An NLP system for extracting and representing knowledge from abbreviated text

Deryle Lonsdale, Merrill Hutchison, Tim Richards, William Taysom
(The BYU NL-Soar Research Group)

Overview

This paper introduces a new system that has been developed specifically for processing abbreviated text from an information-extraction point of view. Each of the three principal components of the system---text preprocessing, parsing for content, and discourse processing---is discussed in turn. Approaches used in the development of the knowledge sources are mentioned, with a particular focus on the linguistic issues involved. Examples of how sentences are processed are given, and strengths of the system are stressed. Finally, conclusions are drawn and future work and applications are sketched.

Introducing LG-Soar

The new system, called LG-Soar, represents the integration of three major processing components: (i) regular-expression-based text preprocessing; (ii) the Link Grammar parser; and (iii) the Soar intelligent agent architecture. The result is a robust, versatile text processing engine useful for difficult-to-handle input.

- Why a new Soar parser? NL-Soar is:
  - Designed for cognitive modeling of natural language use
  - Not (yet) versatile enough to handle grammatically problematic text

The project presented several interesting challenges from an NLP perspective. The overall goal was to mine content from problematic text. Most currently existing systems only perform well on well-structured, completely grammatical text. Another problem was to address complicated linguistic issues in the development of a usable system. Another goal was 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

LG-Soar component of the system operates as follows:
1) An entry is read in from a pre-processed input file.
2) Each entry is split into individual sentences.
3) Each sentence is parsed with the Link Grammar.
4) The discourse representation module creates semantic/discourse representations of link content 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 Perl subroutines and regular expressions 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 its 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 input text is an electronic version of a compiled list of genealogical information about New England settlers in the sixteenth hundreds. The text itself, although difficult to process, is structured well enough to allow 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 semi-structured data.

The preprocessor creates an entry that will consist of a surname, a given name and information about the individual. Surnames always appear in all-caps and head the paragraph of information about a family. Individuals belonging to the family of that surname also always appear in all-caps. 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, or 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 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.

Preparation that must be done prior to running the preparser is to create a list of abbreviations and their interpretations, create complex regular expressions, and create a list of common words. Many of the abbreviations and their substitutions are listed in the information included with 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 hashes to 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. A common word is anything that appears in lower and upper case and is not an abbreviation. 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 hash 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.

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

Why Soar?

Soar is a computer system that has been implemented for use in modeling 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 (Newell 1990), and 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
We were already using it
  - NL-Soar for modeling language use in humans
  - Representing and tracking referents in disourse
  - New system tailored to information-extraction and data-integration tasks: LG-Soar

The input required for LG-Soar processing is thus 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 reprocessed 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 parser output representation, tree structure, is not always conducive to further processing; they are often cumbersome.
There are several reasons why it was decided to use the Link Grammar parser for this application. First of all, the parser is freely available for research purposes. Secondly, it is robust and can handle a much larger range of grammatical, semi-grammatical, and ungrammatical structures 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 engine 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 Link Grammar engine and 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.

Exploring Link Grammar
- What is a link?
  - Two parts, + and –
  - Shows a relationship between pairs of words
    - Subject + verb
    - Verb + object
    - Preposition + object
    - Adjective + adverbial modifier
    - Auxiliary + main verb
  - Labels each relationship
- Potential links are specified by technical rules
- Possible to score linkages, penalize links

Sample link parse

He was killed by the Indians 15 March 1698.

```
+-----------------Xc----------------+
| +------------MVp-----------+        |
| |     +----Jp---+          |        |
+-Ss+---Pv--+-MVp-+  +--Dmc-+      +-TM+--TY-+  |
|   |       |     |  |      |      |   |     |  |
he was.v killed.v by the Indians.n 15 March 1698 .
```

Sample LG rule entries

```
words/words.y:  % year numbers
NN+ or N1a- or AN+ or MV- or ((Xd- & TY- & Xc+) or TY-)
or ({{EN- or NIC-} & (ND+ or OD- or {{@L+} & DD-} &
(=[[Dmcn+] or ((<noun-sub-xnoappositive> or TA-) & (JT- or IN-
or <noun-main-xnoyear>))))));
```
Thomas Smith, Haverhill, married at Andover 6 January 1659, Unice Singletary of Salisbury.

