Machine Translation with Episteme: Linguistic Knowledge Representation, Computational Efficiency and Formal Properties

J. GABRIEL AMORES* & JOSÉ F. QUESADA**

*Universidad de Sevilla
**Centro Informático Científico de Adalucía (CICA)

Abstract

In this paper we describe Episteme, a tool for the development of efficient LFG–based MT systems. Episteme was designed to meet the requirements of modern language engineering as far as the following criteria: efficiency, robustness, reusability, linguistic motivation and capable of handling large amounts of data. The system incorporates a series of novel computational techniques which enhance the overall performance significantly: representation, storage and retrieval of very large feature structure–based knowledge bases, bidirectional event–driven bottom up parsing with top–down predictions and constructive unification with post–copy.

1 Introduction

Episteme was designed to meet the requirements of modern language engineering as far as the following criteria:

1. *Computational properties:* efficiency, reusability (software infrastructure) and scalability.
2. *Linguistic motivation:* representational adequacy (user–friendly), reusability (capable of sharing linguistic knowledge) and theoretical motivation (inspired by unification grammars).
3. *Formal properties:* robustness, completeness and soundness.

The goal of this paper is to show that Episteme meets all those criteria. Section 2 describes the main computational techniques. Section 3 shows the overall architecture of Episteme. The remaining sections go through each of the modules in Episteme: lexical and morphological analysis, parsing and unification, transfer and generation.

2 Computational techniques and formal properties

It has taken approximately two years to develop the tool, until a satisfactory level of robustness and efficiency has been achieved. Our goal has been to obtain a system inspired by unification grammars (Shieber 1986) while achieving the best performance in analysis times. The efficiency of the system results from the implementation of the following techniques:

1. *Representation, Storage and Retrieval of Very Large Feature Structure–based Knowledge Bases.* The lexical module is based on Improved Binary Trees with Vertical Cut (Quesada & Amores 1995). The computational complexity of this technique is $O(log(log(N)))$, where N is the length of the lexicon. We have tested it with artificial dictionaries obtaining that the time necessary to analyze a single lexical item varies from 1 millisecond (with dictionaries of up to 5,000 entries) to 3 milliseconds (with dictionaries of up to 134,000,000 lexical entries).
2. *Bidirectional Event–driven Bottom–Up Parsing with Top–Down Predictions.* From a theoretical point of view, this parsing technique (Quesada 1997b) reduces between 80 to 95% the amount of arcs and nodes in a conventional chart parser. That is, Episteme only generates 20% of structures in the worst case. The practical consequence of this is that we can parse between 2,000 and 5,000 words per second. These results have been tested with grammars including recursion, and local and non–local dependencies, and sentences from 1 (2^0) to 1024 (2^{10}) words. In a different work, it has been demonstrated that the parser is sound and complete (Quesada 1997a:287–339).
3. *Constructive Unification with Post–Copy.* This algorithm incorporates the following strategies: structure–sharing, reversible unification, constructive unification, disunification and post–copying (Quesada 1996). Constructive unification by itself eliminates the problems of pre–copyin over–copying and redundant copying completely. This set of techniques reduces the computational load (memory and time) up to a 98% when compared with basic unification algorithms such as the naïve algorithm or the default use of unification in PROLOG.

3 Machine translation strategy

The basic architecture of Episteme relies on a modular separation between the algorithms which perform specific tasks (parsing, unification, transfer and generation) and the specification languages used to express linguistic knowledge and control commands. So much so, that Episteme could be viewed as containing two layers: a programming language for NLP and a set of functions which perform NLP tasks.

As regards MT strategy, Episteme follows a transfer approach (Hutchins & Somers 1992), which fits perfectly with the double representation which LFG (Bresnan 1982, Dalrymple et al. 1996) assigns to every sentence. Intuitively, the f–structure serves ideally as the input to transfer in a conventional MT system. While the c–structure conveys language–dependent information which is discarded during transfer, grammatical relations render language–independent information of the sort needed during transfer.

Episteme generates one or more (in case of ambiguity) c– and f– structures for each sentence in the source language during the analysis phase. The transfer module goes from source f–structures to target f–structures, and generation goes from target f–structures to target c–structures. The system may be parametrised to yield the first translation or all possible translations of an input sentence.

