A Knowledge Base and Question Answering System Based on Loglan and English

by

Sheldon Linker

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree Doctor of Philosophy

Auburn Alabama
May 9, 2011

Keywords: knowledge base, data base, artificial intelligence

Approved by

Abstract

One of the "holy grails" of computational linguistics has been to have a machine carry out a conversation, and to have some idea of what it is talking about. Loglan's (Brown, 1960 & 1975) machine grammar (Linker, 1980) was a first attempt to carry out such a project using a grammar which was unambiguous, yet able to encompass the whole of human discourse. Writing a logical, speakable language, with a SLR-1 (simple left-to-right parsing, with one look-ahead) grammar, and then reducing that to a functional form results in a language which is hard to use for spoken logic, and is hard to translate into. A more useful way to go is to use the symbols of predicate, first-order logic, second-order logic, and higher-order logic, to use the word-classes of Loglan, to build a functional form from those in combination, and then to work backward from such a functional form to a speakable language, as much like English and Loglan, in priority order, as possible. Such a language is feasible, speakable, understandable, and useful (Linker, 2007). The result was the JCB-English language.

The thesis presented herein is that JBC-English can be improved by a number of means, making the language easier to learn and speak, more concise, and faster to process. The research and development projects detailed herein are to produce an improved version of the language, and the language processing system, which can be effectively used for human and machine discourse, and a demonstration system, which converses in this language, in such a way as to be useful in business and academia.

Acknowledgements

First and foremost, I'd like to thank my wife, who convinced me that I could do this.

Just as important, I'd like to refresh the memory of Professor James Cooke Brown, who started this project in the year I was born.

Last, I'd like to thank the informative and encouraging faculties of Thomas Edison State College, the University of Illinois at Springfield, and Auburn University.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

BNF	Backus-Naur Form
FAQ	Frequently Asked Questions
JCB	James Cooke Brown
KBMS	Knowledge-base management system
PLGS	The "Pretty Little Girls School" ambiguity problem
QA	Question Answering system
SLR-1	Simple left-to-right parsing with a single look-ahead
SQL	Structured (or Standard) Query Language
SVO	A Subject-Verb-Object(s) sentence
VO	A Verb-Subject-Object(s) sentence or fragment
YACC	Yet Another Compiler Compiler

Chapter 1 — Introduction

Since the concept of machine intelligence was first made popular in English-speaking countries by Karel Capek (1920) in the play Rossum's Universal Robots, people have been interested in the possibility of conversing with machines. In a seemingly unrelated development, Sapir and Whorf are said to have developed a psychological hypothesis, that "there may be a linkage between the language one speaks and one’s patterns of thought. " (Beeman, 1987). James Cooke Brown (1960 & 1975) invented the Loglan language as a tool to investigate the Leibnitz conjecture by testing the Sapir-Whorf hypothesis (see below). Loglan was to be a language which was complete enough to express every thought-form expressible in every human language, have a sound set pronounceable by everyone, be adjustable to test parts of the hypothesis, and be totally unambiguous.

It appears that Dr. Brown's Loglan article in Scientific American in 1960 was a turning point in linguistics. Dr. Brown, in his article, cites Leibnitz as the instigator of this line of work:

> The central notion underlying Leibnitz's vision may be stated in a question. Is it true that the "rational power" of the human animal is in any significant measure determined by the formal properties of the linguistic game it has been taught to play?

Many years later, Dr. Brown, Michael Urban, and this author started an association aimed at making Loglan provably unambiguous, and to create a SLR-1 (which stands for "Simple left-to-right parsing with one look-ahead") machine grammar for Loglan

(Linker, 1980). Since then, others have continued this line of research (Brown, 1999) — both at the Loglan Institute, with its Loglan language, and the "rebel" offshoot, the Logical Language Group (2007), with its competing language, Lojban — but have gone only as far as verifying and graphing the grammar of this human-spoken language. The Loglan institute and the Logical Language Group were both effectively working on the same research project, but had differences of opinion. Dr. Brown, the original principal investigator had one idea of where the research and design of the language should be headed, and those who left to form the Logical Language Group had a differing opinion. This author had (and maintains) yet a third opinion, but did not bring it to the public light during Dr. Brown's lifetime.

What of the Sapir-Whorf hypothesis? This author saw three significant effects from learning Loglan and its grammar: (1) The ability to learn languages better and more quickly, (2) the ability to work out a greater variety of problems without benefit of paper, and (3) the sometimes-unfortunate ability to see dozens of ambiguous meanings in what others see as having exactly one reasonable meaning. The best example of this sort of ambiguity is what Dr. Brown called the "Pretty Little Girl's School" problem, in which he points out that the reference could be to a school for girls who are little and pretty, or a pretty school for little girls, or 24 other meanings, including such meanings as a school owned by a very small girl.

What about communications with computers? The basis of this author's research was designed to use the predicate calculus basis of Loglan to create the foundation of a knowledge base integrated with a logic inference engine that answers questions directly from its knowledge base or, when required, make logical inferences and answer questions. The previous research design was limited with respect to a design to deal with a "back end", which communicates in a language of its own. At this point, the JCB-English system can sustain a conversation with a person, build a representative knowledge base, understand the interrelationships, and provide answers to questions.

In this system, a full English-language conversation with the computer involves the following steps:

- For some time to come, the person conversing with the computer will formulate statements, commands, and question in English. Example: "I like my cat."
- The user then translates these items into the controlled English described herein, known as JCB-English. There are many similarities between JCB-English and naturally spoken English; therefore there is little translation to be done. Example: "i like my cat". Whenever JCB-English is used herein, it will be shown in the Chalkboard font (as shown), to distinguish it from English.
- The JCB-English is currently entered into the system via keyboard input, but the program is built to operate as a service. Operating as a service, the C program accepts request packets from various front ends, one of which will be a web interface written as a Java Servlet.
- A parser then translates the language into a data structure, which directly represents the entered utterances.
- Rules of transformation then restructure the representation of the utterance into a more basic logical structure in some cases of grammar detailed below. Example: "i like my cat", when spoken by a user named "sheldon", internally becomes "both "SHELDON" *OWN 1 *CAT// and "sheldon" *LIKE 1 *CAT//"

- Next, certain optimizations may take place to prune the data structure. Example: Changing "both x and true" to "x". Commands are executed at this point.
- Statements and questions are evaluated by a knowledge-base management system (KBMS). This results in statements and questions being rejected as false, rejected for storage because the statement is known to be true, answered, or stored. A series of provers is used. The Instant prover evaluates a statement to be possibly true or false on its own. The Fast prover looks for simple facts that directly prove, disprove, or answer a statement. The Slow prover performs a more complete proof.
- Lastly, the result is translated into JCB-English.

In this author's previous research for this work, a reformulation of the grammar of Loglan into an even more formal form was made, to prepare for the description and implementation of a single-user, functional-form conversational knowledge base.

The JCB-English system was produced. As with most prototypical systems, certain aspects of the language and system providing the language were observed. This paper serves to correct those inadequacies, and to provide the speed, functionality, and ease of use that will make JCB-English both a usable tool and a basis for even further investigation and improvement.

Chapter 2 — Literature Review

Introduction

In reading about question-answering systems, it seems that most of the work concentrates around fitting data into an existing slot, as is the case in most database systems, with manuals and descriptions too numerous to mention, or with finding text containing the data, deciding which are likely to be relevant, and then delivering a list of such documents, delivering the most likely text, or even trying to construct an answer based on the likeliest data. It seems that there is almost no information, though, on the idea of taking an unclassified datum, and storing it into a knowledge base, in a self-defining manner. For instance, one can store "Patrick Henry said 'give me liberty or give me death,'" as plain text, but the likelihood of a system making enough sense of it to make it useful is slim. If one takes a little effort to translate it to something more formal, "Patrick Henry desires that the British give him either liberty or death", or, in JCB-English, ""patrick henry" desire the event "britain" give "patrick henry" the event either "patrick henry" free or "patrick henry" dead".

There is much to be said, too, about the artificial formal languages, such as Loglan and Lojban. Unfortunately, no real-world use has been made of these languages.

On the subject at hand, conversing with the computer in speakable formal language, there is virtually nothing to be found. As Isaac Newton is oft quoted, "…I stand on the shoulders of giants." I stand there only that I may take a flying leap into new territory. Below, the literature review is begun in sections: Patents, Question Answering Systems, and Languages.

Other Research into the Sapir Whorf Hypothesis

One of the best examples of the Sapir Whorf hypothesis was shown in experiments by Phillips & Boroditsky (2003). In the experiments, they showed that, given pictures of various objects, people who spoke (in addition to English) a language in which the object has a female "gender" in their native language saw the object has having female qualities. Similarly, people who spoke a language in which the same object has a male gender in their native language saw the object as having male qualities. Thus showing that language and thought are closely related, and in this type of case, the native language does indeed drive certain thought processes.

The Sapir-Whorf hypothesis is also being investigated as it relates to teaching. Gao (2008) points out that since culture and language are intermixed, they should both be part of a foreign language course, as language won't come easily without culture.

Logical Languages

The subject of machine conversation with people has been rampant in fiction. However, true conversation on a meaningful basis with a computer requires both a formal language and a machine understanding of context. The idea of a speakable formal language was started by Prof. James Brown of the University of Florida, and described as an article in *Scientific American*, in 1960 (Brown, 1960). Later, books on the subject were published (Brown, 1975 and 1978), and became the subject of web-based publication and update (Jennings, 2006). Rather than being the culmination of a research project, things had only begun. Other projects, such as Lojban (LeChevalier, 2006) and Guaspi (Carter,

1991) branched out. Despite grand plans after the first machine grammar was delivered (Linker 1980 and 1981), there had been many half-attempts (Goertzel, 2005), but a machine conversationalist, running a limited subset of Lojban finally was written (Speer & Havasi, 2004). This was a great advance, but still had its limitations, in that full Lojban was not yet supported, and (of course), the common English speaker would not be able to make use of the system. It is also pleasing to note that the Lojban-speaking program is happy with its work, as is evidenced by its use of the word "ua". Proposals for a language partway between Lojban and English, such as "Lojban++" have also been proposed (Goertzel, 2006).

This paper makes heavy use of the interrelatedness of Loglan, thought, and higher-order logic. An excellent overview of all of these, plus the Sapir-Whorf hypothesis appears in Lógica y Lenguajes (Logic and Languages, Laboreo, 2005).

Relatively recently, Norbert Fuchs and his colleagues (Fuchs, et al., 1999) at the Institute for Information at Zurich University have developed a number of schemes, each layered on another, to bring English to usability in logic. They have taken English, and applied a large number of rules against it, and limited it to a subset they call Attempto Controlled English, or ACE (Fuchs, et al., 1999). ACE is a limited subset of English. It is very powerful for a limited subset. The description of the subset takes the form of a manual, part grammar lesson and part programming manual. Attempto Controlled English follows English as closely as its creators could manage; thus there are a very large number of context rules. This is a very strong point in ACE, in that anyone can read ACE; however, the weak point is strong reliance in context rules, and the inherent ability to get into trouble with a misplaced phrase. Given their language specification, the authors next set out to develop the specifications for a Reasoner for ACE, or RACE (Fuchs, et al., 2002). A thesis project was done by Hirtle (Hirtle, 2006) bringing these components together. Unfortunately, ACE has no usable query language at this time, and is limited to the description of facts. Hirtle's project and this one have the possibility of being used together in the future, in that there is a possibility of compiling Attempto into statements acceptable to this project's language or data-store. If so, this project's query language might one day be used to extract information from the resulting, combined knowledge base.

Patents

Some of the patents claiming to involve "universal languages" actually pertain to internal computer coding, and not spoken languages, such as an intermediate compile-to language (Goss et al., 1987), or suggested data structures for storing data for later analysis (Jung, 2005).

There are also patents and patent applications which claim to disclose new information or teach new methods, but do not, even though they have intriguing introductions. These include universal language parsing, in which we are told how to write a compiler (Bralich et al., 1999), a "cognitive processor" of knowledge bases or which understands information (Stier & Haughton, 2002; Suda, 1994; Suda & Jeyachandran, 2003), a logical agent (Jowel & Kessock, 2006), answering in English (Chang, 2006), and even how to build your own fully functional android (Datig, 2002 and 2005).

There are a number of descriptions of question-and-answer methodologies, in which the user presents a command or query, and the computer responds with a series of questions of its own to narrow down the exact nature of the initial command or query. These include pattern-matching to determine the information content and question, much like a game of Twenty Questions (Schramm, 1987; Yano et al., 2002; Matheson, 2006; Zhang & Yang, 2002).

Many patents discuss methods of speeding up processing, some more obvious than others. These include parallelism (Dixon et al., 1992), optimization of the information store (Kautz & Selman, 1993), and hashing (Miller et al., 2002; Brassell & Miller, 2003). In a similar vein, several patents (Eldredge et al., 2004; McConnell & Barklund, 2006; Spiegler & Gelbard, 2002) describe indexing systems that can be applied to existing textual or other forms of data, for quicker retrieval.

A very common method of using natural language is to apply grammatical rules, or picking out sentence fragments or key words to use the utterance or writing to serve as the basis for a formulation of a query, SQL or otherwise, which is then used to query a data base or table (Lamberti et al., 1994; Schwartz, 1993 and 1998; Machihara et al., 2001; Wyss et al., 2002; Hsu & Boonjing, 2002; Metcalf & Dingus, 2004; Sheu & Kitazawa, 2004; Nakamura & Tate, 2005; Ejerhed, 2006; Rosser & Sturges, 2006; and others too numerous to mention) or to perform some command activity (Firman, 1994; Namba et al., 1996; Salanha at al, 2004; Hogenhout & Noble, 2006; Diederiks & Van De Sluis, 2001; Ross et al., 2002 and 2005; Fain & Fain, 2002; Beauregard & Armijo-Tamez, 2006; Dusan & Flanagan, 2002). A similar technique is to verify that the utterance or writing matches a preformulated query template, and then to retrieve query elements from the matching template zones (Appelt, 2003; Harrison et al., 2003; Agarwal & Shahshahani, 2004; Williams & Hill, 2005).

Many systems retrieve information, pulling either entire documents or facts from the documents. This searching can take place based on noun, keyword, or phrase matching (Fujisawa et al., 1991, 1995, and 1996; Haszto et al., 2001; Ho & Tong, 2002; Brown et al., 2003; Fung et al., 2004; Ejerhed, 2006; Tsourikov, 2002), actual grammar-fragment matching (Kupiec, 1996 and 1997; Ford, 2003), statistical likelihoods of having meaningful data (Ahamed, 1998), or a plurality of these techniques (Weber, 2002; Scheneburg et al., 2002; Brody, 2004; Bennett, 2005). An adjunct method to these, reading documents, placing search tags in them, and coming back for matching tags later (Kasravi & Varadarajan, 2006; Pustejovsky & Ingria, 2001) is also described.

There are also a number of inventions dealing with presentation, allowing the computer to present an improved user interface (including drawing faces [Guo et al., 2003]) to humanize the conversation. This includes "human-like responses" (Armstrong, 1998; Hagen & Stefanik, 2005), emulation of an understood system, similar to the Eliza program of the 1970s (Klipstein, 2001), text to speech (Epstein, 2002; Kobayashi et al., 2004; Wang, 2006), and keeping the dialog on track (Coffman, 2003). As an alternative approach, research has been done towards using more natural input. There are an ever-increasing number of products that perform speech recognition (Gould et al., 1999; Strong, 2001 & 2004; Romero, 2002; Bangalore et al., 2003; Wang et al., 2004). The use in speech recognition would be nice in a follow-on to this project, but is not required.

Some systems (Moser et al., 2001; Sukehiro et al., 2004) rather than storing, understanding, or retrieving information, simply translate it to another language.

One system (Hawkinson & Anderson, 2004) uses a tree-structured set of classes, in which numeric class IDs exist in ranges, so that, for instance, Dog might have an index in the Mammal range, and Mammal might have an index in the Animal range (as would Dog). However, full prepositional logic is impossible in this sort of configuration. Here is a simple example: Consider Charles, Prince of Wales. He is a member of the classes Men and Royalty. Any simple indexing scheme using numbers will not do. Prince Charles cannot be assigned any numeric index, which will be both numerically in the Men range and in the Royalty range. The current project, in contrast, uses logic reasoning statement to derive such knowledge. For instance (shown in the English equivalent) "A prince is the son of a king. A son is a male child. A king is royalty."

One system (Tunstall-Pedoe, 2006) uses "objects", which are for the most part nouns or nominative phrases, but which may also be verbs or verb-like phrases. Objects can be grouped to form facts, and facts are queryable. This sort of system can find facts and negative facts quickly, but cannot reason out more complex problems.

Virtually all patent work dealing with question answering systems in the last few years deals with delivering pages, paragraphs, or sentences from a library of purportedly factual documents, rather than formulating answers from the facts themselves.

Question Answering Systems and Related Programming

In their review, Andrenucci and Sneiders (2005) point out a list of approaches being used in question answering systems:

- Natural Language Processing maps user questions into a formal world model and ensures the most reliable answers.
- Information Retrieval powered QA, together with NLP, focus on fact extraction from large amounts of text.
- Template-based QA matches user questions to a number of templates that cover often queried parts of the knowledge domain.

Despite these trends, this project will be using near-natural language data and queries.

The Pegasus language processor (Knöll & Mezini, 2006) uses natural language (currently English and German), and interprets what is said as a series of imperatives. It translates those imperatives to Java. The grammar parsing, of course, is dependent on the input language. Within Pegasus, the basic unit is called an "idea". All in all, the result looks much like COBOL.

The idea that there is a need for a special information retrieval language, perhaps based on Loglan (Ohlman, 1961) or a logical subset of English (Cooper, 1964), is not new. Indeed, these were proposed over 40 years ago.

There have been a number of discussions on the idea of retrieving documents, or fractions thereof, based on input questions (Cardie et al., 2000; Radev et al., 2001; Brill et al., 2002; Ramakrishnan et al., 2003; Sekine & Grishman, 2003). Some take the additional step of allowing the user to narrow down the responses (Small. 2003), or by template (Srihari, Rohini & Li, 2000) or pattern (Roussinov & Robles, 2004) matching, or even reformulating the question to use other such systems as agents (Agichtein et al., 2001 and 2004). Although there are many more articles in this category, the same data tends to repeat, so no further mention of such articles will be made herein.

In the QA1 system, first-order predicate calculus was used to deal with list-controlled data (Green & Raphael, 1968). List-controlled data is likely a good idea, since the true structure and interrelatedness of the data (as relevant) will not be known until the query is issued. However, it seems unlikely that first-order logic would be sufficient for any but the most basic questions. QA1 has the advantage that is can quickly categorize data, and thus search quickly, but the disadvantage of needing to categorize data at input time. The language itself is very Lisp-like. A very similar system, MRPPS (Minker,

1977) uses a more traditional descriptive technique. In a latter paper (1978), Minker describes, in great detail, how one might go about writing a theorem prover into an analysis engine. Since Prolog is available, this project will use Prolog for the time being, but replacing Prolog with Minker's method would yield far greater control in tuning performance. There are also methods whereby some rules can be quickly excluded in such a prover (Joshi, 1977). Yet another system of this nature was created by Furbach et al. (2008), but with the typical limitation of first-order logic, there is still much to be desired.

Some systems (Reiter, 1977; Waltz, 1978; Kang, 2002) attempt to retrieve data from a relational data-base system using natural language queries.

Similarly, but perhaps more clever, one system, QuASM, gathers data from tabular information found on the web (Pinto, 2002). Such a technique could be used to drop data into a strict-grammar system, in that a crawler could gather the tables, and a template could be entered manually for the data, thus effectively adding all of that table's data to the knowledge base. The technique is not a part of the current project, but could be used to enhance future versions.

It has been pointed out (Lita & Carbonell, 2004) that entering data into a question-answering system can be a time-consuming enterprise, and that the data will thus be limited. A method of gaining large amounts of reliable data is proposed. While this technique is not being incorporated into the current project, automatic data gathering could be a useful, later addition. A similar technique, based on similarities (Ramakrishnan, 2004) is also possible.

One innovative system, Cogex (Moldovan, 2003), transforms front-end natural-language queries into a logical first-order predicate form. It does the same with back-end documents. It then uses a theorem prover to find substantial matches, and then returns the source sentence.

The writing of theorem provers has been proposed a number of times, in a number of different ways. However, Prolog is cited as being useful directly as a theorem prover (Loveland, 1986), albeit with a little work. One important source of Prolog techniques, as well as a good list of pitfalls to avoid (such as ways a logic specification can infinite-loop) appears in Prolog Programming for Artificial Intelligence (Bratco, 2001).

If a system is going to answer questions, then it needs to be trustworthy, or at least report the trustworthiness level of its data, as Chen et al. (2006) point out. This is true whether questions are answered by an automatic system, or by a person. It is for this sort of reason that this project will involve rating the veracity of the information and its answers.

Obviously, if the question-answering system can parse English (or some other source language), then it will be fairly precise. Hao et al. (2006) show how to reasonably parse a certain subset of English questions, giving reasonable answers to the most-often type of questions asked. This shows, more by omission than anything else, that a precise language syntax is required for truly precise answers. That is the reason that the current project first attempted the use of Loglan, and has since switched to Loglan-like English. For the purposes of database retrieval, very limited subsets of English have been proposed as a query language (Bernardi et al., 2007).

Some of the question-answering systems are really just FAQ systems, saving pairs of questions and their answers, and either answering directly, or answering a question-answer pair when the new question and the saved question are close enough (Wang et al., 2006).

Burek et al. (2005) describe breaking a sentence down into components, based on linked phrases. They give as an example, "What researchers work in the ALPHA project financed by the Argentine government?" They show that this can be broken down into a sentence describing the ALPHA project, and a question about the researchers on the project. Of course, such method can be applied recursively. This is the type of technique which will be used when linking words like "my" appear in statements or questions in this project.

Bererle (1993) describes a language, Lilog, which is similar to Loglan, first order logic, and the limited English presented below. An example of some Anglicized Lilog is "`forall X: BUILDING(exists Y: TOWER part-of(X,Y) impl CASTLE(X))`". The same sentence in Loglan would look similar, but would be far simpler to say and write. For instance "`forall x:`" is "Ba go" in Loglan. In the limited English presented below, the concept presented above, "for all X, if X is a building, and there is some tower Y, which is a part of X, then X is a castle" would be expressed as "for all x: if both x building and x contains at least 1 tower then x castle".

One system, Chanel (Kuhn & Di Mori, 1995), attempts to learn semantics and grammar on the fly. Herein, though, a fixed grammar will be used.

Specifically Prolog-Based Systems

Baral and Tari (2006) present a project in which grammatical parsing is used to formulate the data and the question into Prolog, and Prolog is then used to formulate the answer to the question. This is an example of an application-specific question-answering system, but what is still desired is a general-purpose question-answering system. Additionally, if the data is to be large, the use of Prolog or its equivalent should be a last resort.

In the work of Marchiori (2004), a project somewhat similar to the current project is discussed. Like this one, and English-like syntax is developed, and Prolog is used to perform the logic. However, the grammar presented is at once too loose ("`JOHN IS 'tall like a tower'.`") and too specific ("`VERB represents 'http://www.w3.org/2003/m2#verb'.`"), so that general discourse would be almost impossible. The same problem was found in attempts to use Loglan grammars. Also, using Prolog for the first cut means that all responses are too slow. Thus, in this project, a more extensive yet fixed grammar will be used, but with a self-extending vocabulary. Additionally, there will be a fast (but limited) prover as the logic engine, with Prolog as a back-up.

Similarly, Greetha & Subramanian (1990) describe a limited English sentence structure that is not only understood in Prolog, but parsed in Prolog. An example given in the work is "John opened the door with a key". Using Greetha & Subramanian's method, the Prolog structure which is first developed is "`sentence(agent(np(propernoun( john))), (vp(verb(opened), (object(np(det(the), noun(door)) ), (instrument(pp(prep(with), np(art(a), noun(key)))))`", which is then simplified to "`sentence(agent(np(john)), verb(opened), object(np(`

`the door)), instrument(pp(with a key)))`". In the project described below, the same original sentence is presented as '`before now open "john" 1 door/ 1 key`', because each predication (verb, for most practical purposes) carries positional arguments. In this example, the translated phrase would be presented to Prolog in a form equivalent to "`time(T<0), open(john, qtty(1,door), qtty(1,key))`". In Greetha & Subramanian's work, the introduced function abbreviations "`np`", "`vp`", "`det`", "`pp`", "`prep`", and "`art`" stand for Noun Phrase, Verb Phrase, DETerminant, Prepositional Phrase, PREPosition, and ARTicle, respectively.

