# A Natural Language Question and Answer System

Chris Callison-Burch and Philip Shilane

June 3, 2000

## 1 Introduction

Our project is a question and answer system that allows natural language question to be asked of a knowledge base of information. Our program is grammar-based system which maps English questions and statements onto predicate logic. User input is parsed, converted to KIF (Knowledge Interchange Format), and sent to a reasoner that does inference using a set of first order logic axioms and returns a solution. In order to implement this program we developed a large scale grammar for English questions based on recent work in Head-driven Phrase Structure Grammar. Highlights of our grammar include the facts that it:

- Distinguishes between questions and propositions,
- Handles a wide range of question constructions, including polar interrogatives, subject and non-subject *wh*-interrogatives, and multiple *wh*-questions,
- Provides a robust, linguistically motivated analysis of long-distance ,
- Creates a mapping between *wh*-words and the semantic arguments that they are associated with.

We tested our system on a family tree domain, but the approach is flexible and can be adapted to other fields. The grammar is general enough to be reused simply by expanding it to include the necessary vocabulary. This is a strong advantage over systems which do not provide such a broad coverage of English, and instead are designed with domain specific questions in mind. The theorem proving program is completely domain independent, though a new knowledge base of facts and inference rules would have to be designed for each new domain. Creating a new knowledge base is often time consuming, but it seems like any system of this type would have this limitation, and there are a number of efforts to create a general purpose, reusable KBs. The main advantage of a question answering system is that a human user can ask questions in natural language instead of generating queries in a logic format.

In the rest of this section, we given an overview of our linguistically motivated approach and then describe the pre-existing parser and theorem which we integrated into our application. In Section 2, we describe some of the mechanisms in

our grammar, give details about our hand-built family tree knowledge base, and end with a description of the code which we use to interface between the two. In Section 3, we give an example session highlighting the question types that our grammar covers. In Section 4, we discuss the coverage of our grammar based on a few user testing sessions. In the final section, we describe future extensions to the system.

## 1.1 A Linguistically Motivated Approach

Our system implements the Ginzburg and Sag (forthcoming) theory of English interrogative constructions. Their theory is formulated in Head-driven Phrase Structure Grammar (HPSG), a constraint based linguistic formalism, and influenced by situation theory semantics. HPSG uses typed feature structures to define a fine-grained typology of language, which provides a comprehensive account of a wide range of syntactic phenomena. Some of the syntactic phenomena specific to English questions include extraction, inversion, and sensitivity to the presence of *wh*-words. Each of these is treated in the theory, and integrated into our implementation.

We modeled the types and constraints described in Ginzburg and Sag (forthcoming) using the LKB (Copestake in preparation) parsing software, and leverage it in interpreting the input to our question and answer system. This type of grammar-based approach allows us to precisely model utterances in a language, and specify a mapping between syntax and semantics. Creating a correspondence between the syntax and semantics of a language allows the surface form of statements to be mapped onto a more abstract, logical representation. We use this logical representation as a way of interfacing with the theorem prover.

The following is an example of the type of representation that our grammar builds for a question:

(1) *Who left?* $\mapsto$

$$\begin{bmatrix} \textit{question} & \\ \text{PARAMS} & \left\langle \begin{bmatrix} \textit{param} & \\ \text{INDEX} & \boxed{1} \\ \text{RESTR} & \left\{ \textit{person-rel}(\boxed{1}) \right\} \end{bmatrix} \right\rangle \\ \text{PROP} & \begin{bmatrix} \textit{proposition} & \\ \text{SIT} & s \\ \text{SOA} & \begin{bmatrix} \text{QUANTS} & \langle\ \rangle \\ \text{NUCL} & \begin{bmatrix} \textit{leave-rel} & \\ \text{LEAVER} & \boxed{1} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Contrast that with the representation for a statement such as "Kim left":

(2) *Kim left* $\mapsto$

$$\begin{bmatrix} \textit{proposition} & \\ \text{SIT} & s \\ \text{SOA} & \begin{bmatrix} \text{QUANTS} & \langle\ \rangle \\ \text{NUCL} & \begin{bmatrix} \textit{leave-rel} & \\ \text{LEAVER} & \textit{kim} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Thus, each question is treated as being about a certain proposition, with a set of variables, or *parameters*, to be determined in an answer. The elements in set of parameters correspond to the *wh*-words in a question, with each *wh*-word introducing one parameter. The PARAMS feature thereby links the parameter for each *wh*-word to the argument position within the *proposition*. The PARAMS feature further introduces restrictions that the referent of the parameter must satisfy, but we're ignoring that for the purposes of our implementation.