Figure 1 shows the overall architecture of the system, including the main components from a structural standpoint, the functional relations among them, and the flow of information during the translation process.

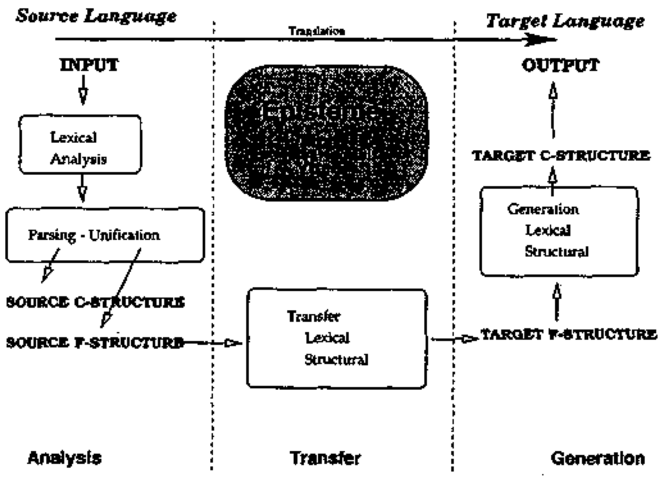


Fig. 1: *Overall architecture of Episteme*

4 Specifying linguistic knowledge in Episteme

Episteme may be described as the core of a tool for the development of MT systems. That is, Episteme is not an MT system itself, but a *shell* to develop MT systems. Accordingly, it is equipped with a series of specification languages whereby the necessary linguistic knowledge is incorporated into the system, and a series of control commands to configure the functioning of the system.

The main concept in Episteme is that of *language*. For each language, we may build an analysis grammar, an analysis lexicon, a set of transfer modules to other languages, a generation grammar, and a generation lexicon. All five components are optional for each language. A configuration command indicates which source and target languages will be active during the translation process.

In addition, Episteme includes a tool to measure the statistical performance of the system, and a "trace" capability which provides detailed information about all the operations involved in the translation process.

In the following sections we will offer a concise description of each of the processes which take place since the system receives a string of words as input until it generates the corresponding translation.

5 Lexical and morphological analysis

The phase of lexical analysis receives a string of words as input and its goal is to output a list of the syntactic category/ies and functional structure/s associated with each word.

5.1 *Simulating morphological analysis through morphological generation and efficient knowledge–base retrieval*

The lexicon is built following Mph–Vtree syntax (Quesada & Amores 1995). Mph defines a sophisticated language for the specification of lexicons for unification grammars. Its output may be linked to Vtree, a powerful system for the efficient storage and retrieval of large feature structure–based knowledge bases. The joint use of both systems creates a specification environment close to the linguist and a very efficient module for lexical and morphological analysis.

From a computational point of view our model is based on inflected forms, as oppossed to other paradigms based on two–level morphology.

Nevertheless, the system does not require that all inflected entries be coded manually. Instead, Mph permits the definition of regular morphological phenomena. An additional advantage is that the same specification language may be used to define lexical redundancy rules and some cases of derivational morphology.

Finally, using Vtree to store and retrieve lexical items once they have been generated by Mph results in excellent times. Specifically, Vtree obtains a performance rate of milliseconds per word, independently of the morphological complexity of the languages involved. Complexity only affects Mph, and the time consumed by Mph to generate all forms is compilation time, which is performed just once.

In sum, a lexical analysis model based on morphological generation such as the one presented here is preferable to models based on morphological analysis in real–time applications and with a large lexicon, where a maximum response time is usually required.

5.2 *Mph*

Mph has been designed for typed feature structures. Namely, it uses the notion of *shape* to refer to complex feature structures permitted in a language. A shape defines the skeleton of a structure, that is, the attributes which may or must be instantiated.

A shape definition is formed by four components: its name or identifier, body, list of indexes and transformational rules of shapes associated

A body definition in a shape is a list of attribute–value pairs.

The last component in a shape definition are transformational rules. One of the goals of transformational rules over shapes is to capture morphological generalisations found in natural languages. They may also be used as lexical redundancy rules to capture transitive alternations in verbs, for example. Each rule contains two components: a pattern and a set of target structures.