The LogAnswer system (Furbach, 2008) parses the question for meaning and formulates its own search plan using a ProLog program, and then does Google-like work, in that it searches documents. Rather than formulating answers, it retrieves sentences or passages, rating each for "Qualität" (quality).

Prolog and the Alternatives

Prolog is an obvious and popular choice for logic programming. Prolog is very different from most other languages, in that it is almost entirely declarative rather than procedural. This makes Prolog difficult to use, even for most experienced programmers. A Prolog manual does not give an explanation of how one might go about actually using Prolog for a project such as this. However, the book Prolog Programming for Artificial Intelligence (Bratco, 2001) does just this. Some key points from apropos to this project from Bratko's book are:

- That if a Prolog program or knowledge base defines a rule on itself, directly or in a loop, that the program may fail in an infinite loop. A trivial example of self-reference is "`a:-a.`" A trivial example of a loop is "`a:-b. b:-a.`" Thus, in a project of this nature, the program should guarantee that no such loop is passed to Prolog. This may or may not prove practical in the given time constraints. If impractical, it should be a future goal. (§2.6.1)

- Because of the way in which Prolog recurses, it is possible that ordering of the clauses passed to it in the knowledge base can involve recursive, depth-first goals, which may prove unsolvable. If a program and query are passed to Prolog, and Prolog infinite-loops for this reason, the result will be a stack overflow message of some sort. In these cases, the driving program (this project) can rearrange the Prolog program such that recursion will be breadth-first. (§2.6.2)
- Exclusive paths — paths, which, if followed, preclude other paths from being tried — can be used to speed program execution, using the "`!`" operator. (§5.1.1)
- If exclusive paths are used, then criteria following the exclusive paths may be omitted, much the same way that in C, one can change "`if (x>0) y(); else if (x<=0) z();`" to "`if (x>0) y(); else z();`". (§5.1.2)
- Prolog uses a closed-world system. Anything that cannot be proven or disproved is considered false. Thus, its use is limited. (§5.4) More on this below.
- Rather than having to generate a new program if the knowledge base is changed, Prolog predicates "`assert`" and "`retract`" can add and remove facts and rules. Additionally, and for the purposes of optimization, "`asserta`" can add facts and rules at the beginning of the consideration list. (§7.4)
- Prolog, in some implementations, has the ability to define a parser. It remains to be seen whether the parsing ability is robust enough for the purposes of this project. If so, some or all of the parser might be written in Prolog, rather than Java. (§21)

- Programs are often written in Prolog for rapid prototyping, and then rewritten in other languages to execute quickly, once the methods or rules are locked in. (§23.1)
- A meta-level executive — in which the Prolog program controls the execution of another Prolog program —can be written almost trivially in Prolog. Use of this type of facility allows various sorts of tracing, explanations of the methods and/or facts used in a proof or determination, and the direction or limitation of depth of exploration. (§23.2.1)
- For full theorem proving, rather than just determination of found or not found, Prolog may have to be supplied with a transform function, giving it the explicit rules of double negation, elimination, distribution, sub-expressions, and De Morgan's laws. (§23.6)

Although Prolog is well-known, it suffers from the major drawback of being a closed-world system, in which Yes is "yes", Maybe and No are "no". This makes Prolog unsuitable as the full-fledged engine behind a conversational knowledge base. Prolog is, however, optimal for a first-cut system, and the debugging thereof, because it is known to work. A better alternative (Boley & Sintek, 1995) is RelFun. RelFun solves the problem of the need for tri-state logic ("yes", "no", and "unproven") neatly:

> Queries to RelFun differ only as follows: they return the truth-value "true" instead of printing the answer "yes"; they signal failure by yielding the truth-value "unknown" instead of printing "no". When we stay in the relational realm of RelFun this makes not much of a difference since "true" can be mapped to "yes" and "unknown" can be mapped to "no". However, when proceeding to RelFun's functional realm, queries will be able to return the third truth-value "false": this is to be mapped to those of Prolog's "no" answers for which the closed-world assumption is justified. In general, however, RelFun does not make the closed-world assumption, and in the absence of explicit negative information modestly yields "unknown" instead of "omnisciently" answering "no".

Other alternatives exist, too. However alternatives such as CP (for Conceptual Programming) provide open-world facilities, but in a completely different manner. For example (as shown by Hartley, 1986):

```
<>
   <- [STATE: (PERSON: John] -
      (POSS) -> [BOOK: * b]],
   <- [EVENT: [GIVE] -
      (AGT) -> [PERSON: John]
      (OBJ) -> [BOOK: * b]
      (RCPT) -> [PERSON: Mary]
   <- [STATE: [PERSON:Mary ] -
      (POSS) -> [BOOK: * b]].
```

Another such alternative is OWL (for Open World Logic), which can be accessed from Prolog (Matzner & Hitzler, 2006). Even more so than Prolog, OWL's differences from the rest of the procedural and declarative languages makes it difficult to use without a lot of OWL experience.

Yet another example is the Lisp-like PowerLoom (Chalupsky, 2005). Although PowerLoom differs significantly from Prolog, PowerLoom actually has a simpler syntax, and a program written for Prolog could be quickly converted to PowerLoom. PowerLoom has the advantage of running on a variety of platforms including Macintosh OS X. Given these data, an extension of the program by porting from Prolog to PowerLoom must be considered for a later phase (or the current project, if time allows).

The Loglan Grammar

The Loglan grammar deserves a full citation in this literature review because it was, to a large extent, two of the steps in writing this proposal. It was the culmination of the project that led to this one, and it was the basis for the first cut at this proposal (in which Loglan was going to be the language in use), and served as the basis for the planning of the English subset in this proposal. The grammar (Brown, 1960; Linker, 1980; Prothero et al., 1994) encapsulates the whole of human language capabilities, in a very small space. Rather than taking the space inline, the grammar and its derivations appear in the indices which follow. The 1994 Loglan Machine Grammar is ©1982-1994, and is used herein with the express written permission of the publisher.

Other Grammar Work

Loglan and JCB-English (as defined below) both have a very limited set of prepositions. In the future a great number of prepositions could be added to JCB-English. Under the current design, descriptions of placement can be made, but not easily. In her paper (2009), Lockwood describes a great number of ways in which language handling of prepositions can work.

Work on the logic of tenses began thousands of years ago by Diodorus Chronus (Galton, 2008), and has been formalized more recently by a number of researchers, beginning with Prior in 1957. Such temporal logic is included here.

The tenses of possibility, such as "will", "may", "can", "must", and the like, are sometimes known as "modes".

Table 1 — Summary/Comparison with other systems

Summary/Comparison with other systems (major examples only — not a complete list)		
Languages	JCB	Easy to learn, Speakable, fully functional for logic definition and theorem proof. Now has a Machine Speaker
	Loglan, Lojban	Hard to learn, Speakable, fully functional for logic definition and theorem proof
	Attempto	Very contextual, so very easy to violate the rules. Has a very limited Machine Speaker, RACE.
	ProLog	Hard for most people to learn and use; not speakable; fully functional for logic definition and theorem proof
	SQL	Hard for most people to learn and use; not speakable; very fast for retrieval and association, but can't apply logic.
Q&A systems	JCB	Uses unambiguous parsing. Has statements and questions. Answers questions with distinct answers. Is not domain-specific.
	Search engines, such as Google	Retrieve documents based on words given. Questions are used to pick words from.
	Template matchers	Retrieves answer templates based on certain linguistic "hits". Attempts to fill in the template from data.
	Structure-based systems	These systems use a number of grammar rules, but since English (and German) grammar is fluid, they take their best guess (highest grammatical point score) or statistically good guesses (from past satisfaction values) to take their guesses on matches.
	Q&A boards, such as Yahoo Answers	These systems rely on users to answer questions.

Chapter 3 — My Thesis, Put Simply

It is possible and feasible to produce a language suitable for a briefly-trained layman to use to enter knowledge into a knowledge base, and to retrieve knowledge from that knowledge base. Further, it is possible and feasible to produce a language processor matched to that language.

The motivation for this work is simple — to help realize the long-sought conversational computer; but at the same time, to produce a system to surpass the capabilities of simple search engines or data bases.

Chapter 4 — The Work Done

The Research Phases

In the first phase of research into the design of JCB-English, a design course was followed that didn't work out well, and that path was abandoned.

In the second phase of research into the design and implementation of JCB-English, the design goals were met, and the outcome was successful. That design an implementation led to this author's previous paper.

Once the author's Master's thesis was completed, further usage tests were performed, in the way of usability and speed research. This research and these tests indicated that a number of improvements could and should be made, and are listed below. The research, design, and implementation goals formed the basis for the present effort.

Completeness

As originally designed, the JCB-English system had a plan calling for a "fast prover" and a "slow prover". Then, when an utterance had been received, the fast prover runs. If the fast prover returns True, False, or Answerable (with a proof text), then the result is returned to the user right away. The fast prover operates by checking direct implications. For each fact in the knowledge base, if that fact can (through direct matching, and not logic manipulation other than decomposition) prove or disprove a statement, or answer a question or query, then a result is in hand, and execution stops. The slow prover was an uncompleted plan. It developed a Prolog program and query to carry out the required logic, but never went so far as to deliver them to RelFun. RelFun is much like Prolog, but rather than True and Unprovable, adds a False response. A complete description can be found at http://relfun.org. Here, the "Slow Prover" was completed, calling RelFun-like code for proof work. Originally, the Slow Prover was built to translate the knowledge and question or candidate new knowledge into the Prolog language for submission to RelFun for external processing. However, there were problems in doing so. This author and the main author of RelFun worked together telephonically to devise a solution. Some parts of the JCB language could not be handled in RelFun, such as "There exists" clauses. The conclusion was that some of the techniques used in RelFun and some of the techniques present in JCB-English would have to be combined, resulting in the current Slow Prover. The Slow Prover, despite its name, can operate fairly quickly. JCB-English has an Instant Prover component, used to see if a statement is on its face true or false, which operates at O(1). The Fast Prover operates at O(n), and is fairly incomplete. The Slow Prover can operate as slowly as O(n!), but typically operates near $O(n^2)$. The Slow Prover can invoke the Medium Prover to handle ∀ and ∃ statement evaluations within the broader investigation of the statement in concert with the knowledge base.

Speed

As Glöckner (2008) points out, speed is a major issue in question answering systems, especially in systems that use parsing and/or proofs to do their work. Two possible methods of increasing speed have been identified.

One speed improvement method was to add a "Medium Prover". The medium prover is a step between use of RelFun-like code to execute full proofs (a slow process) and the quick check provided by the "Fast Prover" as described above. The Medium

Prover evaluates ∀ and ∃ and the negation of these items by enumerating all known items into test sentences, and then calling the Fast Prover for each iteration.

Although it was planned that the Slow Prover should run on a second server while the Medium and Fast Provers run, the Instant and Fast Provers that now run first can complete in less time than it takes to move data to a secondary server, so the use of a secondary server does not increase speed.

User-interface improvement

In the original testbed project, knowledge and input were both read from disk files. The output elicited from the input was delivered to the Java console window, and then knowledge was written to a new disk file for inspection. The second version, after the previous paper was complete was a web interface, in which each user operates in an independent "world". The goal at this stage was to have a usable web and service interface, in which any number of users could log into the system and use it at the same time, each sharing knowledge, but controlled by trust levels. Knowledge is read during start-up, and rewritten to disk on a regular schedule, and once again on shut-down.

In the final system, there are three interfaces. The service program, running on a server, can accept input from a single user, as if service-request packets were arriving, and answer them one at a time. This allows for debugger-based testing. The service can, of course, act as a true service, fielding packets and responding to them. The front-end program appears as a web-page by responding with a web page to HTTP `GET` and `POST` requests, acting as a broker for the service program. User state is maintained purely in HTML, and the server program need maintain extremely little state information. The knowledge base is currently stored as an array of objects in contiguous memory, and so can be read and written very quickly. In this author's tests, read and write time were unnoticeable.

Grammatical improvement over previous JCB-English

There is a very common form of speech in which we list a string of facts. For instance, let's say we want to give facts A, B, C, and D. In English, the three main ways of doing this are as four sentences, four paragraphs, or as a single-sentence list. It doesn't matter which we use. In JCB-English, there were also three ways: As four transmissions, four sub-utterances separated by the word "execute", or as a list of facts in a single utterance, either as "both both both *A* and *B* and *C* and *D*" or "both *A* and both *B* and both *C* and *D*". Either way is cumbersome. There is a difference between using "execute" and "both", in that the "execute" method will accept A as true (in which case the statement will be ignored), false (in which case the statement will be rejected), or plausible (in which case the statement will be retained as knowledge), and then evaluate B, C, and D in turn in the same way. The use of "both" means that the four putative facts are evaluated as a single compound statement. If the statement is plausible, then each of the four facts will be added to the knowledge base separately. If, as a whole, the combined statement is false, then all four sub-facts will be rejected together. In order to make the "both" form simpler to use, "also" is introduced, which acts through the introduction of an additional production into the grammar.

This allows the four facts to be written as "*A* also *B* also *C* also *D*", and has the same meaning as if "both" and "and" had been used. It is grammatically unambiguous because it occurs at the outermost level of the grammar only, and thus cannot bind too soon.

In his original research on speakable unambiguous languages, James Brown brought forth the "Pretty Little Girls' School" example, in which the phrase, known in the Loglan and Lojban communities as "PLGS", has meanings. The various meanings arise from English's ambiguity in binding adjectives to other adjectives or nouns. For instance, one meaning of "pretty little girls' school" has "pretty" modifying "little", meaning "little in a pretty sort of way", and that construct modifying "girls", so that we mean "girls who are little in a pretty sort of way", and finally having all of that modify "school", so that we get "school for girls who are pretty in a little sort of way". In Loglan, where "pretty" is "bilti", "little" is "cmalo", "girl" is "nirli", and "school" is "ckela", this first meaning is translated as "bilti cmalo nirli ckela". In the previous version of JCB English, this would translate as "adjective adjective adjective pretty modifies little modifies girl modifies school". This is cumbersome.

Loglan provides for other orders of adjectival effects by providing "ge" and "gu". In Loglan, adjectives normally associate from left to right. But, "ge" and "gu" form parenthetical markers to limit or rearrange this association. Loglan allows for a missing "gu" when it would occur after the predicate. For instance, "bilti ge cmalo nirli ckela gu" and "bilti ge cmalo nirli ckela" are equivalent.

In order to make use of more manageable simple adjectives, an additional form has been added, allowing the "*adjective predicate*" form, in addition to the previous "adjective *predication* affects *predication*" form.

In any string of two or more predications, a predication to the left of another modifies it as an adjective, binding right-to-left. When left-to-right associations are desired, or arguments are required for adjectival phrases, the older, more verbose form must be used.

Below are three of Loglan's 26 examples of "Pretty Little Girls' School" usage:

Table 2 — Pretty Little Girls School Examples

Standard English	Functional form	Previous JCB English	New JCB English
Pretty little girls' school	(((pretty little) girl) school)	Adjective adjective adjective pretty affects little affects girls affects school	Adjective adjective pretty little affects girl affects school
Pretty little girls' school	(pretty (little (girl school)))	Adjective pretty affects adjective adjective little affects girls affects school	Pretty little girl school
Pretty little girls' school	((pretty (little girls)) school)	Adjective pretty affects adjective little affects girl affects school	Adjective pretty affects little girl affects school

<u>New logic areas</u>

For ease of writing the language, everything in older JCB-English was in Verb-Subject-Object format, even though English is in Subject-Verb-Object format. In order to simplify sentence writing, and have JCB-English look more like standard English, subjects appear at the beginning of a sentence. However, when a predication is used as an argument or adjective, or appears following a tense, it will continue to be in Verb-

Subject-Object argument form. In the first two cases, this is because English uses similar formations. In the last case, this is to avoid ambiguity. Thus, "I like potatoes" could be expressed as "i like the class potato" or "i *like the class *potato//"

Lexical improvement

In its first operational version, JCB English accepted facts (in particular) and the basic concepts of the universe (in general) in the same manner — as statements. For a complete language, such as English, this is often cumbersome. For instance, for the concepts of big (or large) or little (or small), we might have to state the following (shown in English, for clarity):

> For all X, Y, and Z, all of the following is true: If X is bigger than Y, then X is larger than Y. If X is smaller than Y, then X is littler than Y. If X is bigger than Y, then Y is not bigger than X. If X is smaller than Y, then Y is not smaller than X. If X is bigger than Y, then Y is smaller than X. If X is smaller than Y, then Y is bigger than X. If X is bigger than Y, and Y is bigger than Z, then X is bigger than Z. If X is smaller than Y, and Y is smaller than Z, then X is smaller than Z.

For the concept of membership and exclusion, we might have to state items like these (again shown in English):

> For all X, all of the following is true: If X is a cat, then X is not a dog and X is not a rabbit. If X is a dog, then X is not a cat and X is not a rabbit. If X is a rabbit, then X is not a dog and X is not a cat. If X is a dog then X is an animal. If X is a cat then X is an animal. If X is a rabbit then X is an animal.

These are straightforward, but cumbersome. They would take time to load as part of the knowledge base, and would take time to use as a part of the knowledge base during evaluation. To avoid this verbosity and time, and to thwart other problems, the following additions have been made to JCB-English:

Besides storing a knowledge base, JCB-English stores an additional set of knowledge dictionary-like items describing the language and the basic concepts of the universe, apart from facts about the world. This has been implemented as a series of dictionary-defining commands. These commands allow the definition of chaining rules, such as a>b>c and a=b=c, synonyms, antonyms, exclusive membership sets, and strict dictionary items. (See below for a complete description of all grammar items, including commands.) The defining rules shown above in English are greatly simplified using the new dictionary-defining commands:

```
>bigger execute synonyms bigger larger execute antonyms bigger
smaller execute antonyms bigger littler execute set animal cat
execute set animal dog execute set animal rabbit
```

Better Proof of Correctness and Better Speed than available in the Previous JCB-English

The compiler originally used for this project was written by this researcher purely in Java and tested using a test plan. The compiler was supplied with inputs, and behaves in a manner matching the test plan. For this phase of research, the modern equivalent of YACC, Bison (Free, 2010), was used to check the grammar for ambiguity (as it is required that the grammar be unambiguous), and to check whether the grammar is actually in the SLR-1 (simple left-to-right parsing with a single look-ahead) class of languages. (Bison supports general language parsing, so SLR-1 is no longer required; but whether or not a language is SLR-1 is a good measure of its simplicity.) The new compiler was written in C, as was everything but the front-end Java servlet.

A novel technique was used to get the speed required for the system to be usable. An advantage that C has over Java is that C programs can allocate millions of objects at a time. In the tests run in this project, it was not unusual for a simple test run to generate hundreds or even thousands of computational objects during a proof. In C, an object can contain an array without needing a separate array object. If the objects themselves are stored in arrays, and allocated a million at a time, then the C-based prover will perform a memory allocation once per million objects used, as opposed to the Java program's 2,000,000 allocations (for the base class, plus the enclosed array) per million objects used. The objects used in the JCB-English server have another difference from the standard C++ and Java objects. Objects here are polymorphic. For instance, in the previous version, <u>Dyadic-Predication(Exclusively, ConstantPredication(True), Constant-Predication(True))</u> is simplified by creating a new predication, <u>Constant-Predication(False)</u>, and by back-tracking to any predication that linked to the original, and changing that link to bear the object number of the newly created object. Using polymorphic objects, the object which started this process as a Predication-Predication is changed into a Constant-Predication in place, saving even low-level allocation, some of the garbage collection, and saving the need to back-track.

<u>Additional Languages</u>

An investigation was made into having JCB-English both take input in, and produce output in, Spanish ("español-JCB"). One way of making a multilingual parser in YACC is to have the lexer emit one or more tokens to the parser defining the language to use (for instance ENGLISH or SPANISH). The lexer could also switch key-word tables. However, a problem arises in making an attempt to input an unambiguous Spanish-like language.

One major problem in accepting such a Spanish-like language is the manner in which Spanish handles negation. In English, "I like tomatoes" is the opposite of both "I like no tomatoes" and "I don't like tomatoes", and the same as the grammatically horrible "I don't like no tomatoes". In Spanish and related languages, negatives apply throughout, so "Me gustan tomates" and "No me gustan ningunes tomates" are opposites. Using only a single negative strikes a Spanish-speaker as malformed and ambiguous.

Another problem in accepting a Spanish-like language as unambiguous is Spanish's lack of prefix and postfix operators. In English, one can say "apples or tomatoes" and be clear. Similarly, "manzanas o tomates" in Spanish. However, the English "apples or tomatoes and bananas or oranges" is not clear, because we have no rule to determine the "and" and "or" order. To make this clear in English, we need either prefix operation ("both either apples or tomatoes and either bananas or oranges") or the unwieldy postfix operators ("apples or tomatoes, either, and bananas or oranges, either, both"). No such operators exist in Spanish.

Yet another problem in accepting an unambiguous Spanish-like language is the problem of word order. In English, it is possible to rearrange words to suit, and a grammar can be formed that is English-like enough for an English speaker to recognize the words in the grammar for what they are intended to be, for instance, noun-like predications, verb-like predications, or adjective- or adverb-like predications. Additional descriptive phrases can be used. For instance "`red apple`" or "`adjective red affects` apple". In Spanish, the adjectives and adverbs come after the main word, as do the arguments. Any additional words that appear naturally in Spanish to be used to point out the roles of the words would not actually disambiguate the situation.

For these reasons, it was decided that although a Spanish vocabulary could be used for Spanish input, a Spanish-like grammar cannot. Thus, no Spanish input facility has been provided. However JCB-English provides both unambiguous JCB-English output, and ambiguous English output. Ambiguous Spanish output has been added. This required the addition of one extra command, "spanish", followed by the spanish word and the English word. For instance, "spanish gato cat" defines the translation.

Evaluation

Evaluation is at several levels — speakability of the language, unit test, and actual usability of the system. Speakability and unit test have been combined to some extent, in that unit test contains a wide variety of concepts.

- Speakability of the language: Does the speakability of the language actually improve? As can be seen in examples above and below, fewer words are required to say the same thing, and there is now more flexibility in the language.
- Unit test: One very workable method of testing a piece of software is to check each sentence of the manual or description, and see that the feature is present and correct. Another is the aerospace method — checking that every new or changed instruction believed to be reachable is actually used. Some instructions are present for exceptional cases, but are not believed to be actually reachable. Both methods have been used.

- Actual usability of the system: Can the system be used to store data, accept new data, reject data, answer questions, and report that questions cannot be answered? Can normal people do this?
- Timing: The original JCB-English system took a noticeable fraction of a second to read or write a knowledge base, and could take noticeable time to form a proof. "Slow Proofs" were not even implemented. The current system reads, writes, and proves so quickly as to be completely unnoticeable when the fast prover is used, and in a reasonable amount of time when the slow prover is used.

Performance Benchmarks

As a last step, tests were run on various types of statements and queries. In each case not involving chaining logic, JCB-English responded in less than a tenth of a second.

Chapter 5 — Design Considerations

Veracity

In any system which takes in data, and gives results, one must avoid the Garbage In, Garbage Out syndrome. Data base systems typically do this by having trusted users, untrusted users, and excluded users. Trusted users are allowed to write data into the system, update it, and remove it. Untrusted users can only query the system. In a system of this type, those categories are insufficient for a multi-user system. Many times, one hears of one person's reality versus another's. If nothing else, Einstein proposed that the truth, under certain circumstances, can be relative (in that no complete agreement can ever be made on the relative coordinates of two events, not even on their relative times of occurrence (Einstein, 1916)). So, unlike a standard data base system, in which each fact is accepted universally or not, the situation here is of each fact being accepted universally, or pertinent only to a given user and to those users who accept the given user's viewpoint. In JCB-English, the operator speaks with absolute authority. Any user may state an opinion. When proofs are made, only the operator, the user, and other users with sufficient current trust are considered. This may be best shown by example.

The operator says that his cat is large. In so speaking, the operator states absolute, inarguable fact. Bob, a regular person, says that his dog is small. Although Bob is given a great deal of trust, Bob's normal statement is accepted as Bob's belief. Bill, a regular, untrusted user, later says that the operator's cat is small, and that Bob's dog is big.