Beyond being closely matched to the logical representations used in automated reasoners like JTP, the primary advantage of using feature structure notation in a system like this is that it allows an elegant approach to long-distance dependencies. The innovation of early logic/grammar-based question and answer systems, such as the CHAT-80 system (Warren and Pereira 1982), is precisely that they provided an analysis of extraposition within a framework which used context free grammar notation. Our treatment is more elegant in that it does not cause a ballooning in the number of grammar rules, as the "/" notation in CFGs for extraction does. We'll describe our grammar and its treatment of extraposition in more detail in the next section.

## 1.2 The LKB

The LKB (Linguistic Knowledge Building) system is a grammar and lexicon development environment for use with constraint-based linguistic formalisms. The best way to think about the LKB system is as a development environment for a very high-level specialized programming language. Typed feature structure languages are essentially based on one data structure – the typed feature structure, and one operation – unification. This combination is powerful enough to allow the grammar developer to write grammars and lexicons that can be used to parse and generate natural languages. In effect, the grammar developer is a programmer, and the grammars and lexicons comprise code to be run by the system. The LKB system is therefore a software package for writing linguistic programs, i.e., grammars.

It's important not to confuse the LKB system and the grammars which run on it. To be clear, we have had no part in writing the LKB software – it was developed by others in the LinGO group at CSLI, including Ann Copestake, John Carroll, Rob Malouf, and Stephan Oepen – we have used the program to prototype a model of the grammar outlined in the Ginzburg and Sag text.

### 1.3 The JTP

For the knowledge base of our system, we used the Java Theorem Prover (JTP) developed by Gleb Frank in the Knowledge Systems Lab of Stanford's Computer Science Department. JTP handles most first order logic statements. Though different representation languages and reasoning systems can be plugged into JTP, the default application comes with a knowledge base supporting statements written in the Knowledge Interchange Format (KIF). JTP actually implements a subset of KIF and currently lacks the capabilities to handle relations on functions and cardinality. JTP is freely available for academic purposes at http://www.stanford.edu/~gkfrank/jtp/.

JTP can load a knowledge base containing inference rules and facts. Then, queries and statements can be made to JTP through a command line interface. The reasoning component of JTP restricts the number of levels of inference it will complete before reporting no solution. This prevents the reasoner from traveling down infinite search paths, but also causes it to fail to find solutions that do exist in the knowledge base.

## 2 Implementation

### 2.1 Description of the Grammar

In the tradition of HPSG our grammar models utterances as *signs* which associate a given phonology (or string in this case) with a certain syntactic-semantic structure. The meaning of a sign is stored the feature CONTENT (CONT), and the syntactic information, which determines it surface realization is stored in the feature CATEGORY (CAT). The mapping between the structure and meaning takes place in the ARGUMENT-STRUCTURE (ARG-ST).

(3) *Lexical entry for "love"*:

$$\begin{bmatrix} \text{ORTH} & \langle \text{ "love" } \rangle \\ \text{SS|LOC} & \begin{bmatrix} \text{CONT} & \begin{bmatrix} \text{NUCL} & \begin{bmatrix} \textit{love-reln} \\ \text{LOVER } \boxed{1} \\ \text{LOVED } \boxed{2} \end{bmatrix} \end{bmatrix} \\ \text{CAT} & \begin{bmatrix} \text{HEAD} & \textit{verb} \\ \text{ARG-ST} & \langle \begin{bmatrix} \text{LOC|CONT|INDEX } \boxed{1} \end{bmatrix}, \begin{bmatrix} \text{LOC|CONT|INDEX } \boxed{2} \end{bmatrix} \rangle \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Each of the elements on the ARG-ST correspond to syntactic constituents that are realized when grammatical rules are applied to the verb. The semantic relation described by the verb are associated with these syntactic arguments using the coindexation tags. Information about the coindexation elements is added as grammar rules apply. When the the sentence is derived, one can trace the information through the indices and compose the meaning of the sentence.

