeTH into START lets us take advantage of the fruits of labor of thousands of people around the world to provide convenient information access via the World Wide Web.

References

[1] D. Appelt, J. Hobbs, J. Bear, D. Israel, M. Kameyama, and M. Tyson. The SRI MUC-5 JV FASTUS Information Extraction System. In *Proceedings of the Fifth Message Understanding Conference*, 1993.

[2] D. Bikel, S. Miller, R. Schwartz, and R. Weischedel. Nymble: a high-performace learning name-finder. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, 1997.

[3] P. Hayes. NameFinder: Software that finds names in text. In *Proc. of RIAO '94*, Paris, France, 1994.

[4] B. Katz. Using English for indexing and retrieving. In P. H. Winston and S. A. Shellard, editors, *Artificial Intelligence at MIT: Expanding Frontiers*, volume 1. MIT Press, 1990.

[5] B. Katz. Annotating the World Wide Web using natural language. In *Proc. of RIAO '97*, Montreal, Canada, 1997.

[6] W. Lehnert, J. McCarthy, S. Soderland, E. Riloff, C. Cardie, J. Peterson, and F. F. UMASS/HUGHES: Description of the CIRCUS system used for MUC-5. In *Proceedings of the Fifth Message Understanding Conference*, 1993.

[7] NetOwl Extractor technical overview. Technical report, IsoQuest, Inc., March 1997.

[8] Managing text with Oracle8 ConText Cartridge. Technical white paper, Oracle Corporation, June 1997.

[9] Y. Ravin and N. Wacholder. Extracting names from natural-language text. Research Report RC 20338, IBM, 1997.

[10] Y. Ravin, N. Wacholder, and M. Choi. Disambiguation of proper names in text. Research Report 20735, IBM, 1997.