Several steps were taken to enhance the Link Grammar so that it would be able to handle the desired text type. For example, the basic parser only recognizes one month/day order (May 24), whereas Savage uses formats like “24 May”. Similarly, it only recognized years after 1900; this had to be extended back several centuries. It was also necessary to allow years to postmodify verbs, even without prepositions (e.g. died April 1655). Savage also rather idiosyncratically inserted a comma between arguments in verb frames (e.g. “He married 6 July 1694, Ann Lynde.”); constructions like this had to be allowed for. The basic system also recognized dates as direct objects and as comma-introduced appositives, as in constructions like: “He died of smallpox, 24 October 1678.” By penalizing such links, the problem was corrected. Savage also used telegraphic-style prose, such as allowing singular nouns without determiners: “He was son of Thomas.” Rules were added to the grammar to permit these kinds of constructions. Finally several domain-specific words (e.g. “freeman”) had to be added to the system’s general-purpose lexicon.

LG example parses
Thomas Smith, Haverhill, married at Andover 6 January 1659, Unice Singletary of Salisbury.

LEFT-WALL he was v freeman n 1666.

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

The semantic stage

1. Overview

After the text has been preprocessed and the links generated, we reach the heart of the LGSoar project—the translation of the link grammar parse into a representation of anaphoric relationships and semantics. For convenience we chose a subset of Discourse Representation Theory (DRT) as the basis for the representation. The representation of a particular link grammar parse is called a Discourse Representation Structure (DRS). Particulars of DRT are discussed in the next section. In order to manage the link grammar parse to DRS translation, we developed an intermediate data structure called a protoDRS. The details on what information is used to build each successive structure from the previous is outlined in sections 3 and 4. Having constructed the DRS, specific use representations can be generated. In particular, we have implemented translation to a format specifically intended to represent pertinent relationships found in genealogical texts (birth, death, marriage, family relationships). This representation is called Gedcom Data because of its similarity to the gedcom file format for genealogical data.

2. Discourse Representation Theory

/* Note that we get the DRT stuff from Hans and Kamp, cite appropriately, etc. */
/* Perhaps add detail on anaphors. Explain the use of Õ=Ö rather than the same discourse referent. */

DRT was designed as a simple and easily visualizable approach to representing the content of discourse in the context of predication 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 on the discourse referents. For example, given the sentence: “John kicked the ball,” “John” might be assigned the discourse referent u, “the ball” assigned v. Then the conditions placed on
u, v might be: “John(u),” “ball(v)” and “kicked (u,v)” indicating that u denotes John, that v is a ball, and that u kicked v. At present our system only implements the most basic features of DRT; however, additional features and constructions easily be added to the existing framework.

3. Link --> protoDRS

Before discussing the information used to accomplish this reformulation, it is important that we distinguish the protoDRS from the DRS. By the time the DRS construction is complete, all syntax of the source sentence is suppressed, all content is described discourse referents and conditions. The protoDRS in addition to having discourse referents as the arguments of conditions allow pointers to words in the parsed sentence as arguments. For example, the verbal condition in the sentence “John worked in a factory.” will have have as arguments: the discourse referent associated with John, the word “worked” identified as the verb, and the word ”in” identified as introducing a modifying phrase. The Link Grammar parse to protoDRS transition depends entirely on the structure of the Link Grammar parse. 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 verb. The established relationship is that the discourse referent is the subject of the verb phrase.

4. protoDRS --> DRS

For the most part, the protoDRS -->DRS transition consists of removing word pointers from the protoDRS and replacing them with more complex conditions. Also at this level anaphora are 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 good example. The Link Grammar Parses simply treats possessive pronouns as determiners linked to the noun which they modify. In the protoDRS the relationship is represented as condition that marks the pronoun as a determiner and the noun as its argument. During the protoDRS --> 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.

5. DRS--> Gedcom Data

Since the Gedcom Data represents geologically pertinent data; only a few highly specific DRS conditions trigger the Gedcom Data structures. For example, a verbal condition with the verb “died” or “killed” both indicate 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.
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 ?? 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 DRS’s form 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.

Future work and applications

This work has focused on processing one type of text: that of 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. Link Grammar parser versions have also been developed for other languages (e.g. German and French), and 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 also be possible. Another possibility for adding knowledge sources is integration with lexical semantic resources like the WordNet lexical database. 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 perfectly 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.