Our grammar rules build questions of various types, and account for the syntactic phenomena associated with those types. For example, in non-subject *wh-*

interrogative constructions the *wh*-constituent is dislocated from the place that it canonically realized:

(4) Who$_i$ is Diana the mother of $\_\__i$?

I'll sketch our treatment of extraction, since it's crucial for mapping the semantic indices of *wh*-words onto the correct argument positions of the predicate. The term "long-distance dependency" describes syntactic phenomena a constituent is dislocated from the place that it is normally realized, and instead occurs at a potentially unbounded distance from that location. This includes the sort of filler-gap examples associated with extraction:

(5) a. [These bagels]$_i$, I like $\_\__i$. (topicalization)

b. [These bagels]$_i$, they say they like $\_\__i$. (topicalization)

c. [Whose bagels]$_i$ do you like $\_\__i$? (*wh*-interrogative)

d. [From whom]$_i$ did you buy these bagels $\_\__i$? (*wh*-interrogative)

e. [What great bagels]$_i$ they bought $\_\__i$! (*wh*-exclamative)

We use inheritance in the hierarchy of *phrase* types to capture the similarity between topicalized and *wh*-interrogative constructions. Therefore, all constructions from (5) inherit from the same parent, the head-filler phrase:

(6) *hd-fill-ph*:

$$\begin{bmatrix}\textsc{slash} & \boxed{A}\end{bmatrix} \rightarrow \begin{bmatrix}\textsc{loc} & \boxed{1}\end{bmatrix}, \ \mathrm{H}\begin{bmatrix} \textit{phrase} & \\ \textsc{subj} & \langle\ \rangle \\ \textsc{head} & \textit{verb} \\ \textsc{slash} & \left\langle \boxed{1} \right\rangle \oplus \boxed{A} \end{bmatrix}$$

The feature SLASH is used to store all arguments which are not canonically realized by the head-complement constructions. Any argument on a word's argument structure which is not realized in the canonical fashion is marked as a *non-canon-synsem*, and added to the SLASH list. The value of a SLASH list is passed up to a phrase from its head daughter. An element is removed from the SLASH in a head-filler phrase construction. The filler daughter of such constructions is coindexed to the item being removed from the SLASH list thus guaranteeing the dependency between the topicalized element or *wh*-word, and the missing constituent.

Recall that the interpretation that we assigned to questions was based on a PARAMS list which contained the indices of the *wh*-words, as illustrated in (1). The composition of this list takes place through a slightly complicated mechanism. First, the lexical entries for *wh*-words specify that their semantic index is contained in a parameter on a separate list called STORE:

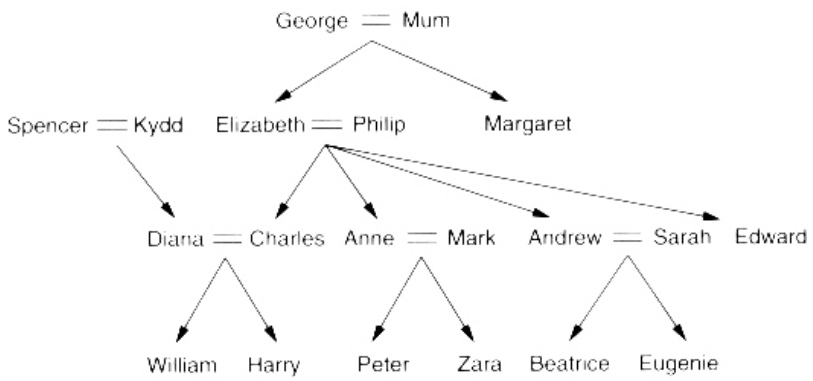


Figure 1: The British Royal Family (as described by Russell and Norvig)

(7) Interrogative *who*:

$$\begin{bmatrix} \text{ORTH} & \langle\ \text{who}\ \rangle \\ \text{SS} & \begin{bmatrix} \text{LOC} & \begin{bmatrix} \text{CAT} & \text{NP} \\ \text{CONT} & \begin{bmatrix} \text{INDEX} & i \end{bmatrix} \\ \text{STORE} & \langle\ \boxed{1}\ \rangle \end{bmatrix} \\ \text{WH} & \boxed{1}\begin{bmatrix} \textit{param} & \\ \text{IND} & i \\ \text{RESTR} & \{\textit{person-rel}(i)\} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The STORE lists of all of a word's arguments are amalgamated, and parameters are retrieved from the STORE into the PARAMS at the level of the question constructions. For a complete description of this process, and the rest of our grammar see Callison-Burch (2000).

