An NLP System for Extracting and Representing Knowledge from Abbreviated Text

Deryle Lonsdale
Merrill Hutchison
Tim Richards
William Taysom

Follow this and additional works at: https://scholarsarchive.byu.edu/dlls

BYU ScholarsArchive Citation
Lonsdale, Deryle; Hutchison, Merrill; Richards, Tim; and Taysom, William (2001) "An NLP System for Extracting and Representing Knowledge from Abbreviated Text," Deseret Language and Linguistic Society Symposium: Vol. 27 : Iss. 1 , Article 6.
Available at: https://scholarsarchive.byu.edu/dlls/vol27/iss1/6

This Article is brought to you for free and open access by the Journals at BYU ScholarsArchive. It has been accepted for inclusion in Deseret Language and Linguistic Society Symposium by an authorized editor of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
This paper presents a new natural language processing (NLP) system called LG-Soar. The system is based at its most fundamental level on the Soar cognitive modeling intelligent agent architecture (Newell 1990). The system represents the integration of three major processing components: (1) regular-expression-based text preprocessing; (2) the Link Grammar parser; and (3) the Soar intelligent agent architecture. The result is a robust, versatile text processing engine useful for difficult-to-handle input. Unlike a related Soar-based NLP system, NL-Soar (Lewis 1993), this new system is not specifically designed for cognitive modeling of natural language use.

The project addresses several interesting challenges from an NLP perspective. The overall goal was to mine content from problematic text. Most existing systems perform well only on well-structured, completely grammatical text. Another goal was to address complicated linguistic issues in the development of a usable system. We also sought to output the information into a variety of usable formats. Finally, the project was meant to test the feasibility of integrating this particular set of components within a unified agent architecture.

The system was designed to handle the parsing of genealogical information from a file containing profiles of several thousand colonial American individuals. This well-known resource, built from Savage’s dictionary (1860–1862), was previously scanned via OCR and placed in raw-text form on the internet.

The LG-Soar system operates as follows:

1. A genealogical entry is read in from a preprocessed input file.
2. Each entry is split into individual sentences.
3. Each sentence is parsed with the Link Grammar parser.
4. The discourse representation module creates semantic/discourse representations (based on parse contents) for all sentences in the entry.
5. Output is generated according to various formats.

These steps are discussed in further detail in the rest of the paper.

**Preprocessing**

The preprocessing stage of LG-Soar uses a collection of subroutines and regular expressions in the Perl scripting language to create machine-readable entries for the parser. Duties of the preprocessor include creating and numbering entries for the individuals found in the input text and reformatting information about those individuals into tokenized, plain-text sentences that can be parsed by LG-Soar.

The input file text is difficult to deal with in its original form. A number of abbreviations used in the text were meant to represent several different words and require analysis of the context to correctly substitute the right word for each abbreviated form. A further complication is manifest in
tokenizing individual sentences. Care must be taken not to truncate the sentence because of an abbreviation. Other problems include place names or words that, after appearing once, are abbreviated later on in the text; an incomplete list of substitutions for abbreviations; and occasional corpus errors. The text itself, although difficult to process, is structured well enough to allow a basic level of automated information extraction. Text similar to the genealogical information used in this project is found on the internet, and the tools used in this project could be adapted for processing this type of semistructured data.

The preprocessor creates an entry that consists of a surname, a given name, and information about the individual. Surnames always appear in full capitals and head the paragraph of information about a family. Individuals belonging to the family of that surname also always appear in full capitals. Information about an individual is parsed and appended to the entry until a new individual is encountered.

Perl was chosen to implement the preprocessor because of its built-in functions for pattern matching and text manipulation. The first step in building the preprocessor was to analyze the input text. A keyword-in-context (KWIC) browser was used to determine the context for a particular interpretation of an abbreviation. Each context is represented as a quoted string. The short string is a type of simple regular expression. Underscores stand for the abbreviation. Complex Perl regular expressions are represented as Perl scalar variables. Parentheses indicate optionality. These contexts are combined into a single line with the word that will replace the abbreviation and are later extrapolated into one large regular expression as the preprocessor is started.