Meta–relations allow us to associate a shape with generation models for new shapes. Despite the use of macros and a transformational rule to obtain the plural of regular nouns, each of the entries above requires specifying all new values, which in fact are the same for all entries. This situation may be simplified by using input forms or *iforms*. These constructions permit to associate a flat, PROLOG–like predicate with a complex feature structure, incorporating all the expressive power of Mph. That is, *iforms* allow the inclusion of macros, functions, multi–word entries, etc.

Effectively, the simultaneous use of shapes and macros permits the design of a hierarchy of typed feature structures. In addition, Mph incorporates a default inheritance strategy, whereby assigning different values to the same attribute does not result in an error.

6 Analysis grammar

From a functional point of view, the parsing (Bunt & Tomita 1996, Quesada 1997a) and unification (Shieber 1986, Quesada 1996) modules in Episteme have been implemented following an *interleaving* strategy. That is, the parser interacts with the unifier during the analysis process.

6.1 *Parsing*

Broadly speaking, the parser in Episteme may be described as a *bidirectional bottom-up chart* (Kay 1980, Quesada 1997a), incorporating *top-down predictions* (Quesada 1997b).

The efficiency of a bottom-up chart parser may be increased if useless arcs are eliminated in the first stages of the process. Top-down predictions have been incorporated in Episteme with that goal.

We have implemented a set of simple and intuitive mathematical relations between the nodes in a grammar which allow us to determine whether certain arcs have no guarantee of sucess on certain occasions.

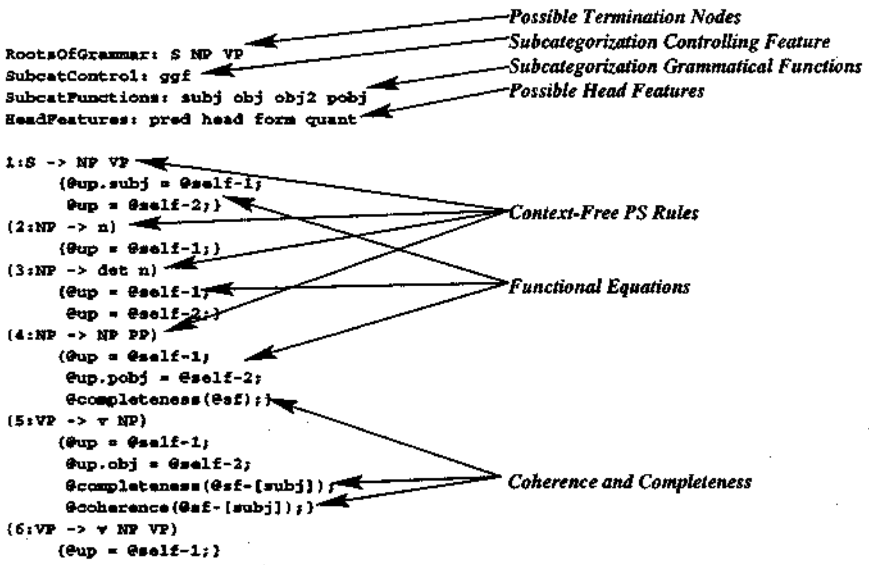
The model of top-down predictions requires that the parser knows all the information regarding possible arc applications over current nodes. This information is obtained through a model of bidirectional event generation.

Our parsing strategy approaches the problem of efficiency from an algorithmic point of view. In addition, Episteme incorporates a computational approach to increase the parsing efficiency further. Namely, the grammar is compiled beforehand, obtaining an internal representation of it which reduces the comparison of strings of characters considerably.

An analysis grammar in Episteme consists of three components:

1. A series of configuration parameters, of which only `RootsOfGrammar` will be used by the parser. `RootsOfGrammar` specifies possible termination symbols in the grammar, thus allowing for grammars with multiple root symbols.
2. A set of context-free productions.
3. Each production may contain a set of functional equations which will be passed on to the unification module.