In the case of Bill's data entry, the statement about the operator's cat being small should be rejected, because the fact that the operator's cat is large is absolute. However, as a regular user, and from Bill's perspective, Bob's dog is indeed big. Thus, if Bill asks for information on Bob's animals, he should be told that Bob's cat is large (as Bob made known as absolute truth), and that Bob's dog is big (as is true from Bill's perspective).

Limitations

Development of the type of system being described here is an open-ended affair. There will always be room for increased functionality and increases in the ability to make logical transformations and solutions. Thus, in implementing a pilot program to demonstrate the concept's viability, certain limitations must be placed on the program. These limitations may include:

- There is currently no shift of pronouns, other than from "you", "i" and "me" to name form. The context rules behind words like "he", "she", "it", "them", "they", and like are difficult to unravel.
- Each utterance (a linguistic term meaning one or more sentences, spoken or entered at once, which can be statements, questions, imperatives, informal languages, and sentence fragments) must be either a collection of statements, or contain exactly one predication containing exactly one question-word. Mixed statements and questions, complex questions, and imperatives are not accepted.

Comparisons

The knowledge base and logic engine accept as input four types of utterance:

- A statement of fact, which may be composed of one or more predications and/or logical connectives,

- a yes/no question,
- a fill-in-the-blank question, and
- direct system directives (commands).

The knowledge base is composed of stored fact-type utterances.

- In answering a yes/no question, the system attempts to prove or disprove the question's predication.
- In the case of a fill-in-the-blank question (using a word, such as "when" or "which"), the logic engine searches the knowledge base for a match (a proof with wild-cards), so that the question can be answered.
- If a new statement is to be added to the knowledge base, and the same knowledge is already present, the new statement is discarded as redundant.
- If a contradiction to the new statement is found (i.e., the new statement can be disproved), then the new statement is not accepted. Note that some users will speak globally, where others speak only for their own frame of reference.
- If a new statement is received, and is plausible (neither provable nor disprovable), then the system accepts the statement into the knowledge base (at the current level of trust).

Here, the rules for what constitutes a match are explained (with the aid of a chart on the following page). The system uses several types of logic. There is an Immediate Prover, which checks whether a statement is true on its face (such as "x=x") or false on its face (such as "x≠x"). There is a Fast Prover, involving only direct matches, and a Slow Prover, which makes use of inference. Proofs are tried in order: Immediate, Fast, and then Slow Provers. If two statements exactly match in form and content, they are equal. (For instance, "I am Sheldon" matches "I am Sheldon" in form and content. Similarly, "I am who" matches, because "who" matches everyone.) If two statements match in form, but do not match in content, the two statements are unequal. (For instance, "I am Sheldon" does not match "I am Bob".) If one statement matches another in content, but not in form, they may still match, by virtue of one of the statements implying the other. For instance, "I have a cat" matches the statement "I have a happy cat". The statement "I have a cat" is a generalization of the statement "I have a happy cat" because they both imply "I have a cat". The second statement refines this knowledge. Thus, the following table of conversations: (Note that these conversations are in English for purposes of clarity of explanation, and not in the language used in a following section.)

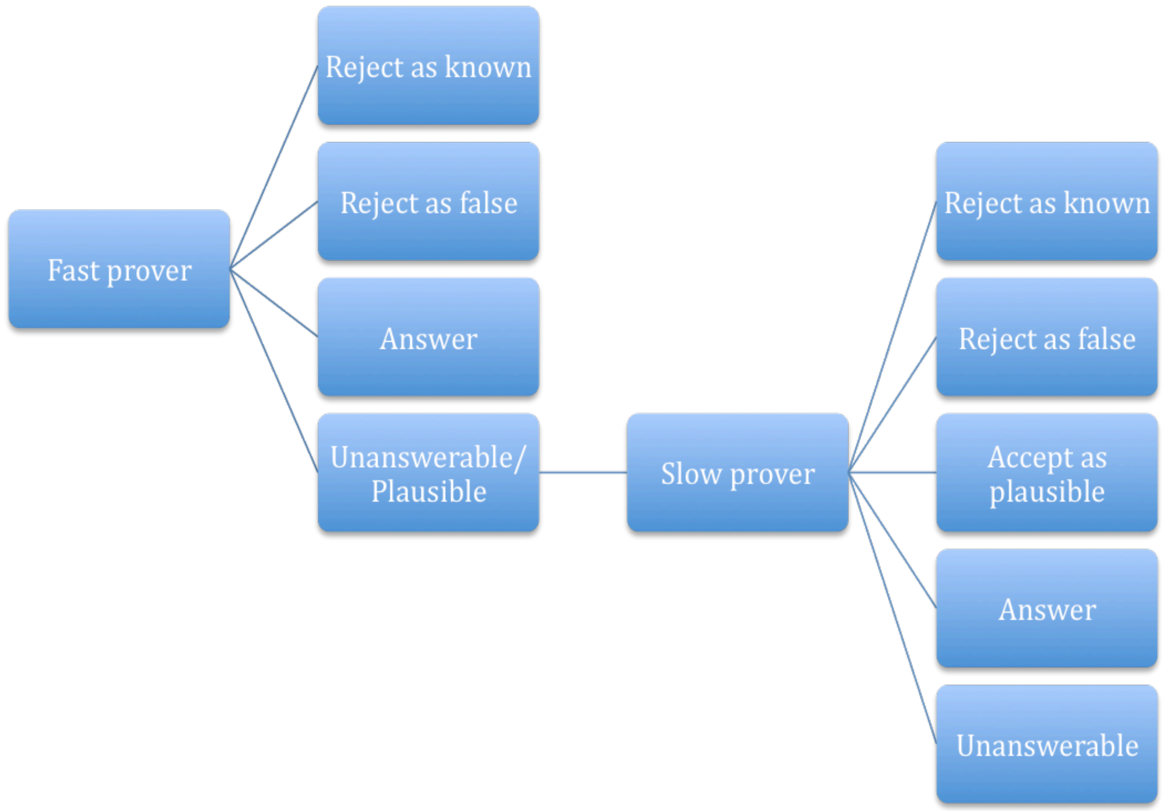


Figure 1 — Proof Flow

Table 3 — Statements and Results (Conversations)

Statements	Result
I have a cat. I have a happy cat. Do I have a happy cat?	(accepted as new information) (accepted as new information; "I have a cat" may be discarded as redundant) Yes
I have a happy cat. I have a cat Do I have a happy cat?	(accepted as new information) (ignored as redundant information) Yes
I have a cat. Some cat is happy. Do I have a happy cat?	(accepted as new information) (accepted as new information) (Unknown, because we have not identified it as the same cat.)
I have a cat. I have no cat.	(accepted as new information) (rejected as contradictory to known information)

Vocabulary

In the existing open-ended mode, the vocabulary was the list of words defined in the grammar proper, plus the words which are recognized in context as predication words. A word which is in context as a predication word is a word that appears in a position which guarantees that it is a verb, and any word with an asterisk immediately before it. In the strict vocabulary mode, only words which are defined as predications and words defined in the grammar are allowed.

Input to the program is (a) persistent information, including dictionary items, grammar nodes, and user accounts, and (b) user input packets, delivered directly or through the Java servlet front end.

Chapter 6 — Implementation

The Choice of Language

The prototype system was written in Java, and generates Prolog. The Prolog component had not fully come into use, but was to be a part of this effort, with RelFun to be used to run the Prolog code. The current version, designed for greater speed and efficiency was written in C languages without the overhead of a generalized garbage collector. In a production setting, this is a project that will process a large amount of data, with the need to respond in a very short amount of time. Objects are used, but they are polymorphic, in that they can change class after creation.

Knowledge Storage Design

The internal storage uses the rules of grammar as the basic storage structure. Many grammar productions are stored as an object representing that grammar rule. Many other grammar productions are transformed into some other grammar rule's storage class. For instance, $\neg(\neg A \wedge \neg B)$ can and will be stored as $A \vee B$. Favoring speed over simplicity, the prover not only deals with $\wedge$ and $\vee$ directly, but also $\rightarrow$, $\leftrightarrow$, and $\oplus$. For external storage, it was found that storing the data as an array of objects, plus a separate symbol table was the fastest (but not the most conservative of disk space).

A novel type of object-oriented programming was used in this project to boost speed. The objects used have several properties different than in normal object-oriented systems:

- To reduce computational time, objects are allocated a million at a time, rather than one at a time.
- Most objects are kept the same size as others, even if this involves considerable padding, so that they can be kept in arrays without needing an array of pointers.
- It is possible to call a member method on a null object. In such a case, the method (in the base class of the pointer's defined type) has a "self" which is itself null. This allows calling a method without having to check before each use whether the object ID is null or valued. Instead, the member routine can contain a single check for the null case.
- Certain specialty objects, all immutable singletons, are never allocated, but have a special object ID which identifies them by class as well as ID. For instance, the "Anything" object is an Argument object with no property fields. No "Anything" objects are ever allocated, but the "Anything" object ID can be used as an object ID, and its methods can come into play.
- Certain immutable objects that have property fields, such as "Name arguments" and "Literal text arguments" have only a single instance per unique value. For instance, the equivalent of 'new NameA("fred")' and 'new NameA("ethel")' would return different object IDs, but two uses of 'new NameA("fred")' would not. Likewise, the "clone" method on these immutables returns the ID of the original object.

Because no current programming language has these desired features, C was used, and these object behaviors were implemented at the application level.

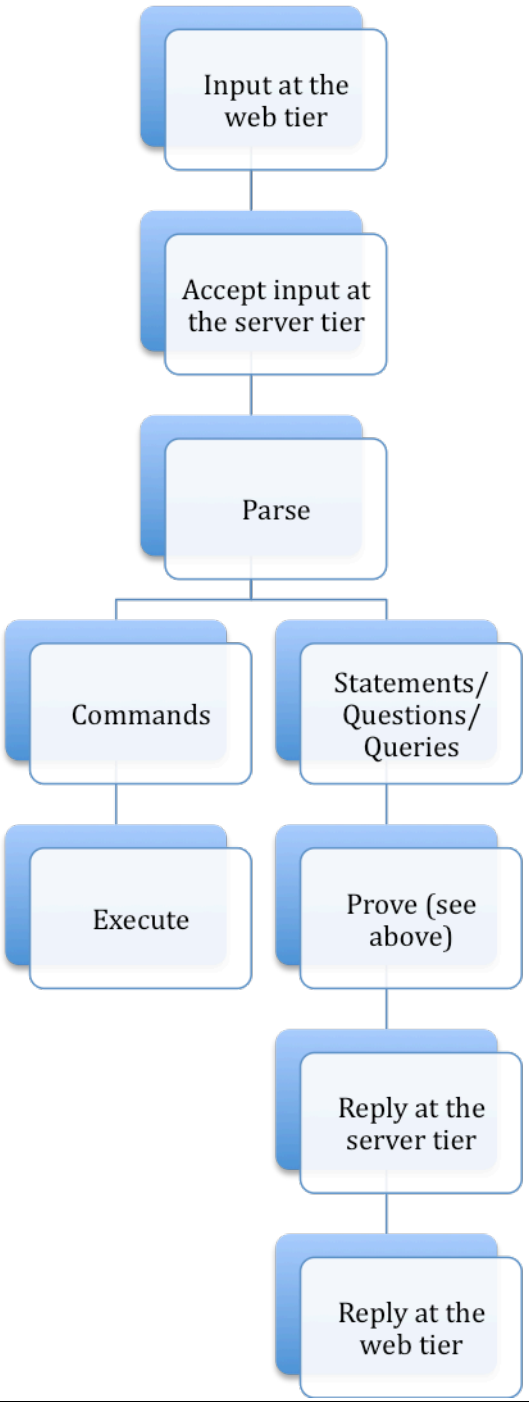


Figure 2 — An Overview of Processing

Inputs

The program needs to read the knowledge base, and then receive a number of utterances. The knowledge base will serve to establish the vocabulary, chaining rules, user list, and the sum of gleaned knowledge (empty at first). Input packets, delivered as HTTP POST messages or and/or direct socket connections, will each contain one utterance, which may contain items separated by the keyword "execute".

Processing

When the program initializes, it must first read the knowledge base. If no vocabulary is present during initialization, then the system has no vocabulary of predicates to start, and any word may be used as a predication. If no knowledge is present during initialization, then the system starts devoid of knowledge, and everything is initially plausible.

The system then starts accepting service requests. There will be a pure service socket, and requests will also be accepted via HTTP transactions. The HTTP page (a Java Servlet) will reformat the users' requests to a form acceptable to the pure service socket. When a message (page or packet) is received, it will process the utterance. The system performs I/O so quickly that it can afford to rewrite after every change.

Each utterance must be checked for a number of things: The utterance must be syntactically and lexically correct, and within the vocabulary if vocabulary is constrained. There can be at most one question word in each utterance. An utterance may also be a command.

Once the preliminary checking is done, the utterance can be evaluated. If the top level is a "both" or "also" compound, then the sub-utterances can be separated, and considered separately. This type of separation is applied recursively. Each is added to the knowledge base one at a time. This involves several steps: The sentence or question is refactored and optimized to have the least possible logic, and least implication possible.

(For instance, "X=X" is always true, and thus should not be added to the data base. As another example, "¬ ¬X" is the same as "X". The sentence is then evaluated against the knowledge base for veracity. If the sentence can be shown to be false, then the entire utterance must be discarded as false. If the sentence can be shown to be true, then the sentence must be discarded as redundant. If the sentence passes, then each of the sentence's and-separated components must be added to the knowledge base. If there is a question present, then it must be answered.

The logic applied above is able to handle adjectives. The subject of adjectives does not usually come up in discussions of logic. So, if the knowledge base held "X is a cat", and the new sentence is "X is a male cat", then it checks out as true. However, it should also be noted that "X is a cat" would be replaced if the sentence is accepted. If the knowledge base held "X is a male cat" and the sentence is "X is a cat", then the new sentence is redundant. If the new sentence is "Y is a dog", then the sentence is plausible, and the system must assume the sentence is true (in the viewpoint of the speaker). Given "X is a cat" as the knowledge base, then these fill-in-the-blank questions would match: "? is a cat", "X ? a cat", "X is ? cat", "X is a ?", and "X is ?".

Outputs

Output from the program occurs at the end of each transmission, responding with acceptance, rejection, or the answer to any given question. Outputs to questions will be in the language shown below. The service socket responds to its requestor. In the case of an HTTP web request, the server socket will return the result to the Applet, which will then wrap the result in a web page and send it to the user.

After modification, the knowledge base is rewritten.

Performance

Performance for the fast-prover is rather quick. Time required is linear with the size of the knowledge base and the size of the input (O(n)). The Slow Prover's performance can be as bad as O(n!), but in practice is closer to O(n^2) when chaining words (such as "big", meaning "bigger than") are not used. Server transaction times for these tests, on a 2.5GHz processor, with both native and Java debuggers running, was observed to be 10 to 67ms in such non-chaining cases.

Chapter 7 — Tests

Tests consist of three types.

- Black box testing exercises each element of the grammar, with an expected result.
- White box testing checks each line of code believed to be executable for proper execution. Exception lines not believed to be executable are not tested.
- Speakability tests involves doing translations, and looking for areas hard to understand or too wordy.

Server Test Results

The tests shown below, are the results of execution. Each shows the input presented directly to the JCB-English server. Where appropriate, the server's outputs, storage, and/or test notes are shown.

Table 4 — Test Cases

Test 1

Input: `either "streak" say 'meow' or "streak" say 'woof'`

Output: This information is being saved<br>

Notes: The statement is already in simplest form, compared against an empty knowledge base. It should be stored as is.

Test 2
Input: (packet has bad credentials)

Output: **Account/Password doesn't match any user**

Notes: Each packet bears credentials, which means that the system doesn't have to maintain any user states.

Test 3
Input: `facts`

Output:

#	Fact
7	`either "STREAK" *SAY 'meow' / or "STREAK" *SAY 'woof' /`

Notes: This command dumps memory. In some instances, the entire dump will be shown. In other instances, only the significant facts are shown. Although the fact was entered without special start ("*") or end ("/") markers, it is displayed with these markers. Knowledge is retained in pure form, without regard to its entry form.

Test 4
Input: (packet header calls for output in English) `my facts`

Output:

#	Fact
7	`either "STREAK" SAY 'meow' or "STREAK" SAY 'woof'`

Notes: The packet called for output in English, rather than JCB-English, so the disambiguating language markers are not present. This test also involved hard-stopping the server without warning, and restarting it. Because the knowledge base is maintained clean, and rewritten in a fraction of a second, maintaining the knowledge base intact should not be a problem.

Test 5
Input: `"streak" say either 'meow' or 'woof'`

Output: This was already known

Notes: The input, after passing through the optimizer, becomes exactly the statement about Streak we already have in the knowledge base, and is thus known.

Test 6

Input: `for all x y sub 2: true`

Output: This was already known

Notes: This is $\forall x, y_2$: True, which optimizes to True.

Test 7

Input: `be my cat`

Output: **Parse error: syntax error, unexpected WORD_N, detected at just before line 1, column 1**

Test 8

Input: `neither me be you nor i be your 1 cat execute`

`facts`

Output: This information is being saved

#	Fact
19	`neither "OPERATOR" *BE "J C B" / nor both "OPERATOR" *BE 1 *CAT / / and "J C B" *OWN 1 *CAT / /`

Notes: Here, "`me`", "`you`", and "`i`" have been mapped to proper names, "operator" for the speaker, and "j c b" for the JCB-English system. "`your`" has been mapped to a separate statement of ownership. In subsequent tests, storage will be shown directly, even though it will be generated with `facts` commands behind the scenes.

Test 9

Input: forget 19

Output: (none)

Memory:(gone)

Test 10

Input: forget "streak" say either 'meow' or 'woof'

Output: (none)

Saved: (gone)

Notes: Again, note that the predication here and in memory did not exactly match, although they had the same meaning, and thus matched.

Test 11

Input: certainly be 'dozen' the sum of 5 and 7 egg

Saved:
```
certainly *BE 'dozen' 12 *EGG//
```

Test 12

Input: on good authority be "streak" the difference between the product of 3 and 3 and the quotient of 16 and 2 cat

Saved:
```
on good authority *BE "STREAK" 1 *CAT//
```

Test 13

Input: i believe the statement cute x is implied by cat x execute

i guess if dog x then friend x

Saved:
```
i believe if *CAT x/ then *CUTE x/

i guess if *DOG x/ then *FRIEND x/
```

Test 14

Input: according to "descartes" if and only if *exist i then think i

Saved:
```
according to "DESCARTES" if and only if *EXIST "OPERATOR"/
then *THINK "OPERATOR"/
```

Notes: Here, "exist" needs an asterisk to prevent it from being considered a keyword.

Test 15

Input: likely exclusively dog "streak" or cat "streak" execute

not i cat

Saved:
```
likely exclusively *DOG "STREAK"/ or *CAT "STREAK"/
not "OPERATOR" *CAT/
```

Test 16

Input: adjective jointly black and brown affects "streak" cat

Saved:
```
adjective jointly *BLACK/ and *BROWN/ affects "STREAK" *CAT/
```

Test 17

Input: who cat

Output: Answer: `"STREAK"`

Test 18

Input: blank "streak"

Output: Answer: `"STREAK" *CAT /`

Test 19

Input: "nemo" blank

Output: There is insufficient information to say

Test 20

Input: adjective go anything emphasis affects "auburn" school also "streak" be the item 1 cat with property i love

Saved:
```
adjective *GO anything emphasis/ affects "AUBURN" *SCHOOL/
"STREAK" *BE 1 *CAT//
"OPERATOR" *LOVE 1 *CAT//
```

Notes: The statement about Auburn is that Auburn is a school, modified by go's second argument. Since "go" is conventionally go(mover,destination,source), Auburn is being modified as being a destination school, or the "go to" place. The property "i love" expands to a separate statement as a way of applying the property to the "cat" argument.

Test 21

Input: most of the class human/ gullible also all of the class politician/ liar

Saved:
```
most of the class *HUMAN/ *GULLIBLE/
all of the class *POLITICIAN/ *LIAR/
```

Test 22

Input: i desire the event the class force/ *with you

Saved:
```
"OPERATOR" *DESIRE all of the event all of the class *FORCE/
*WITH "J C B"//
```

Test 23

Input: i like the number 3.1415926 also all the class cat/ nice also 1000000 dog/ nice execute

how many of the class cat/ nice execute

how many dog/ nice

Output: This information is being saved

Answer: `all of the class *CAT /`

Answer: `1000000 *DOG /`

Saved: `"OPERATOR" *LIKE the number 3.14159/ …`

<u>Test 24</u>
Input: i like both 1 aardvark and 1 bear also i like neither 1 cat nor 1 dog also i like the item 1 elephant is implied by 1 fox also i like if 1 gazelle then 1 hamster also i like if and only if 1 iguana then 1 jaguar also i like exclusively 1 kangaroo or 1 llama

Saved:
```
neither "OPERATOR" *LIKE 1 *CAT// nor "OPERATOR" *LIKE 1 *DOG//
if "OPERATOR" *LIKE 1 *FOX// then "OPERATOR" *LIKE 1 *ELEPHANT//
if "OPERATOR" *LIKE 1 *GAZELLE// then "OPERATOR" *LIKE 1 *HAMSTER//
if and only if "OPERATOR" *LIKE 1 *IGUANA// then "OPERATOR" *LIKE 1 *JAGUAR//
exclusively "OPERATOR" *LIKE 1 *KANGAROO// or "OPERATOR" *LIKE 1 *LLAMA//
"OPERATOR" *LIKE 1 *AARDVARK//
"OPERATOR" *LIKE 1 *BEAR//
```

<u>Test 25</u>
Input: at noon go i my 1 home also on tuesday go i 1 school

Saved:
```
during 2010-08-10 through 2010-08-10 23:59:59 *GO "OPERATOR" 1 *SCHOOL//
```

```
at 2010-08-13 12:00 both *GO "OPERATOR" 1 *HOME// and *OWN
"OPERATOR" 1 *HOME//
```

Test 26

Input:

```
after september work "rachael" 1 job also on or after next week go i
1 school execute
when work "rachael" 1 job
```

Output: This information is being saved

Answer: `after 2010-08-31 23:59:59 true`

Saved:

```
after 2010-08-31 23:59:59 *WORK "RACHAEL" 1 *JOB//
on or after 2010-08-15 *GO "OPERATOR" 1 *SCHOOL//
```

Test 27

Input:

```
potentially burn the class wood execute
when burn the class wood
```

Output: This information is being saved

Answer: `potentially true`

Saved:

```
potentially *BURN all of the class *WOOD//
```

Notes: The answer is "potentially true" because in the JCB-English language, a tense ("potentially") must be applied to a predication ("true").

Test 28

Input:

```
during last month through this year magic a frog
```

Saved:

```
during 2010-07-01 through 2010-12-31 23:59:59 *MAGIC 1
*FROG//
```

Test 29

Input: beginning after today at "auburn u" whatever also before now located 3000000 meters from "auburn u" be "irvine" also on or before yesterday up to a meter from "auburn u" be an egg

Saved:
```
at "AUBURN U" beginning after 2010-08-18 23:59:59 *WHATEVER/
located 3000000 meters from "AUBURN U" before 2010-08-18
03:40:04 *BE "IRVINE"/
up to 1 meters from "AUBURN U" on or before 2010-08-17
23:59:59 *BE 1 *EGG//
```

Test 30

Input: ending before tomorrow located 10 to 100 meters from "auburn u" verbOne also during 11:00 am through 2:00:00 pm at least twenty meters from "auburn u" verbTwo also at 6:00:03 near "auburn u" verbThree

Saved:
```
located 10 to 100 meters from "AUBURN U" ending before 2010-
08-19 *VERBONE/
at least 20 meters from "AUBURN U" during 2010-08-18 11:00
through 2010-08-18 14:00 *VERBTWO/
near "AUBURN U" at 2010-08-18 06:00:03 *VERBTHREE/
```

Test 31

Input: at midnight far from "auburn u" whatever execute

where whatever

Output: …Answer: `far from "AUBURN U" true`

Saved:
```
far from "AUBURN U" at 2010-08-19 *WHATEVER/
```

Test 32

Input: during 15 april whatever execute

tense whatever

Output:
```
...during 2010-04-14 through 2010-04-14 23:59:59 *WHATEVER /...
```

Test 33

Input: at 1 year 2 days from january 1 2010 verbOne also at 1 year before january 1 2010 verbTwo also at may 1 2000 bce verbThree

Saved:
```
during 2000-05-01 bce through 2000-04-30 00:00:01 bce
*VERBTHREE/
at 2009-01-01 *VERBTWO/
at 2011-01-03 23:59:59 *VERBONE/
```

Notes: Note that facts are not necessarily stored in the order received.