Prior to running the preparser, lists of abbreviations and their interpretations, complex regular expressions, and common words must be created. Many of the abbreviations and their substitutions are listed in the information included with the original file (which is subsequently removed from the input text). Unlisted abbreviations are determined by analyzing the corpus. The abbreviations are used as the key that indexes a concatenated string of possible interpretations. Complex regular expressions match a complex type of data, such as a date or occupation, and are built by hand using the KWIC browser. These serve to increase the readability of the context strings because the complexity can be hidden in simple variables that nest the larger, more difficult expressions. The Unix command "grep" is used to create a file of words that appear capitalized. This file is used to create another file that includes any uncapitalized word that matches a word from the capitalized word file.

Sentence boundaries are determined with a simple heuristic: Look at the words preceding and following a period. The sentence ends if the word before the period isn't an abbreviation or the word following the abbreviation is a common word. This heuristic is successful since sentences ending in an abbreviation are uncommon. Abbreviations are processed as the sentence is being concatenated and tokenized. The preparser looks up the abbreviations in the index to find the string of concatenated interpretations. The string is split into individual interpretations that serve as a key to locate the extrapolated context regular expression. The underscores are replaced with the actual abbreviation in the original expression and matched against the line of input. If it matches, the word is replaced. A sample entry and the result of its pre-processing are shown in Figure 1.

Future work on the preprocessor may include techniques to further simplify later stages of processing within the LG-Soar system by splitting long sentences joined by conjunctions into small sentences and using pattern matching to
explicitly replace the subject in sentences that make use of anaphora.

THE SOAR FRAMEWORK

Soar is a computer system that models human cognition (Newell 1990). It is architecturally predisposed to goal-directed problem solving and thus is ideally suited to complex tasks. Implemented in an agent-based framework, it is ideal for web search and similar applications. Its overall design derives from the fact that it was meant to instantiate a unified theory of cognition. More details can be found in the relevant literature. Soar has already been used very successfully in a diverse array of applications.

One of the motivations for using Soar in this project is that NL-Soar has been used successfully for representing and tracking referents in discourse. Because the goal of this work was not to model human cognition directly, it was deemed more appropriate to develop LG-Soar, a new system tailored to information-extraction and data-integration tasks.

LG-Soar processing requires fairly clean (if not completely grammatical) textual input. For the purposes of this paper, it can be assumed that the input to the system is preprocessed text as described previously. The output from the parser part of the system is some representation of structure that will allow for the next stage of processing. Note that the usual NLP parser output representations, tree structures, are not always conducive to further processing; they are often cumbersome and inadequate.

There are several reasons why it was decided to use the Link Grammar parser for this application. First, the parser is freely available for research purposes. Second, it is robust and can handle a much larger range of grammatical, semigrammatical, and ungrammatical structures than traditional parsers can gracefully without failing. Third, the system builds explicit relations suitable for the next stage of processing. The system also runs quite fast, compared to traditional parsers; this is a consideration when handling large volumes of data. It is also written in the C programming language, which facilitates integration with the Soar system. Finally, the LG approach yields a linguistic description that is more appropriate for the task than traditional phrase-structure grammars can provide.

The LG-Soar system was constructed by integrating two systems: Soar and the Link Grammar parser. This was possible since Soar and LG both use C at their lowest levels. In addition, Soar supports Tcl, the Toolkit Command Language, which is used to integrate various computer architectures. Tcl thus acts as "glue" between the Link Grammar engine and the Soar engine. A nontrivial amount of C and Tcl code was therefore written to tie the two systems together. The result is LG-Soar, a system that includes Tcl commands for calling the Link Grammar functions and passing information into the basic Soar processor.

THE LINK GRAMMAR COMPONENT

The Link Grammar parser is a system designed to permit flexible and robust parsing of natural-language text. It
produces a shallow parse, meaning that it does not aim at a complete, theory-dependent, linguistic parse of the sentence with all of its morphological, syntactic, and semantic complexity. Rather, it seeks to describe the major components of a sentence in as simple terms as possible. The basic unit of structure is the link.

Each sentence consists of links, and each link connects two words. Links are of various types and correspond loosely to functional relationships, like associating a subject with its predicate, a verb or preposition with its object(s), an auxiliary verb with its main verb, and so on. A link label specifies the type of relationship between the words at each end of the link. Potential links are specified by highly technical rules. In addition, it is possible to assign a score to overall linkages and also to penalize individual links.

Figure 2 shows a sample link parse for a sentence from an entry in the Savage text.