Figure 2 shows a simple English grammar written in a format acceptable for Episteme.



```
RootsOfGrammar: S NP VP
SubcatControl: ggf
SubcatFunctions: subj obj obj2 pobj
HeadFeatures: pred head form quant

1:S -> NP VP
     {@up.subj = @self-1;
      @up = @self-2;}
(2:NP -> n)
     {@up = @self-1;}
(3:NP -> det n)
     {@up = @self-1;
      @up = @self-2;}
(4:NP -> NP PP)
     {@up = @self-1;
      @up.pobj = @self-2;
      @completeness(@sf);}
(5:VP -> v NP)
     {@up = @self-1;
      @up.obj = @self-2;
      @completeness(@sf-[subj]);
      @coherence(@sf-[subj]);}
(6:VP -> v NP VP)
     {@up = @self-1;}
```

Possible Termination Nodes
Subcategorization Controlling Feature
Subcategorization Grammatical Functions
Possible Head Features
Context-Free PS Rules
Functional Equations
Coherence and Completeness

Fig. 2: *English basic grammar for Episteme*

6.2 *Unification*

The unification module of any unification–based NLP system usually consumes around 80 or 90% of the total computation time. The relevance of this module justifies that we make a special effort in the design of the algorithms and implementation strategies. In addition, we should take into account the linguistic requirements regarding the expressive power that unification grammars usually demand.

Thus, we could divide the unification module in two distinct components. On the one hand, we have the unification algorithm proper, which is independent of any linguistic formalism. On the other, we would find the specification layer, which tries to capture the strategies and notations found in the particular theory being implemented. In our case, the latter has been designed having LFG in mind, although the algorithm is valid for any unification–based formalism.

The core of the module implements a reversible unification strategy, based on disunification and post–copying (Quesada 1996). The strategy relies on a sophisticated data organisation which obviates most copying processes during unification. If unification fails, the disunification algorithm recovers the original data structures faithfully. If unification succeeds the result is copied (post-copied) and the disunification process recovers the original input structures.

The current version allows the use of atomic values, atom negation and disjunction (negated or not), and lists.

As regards the LFG (Bresnan 1982) notation, the unification algorithm covers the basic equational unification (`=`) plus: structure assignment (`=a`), evaluation and conditional execution (`if ...then ...else`), specific functions for the manipulation of character strings and lists (`@concat`, `@member`, `@count`), mathematical (`+,-,*,/`), logical (`!,&&,||`) and relational operators (`==,!=,<,<=,>,>=`), and coherence and completeness controls. Classical LFG (Bresnan 1982, Dalrymple et al. 1996) metavariables $\uparrow$ and $\downarrow$ are called `@up` and `@self-N` in Episteme. The example in Figure 2 includes some of these functions.

7 Transfer

After the analysis phase, the parser and the unifier have obtained one or more constituent structures (c-structure in LFG) and one or more functional structures (f-structure) for each grammatical sentence. According to the machine translation approach we are assuming in Episteme, the next step will consist in transferring the source f-structure into an equivalent f-structure in the target language.[1]

The transfer module is divided in two stages, as in most transfer-based MT systems. First, the *lexical transfer* phase applies translation rules triggered by atomic-valued features. Then, during *structural transfer*, translation rules may modify complex-valued features, which results in changing the internal structure of the sentence.

One of the innovations incorporated in the transfer module is that the user may specify the order in which features are to be transferred, according with the following syntax: `FeatureTransferOrder: <list-1> ... <list-N>`

7.1 *Lexical and structural transfer*

Lexical and Structural Transfer rules are very similar in their syntax. They consist of three main components:

[1] It is possible to set up the number of analysis, transfer and generation outputs we wish to obtain when the input is ambiguous.

1. A triggering feature;
2. A set of conditions; and
3. A set of actions.

Following is a listing of two simple transfer rules. The first will transfer the feature `descr` as `pmod` inserting the appropriate preposition as well. This rule covers the generalisation of transferring premodifying nouns as prepositional phrases with *de* into Spanish: *data structure* => *estructura de datos*.

The second rule changes the internal structure of the verb *give* followed by two objects into a *v NP PP* pattern in Spanish: *to give someone something* => *dar algo a alguien*.