## 2.2 The Knowledge Base

We chose to model a family tree domain in our knowledge base. We created a set of generic inference rules about familial relationships that apply to any family, and then added a set of facts to model the marriages and births of the British Royal Family as shown in Figure 1. (Note that our facts are a bit out of date because we still have Diana and Charles as married.) We felt that this domain would be constrained enough that our question and answer system could handle a reasonable subset of the questions that users would ask.

The knowledge base contains facts about the relations married, parent, and gender. The family tree file is attached as an appendix. The relations in the file are in the form (Parent child parent-of-child).

```
(Married george mum)
(Married spencer kydd)
(Parent margaret george)
(Parent elizabeth mum)
(Male eugenie)
(Female mum)
```

It also contains inference rules for relationships such as daughter, sister, cousin, child, grandparent, etc. JTP is optimized for Horn clauses, so all of our rules build from simple relations in the knowledge base to more complex relations and are unidirectional to increase the performance of the system.

```
;;; Female children are daughters.
(=> (Parent ?x ?p) (Female ?x) (Daughter ?p ?x))
;;; A sister is the daughter of your parent and not you.
(=> (Parent ?x ?p) (Daughter ?p ?y) (not (= ?x ?y)) (Sister ?x ?y))
;;; A cousin has a parent who is your aunt or uncle.
(=> (Or (Aunt ?c ?p) (Uncle ?c ?p)) (Parent ?x ?p) (Cousin ?x ?c))
;;; A grandparent is the parent of a parent.
(=> (Parent ?p ?gp) (Parent ?gc ?p) (Grandparent ?gc ?gp))
```

Our parser handles equality with the relation "equals", so we use rules in our knowledge base about "equals" instead of "=" which JTP handles rather poorly.

```
(=> (equals ?x ?y) (equals ?y ?x))  ;;; equals is reflexive.
```

The KIF specification supports functions through a reduction of an argument in a relation, i.e. (Mother elizabeth) is a function derived from the relation (Mother elizabeth mum). Rather than return truth values as relations do, functions return objects. The example relation would return mum, and should be able to be used interchangeably with "mum" to refer to that object. Unfortunately, JTP does not support the automatic creation of functions yet, so we had to add a set inference rules of the form:

```
(=> (parent ?c ?p) (equals (parent ?c) ?p))
```

to infer that if ?p is the parent of ?c then (parent ?c) is equal to ?p.

## 2.3 Pseudo Code

Our project combines the parsing capabilities of LKB with the theorem prover JTP using our grammar to define the the semantic relations of various sentence constructions. We wrote an additional application to interface between the two existing programs. Our application is written in Perl since the language provided a relatively simple facilities for interacting with other application, as well as the regular expression capabilities necessary for converting between our grammar's semantic representations and KIF. Our code is attached with the family tree knowledge base in the appendix.

```
begin LKB;
begin JTP;

for each sentence S from a user{
   if (LKB can parse S){
     extract semantics Sem from S;
     create a kif statement K from Sem;
     if (Sem is a question){
       ask JTP K;
       print JTP answers;
     }else{
       tell JTP K;
     }
   }else{
     print "Trouble parsing S";
   }
}

end LKB;
end JTP;
```

Our program begins by starting JTP with the famKB.kif file and starting LKB with the grammar file. When a new sentence is entered by a user, the LKB parses it and returns its semantic representation. (If the sentence cannot be parsed, an appropriate error message is printed.)