**Figure 2. Sample link parse**

```
he was.v killed.v by the Indians.n 15 March 1698.
```

For example, here the subject and verb are linked via an Ss (singular subject) link, a determiner and its head noun are associated via a Dmc (determiner) link, and the month and associated year are associated via a TY (time/year) link.

A couple of sample LG rule entries appear in Figure 3.

**Figure 3. Sample LG rule entries**

```
words/words.y: % year numbers
NN+ or Nld or NNa or AN+ or MV- or ((Xd- & TY- & Xc+) or TY-)
or (EN- or Nlc- & (ND+ or OD- or (Ld- & DD-)& (Dmcn+)) or ((noun-sub-xnoappositive) or TA-)& (JT- or IN- or (noun-main-xnoyear))))

<vc-fill>:(K+ & ([@MV+])] & O+n+) or ([O+ or B-] & (K+)) or ([@MV+ & [Xc+] & O+n+])] & [Xc+] & [@MV+];
```

Figure 4 presents a couple of examples.

**SEMANTIC PROCESSING**

After the text has been preprocessed and the syntactic links generated, semantic processing takes over. This involves
the translation of the Link Grammar parse into a representation of discourse objects and their anaphoric relationships. A subset of the linguistic approach called Discourse Representation Theory (DRT) was chosen as the basis for the representation (Kamp & Reyle 1993). A series of intermediate representations of semantic content is followed until the appropriate output format is generated by the system. This section discusses semantic processing and its associated representations.

LG-Soar implements a series of three translations between intermediate semantic representations:

1. converting a syntactic link parse to a protoDRS
2. converting a protoDRS to a DRS
3. converting a DRS to user-directed output formats

The representation of a particular set of semantic entities and relationships is called a Discourse Representation Structure (DRS). A DRS is designed as a simple and easily visualizable means of specifying the content of discourse in the context of its predication, in a manner akin to first-order logic. The approach also places a great deal of emphasis on determining pronoun reference. Any DRS has two kinds of elements: discourse referents and conditions. Discourse referents function basically like variables in logic, and conditions function as predicates over the discourse referents. For example, in the sentence “He was killed by the Indians 15 March 1698,” “he” might be assigned the discourse referent $u$, and “the Indians” assigned $v$. Then conditions placed on $u$ and $v$ might involve the fact that $u$ is represented in the existing framework by a masculine third person singular pronoun, that $v$ refers to some Indians, and finally that $u$ “was killed by” $v$ (or, if the passive voice is disregarded, that $v$ killed $u$). At present the system only implements the most basic features of DRT; however, additional features and constructions can easily be added to the existing framework.

Although the DRS is the principal semantic representation built by LG-Soar, its creation is preceded by that of a protoDRS. The protoDRS derives its information from the Link Grammar.
parse links. It has discourse referents as arguments in conditions and includes pointers to words in the parsed sentence as arguments. For example, the verbal condition in the sentence "John worked in a factory" will have as arguments the discourse referent associated with John, the word "worked" identified as the verb, and the word "in" identified as the introduction to a modifying phrase. The LG-parse-to-protoDRS translation depends entirely on the structure of the Link Grammar parse. First, each link triggers the construction of discourse referents and conditions. After these have been initialized, relationships between them are established. For example, the "S" link connects the main verb of a sentence to its grammatical subject. The link triggers the construction of a discourse referent for the subject and a verbal condition for the relevant verb. Inferred relationships establish that the discourse referent is the subject of the verb phrase. An example of a protoDRS for a sentence is given in Figure 5.

**Figure 5. Sample protoDRS**

\[
\begin{align*}
\text{Thomas}(u), & \quad \text{Eaton}(v), \quad \text{Andover}(x), \quad \text{Unice}(y), \quad \text{Singletary}(z), \\
\text{married at Andover } & \quad \text{January 1659}, \\
\text{Salisbury}(a), & \\
\text{b married } & \quad \text{z, } \quad \text{Indians}(e), \quad \text{d was killed}
\end{align*}
\]