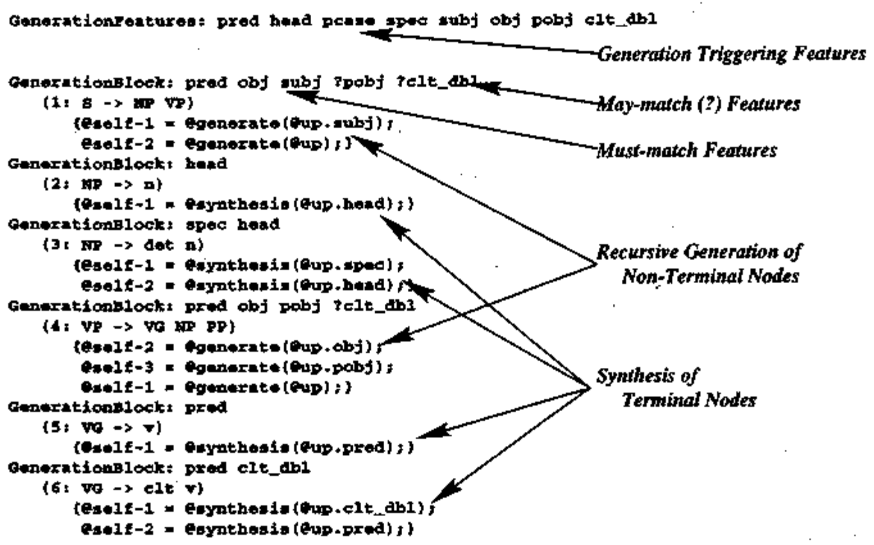
```
StructuralTransfer descr
(=> pmod @do {@target.pmod.pcase =a de; })

LexicalTransfer pred
(give => dar @when (@source.ggf == [subj,obj,obj2])
             @do {@transferas(@source.obj2,@target.obj);
                  @transferas(@source.obj,@target.pobj);
                  @target.pobj.pcase =a a}
        dar )
```

8 Generation

Once the source feature structure has been transferred into an equivalent one in the target language, the next phase involves generating the string of words in the target language. Figure 3 shows a simple generation grammar. Each rule contains four elements:

- A context–free portion specifying the subtree which will be generated by this rule, for example:
  ```
  (1:S -> NP VP)
  ```
- A set of functional equations specifying how to proceed with the generation process for each of the nodes generated by the rule:
  ```
  {@self-1 = @generate(@up.subj);
   @self-2 = @generate(@up);}
  ```
- In addition, every rule must be preceded by the list of features triggering it:
  ```
  GenerationBlock: pred obj subj ?pobj ?clt_dbl
  ```

  Episteme allows several generation rules to share the same triggering generation features, which leads to the idea of a *generation block*, or set of generation rules. In the example in Figure 3 generation blocks contained a single rule each. However, real applications demand this capability.



```
GenerationFeatures: pred head pcase spec subj obj pobj clt_dbl

GenerationBlock: pred obj subj ?pobj ?clt_dbl
   (1: S -> NP VP)
      {@self-1 = @generate(@up.subj);
       @self-2 = @generate(@up);}
GenerationBlock: head
   (2: NP -> n)
      {@self-1 = @synthesis(@up.head);}
GenerationBlock: spec head
   (3: NP -> det n)
      {@self-1 = @synthesis(@up.spec);
       @self-2 = @synthesis(@up.head);}
GenerationBlock: pred obj pobj ?clt_dbl
   (4: VP -> VG NP PP)
      {@self-2 = @generate(@up.obj);
       @self-3 = @generate(@up.pobj);
       @self-1 = @generate(@up);}
GenerationBlock: pred
   (5: VG -> v)
      {@self-1 = @synthesis(@up.pred);}
GenerationBlock: pred clt_dbl
   (6: VG -> clt v)
      {@self-1 = @synthesis(@up.clt_dbl);
       @self-2 = @synthesis(@up.pred);}
```

Generation Triggering Features
May-match (?) Features
Must-match Features
Recursive Generation of Non-Terminal Nodes
Synthesis of Terminal Nodes

Fig. 3: *Simple generation grammar*

- A condition of type `@when` which evaluates an expression. Conditions during generation make use of the same specification language and expressions evaluation mechanism designed for unification and transfer.

Another innovation of Episteme is the inclusion of optional features in the specification of the generation block. This avoids unnecesary repetitions in the generation component. For example, rule 1 in Figure 3 will be used with structures that must contain obligatorily a `pred`, `subj` and `obj` and may contain a `pobj` and/or a `clt_dbl` optionally.

Finally, lexical generation makes use of the same model (Mph–Vtree) described previously for lexical analysis.

9 Episteme at work

This section illustrates how Episteme may solve a complex translation problem between English and Spanish. Resultative constructions in English include in the predicate the manner in which the action takes place. If the action has a result it is described by means of an adjunct or a secondary predicate. On the contrary, the predicate in Spanish denotes the result and the adjunct denotes the manner.