Test 34

Input both beginning 1997-04-01 noon cat "streak" and ending noon january 1 2004 big dog "fido" also "whitehouse" white house

Saved:
```
beginning 1997-04-01 12:00 *CAT "STREAK"/
ending 2004-01-01 12:00 adjective *BIG/ affects "FIDO" *DOG/
adjective *WHITE/ affects "WHITEHOUSE" *HOUSE/
```

Test 35

Input: "streak" cat execute

is "streak" cat execute

is "arrow" cat

Output: This information is being saved

Yes

There is insufficient information to say

Test 36

Input: some of the class politician/ honest also little of the class student/ stupid also x blue quantity y house

Saved:
```
some of the class *POLITICIAN/ *HONEST/
little of the class *STUDENT/ *STUPID/
subject x *BLUE quantity x *HOUSE//
```

Test 37

Input: there exists x such that false

Output: That would contradict information already held

Notes: The optimizer evaluates this, and the prover does not run.

Test 38

Input: there exists at least 1 x such that x happy

Saved:
```
there exists at least 1 x such that x *HAPPY/
```

Test 39

Input: there exists 1 x such that x be "luna" also there exists up to 27 x such that x go "luna" also there does not exist x such that both x go "disneyland" and x sad

Saved:
```
there exists 1 x such that x *BE "LUNA"/
there exists up to 27 x such that x *GO "LUNA"/
for all x: not both x *GO "DISNEYLAND"/ and x *SAD/
```

Notes: As of this test, the entire question and statement grammar-space has been tested.

Tests which follow cover command grammar and program logic.

<u>Test 40</u>

Input:
```
user "abe" password 'aaa' execute

trust "abe" 1 execute

user "bob" password 'bbb' execute

trust bob 0.5 execute

user "sales" password 'sss' execute

trust "sales" 0 execute

drop "sales "
```

Notes: Effects were verified by inspection of the user table's storage.

<u>Test 41</u>

Input

(Present credentials as "abe") `cat "streak"`

(Present credentials as "bob") `"fido" dog`

(Presentation of "sales" credentials was rejected)

(Present credentials as "operator") `consider facts execute`

```
who cat execute

who dog execute

consider opinion execute

who cat execute

who dog
```

Output: Answer: `"STREAK"`

There is insufficient information to say

Answer: `"STREAK"`

Answer: `"FIDO"`

Notes: With only certain facts in play, we can't take Bob's statement into consideration, because his trust level is only 50%. We can take Abe's statement into consideration, because his trust level is 100%.

Test 42

Input:
```
consider opinion at 0.4 execute

who dog execute

consider opinion at 0.6 execute

who dog
```

Output: Answer: `"FIDO"`

There is insufficient information to say

Notes: At a consideration level of 0.4, Bob's 0.5 trust level is useful. At a consideration level of 0.5, Bob's trust rating falls under the line.

Test 43

Input:
```
forget "bob" facts
```

Saved:
```
subject "STREAK" *CAT/
```

Test 44

Input: password 'ooo'

(present new credentials)

Test 45

Input: `"streak" cat also for all x: if x cat then x happy execute`

`who happy`

Output: …Answer: `"STREAK"`

Notes: This is the first test showing inference, rather than data retrieval.

Test 46

Input: `after noon go i`

Output: This information is being saved

Input: `after 13:00 go i`

Output: This information is being saved

Input: `after 11:00 go i`

Output: This was already known

Notes: This and the nearby following tests are exercising otherwise untested areas of the Fast prover.

Test 47

Input: `after x seconds before noon go i`

Output: This information is being saved

Input: `after y seconds before noon go i`

Output: This information is being saved

Input: `after x seconds before noon go i`

Output: This was already known

Test 48

Input: ending before monday go i

Output: This information is being saved

Input: ending before sunday go i

Output: This information is being saved

Input: ending before tuesday go i

Output: This was already known

Test 49

Input: during this month happy i

Output: This information is being saved

Input: during this week happy i

Output: This was already known

Test 50

Input: located 1000 to 1500 meters from "comfort inn" be "intuit"

Output: This information is being saved

Input: located 1500 to 1000 meters from "comfort inn" be "intuit"

Output: This was already known

Input: located 1000 to 1500 meters from x be "intuit"

Output: This information is being saved

Input: according to "ted" located 1000 to 1500 meters from x be "intuit"

Output: This information is being saved

Test 51

Input: potentially go i

Output: This information is being saved

Input: go i

Output: This information is being saved

Test 52

Input: i like the sum of 2 and x cat execute

i like the product of 2 and x cat

Notes: The Fast Prover correctly recognized the difference.

Test 53

Input: i like either a cat or a dog execute

either i like 1 cat or i like 1 dog

Output: This information is being saved

This was already known

Notes: The dyadic "or" was evaluated properly, using code that recognizes any operator. "And" is a special case, though, in that the facts for "and" are stored separately.

Test 54

Input: not "streak" dog execute

not "streak" fish execute

i like the number 3.1415 execute

i like the number 2.7818 execute

there exists x such that x happy execute

there exists x such that x happy

Notes: Each new fact was accepted. The repeated fact was called out as such.

<u>Test 55</u>

Input: there exists at least 3 x such that x go

Output: This information is being saved

Input: there exists up to 6 x such that x go

Output: This information is being saved

Input: there exists at least 4 x such that x go

Output: This information is being saved

Input: there exists at least 2 x such that x go

Output: This was already known

Input: there exists 5 x such that x weird

Output: This information is being saved

Input: there exists 6 x such that x weird

Output: That would contradict information already held

<u>Test 56</u>

Input: "x   y   z   " like the number .3/

Saved:
```
"X Y Z" *LIKE the number 0.3/
```

<u>Test 57</u>

Input: at 1900-5-5 go i also at 1904-5-5 go i also during september go i also during january go i also at last century go i

Saved:
```
during 1900-05-05 through 1900-05-05 23:59:59 *GO "OPERATOR"/
during 1904-05-05 through 1904-05-05 23:59:59 *GO "OPERATOR"/
during 2010-09-01 through 2010-09-30 23:59:59 *GO "OPERATOR"/
during 2011-01-01 through 2011-01-31 23:59:59 *GO "OPERATOR"/
during 1900-01-01 through 1999-12-31 23:59:59 *GO "OPERATOR"/
```

Test 58

Input: there exists x y z such that x go y z

Output: This information is being saved

Input: there exists z y y x such that x go y z

Output: This was already known

Test 59

Input: located 0 to 100 meters from b like i c also i like the sum of 0 and d aardvark also i like the sum of e and 0 baboon also i like the difference between f and 0 cat also i like the product of 0 and g dog also i like the product of 1 and h elephant also i like the product of j and 0 fox also i like the quotient of 0 and k gazelle also i like the quotient of L and 1 iguana

Saved:
```
"OPERATOR" *LIKE quantity d *AARDVARK//
up to 100 meters from b *LIKE "OPERATOR" c/
"OPERATOR" *LIKE quantity e *BABOON//
"OPERATOR" *LIKE quantity f *CAT//
"OPERATOR" *LIKE 0 *DOG//
"OPERATOR" *LIKE quantity h *ELEPHANT//
"OPERATOR" *LIKE 0 *FOX//
"OPERATOR" *LIKE 0 *GAZELLE//
"OPERATOR" *LIKE quantity L *IGUANA//
```

Notes: These are optimizer tests.

Test 60

Input: i like twelve both bagel and donut

Saved:
```
"OPERATOR" *LIKE 12 *BAGEL//

"OPERATOR" *LIKE 12 *DONUT//
```

Test 61

Input: not not i like 1 aardvark also not neither i like 1 baboon nor i like 1 cat also not either i like 1 dog or i like 1 elephant also not if and only if i like 1 fox then i like 1 gazelle also not exclusively i like 1 hare or i like 1 iguana

Saved:
```
"OPERATOR" *LIKE 1 *AARDVARK//

either "OPERATOR" *LIKE 1 *BABOON// or "OPERATOR" *LIKE 1
*CAT//

neither "OPERATOR" *LIKE 1 *DOG// nor "OPERATOR" *LIKE 1
*ELEPHANT//

exclusively "OPERATOR" *LIKE 1 *FOX// or "OPERATOR" *LIKE 1
*GAZELLE//

if and only if "OPERATOR" *LIKE 1 *HARE// then "OPERATOR"
*LIKE 1 *IGUANA//
```

Test 62

Input: x be x

Output: This was already known

Test 63

Input: There exists at least 0 x such that x be a unicorn

Output: This was already known

Test 64

Input: (Output in ambiguous English)
```
x sub 1 be "'<  &' execute

facts
```

Output:
```
This information is being saved<br><table border=2><tr><td align=center><b>
#</b></td><td align=center><b>Fact</b></td></tr><tr><td align=center
valign=center>1</td><td><tt>x<sub>1</sub> *BE '''&lt;&nbsp; &nbsp;
&amp;' / </tt></td></tr></table><br>
```

Notes: This is an output test.

Test 65

Input:
```
according to "random guy" cat "streak" execute

who cat execute

consider opinion at 0.6 execute

who cat
```

Output: This information is being saved

Answer: "STREAK"

There is insufficient information to say

Notes: This is a trust test.

Test 66

Input:
```
"streak" cat execute

for all y: not y cat execute

for all z: z cat
```

Output: This information is being saved

That would contradict information already held

This information is being saved

Notes: This is the first test in the Medium prover test series. The Medium Prover used to be a separate item, but is now a part of the Slow Prover.

<u>Test 67</u>

Input:

```
both "streak" and "arrow" cat execute

there exists x such that x cat execute

there exists at least 2 x such that x cat execute

either "streak" toaster or there exists x such that x cat
```

Output:

```
This information is being saved

This was already known

This was already known

This was already known
```

<u>Test 68</u>

Input:

```
>big execute

"rachael" big "shannah" also "shannah" big "rebecca" execute

is "rachael" big "rebecca"
```

Output:

```
This information is being saved

Yes
```

Notes: With this test, the first dictionary test and the first extended logic test have succeeded.

<u>Test 69</u>

Input:

```
=same execute
```

```
"auburn university" same "auburn u" also "auburn u" same "auburn"

execute

is "auburn university" same "auburn"
```

Output: This information is being saved

Yes

<u>Test 70</u>

Input:

```
antonyms big small execute

"rachael" big "shannah" execute

"shannah" small who
```

Output: This information is being saved

Answer: `"RACHAEL"`

<u>Test 71</u>

Input:

```
synonyms big large execute

"rachael" large who
```

Output: Answer: `"SHANNAH"`

<u>Test 72</u>

Input:

```
set animal cat execute

set animal dog execute

"streak" cat execute

who cat execute

who animal execute

is "streak" dog
```

Output: This information is being saved

Answer: "STREAK"

Answer: "STREAK"

No

Notes: Here, "animal" is being defined as an exclusive set or class that includes "cat". "Animal" also includes "dog". Streak is a cat. Asking "Who's a cat" and "Who's an animal" both respond with "Streak". Streak is known not to be a dog because Streak is known to be a cat, and nothing can be both a cat and a dog.

Test 73

Input: facts

Output:

#	**Fact**
1	"RACHAEL" *BIG "SHANNAH" /
4	"STREAK" *CAT /
Dictionary item 1	SMALL is an antonym of BIG
Dictionary item 2	LARGE is a synonym of BIG
Dictionary item 3	CAT is a member of ANIMAL
Dictionary item 4	DOG is a member of ANIMAL

Test 74

Input: :happy execute

i happy execute

i sad

Output: …**Warning**: Strict dictionary use is in effect, but the predication is not in the dictionary…

Test 75

Input:

```
"streak" cat execute

"streak" black cat execute

"fido" big dog execute

"fido" dog
```

Output:

```
This information is being saved

This information is being saved

This information is being saved

This was already known
```

Notes: Adjectives are special, in that they increase specificity.

Test 76

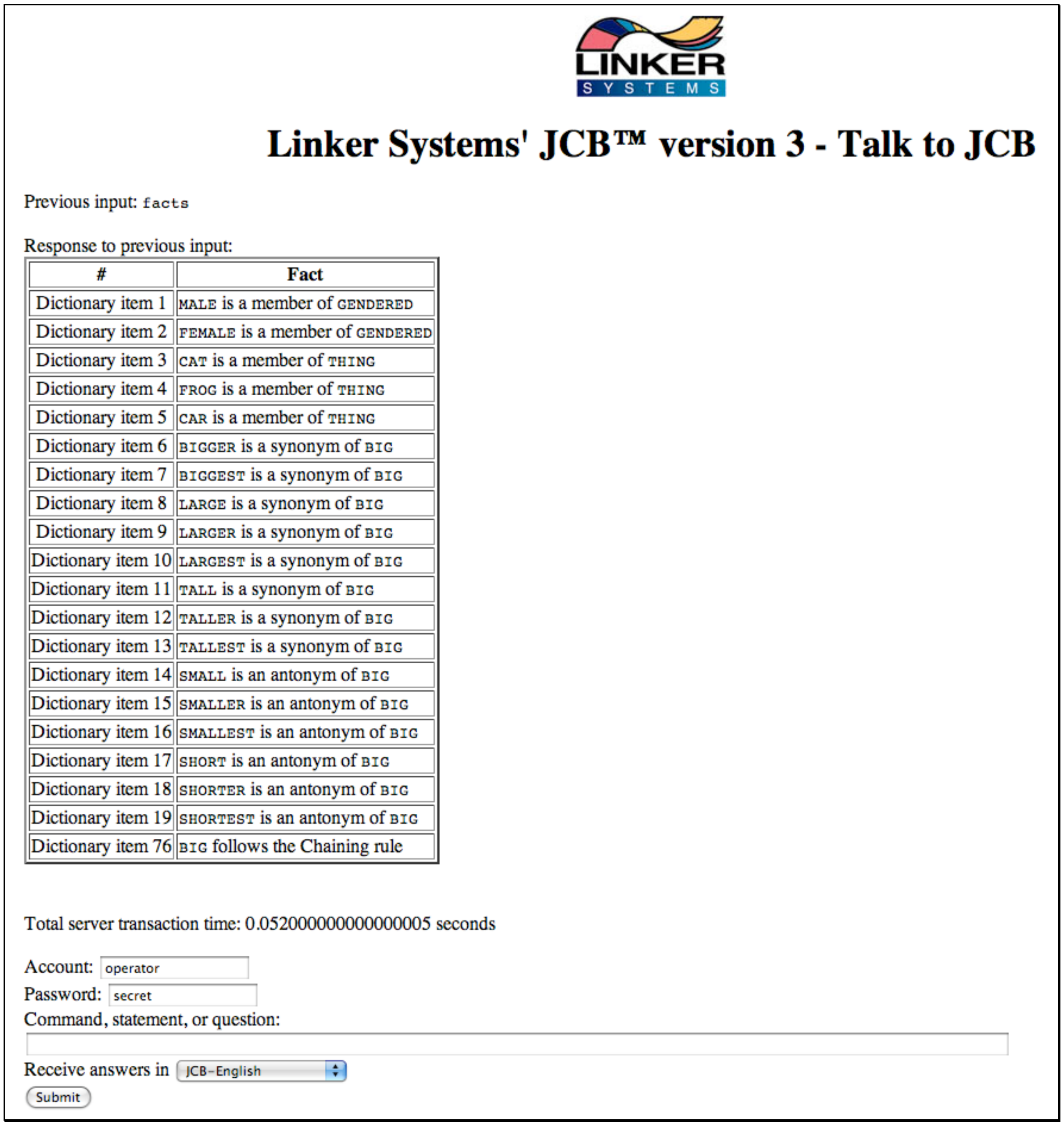
Input:

```
"streak" say 'meow'
```

Saved:

```
"STREAK" *SAY 'meow'/
```

Test 77
The front-end in English:



Linker Systems' JCB™ version 3 - Talk to JCB

Previous input: `facts`

Response to previous input:

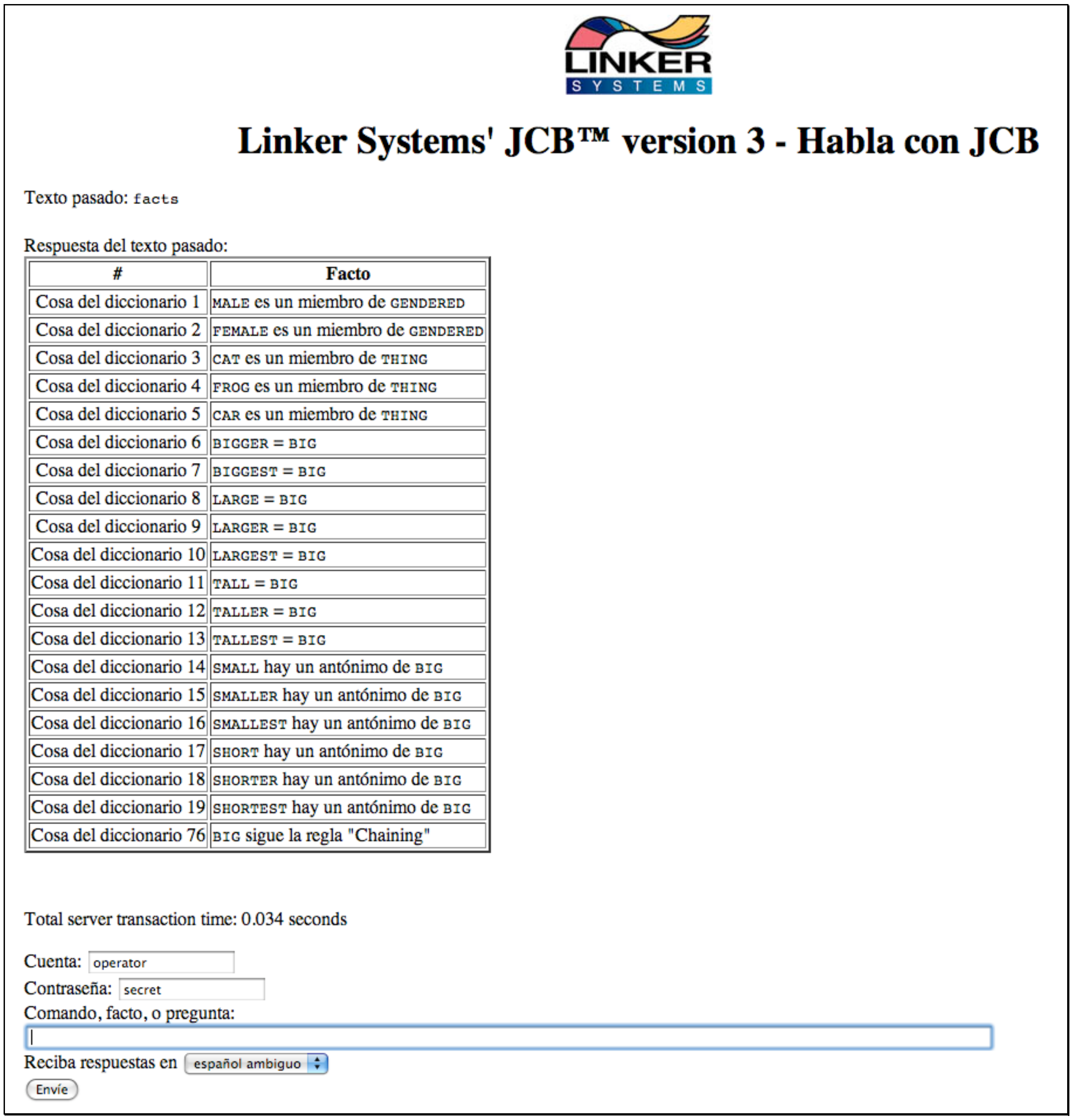
#	Fact
Dictionary item 1	MALE is a member of GENDERED
Dictionary item 2	FEMALE is a member of GENDERED
Dictionary item 3	CAT is a member of THING
Dictionary item 4	FROG is a member of THING
Dictionary item 5	CAR is a member of THING
Dictionary item 6	BIGGER is a synonym of BIG
Dictionary item 7	BIGGEST is a synonym of BIG
Dictionary item 8	LARGE is a synonym of BIG
Dictionary item 9	LARGER is a synonym of BIG
Dictionary item 10	LARGEST is a synonym of BIG
Dictionary item 11	TALL is a synonym of BIG
Dictionary item 12	TALLER is a synonym of BIG
Dictionary item 13	TALLEST is a synonym of BIG
Dictionary item 14	SMALL is an antonym of BIG
Dictionary item 15	SMALLER is an antonym of BIG
Dictionary item 16	SMALLEST is an antonym of BIG
Dictionary item 17	SHORT is an antonym of BIG
Dictionary item 18	SHORTER is an antonym of BIG
Dictionary item 19	SHORTEST is an antonym of BIG
Dictionary item 76	BIG follows the Chaining rule

Total server transaction time: 0.052000000000000005 seconds

Account: operator
Password: secret
Command, statement, or question:

Receive answers in JCB-English

Test 78
The font-end in Spanish:



Linker Systems' JCB™ version 3 - Habla con JCB

Texto pasado: `facts`

Respuesta del texto pasado:

#	Facto
Cosa del diccionario 1	MALE es un miembro de GENDERED
Cosa del diccionario 2	FEMALE es un miembro de GENDERED
Cosa del diccionario 3	CAT es un miembro de THING
Cosa del diccionario 4	FROG es un miembro de THING
Cosa del diccionario 5	CAR es un miembro de THING
Cosa del diccionario 6	BIGGER = BIG
Cosa del diccionario 7	BIGGEST = BIG
Cosa del diccionario 8	LARGE = BIG
Cosa del diccionario 9	LARGER = BIG
Cosa del diccionario 10	LARGEST = BIG
Cosa del diccionario 11	TALL = BIG
Cosa del diccionario 12	TALLER = BIG
Cosa del diccionario 13	TALLEST = BIG
Cosa del diccionario 14	SMALL hay un antónimo de BIG
Cosa del diccionario 15	SMALLER hay un antónimo de BIG
Cosa del diccionario 16	SMALLEST hay un antónimo de BIG
Cosa del diccionario 17	SHORT hay un antónimo de BIG
Cosa del diccionario 18	SHORTER hay un antónimo de BIG
Cosa del diccionario 19	SHORTEST hay un antónimo de BIG
Cosa del diccionario 76	BIG sigue la regla "Chaining"

Total server transaction time: 0.034 seconds

Cuenta: operator
Contraseña: secret
Comando, facto, o pregunta:

Reciba respuestas en español ambiguo

Speakability Improvement Comparisons

Here, the speakability of the previous version of JCB-English is compared. The first example is an informal text, and can be translated loosely:

> Once upon a time, there were three little pigs. One pig made a house of straw, one made a house of sticks, and one made a house of bricks.

This could be translated in the previous version of JCB-English as:

> before now both be x 3 adjective little affects pig and both member x sub 1 x and both member x sub 2 x and both member x sub 3 x and both build x sub 1 1 adjective straw affects house and both build x sub 2 1 adjective stick affects house and build x sub 3 1 adjective brick affects house

Using the improved grammar, this could be better translated as:

> before now both x be 3 little pig also both x sub 1 and both x sub 2 and x sub 3 member x also x sub 1 build a straw house also x sub 2 build a stick house and x sub 3 build a brick house

As an example of a more formal document, I cite the oldest formal document still in good standing. Since this document is older than English, proper translation should be from the original language, and not from an English translation. Below are shown the original, modern but formal English, and JCB-English. First, the original:

בראשית ברא אלהים את השמים ואת הארץ
והארץ היתה תהו ובהו...

Rendered into rather exact English, word-by-word (letter-by-letter, in some cases), that becomes:

> At-[the]-root-[of everything]*, created *elohim*† the-depths-of-space‡ and the-land. And-the-land was unformed and-void…

*Root word is "head"

†*elohim* can be translated as G-d, god, gods, spirit, or spirits (since the sentence is in the singular, but the word is in the pleural, this is taken to mean the royal conjugation)

‡Root word is "water"

In the previous version of JCB-English, this became:

> Both there exists one x such that x be elohim and both for all y
>
> if it is not the case that be 1 elohim y then create 1 elohim y

and both be 1 elohim both 1 god and both 1 spirit and 1 monarch and both create one elohim both the class space and the class land and ending before now both it is not the case that form anything the class land is true and void the class land

In the current version of JCB-English, this becomes:

There exists one x such that x be elohim also for all y: if not y be 1 elohim then 1 elohim create y also 1 elohim be both 1 god and both a spirit and a monarch also one elohim create both the class space and the class land also ending before now not anything form the class land and void the class land

Thus, it has been shown that text can, in general, be translated to JCB-English (including an inline definition of a hard-to-translate word). In these two examples (picked before the grammar improvements), the new grammar results in smaller utterances.