As an example, the sentence "Anne is a the parent of Zara." returns the following:

```
 [PROPOSITION
  SOA:   [SOA
           NUCL:  [R-EQUALS
                    ARG2:  [R-PARENT
                             ARG1:  (ZARA)]
                    ARG1:  (ANNE)]]
  SIT:   (SIT)]
NIL
```

The question "Who are the parents of Zara?" returns

```
 [QUESTION
  PARAMS:  [*LIST-OF-PARAMS*
           LIST:  [NE-LIST-OF-PARAMS
                    FIRST:  [PARAM
                             INDEX:  <1> =      (INDEX)
                             FOC:   (BOOLEAN)]
                    REST:  <2> =       (LIST-OF-PARAMS)]
           LAST:  <2>]
```

```
  PROP:  [PROPOSITION
           INDEX:  <0>
           SOA:   [R-SOA
                    NUCL:  [R-EQUALS
                             ARG2:  [R-PARENT
                                      ARG1:  (ZARA)]
                             ARG1:  <1>]]
           SIT:   (SIT)]]
NIL
```

We first check if the sentence is a query or a statement, and then translate the representation into a KIF statement. The PROPOSITION and QUESTION tags distinguish between types, and signal whether the user input is an assertion or a query. Queries and assertions are given to JTP by changes through the interactive command line to either "tell" or "ask".

Next, we translated the representation into KIF using regular expressions on the relations contained within the proposition. Observe that questions contain an embedded proposition as shown above. Object constants are surrounded by parentheses, and variables are numbers surrounded by angled brackets, which correspond to coindexations with objects on the PARAMS list. A KIF statement is then produced of the form (relation arg1 arg2 ... argN) to capture to ordering, and allows any of the arguments to contain embedded relations.

The KIF statements for the above sentences are:

```
``Anne is the parent of Zara.''    (equals  anne (parent  zara))
``Who are the parents of Zara?''   (equals  ?1 (parent  zara))
```

These KIF statements are sent to JTP, which returns a response. For questions without variables the response will be "Yes" or "No", and for *wh*-questions a list of constants which satisfy the variables in the proposition is returned. Since there are often multiple ways of reaching the same answer to a query with JTP, we use the get-setof command. This eliminates duplicates in the solution.

If an answer cannot be found within the limitations of JTP's search depth then an error message is returned. JTP has poor memory management and once it runs out of memory, it is unable to handle any further inputs. This problem only arose occasionally during testing.

# 3 Example Session

```
Loading default family tree KB: ~philips/cs224n/project/famKB.kif
Loading ~philips/cs224n/project/lkb/libacl503.so.
Starting
.......................................................

ASK> Is George the father of Elizabeth?
Asking JTP: (equals  george (father  elizabeth))
```

```
Yes

ASK>  Does Charles love Diana?
Asking JTP: (love  charles diana)
No.
```

Our program parses the polar interrogative question, and asks JTP if the relation holds in the KB. JTP goes through a set of inference rules, to derive that since George is male, and George is the parent of Elizabeth, that George is the father of Elizabeth. The program verifies the question and happily return "Yes."

If a predicate is not present in the knowledge base (as is the case with te predicate love), or is proved false then the program return "no." The database can be updated by making a statement, and then can be queried again:

```
ASK> Charles does love Diana.
Telling JTP: (love  charles diana)

ASK> Does Charles love Diana?
Asking JTP: (love  charles diana)
Yes
```

When a *wh*-question is posed to our program, it correctly identifies the argument that the *wh*-word corresponds to, and inserts a variable into the KIF query:

```
ASK> Who are the grandchildren of Philip?
Asking JTP: (equals  ?1 (grandchild  philip))
s = (setof (william) (harry) (peter) (zara) (beatrice) (eugenie))

ASK> Who is Charles married to?
Asking JTP: (married  charles  ?2)
s = (setof (diana))

ASK> Who is Anne the mother of?
Asking JTP: (equals  anne (mother  ?2))
s = (setof (peter) (zara))
```

Notice that our program correctly identifies the argument of the *wh*-word, regardless of where it is extracted from. Our robust analysis of long distance dependencies allows this. Our program also keeps track of indices in multiple *wh*-questions:

```
ASK> Who is the mother of who?
Asking JTP: (equals  ?1 (mother  ?2))
s = (setof (mum elizabeth) (mum margaret) (kydd diana) (elizabeth charles)
   (elizabeth anne) (elizabeth andrew) (elizabeth edward)
   (diana william) (diana harry) (anne peter) (anne zara)
   (sarah beatrice) (sarah eugenie))
```