The blacksmith hammered the metal flat (resultative)

These constructions pose a serious problem to MT systems, since what is a verb in English becomes an adjunct in Spanish and vice versa.

(A) *The blacksmith hammered the metal flat* ->
El herrero aplanó el metal a martillazos

Our analysis is based on the assumption that resultative constructions undergo a process of predicate composition in English (Amores & Alvarez 1994). Although our analysis is inspired in LFG (Bresnan 1982), it does not reflect the official trend in the theory.

Resultative constructions are identical in their constituent organisation to so–called *depictive* constructions, in which the adjunct does not describe the result of applying an action over an object, but rather, the state of the object (or subject) when the action took place.

(B) *The waiter served the fish raw* ->
El camarero sirvió el pescado crudo

Therefore, the system must be capable of differentiating each type of construction first, and then apply the corresponding translation rules.

The resultative construction will be identified during analysis taking into account whether the main verb allows this type of construction, signalled by the presence of the (result:yes) feature. If so, a new complex predicate will be formed, composed of the main verb and the head of the Adjective phrase. This operation is performed by a special function (@concat()), whose result will overwrite (=a) the original value of pred, assigned by the application of an @up = self-1 functional equation. If the triggering feature was not found, the Adjective phrase will be analysed as an adjectival adjunct (aadj) of the object by default.

```
(4:VP->v NP ADJP)
        {@up = @self-1;
         @up.obj = @self-2;
         @if (@self-1.result == yes)
         @then {
                @up.pred =a @concat(@self-1.pred,"-",
                                    @self-3.pred); }
         @else {
                @up.obj.aadj = @self-3; }
        }
```

The complex predicate *hammer-flat* will be translated as *aplanar* during the transfer phase, and the `manner` feature will hold the manner of action (`manner:"a martillazos"`). The other translation assignments are trivial. The transfer phase inserts gender features as well, which were not present in English.

REFERENCES

Amores, J. Gabriel & Gloria Alvarez. 1994. "Resultative and Depictive Constructions in English: An LFG-Based Approach for Machine Translation". *Lenguajes Naturales y Lenguajes Naturales X* ed. by Carlos Martín Vide. Barcelona: PPU.

Bresnan, Joan, ed. 1982. *The Mental Representation of Grammatical Relations.* Cambridge, Mass.: The MIT Press.

Bunt, Harry & Masaru Tomita, eds. 1996. *Recent Advances in Parsing Technology.* Kluwer Academic.

Dalrymple, Mary, Ronald M. Kaplan, John T. Maxwell III & Annie Zaenen, eds. 1995. *Formal Issues in Lexical-Functional Grammar* (= *CSLI Lecture Notes*, 47). Stanford, California: Center for the Study of Language and Information.

Hutchins, W. John & Harold L. Somers. 1992. *An Introduction to Machine Translation.* London: Academic Press.

Kay, Martin. 1980. "Algorithm Schemata and Data Structures in Syntactic Processing". *Report CSL-80-12.* Palo Alto, Calif.: Xerox, Palo Alto Research Center.

Quesada, José F. 1996. "Unificación Constructiva, Estratégica, con Compartición de Estructuras y Post-Copia". *Procesamiento del Lenguaje Natural* 19:148-158.

——. 1997a. *El Algoritmo SCP de Análisis Sintáctico Mediante Propagación de Restricciones.* Ph.D. dissertation, University of Seville. Spain.

——. 1997b. "A General, Sound and Efficient Natural Language Parsing Algorithm based on Syntactic Constraints Propagation". *Proceedings of the VII Conference AEPIA'97* ed. by Vicent Botti, 775–786. University of Malaga: Torremolinos, Spain.

—— & J. Gabriel Amores. 1995. "A Computational Model for the Efficient Retrieval of Very Large Structure-Based Knowledge Bases". *Proceedings of the Knowledge Representation, Use and Storage for Efficiency (KRUSE'95) Symposium* ed. by Gerard Ellis, Robert A. Levinson, Andrew Fall & Veronica Dahl, 86–96. Santa Cruz, California.

Shieber, Stuart M. 1986. *An Introduction to Unification-based Approaches to Grammar* (= *CSLI Lecture Notes*, 4). Stanford, California: Center for the Study of Language and Information.