Chapter 8 — The Final Grammar

Below, the grammar is shown in both YACC format (as compiled by Bison, with lexical comments), and in BNF as commonly used. An explanation accompanies the BNF description of the language.

The language definition, in YACC

```
%token ACCORDING ADJECTIVE AFFECTS AFTER ALL ALSO AM AND ANYTHING AT
%token AUTHORITY BCE BEFORE BEGINNING BETWEEN BLANK BOTH BY CLASS
%token CONSIDER DIFFERENCE DO DOES DROP DURING EITHER EMPHASIS
%token ENDING EVENT EXCLUSIVELY EXECUTE EXIST EXISTS FACTS FAR FOR
%token FORGET FROM GOOD HOW IF IMPLIED IS ITEM JOINTLY LAST LEAST
%token LOCATED MANY MIDNIGHT MY NEAR NEITHER NEXT NOON NOR NOT NOW
%token NUMBER OF ON ONLY OPINION OR PASSWORD PM POTENTIALLY PRODUCT
%token PROPERTY QUANTITY QUOTIENT STATEMENT SUB SUCH SUM THEN THE
%token THEN THERE THIS THROUGH TO TODAY TOMORROW TRUST UP USER WITH
%token YESTERDAY YOU YOUR
%token BELIEVE_X /* believe guess */
%token DAY_N     /* Sunday-Saturday */
%token END_X     /* END / */
%token LETTER_N  /* b-h j-z */
%token LIKELY_X  /* certainly likely */
%token ME_X      /* I me */
%token METER_S   /* meter meters */
%token MONTH_N   /* January-December */
%token NOISE_X   /* than */
%token PAIR_X    /* antonyms set synonyms */
%token PERIOD_N  /* second seconds minute minutes hour hours week
                    weeks month months year years decade decades
                    century centuries millennium millennia */
%token SOME_X    /* little most some */
%token TENSE_Q   /* tense when where */
%token TRUE_N    /* false true */
%token WHAT_X    /* what which who */
%token NUMBER_N  /* digits, with or without a decimal point, or A AN
                    HALF ONE-NINETEEN TWENTY-NINETY THE */
%token DOUBLEQUOTEDSTRING_N /* " letters and spaces " */
%token SINGLEQUOTEDSTRING_N /* ' any symbols, which may include
                               doubled-single-quotes ' */
%token WORD_N               /* letters */
%%
```

```
Transmission   : Utterance
               | Transmission EXECUTE Utterance ;

Utterance      : AlsoList
               | IS AlsoList TRUE_N
               | CONSIDER FACTS
               | CONSIDER OPINION
               | CONSIDER OPINION AT NUMBER_N
               | USER DOUBLEQUOTEDSTRING_N PASSWORD
                      SINGLEQUOTEDSTRING_N
               | PASSWORD SINGLEQUOTEDSTRING_N
               | DROP DOUBLEQUOTEDSTRING_N
               | MY FACTS
               | FACTS
               | FORGET Svo
               | FORGET NUMBER_N
               | FORGET DOUBLEQUOTEDSTRING_N FACTS
               | TRUST DOUBLEQUOTEDSTRING_N NUMBER_N
               | '>' WORD_N
               | '=' WORD_N
               | ':' WORD_N
               | PAIR_X WORD_N WORD_N ;

AlsoList       : Svo
               | AlsoList ALSO Svo ;

Svo            : SSimple
               | BOTH AlsoList AND Svo
               | EITHER AlsoList OR Svo
               | NEITHER AlsoList NOR Svo
               | THE STATEMENT AlsoList IS IMPLIED BY Svo
               | IF AND ONLY IF AlsoList THEN Svo
               | IF AlsoList THEN Svo
               | EXCLUSIVELY AlsoList OR Svo
               | ADJECTIVE Vo AFFECTS Svo
               | JOINTLY Svo AND Svo
               | TRUE_N
               | FOR ALL VariableList ':' Svo
               | THERE EXISTS VariableList SUCH THAT Svo
               | THERE EXISTS AT LEAST Number VariableList SUCH THAT
                       Svo
               | THERE EXISTS NumberOr VariableList SUCH THAT Svo
               | THERE EXISTS UP TO Number VariableList SUCH THAT Svo
               | THERE DOES NOT EXIST VariableList SUCH THAT Svo
               | NOT Svo
               | Tense Vo ;

Vo             : Simple
               | BOTH Vo AND Vo
               | EITHER Vo OR Vo
               | NEITHER Vo NOR Vo
               | THE STATEMENT Vo IS IMPLIED BY Vo
               | IF AND ONLY IF Vo THEN Vo
               | IF Vo THEN Vo
               | EXCLUSIVELY Vo OR Vo
               | ADJECTIVE Vo AFFECTS Vo
               | JOINTLY Vo AND Vo
```

```
                | NOT Vo
                | Tense Vo ;

SSimple         : Argument Simple
                | Argument IS Simple
                | Argument NUMBER_N Simple
                | Argument IS NUMBER_N Simple ;

Simple          : Simplest
                | WORD_N Simple ;

Simplest        : WORD_N ArgumentList MayEnd
                | BLANK ArgumentList MayEnd ;

MayEnd          : END_X
                | /*EMPTY, CONTEXTUAL*/ ;

ArgumentList    : /*EMPTY*/
                | ArgumentList TO Argument
                | ArgumentList FROM Argument
                | ArgumentList NOISE_X Argument ;

Argument        : Some Vo
                | WHAT_X
                | ANYTHING
                | EMPHASIS
                | Variable
                | BOTH Argument AND Argument
                | EITHER Argument OR Argument
                | NEITHER Argument NOR Argument
                | THE ITEM Argument IS IMPLIED BY Argument
                | IF AND ONLY IF Argument THEN Argument
                | IF Argument THEN Argument
                | EXCLUSIVELY Argument OR Argument
                | ME_X
                | YOU
                | MY Argument
                | YOUR Argument
                | SINGLEQUOTEDSTRING_N
                | DOUBLEQUOTEDSTRING_N
                | Some THE CLASS Vo
                | THE CLASS Vo
                | THE EVENT Svo
                | THE NUMBER Number
                | THE ITEM Argument WITH PROPERTY Svo ;

Some            : NumberOr Of
                | ALL Of
                | SOME_X Of ;

NumberOr        : QUANTITY Variable
                | NumberLow ;

Number          : Variable
                | NumberLow ;

NumberLow       : NUMBER_N
                | THE SUM OF Number AND Number
                | THE DIFFERENCE BETWEEN Number AND Number
                | THE PRODUCT OF Number AND Number
                | THE QUOTIENT OF Number AND Number
                | HOW MANY ;

Of              : /*EMPTY*/
                | OF ;

Variable        : LETTER_N SUB NUMBER_N
                | LETTER_N ;

VariableList    : VariableList Variable
                | Variable ;

Tense           : AT TimeReference
                | DURING TimeReference
                | NEAR Argument
                | FAR FROM Argument
                | POTENTIALLY
                | TENSE_Q
                | ON GOOD AUTHORITY
                | LIKELY_X
                | ME_X BELIEVE_X
                | ACCORDING TO Argument
                | UP TO Distance FROM Argument
                | LOCATED Distance FROM Argument
                | AT Argument
                | AT LEAST Distance FROM Argument
                | LOCATED Number TO Distance FROM Argument
                | ON OR AFTER TimeSpec
                | ON TimeSpec
                | AFTER TimeSpec
                | BEGINNING AFTER TimeSpec
                | BEGINNING TimeSpec
                | ON OR BEFORE TimeSpec
                | BEFORE TimeSpec
                | ENDING BEFORE TimeSpec
                | ENDING TimeSpec ;

TimeReference   : TimeSpec
                | TimeSpec THROUGH TimeSpec ;

TimeSpec        : SimpleTime
                | Offsets FROM SimpleTime
                | Offsets BEFORE SimpleTime ;

SimpleTime      : NOW
                | Time Date
                | Date Time
                | Date
                | Time
                | THIS PERIOD_N
                | LAST PERIOD_N
                | NEXT PERIOD_N
                | MONTH_N ;

Date            : DateLow BCE
                | DateLow
                | DAY_N
                | TODAY
                | TOMORROW
                | YESTERDAY ;
DateLow         : NUMBER_N '-' NUMBER_N '-' NUMBER_N
                | MONTH_N NUMBER_N NUMBER_N
                | NUMBER_N MONTH_N ;
Time            : TimeLow AM
                | TimeLow
                | TimeLow PM
                | NOON
                | MIDNIGHT ;
TimeLow         : NUMBER_N ':' NUMBER_N ':' NUMBER_N
                | NUMBER_N ':' NUMBER_N ;
Distance        : Number METER_S ;
Offsets         : Offset
                | Offsets Offset ;
Offset          : NumberOr PERIOD_N ;
%%
```

The language definition, in BNF, with explanation following

```
Transmission  → [Utterance execute]… Utterance

Utterance     → AlsoList
                | is AlsoList
                | consider facts
                | consider opinion [at NUMBER]
                | [user "string"] password 'string'
                | drop "string"
                | [my] facts
                | forget Predication
                | forget NUMBER
                | forget "string" facts
                | trust "string" [at NUMBER]
                | do not trust "string"
                | >WORD
                | =WORD
                | :WORD
                | {antonyms synonyms set spanish} WORD WORD

AlsoList      → [Predication also]… Predication

Predication   → [Argument] [IS] [A] [WORD…] {WORD blank} [[{to from than}]
                Argument…] [{end /}]
                | both AlsoList and Predication
                | {either exclusively} AlsoList or Predication
                | neither AlsoList nor Predication
```

```
               | the statement AlsoList is implied by Predication
               | if [and only if] AlsoList then Predication
               | {true false}
               | for all VariableList: Predication
               | there exists [at least NumberOr] Variable… such that
                 Predication
               | there exists [Number] Variable… such that Predication
               | there exists up to NumberOr Variable… such that Predication
               | there does not exist Variable… such that Predication
               | not Predication
               | adjective AlsoList affects Predication
               | jointly AlsoList and Predication
               | Tense Predication

Argument     → Number [of] Predication
               | {all little most some} [of] Predication
               | {what which who}
               | anything
               | emphasis
               | Variable
               | both Argument and Argument
               | {either exclusively} Argument or Argument
               | neither Argument nor Argument
               | the item Argument is implied by Argument
               | if [and only if] Argument then Argument
               | {me i you}
               | {my your} Argument
               | "string"
               | 'string'
               | [{all little most some} [of]] the class Predication
               | the event Predication
               | the number Number
               | the item Argument with property Predication

Number       → [quantity] Variable
               | NUMBER
               | the {sum product quotient} of Number and Number
               | the difference between Number and Number
               | how many

Variable     → LETTER [sub NUMBER]

Tense        → {at during} TimeSpec [through TimeSpec]
               | {at near} Argument
               | far from Argument
```

| {tense when where potentially certainly likely}
| on good authority
| i{believe guess}
| according to *Argument*
| up to *NumberOr* meter[s] from *Argument*
| at least *NumberOr* meter[s] from *Argument*
| located [*NumberOr* to] *NumberOr* meter[s] from *Argument*
| on [or {before after}] *TimeSpec*
| beginning [after] *TimeSpec*
| ending [before] *TimeSpec*
| {before after} *TimeSpec*

TimeSpec → [*Offset…* {from before}] *SimpleTime*

SimpleTime → now
| *Time* [*Date*]
| *Date* [*Time*]
| {this last next} {second minute hour day week month year decade century millennium}
| {january-december}

Date → *DateLow* [bce]
| {sunday-saturday}
| {today tomorrow yesterday}

DateLow → NUMBER-NUMBER-NUMBER
| {january-december} NUMBER [NUMBER]

Time → NUMBER:NUMBER[:NUMBER] [{am pm}]
| {noon midnight}

Offset → *Number* {second[s] minute[s] hour[s] day[s] week[s] month[s] year[s] decade[s] century centuries millennium millennia}

<u>Explanation of the BNF grammar</u>

A *Transmission* is composed of one or more *Utterance*s.

An *Utterance* is a statement of fact (an *AlsoList* not containing a question-word), a question (is *AlsoList*), a query (an *AlsoList* containing a question-word), a command, or a dictionary definition. The consider facts command requires that only statements at a trust level of 1 be considered in answering questions. Consider opinion allows any trust level to come into play. If a number is specified, then any statement at or above that trust level will be applied. The `password` command changes a user's password. `My facts` shows a list of facts that the user created. `Facts` alone lists all facts, including dictionary definitions. `Forget`, followed by a *Predication* or a fact number, drops a fact from the knowledge base. The operator can drop any fact, but users can only drop their own. The following commands can only be executed by the operator. User accounts can be created and dropped using the `user` and `drop` commands, respectively. A dropped user can no longer log in, but the user's facts are retained. `Forget "`*user name*`" facts` drops the knowledge, whether or not the user is still valid. Like `consider`, `trust` and `do not trust` establish a trust level (between 0 and 1) for a given user. The remaining commands set up a dictionary. `>` defines a word as having one-way comparison chaining. For instance, `>big` establishes "big" with two arguments to behave symbolically like the ">" symbol, effectively making it the English word "bigger". Similarly `=` defines a word as having two-way comparison chaining. For instance, `=same` establishes "same" with two arguments to behave symbolically like the "=" symbol. The : symbol defines a word as being in the dictionary, and puts the dictionary into Strict mode, in which words not already in the dictionary are flagged as a problem. The `antonyms` command establishes the second word as the opposite of the first, and any use of the second word will be translated to a use of the first word. For instance, `antonyms big small` makes "small" the opposite of "big", and any use of "small" will become a use of "big". For instance after `antonyms big small`, the statement `small a cat/ an elephant` (a cat is smaller than an elephant) will be stored as `big an elephant/ a cat`. The `synonyms` command replaces subsequent uses of the second word with uses of the first word. The set command establishes mutual exclusion sets. This is best shown by example. Set animal cat states that all cats are animals. Set animal elephant also establishes elephants as animals, but also states that no thing can be both a cat and an elephant. The spanish command establishes a Spanish language translation. For instance, spanish gato cat establishes "gato" as the Spanish word for "cat".

An *AlsoList* is one or more predications separated by the word also. Also has the same meaning as ^ in logic, and is the only infix operator. It allows afterthought connectors, while all other operators are Polish or prefix operators, and bind explicitly.

A *Predication* is the main statement or question type in JCB-English. A *Predication* may by prefixed by one or more nots (which negate the rest of the predication), *Tense*s, and words used as adjectives. The most basic predications are a predicate word followed by one or more arguments. Following the argument list, end or / may be added. The use of end or / is context sensitive, and is used when meaning would otherwise not be clear. For instance, an elephant big a cat would be considered by JCB-English to be two arguments, one an "elephant" type of "big", and another which is a cat. Using instead an elephant/ big a cat allows JCB-English to understand that "big" has two arguments, since the slash terminates the "elephant" predication. Placing an asterisk in front of a predication word forces that word to be used as a predication word, even if the word is a keyword. If, instead of a predicate word, the word blank is used, then the predication is a fill-in-the-blank query. For instance, an elephant/ blank a cat would respond with "big". The prefix operators both, either, exclusively, neither, is implied by, if, if and only if, for all, there exists, and there does not exist have the same meaning as they do in classical logic. Adjective words, as described above, bind right to left. The adjective operator allows explicit adjective binding. The jointly operator behaves like English's "and jointly" For instance, i like an adjective jointly green and blue affects ball means that I like a ball which is somehow green and blue at the same time. i like an adjective both green and blue affects ball means that I like a green ball, and that I like a blue ball. Last, true and false are constant predications, and are probably not useful in standard discourse (since they will almost always be factored out during the optimization phase), but are very useful for testing purposes. Early in testing, it was found that people like to add "is", "is a", "is the", "a", or "the" between the subject and the first predicate word. The grammar was expanded to allow for this extraneous verbiage. Some so-called little words (as defined in English) are allowed the user, but ignored by the parser, to make input easier. In some contexts, where the presence or absence of such a word does not add or remove any meaning, a, an, from, is, than, the, and to are ignored (by virtue of the parser recognizing the word in a position where the word doesn't matter, and skipping over the word).

An *Argument* is most typically a number, followed by a predication. Usually, the predication will have no arguments itself. Thus, a cat is a typical argument, with the "number" a and the predication cat. The number can be a distinct number, or all, little, most or some of. The operators both, either, exclusively, neither, is implied by, if, and if and only if can be applied to arguments. Computationally, this is handled by forming separate predications, none of which contain arguments with operators within them. Direct, possessional, and query-forming pronouns, `me`, `i`, `you`, `my`, `your`, `what`, `which`, and `who` can be used, as well as variables. Proper names can be used by enclosing them in double-quotes. In names, capitalization and spacing are not significant. Literal text strings can be used as such by enclosing the string in single quote. Capitalization and spacing are significant within such strings. Within a literal text string, a single quote can be represented using two single-quote characters. Classes of things can be represented using `the class`. Events and numbers can be used as arguments using `the event` and `the number`, respectively. `With property` allows short-hand use. `With property` is never stored into the knowledge base. For instance `i take an apple with property i like` is evaluated and stored as `i like an apple` and `i take an apple`. The argument `anything` is a place-holder, in that `x go y` and `x go y anything` are the same. `Anything` is most useful when an argument needs to be skipped. "`Emphasis`" is a special and complicated case. Let's say we define the predicate "`x go y z t`" as meaning "x goes from y to z over route t." "`Go restaurant`" would mean a restaurant in a going sort of way, which doesn't make much sense. Let's say that we wanted to talk about a "destination restaurant". That would mean that we need to modify "restaurant" with "destination". Since "destination" is the third argument of "go", we'd use "`Adjective go anything anything emphasis modifies restaurant`".

NumberOr means a *Number* (as defined below) or a *Variable*.

Wherever *Number* appears in the grammar, a number can be used, either as digits (with or without a decimal point) or as a single word. The prefix operators `sum`, product, quotient, and difference may be used. Questions are formed using how many in place of a number. A variable can be used as if a number, too, when prefixed by quantity to distinguish it from an argument.

A *Variable* is a letter, other than A or I, optionally followed by sub and an integer. For instance x sub 2 is "x_2".

*Tense*s can set time or place, as in most languages, and belief, as in Hopi. Time tenses include after, at, before, beginning, during, ending, on, and through, alone or in consort. Distances can use at, at least, far, located, near, and up to define position. Belief uses according to, certainly, likely, i believe, i guess, on good authority, and potentially. When and where are specific questions. Tense is a general question, and can be answered with time, location, and/or believability. Time specifications can include offsets, from a second to millennia, as well as the words am, bce, last, next, midnight, now, pm, this, today, tomorrow, yesterday, day names, and month names, all of which have the same meanings here as in English.

Chapter 9 — Acceptance Testing

The main question answering system types, as mentioned above, are represented by Google, Prolog, SQL, and Yahoo Answers. Of this list, only SQL is suitable for user acceptance testing. User acceptance testing was performed as described below with the help of 30 volunteer test subjects. The entrance criteria for test subjects were that they were aged 19 or more, and that they had no prior knowledge of SQL.

Tests that Cannot be Run

Google can search documents for results, but there is little or no control of what data gets in, or how. There is no cross-document correlation done. There is no method to deal with veracity. Finally, Google does not present its results as answers. Instead, it presents a list of documents; often thousands of documents. For purposes of testing, there is no way for users to directly enter data into the system.

Prolog is a system directly built for the entry of facts, correlation of those facts, and for answering questions regarding those facts and getting back reasoned responses. Unfortunately, Prolog requires that the Prolog program be compiled before queries can be made. There is no reasonably convenient way to intermix the entry of facts, rules, and questions into Prolog, since this would require a recompilation each time the user wants to switch from fact or rule entry to questions. Additionally, the difficulty of sharing data entry with multiple (n) users is $O(n^2)$.

In concept, Yahoo Answers is the closest system to what is being attempted here. Questions can be entered into Yahoo Answers, and facts may likewise be entered. The two main differences are that in Yahoo Answers, questions are entered first, and the facts, in the form of answers, are entered later. The second big difference, of course, is that Yahoo Answers uses volunteer human labor. This leads to delays of minutes to days in getting answers. Simply put, there isn't enough time to get answers from Yahoo Answers or to test the system's use.

Tests that were Run

There were two test groups, adults who are not programmers and do not know any SQL (23 subjects), and adult programmers who do not know any SQL (7 subjects). The number of programmers who do not know SQL is low, since finding programmers who do not know any SQL is fairly difficult. Each test group was taught JCB-English, for up to 15 minutes, and taught SQL, for up to 15 minutes. Everyone who registered initial interest agreed to participate and began instruction.

After training, each subject was asked to perform a series of data entry and data retrieval tasks, with the total task time for each language limited to 15 minutes. The training forms (used as "cheat sheets") used, the list of tasks, and questionnaires used each appear following this document.

Completion Metrics

Each of the following was measured directly by the author, and compared with SQL only. No comparison can be made to any other commonly used system, as no other commonly-used system can switch between data entry and answering mode on the fly. Below, each area with a light green background indicates a result in the planned range.

Question	JCB	SQL
How long does it take the system read & write the test knowledge base?	<0.1s	<0.1s
How long does it take the system to compile a typical query?	<0.1s	<0.1s
How long does the fast prover typically take?	<0.1s	<0.1s
Typical Slow Prover time, without chaining rules:	<0.1s	n/a
Typical Slow Prover time, with chaining rules:	<1.3s	n/a
Can the system operate with multiple users in the same knowledge space?	Yes	Yes
Given that the size of the sample translations presented in the original thesis are 100%, what is the size of those same translations in the current language?	80%	n/a
Given that the size of the PLGS problems' translations are 100%, what is the current minimum size of those same translations?	66%	n/a
Can sentences be entered in a more English-like subject-verb-object manner?	Yes	No
Can the concept of chaining (i.e., if a>b, and b>c, then a>c) be entered as a simple command or statement, rather than formulating a theorem covering the matter?	Yes	No
Can the concept of equivalence (i.e., if a=b, and b=c, then a=c) be entered as a simple command or statement, rather than formulating a theorem covering the matter?	Yes	No
Can the concept of membership in a group, such as cats and dogs being members of mammals, be entered as a simple command or statement, rather than formulating a theorem covering the matter?	Yes	No

Question	JCB	SQL
Can the concept of exclusion within a group, such as being a cat and being a dog being mutually exclusive, be entered as a simple statement or command, rather than formulating a theorem covering the matter?	Yes	No
Can the concept of synonyms (i.e., big, bigger, biggest, large, larger, largest, huge, immense, … are all the same predication) be entered as a simple command or statement, and handled in the parse or lex phase, thus alleviating the need for formulation of theorems, and also alleviating the need of execution at proof-time?	Yes	No
Can the concept of antonyms (i.e., and small being the same predication, but with arguments reversed, such as Big(x,y)↔Small(y,x)) be entered as a simple command or statement, and handled in the parse or lex phase, thus alleviating the need for formulation of theorems, and also alleviating the need of execution at proof-time?	Yes	No
Is the grammar proven to be SLR-1, and thus completely syntactically unambiguous?	Yes	Yes
Percentage of lines tested:	100%	n/a
Can the system operate with or without a "strict vocabulary" mode, in which "strict" means that only pre-approved predication words are available?	Yes	n/a
Can all previously documented features still be used?	Yes	n/a
In which additional languages, can the system output?	Spanish	n/a

The following tables each derive their data from observations of the subject, or from questionnaire answers by the subjects. Below, a light green background indicates whether JCB-English or SQL had a better (rather than higher or lower) result.

Subjects who completed the exercise (after having started):

	Non-programmers	Programmers	Overall
JCB-English	100%	100%	100%
SQL	91%	100%	93%

Average learning time, ± standard deviation, measured in minutes, but shown here in minutes and seconds, with a 15-minute maximum:

	Non-programmers	Programmers	Overall
JCB-English	11:08 ±2:35	11:26 ±1:54	11:12 ±2:25
SQL	13:06 ±1:55	12:43 ±2:22	13:00 ±2:00

The expectation with respect to learning time was that programmers would have approximately the same or better learning time for JCB-English than would non-programmers. However, non-programmers learned JCB-English faster than did programmers.