An additional feature of the grammar which our program is able to exploit is the possessive:

```
ASK> Who are Anne's children?
Asking JTP: (equals  ?1 (child  anne))
s = (setof (peter) (zara))

ASK> Whose mother is Mum?
Asking JTP: (equals  (mother  ?1) mum)
s = (setof (elizabeth) (margaret))
```

# 4 Coverage of Questions

We wanted to determine the coverage of our question answering system, so we surveyed Stanford undergraduates. Our survey described a family tree as containing information about marriage, gender, and children, and asked them to list the questions they might pose to the system. Our system is able to cover many of the simpler questions from the survey, but the survey showed several limitations of our approach. The types of questions gathered from the survey are analyzed below.

## 4.1 *Wh*-Questions:

```
Who is Bernard's mother?
George is married to whom?
Whose kid is Elizabeth?
What gender is Pat?
```

Our system is able to answer these types of questions. The semantics of these sentences has a direct mapping into our knowledge base, and most of our development time was spent checking these types of sentences.

## 4.2 *How Many* Questions:

```
How many children does George have?
How many daughters does Mary have?
```

The semantics for "How many" questions has not been fully determined yet, and is not included in our grammar. Even if we could parse these sentences, JTP does not support cardinality functions, so we would need to handle counting within our interface code.

## 4.3 *Does Any* Questions:

```
Does Margaret have any brothers?\\
Does George have any grandchildren?
```

The representation of sentences containing the word *Any* require an existential quantifier, and we are currently only doing predicate logic and do not cover quantification or scope ambiguities.

### 4.4 Limitations of the KB and Grammar

```
How many pairs of brothers and sisters are there in the family?
How many only children are there?
What is the biggest immediate family grouping (parents plus children)?
What is the shortest number of generations between repetitions of a
name in the female line?
```

These are some of the more challenging we received. To handle these queries, we would have to expand the grammar to handle *there, pairs, immediate, shortest, number, generations, repetitions, name,* and *line.* The semantics for many of these words requires real-world knowledge. For the definition of generation, we could define it as the level of the tree a person exists on counting from the root of the tree. This assumes that the family tree is connected and everyone descends from the root. We could add rules to state that spouse have the same generation number, and parents have a generation level one less than their children. This would handle Diana and her parents. If our knowledge base included two separate family trees, the British Royal Family and the Kennedy Family, we would want to claim that John F. Kennedy Jr. and Princess Diana should be in the same generation because they were born within two years of each other.

## 5 Future Work

While our program does cover a wide range of questions, it is lacking an analysis for questions such as "Which man is the grandparent of Anne?" and "How many children does George have?". Our grammar does allow us to parse questions where the *wh*-word is contained within a filler phrase, but we haven't been able to come up with a treatment of the semantics for such examples. Ideally, we would like the meaning of "Which man is the grandparent of Anne?" to be represented in KIF as (and (man ?x) (grandparent ?x anne)). Because the word *man* is selects for the word *which* as a determiner, rather than having the interrogative select for *man*, it's difficult to see how we can add the restriction (man ?x) which is contributed by the *which* phrase. It's a similar problem for *how many* phrases.

A further extension that we would like to do, is to produce natural language answers rather than returning a list of variables. It would be simple to convert the KIF statements back into our grammar's feature structure representation, and since unification based grammars are reversible it would be theoretically possible to generate English strings which our grammar would assign as having that meaning. Unfortunately however, the LKB is only able to generate from grammars which formulate their semantics in a style similar to the Minimal Recursion Semantics formalism, and our semantics isn't formulated in that style.

## References

Callison-Burch, Chris. 2000. A computer model of a grammar for english questions. Undergraduate honors thesis, Stanford University, ms.

Copestake, Ann. in preparation. Implementing typed feature structure grammars. Stanford University: CSLI.

Ginzburg, Jonathan, and Ivan Sag. forthcoming. English interrogative constructions. Stanford University: CSLI.

Warren, David, and Fernando Pereira. 1982. An efficient easily adaptable system for interpreting natural language queries. American Journal of Computational Linguitics 8.110–122.