Once a protoDRS has been built, a DRS can then be created from it. For the most part, the protoDRS-to-DRS transition involves transferring conditions from the protoDRS while removing word pointers, and also formulating relationships with complex conditions in the DRS. During this stage of processing, anaphoric relations are also determined. The rules for the construction of the DRS from the protoDRS make use of some knowledge beyond that which is expressed in the Link Grammar parse. Possessive pronouns are a good example. The Link Grammar parser simply treats possessive pronouns as determiners linked to the noun which they modify. In the protoDRS the relationship is represented as a condition that marks the pronoun as a determiner and the noun as its argument. During the protoDRS-to-DRS transition, the determiner is checked against a list of possessive pronouns. If the determiner is a possessive pronoun a "pos-s" condition is added to the protoDRS. This new condition marks the possessive pronoun as the possessor and the noun of the determiner condition as the possessed. The presence of the "pos-s" condition then triggers its transfer to the DRS. A sample DRS is given in Figure 6.

**Figure 6. Final DRS for sample sentence**

\[
\begin{align*}
\text{u, v, x, y, z, a, b, c, d, e} \\
\text{Thomas}(u), & \quad \text{Eaton}(v), \\
\text{Andover}(x), & \quad \text{Unice}(y), \quad \text{Singletary}(z), \\
\text{Salisbury}(a), & \\
\text{b married z, } & \quad \text{b was c, } \quad \text{d killed}
\end{align*}
\]

The architecture of LG-Soar allows for the arbitrary extension and branching of this three-step series of translations. Robustness is achieved by ignoring parts of a sentence from which no acceptable parse can be determined. A similar criterion is used during translation phases. At each level or representation, only certain "triggering" configurations allow for the
generation of structure at the subsequent level. This allows certain links to be ignored during the first phase. During the second phase, semantically uninterpretable conditions, perhaps arising from a massive failure in the Link Grammar parse, do not prevent the salvaging of some relationships. At each phase, only relevant information is transferred. In future work, additional knowledge sources can be used at some levels in order to make distinctions that were not explicitly represented at the previous level.

By the time the DRS construction is complete, the syntax of the source sentence has been suppressed, and all content is described as discourse referents and conditions. Having constructed the DRS, specific user-oriented output representations can be generated.

### Output Formats

The system is capable of outputting the extracted information in a variety of formats. For example, predicate-argument relationships such as those depicted in Figure 6 can be output directly. DRT has defined a data structure called discourse representation structures; the data can be output in DRS format as well. A tool called CLIG (computational linguistics interactive grapher) has a Tcl/C implementation; it was integrated into LG-Soar successfully to output DRSs from the extracted information. Potentially most useful, though, is the GEDCOM (genealogical data communication) format, which is the de facto standard for exchanging genealogical data. The LG-Soar system is capable of outputting the extracted information in GEDCOM format, which can be used by a large variety of personal history products. Only a few highly specific and pertinent DRS conditions trigger the GEDCOM data structures. For example, a verbal condition with the verb “died” or “killed” indicates someone’s death. The advantage of constructing the GEDCOM data structure from the DRS (as opposed to, say, the Link Grammar parse) is that the DRS as a semantic representation denotes many possible syntactic constructions identically. So rules for constructing the GEDCOM data at the DRS level can easily cover more possible sentences than rules at a previous level.

### Future Work and Applications

This work has focused on processing one type of text: Savage’s monumental work. However, the goal was to develop a much more widely applicable system. For example, only English text was addressed in this paper, yet many languages follow the same conventions observed in Savage’s text, particularly for biographical and genealogical information. Because Link Grammar parser versions have also been developed for other languages (e.g., German and French), it should be possible to integrate them into LG-Soar. The processing of semistructured text was the focus of this paper; however, handling completely unstructured (i.e., free) text should also be possible within our approach. In addition, completely structured text (e.g., from a spreadsheet) should likewise be possible. Additional knowledge sources could be added, such as lexical semantic resources like the WordNet lexical database (Fellbaum 1998). WordNet has been integrated with other Soar projects, and having this resource in the system will allow some automatic inferencing that is now being hand-coded in the discourse section (e.g., the fact that if someone is killed by the Indians on a particular date, that date is his death date). Finally, another exciting aspect of the LG-Soar system follows from the fact that Soar is a machine fully capable of autonomous learning. Though machine learning was turned off in the development of the system as described in this paper, it is per-
fectly reasonable to assume that many aspects of the task as described can be learned by the system. This should allow it to deal with unseen difficulties and to further optimize processing.

NOTES

1. This example reflects previous factoring of the data from its original presentation layout.

REFERENCES