Average task time, ± standard deviation, measured in minutes, but shown here in minutes and seconds, with a 15-minute maximum:

	Non-programmers	Programmers	Overall
JCB-English	8:34 ±3:40	10:26 ±4:05	9:00 ±3:47
SQL	15:00 ±0:00	15:00 ±0:00	15:00 ±0:00

Again, the expectation was that programmers would be able to perform more tasks than would non-programmers using JCB-English. Again, the non-programmers did better at JCB-English than did the programmers.

Average percentage of tasks completed, ± standard deviation:

	Non-programmers	**Programmers**	**Overall**
JCB-English	99% ±5%	90% ±17%	97% ±10%
SQL	29% ±29%	52% ±29%	34% ±30%

The expectation was that programmers would complete more tasks than would non-programmers. However, the opposite was true. Programmers still did better at JCB-English than they did at SQL.

Average number of help instances, ± standard deviation:

	Non-programmers	**Programmers**	**Overall**
JCB-English	1.2 ±2.2	0.6 ±0.8	1.0 ±2.0
SQL	3.2 ±2.8	1.7 ±0.8	2.8 ±2.5

Adjusted average number of completed tasks per 15-minute period (counting each help instance as equivalent to 5 minutes):

	Non-programmers	**Programmers**	**Overall**
JCB-English	36.1 ±26.0	30.9 ±24.6	34.9 ±25.4
SQL	3.6 ± 3.9	7.0 ± 3.4	4.4 ± 4.0

Part of the reason that the standard deviation is so high for the number of tasks completed in a 15-minute period is that seven of the subjects were able to score between 79 and 105 tasks per 15-minute period.

The language is… (Terrible=0; Wonderful=4)

	Non-programmers	Programmers	Overall
JCB-English	2.9 ±0.8	2.9 ±1.1	2.9 ±0.8
SQL	1.2 ±0.9	1.4 ±1.4	1.3 ±1.0

Learning the language is… (Difficult=0; Easy=4)

	Non-programmers	Programmers	Overall
JCB-English	3.6 ±0.8	3.7 ±0.5	3.6 ±0.8
SQL	1.0 ±1.2	1.9 ±1.5	1.2 ±1.3

Using the language is… (Difficult=0; Easy=4)

	Non-programmers	Programmers	Overall
JCB-English	3.4 ±1.0	2.9 ±1.1	3.3 ±1.0
SQL	1.0 ±1.2	1.4 ±1.0	1.1 ±1.2

Using the language is… (Frustrating=0; Satisfying=4)

	Non-programmers	Programmers	Overall
JCB-English	3.0 ±1.1	2.7 ±1.1	2.9 ±1.1
SQL	1.0 ±0.9	1.4 ±1.0	1.1 ±0.9

The system is… (Too slow=0; Fast enough=4)

	Non-programmers	Programmers	Overall
JCB-English	3.1 ±0.9	2.4 ±1.8	2.9 ±1.2
SQL	1.4 ±1.5	2.1 ±1.6	1.6 ±1.5

The experience was… (Dull=0; Stimulating=4)

	Non-programmers	Programmers	Overall
JCB-English	3.1 ±0.8	3.1 ±1.5	3.1 ±1.0
SQL	1.8 ±1.4	2.4 ±1.5	2.0 ±1.4

The language is… (Rigid=0; Flexible=4)

	Non-programmers	Programmers	Overall
JCB-English	2.4 ±1.1	3.0 ±1.0	2.6 ±1.1
SQL	0.5 ±0.7	1.1 ±1.2	0.7 ±0.9

Organization of the interface is… (Confusing=0; Very clear=4)

	Non-programmers	Programmers	Overall
JCB-English	3.0 ±1.3	2.9 ±1.5	3.0 ±1.3
SQL	0.9 ±0.8	2.9 ±1.3	1.4 ±1.3

The prompts are… (Confusing=0; Very clear=4)

	Non-programmers	Programmers	Overall
JCB-English	3.1 ±1.1	3.0 ±1.5	3.1 ±1.2
SQL	1.4 ±1.1	1.6 ±1.0	1.4 ±1.1

Error messages are… (Helpful=0; Not helpful=4)

	Non-programmers	Programmers	Overall
JCB-English	3.0 ±1.3	3.0 ±1.5	3.0 ±1.3
SQL	1.5 ±1.5	1.7 ±1.3	1.6 ±1.4

Remembering the words and commands is… (Difficult=0; Easy=4)

	Non-programmers	Programmers	Overall
JCB-English	3.1 ±1.1	3.7 ±0.5	3.2 ±1.1
SQL	0.9 ±1.1	1.4 ±1.4	1.0 ±1.1

Using this system at work would allow me to accomplish tasks quickly. (Unlikely=0; Likely=4)

	Non-programmers	Programmers	Overall
JCB-English	2.5 ±1.6	2.8 ±1.5	2.6 ±1.5
SQL	0.7 ±1.0	1.7 ±1.4	0.9 ±1.2

Using this system at work would allow me to accomplish more. (Unlikely=0; Likely=4)

	Non-programmers	Programmers	Overall
JCB-English	2.5 ±1.5	2.3 ±1.4	2.5 ±1.4
SQL	0.7 ±0.8	1.3 ±1.0	0.8 ±0.9

It would be easy for me to get this language to do what I want it to. (Unlikely=0; Likely=4)

	Non-programmers	Programmers	Overall
JCB-English	2.9 ±1.4	3.4 ±0.5	3.0 ±1.2
SQL	1.3 ±1.4	2.0 ±1.4	1.5 ±1.4

If I were to use this system, I'd become a… (Novice=0; Expert=4)

	Non-programmers	Programmers	Overall
JCB-English	3.5 ±0.8	3.4 ±0.8	3.4 ±0.8
SQL	1.5 ±1.4	2.9 ±1.2	1.9 ±1.5

This language is… (Inaccurate=0; Accurate=4) Note that there is an objectively correct answer for both languages of Accurate. Note that this question is purely a matter of perception, as both languages are entirely accurate.

	Non-programmers	Programmers	Overall
JCB-English	3.3 ±1.1	3.3 ±1.5	3.3 ±1.2
SQL	1.9 ±1.2	2.6 ±1.5	2.1 ±1.3

Compared to other computer systems, this language is more… (Contrived=0; Natural=4)

	Non-programmers	Programmers	Overall
JCB-English	3.1 ±1.1	3.8 ±0.4	3.3 ±1.0
SQL	0.8 ±1.0	0.8 ±0.8	0.8 ±0.9

Using this language, you have to remember the conversation… (Totally=0; Not at all=4)

	Non-programmers	Programmers	Overall
JCB-English	2.2 ±1.4	3.1 ±0.9	2.5 ±1.3
SQL	1.1 ±1.5	0.6 ±0.8	0.9 ±1.4

The responses to this question are a bit surprising, in that the objective answer is 0 in JCB-English, and 4 in SQL.

This language does what I would expects (fulfils the promise)… (Not at all=0; Wonderfully=4)

	Non-programmers	Programmers	Overall
JCB-English	3.3 ±1.0	3.7 ±0.5	3.3 ±0.9
SQL	1.6 ±1.2	2.4 ±1.5	1.8 ±1.3

This language's capabilities… (Fall short=0; Are great=4)

	Non-programmers	Programmers	Overall
JCB-English	3.1 ±1.0	2.7 ±1.0	3.0 ±1.0
SQL	1.6 ±1.3	1.6 ±1.0	1.6 ±1.2

Fixing a mistake is… (Difficult=0; East=4)

	Non-programmers	Programmers	Overall
JCB-English	3.3 ±1.1	3.3 ±1.5	3.3 ±1.2
SQL	1.5 ±1.7	1.4 ±1.5	1.5 ±1.6

I want to book-mark a web-site running this language… (Not=0; For sure=4)

	Non-programmers	Programmers	Overall
JCB-English	2.2 ±1.5	2.6 ±1.4	2.3 ±1.4
SQL	0.6 ±1.1	1.1 ±0.9	0.7 ±1.0

The same data has to be entered… (Repeatedly=0; Once=4) Note that for both languages, the objectively correct answer is Once. This question is one of impressions or training, as it can be shown that in both languages, the objective answer is 0.

	Non-programmers	Programmers	Overall
JCB-English	3.3 ±1.1	2.3 ±1.9	3.1 ±1.3
SQL	1.9 ±1.8	2.4 ±1.8	2.0 ±1.8

Statements can be undone (forgotten)… (Not at all=0; Easily=4)

	Non-programmers	Programmers	Overall
JCB-English	3.4 ±1.1	3.7 ±0.5	3.5 ±0.9
SQL	1.9 ±1.4	2.0 ±1.2	1.9 ±1.3

The language is useful for storing data. (Not at all=0; Very useful=4)

	Non-programmers	Programmers	Overall
JCB-English	3.3 ±0.9	3.3 ±1.5	3.3 ±1.0
SQL	2.0 ±1.3	2.6 ±1.6	2.2 ±1.4

The language is useful for retrieving data. (Not at all=0; Very useful=4)

	Non-programmers	**Programmers**	**Overall**
JCB-English	3.5 ±0.8	3.1 ±1.5	3.4 ±1.0
SQL	2.0 ±1.5	2.6 ±1.1	2.2 ±1.4

This language can do some of the mental work for me. (Not at all=0; Certainly=4)

	Non-programmers	**Programmers**	**Overall**
JCB-English	2.9 ±1.4	2.9 ±1.5	2.9 ±1.4
SQL	1.1 ±1.4	1.7 ±1.4	1.2 ±1.4

Conclusion

The main objectives of this research were to produce a grammar that is similar to English, using techniques from Loglan and other logic areas, and to produce a system which responds to the language so produced logically, coherently, usefully, and relatively quickly. These goals have been accomplished: The current grammar is relatively small, easy to understand, and useful. The Slow Prover is fast enough, using a single laptop computer, to respond to users when using a small knowledge-base size. The system can, in the future, be scaled into a useful tool with broad range. The system can produce its output multilingually (currently using JCB-English, English, Spanish), and unambiguously in JCB-English.

Tests performed on the system by actual prospective users, known in the computer industry as "alpha testers", had a number of expected results, and a few unexpected results. The expected results were that this new language is indeed easy for the average person to learn and use. Also as expected is the fact programmers outperformed non-programmers using control language, SQL. The unexpected results were that the user acceptance level, in every category except production of error messages, was greatly positive. Another unexpected result is that the non-programmers outperformed programmers in many aspects of learning and using JCB-English. A suspicion is that there may be left-brain and right-brain affinity in languages and careers.

If a single-sentence conclusion can be offered, then that conclusion is this: JCB-English may well prove to be a useful tool for information storage and retrieval, and for expert and logical processing of that knowledge.

Chapter 10 — Future Work

Although a great deal of progress and improvement has been made, there is always additional work to do. Some of the future work involves straightforward development, while other work involves some research element. Below appears a list of such work:

<u>Research</u>

- In order to allow for direct input in languages other than English, appropriate grammars will have to be researched. As of this writing, it is not certain that languages outside of the Germanic family (including English, the result of at least one "mixed marriage") can be used for logic without modifications to the grammar so great as to make the new language unrecognizable.
- The addition of COBOL-like noise words at various locations in the grammar may obviate the need to "/" or "end" in various contexts or circumstances. It would bear scrutiny to determine if a grammar simplification can occur because of this.

<u>Research and Development</u>

- The system, as implemented is fast, but runs on a single processor. For widespread use, the system will have to be modified to run simultaneously on many processors.
- Since numbers and variables are present, the system's provers can be expanded to handle straight algebra problems, as well as word problems involving any branch of mathematics.

- During the human-subject ("alpha") testing, a number of suggestions were made. One of the more important suggestions was the proposition that if Streak is a multicolored cat, then Streak is a cat, and Streak is multicolored. The JCB-English language, as currently defined, recognizes from '"Streak" is a multicolored cat.' that Streak is a cat and that Streak is a multicolored cat. It does not directly recognize the independent concept that Streak is multicolored. Adding this capability would involve some research and some development. The research involved is to determine which of the following is true of any noun-adjective-predicate set A:P(N):
  - The set can be expressed as two noun-predicate sets, A(N) and P(N).
  - The adjective may or may not be separable, and this separability can be listed in the dictionary, so that A:P(N) can be separated to A(N) and P(N) if the dictionary entry for A so allows.
  - The predication may or may not be separable, and this separability can be listed in the dictionary, so that A:P(N) can be separated to A(N) and P(N) if the dictionary entry for P so allows.
  - The adjective-predication pair may or may not be separable, and this separability can be listed in a table, so that A:P(N) can be separated to A(N) and P(N) if A:P appears in the Separable table.

  Once the rules for separability are determined, the addition of a SEPARABLE rule may be in order.
- Currently, the JCB-English parser recognizes certain words as COBOL-like "noise" words. An example is the word "than". There are certainly more such noise words,

and those words should be added to the lexer's list of such words. Some of the words listed currently as noise could actually be used to rearrange arguments.

Development

- The system can make use of a complete dictionary, defining all predicates, and the meanings of each predicate's arguments. This would include set classifications, synonyms, and antonyms.
- The system can also make use of a complete Spanish dictionary, defining translations for each predicate.
- The system can make use of translation dictionaries for additional languages.
- The system currently uses 8-bit characters. To use certain other languages, 16-bit characters would be more appropriate.
- The system currently uses brute force for its proofs, without making a determination as to which path would be the most likely to form a conclusion. Similarly, no cuts are used. Addition of optimization would be helpful.
- The system currently allows facts determined by the slow prover, but not used in the final proof, to go unreferenced, and thus to be swept away by the Garbage Collector. An alternative plan would be to retain such facts.
- The optimizer should be able to recognize predicate terms used but unknown, and determine (using truth tables or other means) whether use of the term can be optimized out completely.
- The current grammar allows for assignment of ownership using the insertion of "my" or "your". This typically appears in a sentence in this manner: "I like my one cat." The presence of "my" or "your" absolutely implies that an argument (as defined in the

grammar) follows. An additional grammar rule (or rules) can be added to recognize an argument with a missing quantifier. The missing quantifier would assumed to be "one", which would allow simplification of this example sentence to "I like my cat."

- In the event of a syntax error, when the parser can determine what the user likely meant, the user interface should not only report the error, but should report (in the text entry area) the most likely correct text, so that the user could then press Return or Enter, or click on Submit, without having to retype or correct the entry.

References

Agarwal & Shahshahani, patent application #2004-0085162, *Method and apparatus for providing a mixed-initiative dialog between a user and a machine*, May 6, 2004.

Agichtein, Eugene et al. (2001). Learning search engine specific query transformations for question answering. *Proceedings of the 10th international conference on World Wide Web*, 2001, pages 169-178. ACM.

Agichtein, Eugene et al. (2004). Learning to find answers to questions on the Web. *ACM Transactions on Internet Technology*. May, 2004, pages 129-162. ACM.

Ahamed. Patent #5,809,493. *Knowledge processing system employing confidence levels.* September 15, 1998.

Andrenucci, Andrea & Sneiders, Eriks. (2005). Automated Question Answering: Review of the Main Approaches. *Proceedings of the Third International Conference on Information Technology and Applications,* 2005. IEEE.

Appelt et al. Patent application #2003-0078766. *Information retrieval by natural language querying.* April 24, 2003.

Armstrong. Patent #5,855,002. *Artificially intelligent natural language computational interface system for interfacing a human to a data processor having human-like responses*. December 29, 1998.

Bangalore et al. Patent application #2003-0130841. *System and method of spoken language understanding in human computer dialogs*. July 10, 2003.

Baral, Chitta and Tari, Luis. (2006). Using AnsProlog with Link Grammar and WordNet for QA with deep reasoning., 2006. IEEE.

Beauregard & Armijo-Tamez. Patent-related document #RE-39,090. *Semantic user interface.* May 2, 2006.

Beeman, William et al. (1987). Hypertext and Pluralism: From Lineal to Non-lineal Thinking. *Hypertext Papers,* November, 1987, pages 67-88.

Beierle, Christoph et al. (1993). Knowledge Representation for Natural Language Understanding: The LLILOG Approach. IEEE Transactions on Knowledge and Data Engineering, June 1993, Pages 386-401. IEEE.

Bennet. Patent application #2005-0080614. *System & method for natural language processing of query answers*. April 14, 2005.

Bennet. Patent application #2005-0086049. *System & method for processing sentence based queries*. April 21, 2005.

Bernardi et al. (2007). Lite Natural Language. Free University of Bozen-Bolzano, Italy. <http://www.inf.unibz.it/~thorne/perso/lite.pdf>. [2010, February 21].

Boley, Harold & Sintek, Michael. (1995). Open-World Datalog. <http://www.dfki.uni-kl.de/vega/relfun+/extension/subsection3_3_1.html> [2009, December 24].

Bralich et al. Patent #5,878,385. *Method and apparatus for universal parsing of language*. March 2, 1999.

Brassell & Miller. Patent #6,553,372. *Natural language information retrieval system.* April 22, 2003.

Bratko, Ivan. (2001). *Prolog Programming for Artificial Intelligence.* Harlow, England: Pearson Addison Wesley.

Brill, Eric, et al. (2002). An Analysis of the AskMSR Question-Answering System. *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, July 2002, pages 257-267. ACM

Brody. Patent application #2004-0215612. *Semi-boolean arrangement, method, and system for specifying and selecting data objects to be retrieved from a collection*. October 28, 2004.

Brown et al. Patent #6,665,666. *System, method and program product for answering questions using a search engine*. December 16, 2003.

Brown, James. (1960). Loglan. *Scientific American*. June, 1960. Pages 53-63. Scientific American, Inc., Ney York, New York.

Brown, James. (1975). *Loglan 1*, Loglan: A Logical Language, Gainesville, Florida: The Loglan Institute:

Brown, James. (1975). *Loglan 2*, Loglan: A Logical Language, Gainesville, Florida: The Loglan Institute:

Brown, James. (1978). *Loglan 3*, Loglan: A Logical Language. Xerox Press.

Brown, James. (1978). *Loglan 4*, Loglan: A Logical Language. Xerox Press.

Brown, James. (1978). *Loglan 5*, Loglan: A Logical Language. Xerox Press.

Brown, James. (1999). Loglan 1: A Logical Language. <http://www.loglan.org/Loglan1/forward.html> [2009, December 24].

Burek, Gaston et al. (2005). Hybrid Mappings of Complex Questions over an Integrated Semantic Space. *Proceedings of the 16th International Workshop on Database and Expert Systems Applications, 2005.* IEEE.

Capek, Karel. *Rossum's Universal Robots* (Broadway play), 1920.

Cardie, Claire et al. Examining the role of statistical and linguistic knowledge sources in a general-knowledge question-answering system, *Proceedings of the sixth conference on Applied natural language processing*, 2000, pages 180-187. Morgan Kaufmann Publishers Inc.

Carter, James. (1991). Guaspi, an Artificial Natural Language. <http://www.math.ucla.edu/~jimc/guaspi/acmpaper.html>. [2009, December 24].

Chalupsky, Hans. (2006). PowerLoom Documentation. <http://www.isi.edu/isd/LOOM/PowerLoom/documentation> [2009, December 24].

Chang. Patent application #2006-0123045. *Natural language expression in response to a query*. June 8, 2006.

Chen, Wei et all. (2006). A User-Reputation Model for a User-Interactive Question Answering System. *Proceedings of the Second International Conference on Semantics, Knowledge, and Grid*, 2006. IEEE.

Coffman. Patent application #2003-0014260. *Method and system for determining and maintaining dialog focus in a conversational speech system*. January 16, 2003.

Cooper, William. (1964). Fact Retrieval and Deductive Question-Answering Information Retrieval Systems, *Journal of the ACM*, April 1964, pages 117-137. ACM

Datig. Patent application #2002-0198697. *Universal epistemological machine (a.k.a. android)*. December 26, 2002.

Datig. Patent application #2005-0005266. *Method of and apparatus for realizing synthetic knowledge processes in devices for useful application*. January 6, 2005.

Diederiks & Van De Sluis. Patent application #2001-0056364. *Method of interacting with a consumer electronics system*. December 27, 2001.

Dixon et al. Patent #5,088,048. *A method to speed up problem solvers and provers*. February 11, 1992.

Dusan & Flanagan. Patent application #2002-017805. *System and method for adaptive language understanding by computers*. November 28, 2002.

Einstein, Albert. (1916). Die Gundlage der allgemein Relativitätstheorie (The Basis of General Relativity Theory). *Annalen der Physik (Annals of Physics),* 1916 #7, pages 769-821.

Ejerhed. Patent #7,058,564. *Method of finding answers to questions*. June 6, 2006.

Eldredge et al. Patent #6,697,801. *Methods of hierarchically parsing and indexing text*. February 24, 2004.

Epstein. Patent application #2002-0123891. *Hierarchical language models*. September 5, 2002.

Fain & Fain. Patent application #2002-0169597. *Method and apparatus providing computer understanding and instructions from natural language*. November 14, 2002.

Firman. Patent #5,377,303. *Controlled computer interface*. December 27, 1994.

Ford. Patent application #2003-0144831. *Natural language processor*. July 31, 2003.

Free Software Federation. (2010). Bison 2.4.2. Free Software Foundation. <http://www.gnu.org/software/bison/manual/html_mono/bison.html> [2010, May 1]

Fuchs, Norbert et al. (1999). Attempto Controlled English Language Manual Version 3.0. Geneva, Switzerland: Institut für Informatik der Universität Zürich.

Fuchs, Norbert & Schwertel, Uta. (2002). Reasoning in Attempto Controlled English. *Technical Report*, January, 2002. Institut für Informatik, Universität Zürich.

Furbach, Ulrich, et al., (2008). LogAnswer - A Deduction-Based Question Answering System. <http://www.uni-koblenz.de/~bpelzer/publications/FGHP08_IJCAR08_prel.pdf>. [2009, December 29].

Fujisawa et al. Patent #5,404,506. *Knowledge based information retrieval system*. April 4, 1995.

Fujisawa et al. Patent #6,182,062. *Knowledge based information retrieval system*. January 30, 2001.

Fujisawa et al. Patent #5,555,408. *Knowledge based information retrieval system*. September 10, 1996.

Fung et al. Patent #6,687,689. *System and methods for document retrieval using natural language-based queries*. February 3, 2004.

Furbach et al. (2008). LogAnswer - A Deduction-based Question-answering System. <http://www.uni-koblenz.de/~bpelzer/publications/FGHP08_IJCAR08_prel.pdf>. [2010, February 21].

Gao, Sixia. (2008). Study on College English Teaching Strategy. *Asian Social Science*, volume 4, number 11, pages 93-99. Toronto, Ontario: Canadian Center of Science and Education.

Galton, Antony. (2008). Temporal Logic. *Stanford Encyclopedia of Logic.* Stanford University. <http://plato.stanford.edu/entries/logic-temporal>. [2010, February 21].

Glöckner, Ingo. (2008). Towards Logic-Based Question Answering under Time Constraints. *Proceedings of the International MultiConference of Engineers and Computer Scientists 2008*, volume 1. Hong Kong: IMECS.

Goertzel, Ben. (2005). Potential Computational Linguistics Resources for Lojban. <http://www.goertzel.org/new_research/lojban_AI.pdf> [2009, December 24].

Goertzel, Ben. (2006). Lojban++: An Efficient, Minimally Ambiguous, User-Friendly Natural-Like Language for Human-Computer, Computer-Computer and Human-Human Communication. <http://www.goertzel.org/papers/lojbanplusplus.pdf> [2009, December 24].

Goss et al. Patent #4,667,290. *Compilers using a universal intermediate language*. May 19, 1987.

Gould et al. Patent #5,920,836. *Word recognition system using language context at current cursor position to affect recognition probabilities*. July 6, 1999.

Green, C. & Raphael, Bertram. (1968). The use of theorem-proving techniques in question-answering systems. *Proceedings of the 1968 23rd ACM national conference table of contents*, 1968, pages 169-181. ACM.

Greetha, T. & Subramanian, R. (1990). Representing Natural Language with Prolog, IEEE Software, March 1990, Pages 85-92. IEEE.

Guo et al. Patent application #2003-0144055. *Conversational interface agent*. July 31, 2003.

Hagen & Stefanik. Patent application #2005-0010415. *Artificial intelligence dialogue processor*. January 13, 2005.

Hao, Tianyong et al. (2006). Semantic Pattern for User Interactive-Question Answering. Proceedings of the Second International Conference on Semantics, Knowledge, and Grid, 2006. IEEE.

Harrison et al. Patent application #2003-0069880. *Natural language query processing*. April 10, 2003.

Hartley, Roger. (1986). An Overview of Conceptual Programming. <http://www.cs.nmsu.edu/~rth/publications/overviewCP.pdf> [2009, December 24].

Haszto et al. Patent #6,192,338. *Natural language knowledge servers as network resources*. February 20, 2001.

Hawkinson & Anderson. Patent application #2004-0122661. *Method, system, and computer program product for storing, managing and using knowledge expressible as, and organized in accordance with, a natural language*. June 24, 2004.

Hirtle, David. (2006). Translator: A Translator from Language to Rules. <http://www.ruleml.org/translator> [2009, December 24].

Ho & Tong. Patent #6,498,921. *Method and system to answer a natural-language question*. December 24, 2002.

Hogenhout & Noble. Patent #7,062,428. *Natural language machine interface*. June 13, 2006.

Hsu & Boonjing. Patent application #2002-0059069. *Natural language interface*. May 16, 2002.

Jennings, James, editor. (2006). Loglan.org. <http://www.loglan.org> [2009, December 24].

Joshi, Aravind. (1977). Some extensions of a system for inferencing on partial information. *ACM SIGART Bulletin, Deductive inference: question answering/theorem proving*, June 1977, page 7. ACM.

Jowel & Kessock. Patent #7,103585. *System and method for logical agent engine*. September 5, 2006.

Jung. Patent #6,950,827. *Methods, apparatus, and data structures for providing a uniform representation of various types of information*. September 27, 2005.

Kang, In-Su et al. (2002). Database Semantics Representation for Natural Language Access, *Proceedings of the First International Symposium on Cyber Worlds*, 2002. IEEE.

Kasravi & Varadarajan. Patent #7,085,750. *Method and system for mapping a hypothesis to an analytical structure*. August 1, 2006.

Kautz & Selman. Patent #5,259,067. *Optimization of information bases*. November 2, 1993.

Klipstein. Patent application #2001-0049597. *Method and system for responding to a user based on a textual input*. December 6, 2001.

Knöll, Roman & Mezini, Mira. (2006). Pegasus: first steps toward a naturalistic programming language. *Companion to the 21st ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, 2006, pages 542-559. ACM.

Kobayashi et al. Patent application #2004-0054519. *Apparatus and methods for developing conversational applications*. March 18, 2004.

Kuhn, Roland and Di Mori, Renato. (1995). The Application of Semantic Classification Trees to Natural Language Understanding. IEEE Transactions on Pattern Analysis and Machine Intelligence, May 1995, Pages 449-460. IEEE.

Kupiec. Patent #5,519,608. *Method for extracting from a text corpus answers to questions stated in natural language by using linguistic analysis and hypothesis generation*. May 21, 1996.

Kupiec. Patent #5,696,962. *Method for computerized information retrieval using shallow linguistic analysis*. December 9, 1997.

Laboreo, Daniel. (2005). Lógica y lenguajes (Logic and languages). <http://www.danielclemente.com/apuntes/ales/hl/html/hl.es.xhtml> [2009, December 24].

Lamberti et al. Patent #5,377,103. *Constrained natural language interface for a computer that employs a browse function*. December 27, 1994.

LeChevalier, Robert, editor. (2006). Lojban.org. <http://www.lojban.org/tiki/Lojban> [2009, December 24].

Linker, Sheldon. (1980). A Partial Machine Grammar of Loglan. *The Loglanist* (also published under the name *La Loglantan*), June 1980, pages 21-32. Gainesville, Florida: The Loglan Institute.

Linker, Sheldon. (1981). An Alternative 'MEX' Proposal. *The Loglanist* (*La Loglantan*), February, 1981, pages 16 & 17. Gainesville, Florida: The Loglan Institute.

Linker, Sheldon. (2007). A Knowledge Base and Question Answering System based on Loglan and English. Thesis. Springfield, Illinois: The University of Illinois.

Lita, Luvian & Carbonell, Jaime. (2004). Unsupervised question answering data acquisition from local corpora, *Proceedings of the thirteenth ACM international conference on Information and knowledge management table of contents*, 2004, pages 607-614. ACM.

Ljungberg & Holm. (1996). Speech acts on trial. *Scandinavian Journal of Information Systems*. 1996.

Lockwood, Kate. (2009). Using Analogy to Model Spatial Language Use and Multimodal Language Capture. Northwestern University, Evanston, Illinois. <http://www.qrg.northwestern.edu/papers/Files/QRG_Dist_Files/QRG_2009/Lockwood_dissert_09.pdf> [2010, February 21].

Logical Language Group, Inc. (wiki contributorship). (2007). Lojban.org Home Page. <http://www.lojban.org/tiki/tiki-index.php?page=Home+Page&bl&bl=y> [2007, March 1].

Loveland, D. (1986). Automated theorem proving: mapping logic into AI. *Proceedings of the ACM SIGART international symposium on Methodologies for intelligent systems*, 1986, pages 214-229. ACM.

Machihara et al. Patent #6,233,578. *Method and system for information retrieval*. May 15, 2001.

Marchiori, Massimo. (2004). Towards a People's Web: Metalog, *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence*, 2004. IEEE.

Matheson. Patent #7,024,368. *Man-machine dialogue system, controls dialogue between systems and user using dialogue specification employing augmented transition networks propagating tokens*. April 4, 2006.

Matzner, Tobias & Hitzler, Pascal. (2006). Any-World Access to OWL through Prolog. <http://www.aifb.uni-karlsruhe.de/WBS/phi/resources/publications/prowlog.pdf> [2007, March 14].

McConnell & Barklund. Patent #6,993,475. *Methods, apparatus, and data structures for facilitating a natural language interface to stored information*. January 31, 2006.

Metcalf & Dingus. Patent application #2004-0044515. *Automated natural language inference system*. March 4, 2004.

Miller et al. Patent #6,393,428. *Natural language information retrieval system*. May 21, 2002.

Minker, Jack. (1977). Control structure of a pattern-directed search system. *Deductive inference: question answering/theorem proving*, June 1977, pages 7-14. ACM.

Minker, Jack. (1978). Search strategy and selection function for an inferential relational system. *ACM Transactions on Database Systems*, March 1978, pages 1-31. ACM.

Moldovan, Dan et al. (2003). COGEX: a logic prover for question answering. *Proceedings of the 2003 Conference of the North American Chapter of the*

Association for Computational Linguistics on Human Language Technology, 2003, pages: 87-93. Association for Computational Linguistics

Moser et al. Patent #6,275,789. *Method and apparatus for performing full bidirectional translation between a source language and a linked alternative language*. August 14, 2001.

Nakamura & Tate. Patent #6,941,295. *Data access system*. September 6, 2005.

Namba et al. Patent #5,555,169. *Computer system and method for converting a conversational statement to computer command language*. September 10, 1996.

Ohlman, Herbert. (1961). Pro a Special IR Language. *Design, Implementation and Application of IR-Oriented Languages*, 1961, pages 8-10. ACM.

Phillips, Web & Boroditsky, Lera. (2003). Can quirks of grammar affect the way you think? Grammatical gender and object concepts. *Proceedings from the 25th Annual Meeting of the Cognitive Science Society*, 2003, pages 928-933.

Pinto, David et al. (2002). QuASM: a system for question answering using semi-structured data. *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, 2002, pages 46-55. ACM.

Prothero, Jeff. (1990). Design and Implementation of a Near-optimal Loglan Syntax. <http://www.rickharrison.com/language/plan_b.html> [2009, December 24].

Prothero, Jeff, et al. (1994). Loglan Grammar as of Dec 94. <http://loglan.org/Misc/grammar80.y> [2009, December 24].

Pustejovsky & Ingria. Patent application #2001-0039493. *Answering verbal questions using a natural language system*. November 8, 2001.

Radev, Dragomir et al. (2001). Mining the web for answers to natural language questions. *Proceedings of the tenth international conference on Information and knowledge management*, 2001, pages 143-150. ACM.

Ramakrishnan, Ganesh et al. (2003). Question Answering via Bayesian inference on lexical relations. *Proceedings of the ACL 2003 workshop on Multilingual summarization and question answering,* 2003, pages 1-10. Association for Computational Linguistics.

Ramakrishnan, Ganesh et al. (2004). Is question answering an acquired skill?. *Proceedings of the 13th international conference on World Wide Web table of contents*, 2004, pages 111-120. ACM.

Reiter, Raymond. An approach to deductive question-answering systems. *ACM SIGART Bulletin, Natural Language interfaces*, 1977, pages 41-43. ACM.

Romero. Patent application #2002-0111803. *Method and system for semantic speech recognition*. August 15, 2002.

Ross et al. Patent application #2002-0138266. *Method and apparatus for converting utterance representations into actions in a conversational system*. September 26, 2002.

Ross et al. Patent #6,950,793. *System and method for deriving natural language representation of formal belief structures*. September 27, 2005.

Rosser & Sturges. Patent application #2006-0069546. *Autonomous response engine*. March 30, 2006.

Roussinov, Dmitri & Robles, Jose. (2004). Self-learning web question answering system. *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, 2004, pages 400-401. ACM.

Salanha at al. Patent #6,714,939. *Creation of structured data from plain text*. March 30, 2004.

Scheneburg et al. Patent application #2002-0133347. *Method and apparatus for natural language dialog interface*. September 19, 2002.

Schramm. Patent #4,670,848. *Artificial intelligence system*. June 2, 1987.

Schwartz. Patent #5,812,840. *Database query system*. September 22, 1998.

Schwartz et al. Patent #5,197,005. *Database retrieval system having a natural language interface*. March 23, 1993.

Sekine, Satoshi & Grishman, Ralph. (2003). Hindi-english cross-lingual question-answering system. *ACM Transactions on Asian Language Information Processing*, September 2003, pages 181-192. ACM.

Sheu & Kitazawa. Patent application #2004-0088158. *Structured natural language query and knowledge system*. May 6, 2004.

Small, Sharon et al. (2003). HITIQA: an interactive question answering system a preliminary report. *Proceedings of the ACL 2003 workshop on Multilingual summarization and question answering*, 2003, pages 46-53. Association for Computational Linguistics.

Speer, Rob & Havasi, Catherine. (2004). Meeting the Computer Halfway: Language Processing in the Artificial Language Lojban, *Proceedings of MIT Student*

Oxygen Conference, MIT. <http://sow.lcs.mit.edu/2004/proceedings/Speer.pdf> [2007, March 5].

Spiegler & Gelbard. Patent application #2002-0087567. *Unified binary model and methodology for knowledge representation and for data and information mining*. July 4, 2002.

Srihari, Rohini & Li, Wei. (2000). A question answering system supported by information extraction. *Proceedings of the sixth conference on Applied natural language processing*, 2000, pages 166-172. Morgan Kaufmann Publishers Inc.

Stier & Haughton. Patent #6,499,024. *Method and system for development of a knowledge base system*. December 24, 2002.

Strong. Patent #6,311,157. *Assigning meanings to utterances in a speech recognition system*. October 30, 2001.

Strong. Patent #6,704,710. *Assigning meanings to utterances in a speech recognition system*. March 9, 2004.

Suda. Patent #5,282,265. *Knowledge information processing system*. January 25, 1994.

Suda & Jeyachandran. Patent application #2003-0144977. *Information processing system which understands information and acts accordingly and method therefor*. July 31, 2003.

Sukehiro et al. Patent application #2004-0205671. *Natural language processing system*. October 14, 2004.

Tsourikov. Patent application #2002-0116176. *Semantic answering system and method.* August 22, 2002.

Tunstall-Pedoe. Patent #7,013,308. *Knowledge storage and retrieval system*. March 14, 2006.

Yano et al. Patent #6,466,899. *Natural language dialogue apparatus and method*. October 15, 2002.

Waltz, David. (1978). An English language question answering system for a large relational database. *Communications of the ACM*, July 1978, pages 526-539. ACM.

Wang. Patent application #2006-0190268. *Distributed language processing system and method of outputting intermediary signal thereof*. August 24, 2006.

Wang et al. Patent application #2004-0225499. *Multi-platform capable inference engine and universal grammar language adapter for intelligent voice application execution*. November 11, 2004.

Wang, Chun-Chia et al. (2006). An Application of Question Answering System for Collaborative Learning. *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems Workshops,* 2006. IEEE.

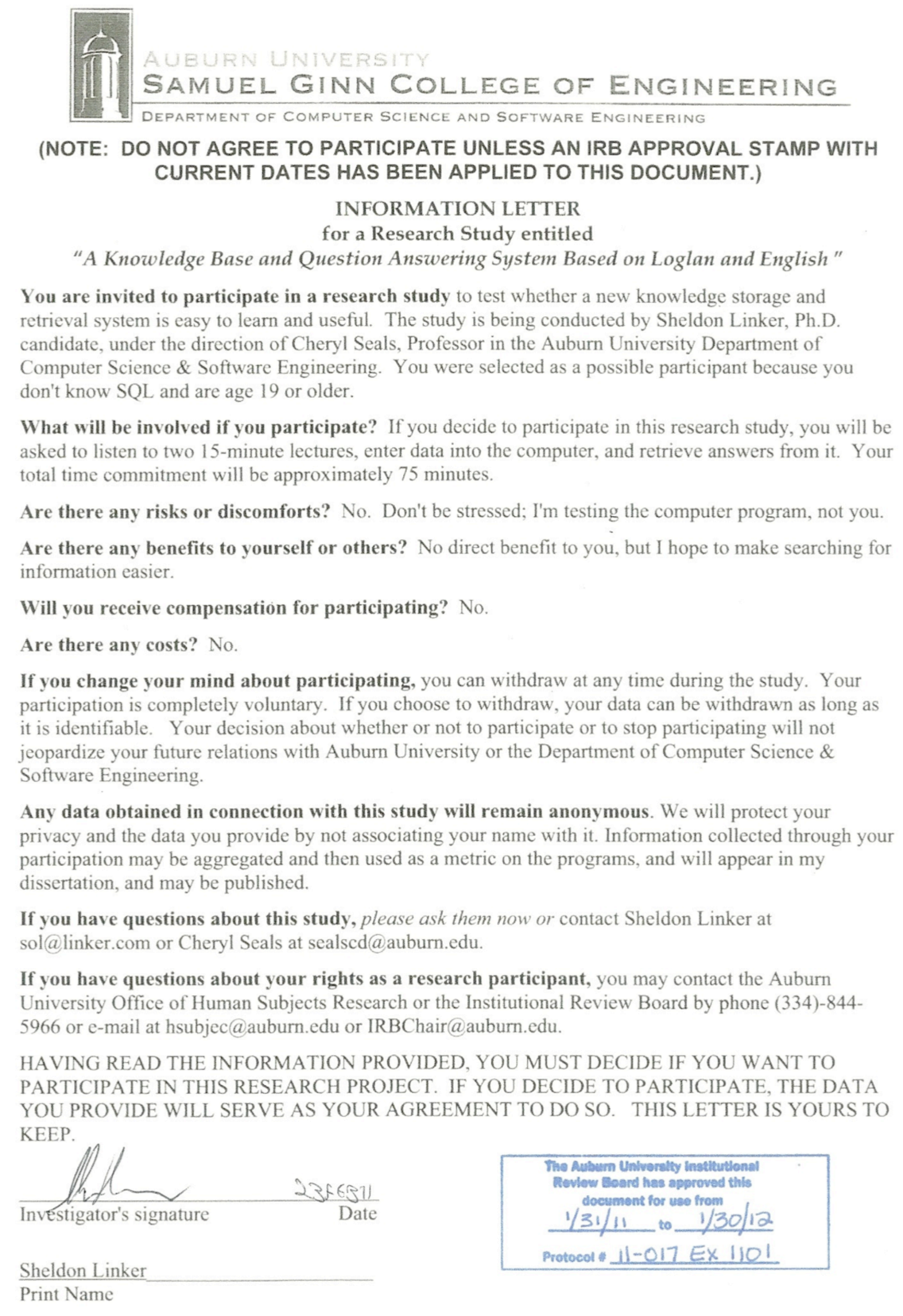
Weber. Patent #6,499,013. *Interactive user interface using speech recognition and natural language processing*. December 24, 2002.

Williams & Hill. Patent application #2005-0105712. *Machine learning*. May 19, 2005.

Wyss et al. Patent application #2002-0026435. *Knowledge-base system and method*. February 28, 2002.

Zhang & Yang. Patent application #2002-0077815. *Information search method based on dialog and dialog machine*. June 20, 2002.

Appendix I — Institutional Review Board Approval



AUBURN UNIVERSITY
SAMUEL GINN COLLEGE OF ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

(NOTE: DO NOT AGREE TO PARTICIPATE UNLESS AN IRB APPROVAL STAMP WITH CURRENT DATES HAS BEEN APPLIED TO THIS DOCUMENT.)

INFORMATION LETTER
for a Research Study entitled
"A Knowledge Base and Question Answering System Based on Loglan and English"

You are invited to participate in a research study to test whether a new knowledge storage and retrieval system is easy to learn and useful. The study is being conducted by Sheldon Linker, Ph.D. candidate, under the direction of Cheryl Seals, Professor in the Auburn University Department of Computer Science & Software Engineering. You were selected as a possible participant because you don't know SQL and are age 19 or older.

What will be involved if you participate? If you decide to participate in this research study, you will be asked to listen to two 15-minute lectures, enter data into the computer, and retrieve answers from it. Your total time commitment will be approximately 75 minutes.

Are there any risks or discomforts? No. Don't be stressed; I'm testing the computer program, not you.

Are there any benefits to yourself or others? No direct benefit to you, but I hope to make searching for information easier.

Will you receive compensation for participating? No.

Are there any costs? No.

If you change your mind about participating, you can withdraw at any time during the study. Your participation is completely voluntary. If you choose to withdraw, your data can be withdrawn as long as it is identifiable. Your decision about whether or not to participate or to stop participating will not jeopardize your future relations with Auburn University or the Department of Computer Science & Software Engineering.

Any data obtained in connection with this study will remain anonymous. We will protect your privacy and the data you provide by not associating your name with it. Information collected through your participation may be aggregated and then used as a metric on the programs, and will appear in my dissertation, and may be published.

If you have questions about this study, *please ask them now or* contact Sheldon Linker at sol@linker.com or Cheryl Seals at sealscd@auburn.edu.

If you have questions about your rights as a research participant, you may contact the Auburn University Office of Human Subjects Research or the Institutional Review Board by phone (334)-844-5966 or e-mail at hsubjec@auburn.edu or IRBChair@auburn.edu.

HAVING READ THE INFORMATION PROVIDED, YOU MUST DECIDE IF YOU WANT TO PARTICIPATE IN THIS RESEARCH PROJECT. IF YOU DECIDE TO PARTICIPATE, THE DATA YOU PROVIDE WILL SERVE AS YOUR AGREEMENT TO DO SO. THIS LETTER IS YOURS TO KEEP.

[illegible] 23FEB11
Investigator's signature Date

Sheldon Linker
Print Name

The Auburn University Institutional Review Board has approved this document for use from 1/31/11 to 1/30/12
Protocol # 11-017 EX 1101

Appendix II — A Loglan Primer

Loglan is a predicate-based language. In Loglan, the difference between a verb, a noun, and an adjective or adverb is where the predication is placed in the language structure. For instance,

Table 5 — Loglan Translations

Loglan	Literal translation	Idiomatic translation
Cmalo	Small	Be small.
Tu cmalo	You small	You have the capacity to be small; You are smaller than [some x]
Tu na cmalo	You now small	You are small
Bilti cmalo ckela	Pretty small school	Prettily small school: A school that is small, such that the Smallness is a pretty form of smallness
Bilti ge cmalo ckela	Pretty type of small school	Pretty small school: A small school, which is pretty
Bilti e cmalo ckela	Pretty and small school	Something which is pretty, and a small school
Bilti e cmalo gu ckela	Pretty and small type of school	A school, which is pretty and small

Loglan	Literal translation	Idiomatic translation
Le ckela ga cmalo	The school is small	The school is small

Of course, one need not use the Loglan vocabulary to carry out Loglan's functions. Loglan's vocabulary was designed to be unambiguously parseable in continuous verbal stream. However, there is no reason that an English vocabulary could not be used in a printed communication. For instance,

English (ambiguous): Pretty little girls' school

Loglan (unambiguous): Bilti ge cmalo nirli ckela

Hybrid (unambiguous): Pretty type of school for girls who are small

Loglan was designed by Dr. Brown to be unambiguous. By the time Michael Urban and this author joined the project, Dr. Brown had a grammar, written in a BNF-like notation. The grammar was then translated into YACC input format. With some hints from Michael Urban and Dr. Brown, this author came up with a grammar which met the requirements of Loglan, and which was provably unambiguous. Two versions of this were published in The Loglanist (a.k.a. La Loglantan, Linker 1980 & 1981). Later, Jeff Prothero (1990) and others (Prothero et al., 1994) made continuing improvements to the Loglan grammar, and wrote a series of parsers for Loglan. Unfortunately, these parsers never went past graphing the sentences, and stating whether a given "utterance" of Loglan was legal or not.

Appendix III — Transformations of Loglan into Functional Form

The Starting Point: The Loglan Grammar

Some say that the Loglan grammar was superseded by Lojban, but for the present purposes, the simpler the better. The text shown in courier is as it is delivered by HTTP and published by The Loglan Institute, Inc (Prothero et al., 1994). This code is used and reproduced here with specific permission. Various "trials" are referenced. Trial 1 was written by myself, with the extensive testing and modification help of Mike Urban and James Brown. Trial 2 was also this author's work. The additional 78 trials represent additions and changes to the work, done by others.

C-code and some other items have been stripped for clarity. Items shown in this typeface are this author's comments.

```
/* GRAMMAR 80           Loglan grammar as of Dec 94
Copyright ©1982, 1984, 1986-1995 by The Loglan Institute, Inc.
Created in Jan-Feb 82 from JSP's Aug 81 grammar by SWL & JCB,
Modified in Mar 82, Dec 83, Mar 84, and Dec 86 - Jun 87 by
JCB. and in 1987-90 by RAM.
```

Translation: Sheldon Linker and James Cooke Brown worked together on a grammar, which was converted to a full parser by Jeff Prothero, and later improved by Robert McIvor. By 1981, this author had left UCLA, and had begun to work for TRW. TRW was kind enough to allow this author to continue the work on this project at their facility, and cleared the work for publication.

```
Trial 80 was created in Dec 94, include luo and lou, mea, nuo,
fuo, and juo. The preparser was adjusted to allow for the
other Keugru-mediated changes. Still in abeyance is whether
```

```
duo, dui should go to the bi lexeme, whether fio, foi and suo
should be included in advance of approval.
```

Other comments on various trials ranging from 1987-1993 have been stripped out, as well as YACC elements that have no effect.

```
*/
%token A1 /* a1 zea used for A when connecting predicates */
%token A2 /* a2 used for A when connecting linkargs or
             modifiers  */
%token A3 /* a3 used for A when connecting argmods */
%token A4 /* ha a e o u also CPDs anoi, apa, noanoi, etc.
             Used for all other A */
%token ACI /* recognized by CPD-lexer */
%token AGE /* recognized by CPD-lexer. */
%token BI /* bi bia bie cie cio */
%token BAD
%token CA /* ca ce co cu also CPDs noca, canoi, nocanoi, etc.
             */
%token CI /* ci */
%token CUI /* cui */
%token DA /* ba be bo bu da de di do du mi tu mu ti ta tao tio
             tua mia mie mii mio miu mua mue mui muo muu toa
             toi tue tui tuo tuu suo */
%token DIE /* die fie kae nue rie */
%token DIO /* beu cau dio foa kao jui neu pou goa sau veu
              zua zue zui zuo zuu lae lue */
%token DJAN /* all C-final words found by lexer */

%token FI /* fi */
%token GA2 /* ga */
%token GE /* ge */
%token GEU /* geu */
%token GI /* gi goi */
%token GO /* go */
%token GU /* gu */
%token GUE /* gue */
%token GUI /* gui */
%token GUO /* guo */
%token GUU1 /* guu */
%token GUU2 /* guu2 */
%token HOI /* hoi */
%token I /* i also CPDs ifa, inusoa, etc. */
%token ICA /* all eeskeks, recognized by lexer */
%token ICI /* ici & icaci-type words, all recognized by CPD-
              lexer */
%token IE /* ie */
%token IGE /* ige & icage-type words, all recognized by CPD-
              lexer */
```

```
%token JE /* je */
%token JI /* ji ja jie jae pe */
%token JIO /* jio jao */
%token JO /* jo also CPDs rajo, tojo, etc. */
%token JUE /* jue */
%token KA1 /* ka1 used for KA when connecting linkargs */
%token KA2 /* ka2 used for KA when connecting predicates */
%token KA3 /* ka ke ko ku also CPDs kanoi, nuku, nukunoi,
             kouki, nukouki,etc. For the rest */
%token KOU /* kou moi rau soa these are pa words separated out
             for the lexer */
%token KI /* ki also the CPD  kinoi */
%token KIE /* kie */
%token KIU /* kiu */
%token LAO /* lao */
%token LAU /* lau lou */
%token LE /* le la lo lea leu loe lee laa */
%token LEPO /* recognized by CPD-lexer*/
%token LI /* li */
%token LIE /* lie */
%token LIO /* lio */
%token LIU /* liu lii niu */
%token LU /* lu */
%token LUA /* lua luo */
%token SOI /* soi */
%token MA /* ma si to recognize initial vowels in acronyms, NI
            otherwise */
%token ME /* me mea */
%token NI /* ho ni ne to te fo fe vo ve pi re ro ru sa se so
            su mo kua gie giu hie hiu kue nea nio pea pio suu
            sua tia zoa zoi also CPDs neni, nenisei, iesu,
            ietoni, etc. */
%token NO1 /* no1 used for NO + mod shown by PA */
%token NO2 /* no2 used for NO + markpred shown by PO, ZO or
             PA1 */
%token NO3 /* no3 used for NO + argument */
%token NO4 /* no For all other no's */
%token NOI /* noi */
%token NU /* nu fu ju nuo fuo juo also CPDs nufu, nufuju, nuto
            (=nu), nute (=fu), nufo (=ju), nufe, nuso, etc.
            */
%token PA1 /* pa1 used for PA and GA when inflecting a
             predicate */
%token PA2 /* va vi vu  pa na fa gia gua pia pua nia nua fia
             fua via vii viu ciu coi dii duo gau kii lia lui
             mou hea peu rui sea tie fio foi also CPDs pana,
             pazi, pacenoina, etc. For the rest of PAs*/
%token PAUSE /* , # */
%token PO /* po pu */
%token PREDA /* he dua dui bua bui all preda-forms words; also
                all pred-wds found by lexer, CPDs like rari,
                  nenira, sutori, etc.; also acronyms like ebai,
                  baicai, ebaicai, ebaiocai, haitosaiofo, etc.,
                  */
%token RA /* ra ri */
%token HUE /* hue */
%token SUE /* sue sao */
%token TAI /* gao forms like ama bai cai tai tei are
                recognized by the lexer; CPDs like baicai,
                ebaicai, ebaiocai, haitosaiofo, etc., belong to
                PREDA */
%token UI /* ua ue ui uo uu oa oe oi ou ia ii io iu ea ei eo
               eu ae ai ao au bea biu buo cea cia coa dau dou
               fae fao feu gea kuo kuu rea nao nie pae piu saa
               sui taa toe voi zou loi loa sia sii siu cao ceu
               also CPDs nahu, vihu, kouhu, duohu, nusoahu, etc.
               */
%token ZE1 /* ze also used by the preparser to recognize
                acronymic PREDA's  */
%token ZE2 /* ze2 used for ZE + argsign */
%token ZI /* zi za zu used by the preparser to recognize pazi-
               CPDs and acronymic PREDA's */
%token ZO /* zo used by the preparser to recognize acronymic
               PREDA's; otherwise zo would be a member of PO */
%start utterance
%%

err               : error
                  ;
guo               : GUO
                  | GU
                  | err
                  ;
gui               : GUI
                  | GU
                  | err
                  ;
gue               : GUE
                  | GU
                  | err
                  ;
guu               : GUU1
                  | GU
                  | err
                  ;
lua               : LUA
                  | err
                  ;
geu               : GEU
                  | err
                  ;
gap               : PAUSE
```

```
                | GU
                | err
                ;
juelink         : JUE argument
                ;
links1          : juelink
                | juelink links1 gue
                ;
links           : links1
                | links A2 links1
                | KA1 links KI links1
                ;
jelink          : JE argument
                ;
linkargs1       : jelink gue
                | jelink links gue
                ;
linkargs        : linkargs1
                | linkargs A2 linkargs1
                | KA1 linkargs KI linkargs1
                ;
predunit1       : PREDA
                | SUE
                | NU PREDA
                | GE despredE geu
                | NU GE despredE geu
                | ME argument gap
                ;
predunit3       : predunit2
                | predunit2 linkargs
                ;
predunit2       : predunit1
                | NO4 predunit2
                ;
predunit        : predunit4
                | predunit ZE1 predunit4
                ;
predunit4       : predunit3
                | PO predunit3
                ;
despredA        : predunit
                | kekpredunit
                | predunit CI despredA
                ;
kekpredunit     : NO4 kekpredunit
                | KA2 predicate KI predicate
                ;
despredB        : despredA
                | CUI despredC CA despredB
                ;
despredC        : despredB
```

```
                | despredC despredB
                ;
despredD        : despredB
                | despredD CA despredB
                ;
despredE        : despredD
                | despredE despredD
                ;
descpred        : despredE
                | despredE GO descpred
                ;
senpred1        : predunit
                | predunit CI senpred1
                ;
senpred2        : senpred1
                | CUI despredC CA despredB
                ;
senpred3        : senpred2
                | senpred3 CA despredB
                ;
senpred4        : senpred3
                | senpred4 despredD
                ;
sentpred        : senpred4
                | senpred4 GO barepred
                ;
mod1            : PA2 gap
                | PA2 argument gap
                ;
mod             : mod1
                | NO1 mod1
                ;
kekmod          : KA3 modifier KI mod
                | NO3 kekmod
                ;
modifier        : mod
                | kekmod
                | modifier A2 mod
                ;
name            : DJAN
                | name CI DJAN
                | name predunit
                | name DJAN
                ;
mex             : NI
                | mex NI
                ;
descriptn       : LE descpred
                | LE mex descpred
                | LE arg1 descpred
                | LE mex arg1a
```

```
                        ;
voc                     : HOI descpred gap
                        | HOI name gap
                        | HOI DA
                        | HOI gap
                        | name gap
                        ;
arg1                    : LIO mex gap
                        | LIO descpred gap
                        | LIO term gap
                        | LE name gap
                        | descriptn gap
                        | descriptn name gap
                        | LI utterance LU
                        | LI LU
                        | LIU
                        | LIE
                        | LAO
                        | LEPO uttAx guo
                        | LEPO sentence guo
                        ;
arg1a                   : DA
                        | TAI
                        | arg1
                        | voc
                        ;
argmod1                 : JI argument
                        | JI modifier
                        | JI predicate gui
                        | JIO uttAx gui
                        | JIO sentence gui
                        ;
argmod                  : argmod1
                        | argmod A3 argmod1 gap
                        ;
arg2                    : arg1a
                        | arg2 argmod gap
                        ;
arg3                    : arg2
                        | mex arg2
                        ;
indef1                  : mex descpred
                        ;
indef2                  : indef1 gap
                        | indef2 argmod gap
                        ;
indefinite              : indef2
                        ;
arg4                    : arg3
                        | indefinite
                        | arg4 ZE2 arg3
```

```
                | arg4 ZE2 indefinite
                ;
arg5            : arg4
                | KA3 argument KI argx
                ;
arg6            : arg5
                | DIO arg6
                | IE arg6
                ;
argx            : arg6
                | NO3 argx
                ;
arg7            : argx
                | argx ACI arg7
                ;
arg8            : arg7
                | arg8 A4 arg7
                ;
argument        : arg8
                | arg8 AGE arg8
                | argument GUU2 argmod gap
                | LAU wordset
                ;
term            : argument
                | modifier
                ;
terms           : term
                | terms term
                ;
wordset         : words lua
                | lua
                ;
words           : word
                | words word
                ;
word            : arg1a gap
                | NI gap
                | UI gap
                | PA2 gap
                | DIO gap
                | predunit1 gap
                | indef2
                ;
termset1        : terms guu
                ;
termset2        : termset1
                | termset2 A4 termset1
                | KA3 termset2 KI termset1
                ;
termset         : termset2
                | guu
                ;
barepred        : sentpred termset
                | kekpred termset
                ;
markpred        : PA1 barepred
                | PO gap sentence gap
                | NO4 markpred
                ;
backpred1       : barepred
                | markpred
                | NO2 backpred1
                ;
backpred        : backpred1
                | backpred1 ACI backpred
                ;
bareekpred      : barefront A1 backpred
                ;
barefront       : barepred
                | bareekpred termset
                ;
markekpred      : markfront A1 backpred
                ;
markfront       : markpred
                | markekpred termset
                ;
predicate2      : barefront
                | markfront
                | NO2 predicate2
                ;
predicate1      : predicate2
                | predicate2 AGE predicate1
                ;
identpred       : BI termset
                | NO4 identpred
                ;
kekpred         : kekpredunit
                | kekpred despredD
                ;
predicate       : predicate1
                | identpred
                ;
gasent          : PA1 barepred GA2 terms
                | NO2 gasent
                ;
statement       : gasent
                | terms gasent
                | terms predicate
                ;
keksent         : KA3 sentence KI uttA1
                | KA3 gap sentence KI uttA1
                | KA3 headterms sentence KI uttA1
                | NO3 keksent
                ;
sen1            : predicate
                | statement
                | keksent
                ;
sentence        : sen1
                | sentence ICA sen1
                ;
headterms       : terms GI
                | headterms terms GI
                ;
uttA            : A4
                | IE
                | mex
                ;
uttAx           : headterms sen1
                ;
uttA1           : uttA
                | uttAx
                | NO4
                | terms
                | links
                | linkargs
                | sen1
                | argmod
                | terms keksent
                ;
freemod         : UI
                | SOI
                | DIE
                | NO4 DIE
                | KIE utterance KIU
                | HUE statement gap
                | HUE terms gap
                | JO
                ;
neghead         : NO4 gap
                ;
uttC            : uttA1
                | neghead uttC
                ;
uttD            : uttC
                | uttC ICI uttD
                ;
uttE            : uttD
                | uttE ICA uttD
                ;
uttF            : uttE
                | uttE I uttE
                ;
```

```
utterance        : I
                 | freemod
                 | uttF
                 | I uttF
                 | ICA uttF
                 | uttE IGE utterance
                 ;
%%
```

As a result of the error construct, anywhere Gu is required, but does not appear, that disappearance is forgiven, and Gu assumed. This is a context rule similar to the usage of "else" in languages in the C and Cobol families. If statements have an Else clause, but if the clause is missing, its absence does not affect anything. Similarly, in JCB-English, some constructs need to be explicitly ended in some context. In each such case, "**end**" or "/" is required.

Changes Already Made

The following changes are detailed in this author's previous paper.

• Change From YACC Input to BNF-Like Notation

• Simplification of the Representative Form

• An Increase in Formality

• Taking Advantage of a Written-Only Form

• Limiting the Choice of Word Order

• Converting to the Functional View

• The first-cut functional form

Appendix IV — Previous Redesign of the Language

The interactive language is designed based on English, propositional grammar, and Loglan. The language extends to higher-order logic, including the ability to extend to modifiers of every sort allowed by every language. Below is the derivation.

Table 6 — The Symbols of Logic

Symbol	**Meaning**	**Functional**
∀	For all	for all *variable… expression*
∃	There exists	there exists *variable…* such that *expression* there exists *quantifier numeric expression variable* such that *expression*
∄	There does not exist	there does not exist *variable* such that *expression*
∈, ∉, ∩, ∪, ⊂, ⊃, ⊆, ⊇, ⊄, ⊄, ⊈, ⊉, {⋯}, [⋯]	Set and list notation	These are predicates for now, but should be added in this project as primitives later.

Symbol	**Meaning**	**Functional**
∧	And	both *logic expression* and *logic expression*
∨	Or	either *logic expression* or *logic expression* neither *logic expression* nor *logic expression*
<, ≤, =, ≥, >, ≠	Equality notation	These are predicates for now, but should be added in this project as primitives later.
(…)	Grouping	(not needed)
P*x*	Predication	(see below)
F*x*	Function	(see below)
+, -, ·, ÷	Arithmetic	the sum of *numeric expression* and *numeric expression* the difference between *numeric expression* and *numeric expression* the product of *numeric expression* and *numeric expression* the quotient of *numeric expression* and *numeric expression*
⇐	Is implied by	the statement *expression* is implied by *expression*

Symbol	Meaning	Functional
$\Rightarrow$	Implies	if *expression* then *expression*
$\Leftrightarrow$	If and only if	if and only if *expression* then *expression*
$\oplus$	Exclusive or	exclusively *logic expression* or *logic expression*
$\neg$	Not	it is not the case that *logic expression* is true
true, false	Logicals	true false
$\Diamond$	Uncertainty	perhaps, as a guess, as a belief
$\Box$	Certainty	certainly

Table 7 — Items Inherited, or not, from Loglan

Class & Meaning	Functional
Preda Dio Ga Ge Geu Gu: Predicates and adjectives	**lexeme/* **lexeme argument ... /* "*" (handled lexically) and "/" can be omitted if doing so would not contextually cause a problem. One predicate can modify another: adjective *predication* affects *predication* Thus, to say "pretty little girls' school", having a primary meaning of a school for girls who are little, and that such school is pretty would be said (or written) as "adjective *pretty/ affects adjective adjective *little/ affects *girl/ affects *school"
Ze: Jointly	Let's say that one has a red-and-blue ball. It is not "a red ball and a blue ball", not is it "a red ball in a blue way", nor "a blue ball in a red way". It is "a ball in a way that is jointly red and blue". One could also say that it is "a ball and it is jointly red or blue. For this, "jointly", which is tightly binding is introduced. jointly *predication* and *predication*

Class & Meaning	Functional
A Aci Age Bi Ca Cui Go Guu I Ica Ici Ige Ka Ki Lau Lua No: Operators	(Handled in the first-order logic section)
I: Period	A separation between two executables (utterances or queries), even though they are delivered together *executable* execute *executable*
Ha: Question	As a variable: what Alternate form: which As a predicate: blank

Class & Meaning	Functional
Nu: Rearranger	In Loglan, the Nu-class words rearrange arguments. We do not need that here, because there is a simpler way to do this. First, any place an argument can appear, the word "anything" can appear in its stead, as an unconstrained value. Second, when a predicate is used as an adjective or adjectival phrase, we need to know how. For this, the word "emphasis" is chosen. For instance, Go(x,y,z) means the going from x to y over path z. Thus, "adjective *go/ affects *drive" would mean to drive in a going sort of way, as would "adjective *go emphasis/ affects *drive". However, "adjective *go anything emphasis/ affects *drive" would mean to drive in a coming sort of way. anything emphasis

Class & Meaning	**Functional**
Da: It Bi: Is	In English, "it", "he", "she", "him", "her", "they", and "them" all have meanings that are ambiguous. You have to guess the same meaning that the speaker is using. In Loglan, the meanings for "Da" through "Du", and their subscripted forms are unambiguous, but follow fairly complicated rules and are hard to follow in conversation. Here, for simplicity and formality, as well as ease of understanding, a special predication (verb) will be used: be *argument argument* Rather than implying that "it" is "a school" through context rules, one will say "x is a school" explicitly.
Ba: x	This class of word, in Loglan, represent true variables. They appear in ∀ and ∃ situations. Thus, in this language, the scope of each variable is only with its "for all" or "there exists" phrase. These variables will be represented by letters, other than "a" or "i".

Class & Meaning	**Functional**
Mi: Me, I	me i Either of these terms, used interchangeably, represents the speaker. When used, these tokens will be replaced by the known name of the user. On output, the name of the particular knowledge-base computer system will be replaced by "me" or "i". For instance, "I like you", translated as "like i you" might be stored as "like "Sheldon Linker" "hal.linker.com"". If this text were read out of the system, it might be translated back as "*like you me/". If the data were read back to a different user, or on a different system, the "you" and "me" substitutions would not occur.
Tu: You	you Operates similarly to "me" and "i". See above.
Ci: -sub-	*variable name* sub *number* Since 24 variables may not be enough, subscripts are provided. For instance, "x sub 2".

Class & Meaning	**Functional**
Miu: We	In Loglan, there are a number of words for "we" and other compounds. In order to keep this language more English-like, and at the same time more like logic, and easier to follow, such compounds must be defined using "let". For instance, "let W be you and me", or "let U be 'ted' and i".
Li Lie Liu Lu Soi Sue Tai: Quoted items	In Loglan, each of these syntactically marks a type of sound as a quoted string. In this language, quoted strings will always act as arguments. The meaning of the argument is "the literal text…". '*literal text*' To allow a quotation mark inside a string, a pair of quotes will do. Thus, the literal string "don't" would be represented as "'don''t'". That may be a bit hard to read, so here is an easier version: The string consisting of a single-quote character, the letter D, the letter O, the letter N, a single-quote character, another single quote character, the letter T, and a final single-quote character.

Class & Meaning	Functional
Djan Lao: Names	Names representing a person, place, thing, or named concept always act as arguments, and are represented as double-quoted strings. Because these name strings are marked names, and not literal strings, only letters and numbers should be considered, with internal white space being considered as a single space character. Thus, ""James Cooke Brown"" would have the same meaning (and storage) as "" james cooke brown "". Other examples of names might be ""Shangri-la"", ""Fido"", and perhaps even ""The theory of Relativity"". "*name*"
Die Ui: !	Attitudinal indicators are not needed, as each indication can be just as easily expressed as a statement.
Hoi Hue: Hey	Vocative markers, used in Loglan to gain or redirect attention, are not needed, as we are engaged in point-to-point communication.
Jo Kie Kiu: metaphor	Metaphors and parenthetic additions are better stated as separate sentences, and so are not borrowed from English or Loglan.
Ie Gue Gui Je Ji Jio Jue Le: The,	In English and Loglan, it is possible to say "the house". However, for the purposes here, "the house" is still too ambiguous, as it requires that the listener to understand which house is <u>meant</u> by the speaker, using some form of plausibility context. Thus, a direct "the" is inappropriate. There

Class & Meaning	Functional
connectives, descriptives	are two ways around this. First, one could say that the Whitehouse is a white house with qualification: "adjective *white/ affects *house "Whitehouse"". Second, one could declare that the Whitehouse is white and a house in two separate clauses: "both *white "Whitehouse" and *house "Whitehouse"". Third, one could assign a variable: "let W be "Whitehouse" execute both *white W and *house W". A form had been considered which would be something on the order, "the house identified by…", but that is (a) redundant with the above, and (b) a problem for execution order. Thus, "the" and the various "which is" and "known as" type words and clauses are rejected, and the slightly wordier forms just mentioned will suffice. However, a general form of "of" is provided: the item *argument* with property *predication* In this form, which is really a short-hand, a property can be linked to a thing. This is best shown by example: If we take a predicate Own to be Own(owner,owned), then "my house" could be "the item 1 *house/ with property *own me". "emphasis" will be used to point out other meanings. For instance, Go is Go(traveler,whence,wither,path). Thus, "the restaurant I am going to" is "the item 1 *restaurant with property *go i anything emphasis". Internally, anything of this form

Class & Meaning	Functional
	will split the predication into two components, and introduce an unnamed variable. as shown above. Yet a further short-hand is provided: my *argument* This is equivalent to "the item *argument* with property *own me". Of course, we also need: your *argument*

Class & Meaning	Functional
Ni: #, all, some	One way to turn a predication into an argument is to put number in front of it. A literal number or numeric expression will quantify the item. For instance, "1 *apple" would be an apple. "2 *apple" would be two apples. "0.5 *apple" would be half an apple. To get to "the apple", One would need to first identify a particular item, using means described above. When tempted to use "the", note that the item must have already been fully qualified, and thus "1" will do. *numeric expression predicate expression* Loglan's Ni class also has other numeric forms. Special words mean "all", "most", and "some". Thus, these words can be used to quantify a predication into an argument. all of *predicate expression* most of *predicate expression* some of *predicate expression* little of *predicate expression* This form can also be used in questioning. how many *predicate expression* Note that "most of *apple" means "most of an apple", and not "most apples".

Class & Meaning	**Functional**
Lo: class	Items can be taken as a class. For instance, "the class *apple" means "the class consisting of all apples", or just "apples" in some uses. the class *predicate* Quantifiers can be applied with classes. So, "half of all apples" would be "0.5 class of *apple". "some apples" (meaning "a certain percentage of all apples", rather than "a certain number of apples") would be "some of the class *apple". *numeric expression* **the class** *predicate expression* most of the class *predicate expression* some of the class *predicate expression* little of the class *predicate expression* how many the class *predicate expression*
Guo Po: event	One may speak of an event. The loss of attitudinal indicators requires it. For instance, rather than saying "May The Force be with you", without the indicators, this would be "I desire the event: The Force is with you", or "*desire i the event *with "The Force" you". Likewise, properties can be described, such as happiness. the event *predication* the property *predicate expression*

Class & Meaning	Functional
Lio: number	Using this phrasing, a number can be turned into an argument. the number *numeric expression* For instance, "the number 7"
Pa: tenses	Tenses in Loglan and in this language inherit from every language in the world, including Hopi. Each tense modifies everything to its right, and tenses can be stacked. Unprivileged users get a default tense which tags an event as their opinion. Time tenses are listed below, to be followed by other tenses. at *time reference predication* on [or after] *time reference predication* after *time predication* beginning [after] *time predication* [on or] before *time predication* ending [before] *time predication* potentially *predication* during *time predication* This last tense is the "-ing" tense.
time	Time references can be a single time, or a period *time*

Class & Meaning	Functional
	time through *time* *date* *date time* *time* now today this *period* last *period* next *period* *offset…* from *time* *offset…* before *time* tomorrow yesterday *named date* Dates can be in the form *yyyy*, *yyyy-mm*, or *yyyy-mm-dd*. They can be followed be ad, ce, bc, or bce. Times can be in the form *hh*, *hh*:*mm*, or *hh*:*mm*:*ss*. All times are assumed to be GMT. Period names can be second, minute, hours, day, week, month, year. Offsets can be:

Class & Meaning	Functional
	numeric expression period **a** *period* Periods may be in the pleural. Named dates may include sunday-saturday and january-december.
Tenses	Other tenses include: certainly *predication* on good authority *predication* likely *predication* as a belief *predication* as a guess *predication* according to *argument predication* located at [a distance of [up to] *distance* from] *argument predication* located at a distance of at least *distance* from *argument predication* located at a range of *distance* to *distance* from *argument predication* near *argument predication* far from *argument predication*

Class & Meaning	**Functional**
Distance	a meter *numeric expression* meters
tenses?	Questions can be asked as tenses. Some are specific, and the last is general. when where tense

The Semantics of Predicates

Each predicate is transparent. It does not matter to the language what the predicate means to you. However, we need to understand what it means. An example would be Go. "Go(a,b,c,d)" we will take to mean "A goes to B from C over path D". If we take "Come(a,b,c,d)" to mean "A comes from B to C over path D", then we can define "$\forall$a,b,c,d, (Go(a,b,c,d) $\Leftrightarrow$ Come(a,c,b,d))". Missing arguments are assumed unspecified, so that "$\forall$x, (P(x) $\Leftrightarrow$ $\exists$y, P(x,y))".

Commands

A number of commands need to be present to control the system. These are separate and distinct from statements and queries:

Table 8 — Commands

Command	**Syntax**
Tell the system to trust a given user's statements from now on, either completely or to a given extent	trust *user* [to level *number*]
Tell the system to stop trusting a user's future statements.	do not trust *user*
Add an authorized (untrusted) user.	add user *user* [password *password*] name of user *user* is "name"
Remove further access for a user, without affecting knowledge gained from that user.	drop user *user*
Forget a particular statement.	forget *predication*
Log off	bye
Log on	i am *user* [with password *password*]
Dump the knowledge base as a series of statements, numbered.	list facts [i control]
Drop a particular knowledge item from the knowledge base.	forget fact *number*

Command	**Syntax**
Drop all information from a particular untrusted user (used as a maintenance item, or for malevolent users)	forget what *user* said
Retroactively promote a user's statements to trusted status	trust what *user* said
Speak in unprivileged state (the default, even for the privileged)	commoner
Speak in privileged state (for those privileged)	ex progmatica
Acceptance of various levels of veracity	consider facts consider opinion [to level *level*]