Meeting the Computer Halfway: Language Processing in the Artificial Language Lojban

Rob Speer RSPEER@MIT.EDU
Catherine Havasi HAVASI@MIT.EDU
77 Massachusetts Ave #32D-712, Cambridge, MA, 02139 USA

1. Introduction

There is one big problem that any natural language researcher faces: all that pesky natural language. In constructing a computational representation of English, or any other natural language, one soon gets bogged down in ambiguity, polysemy, vague grammar rules, and so on.

Getting from English text to a semantic representation is the subject of incredible amounts of NLP research, but currently, any project that wants to work at the semantic level – such as computer reasoning, question answering, or story understanding – has to take shortcuts to get there. Some systems do this by working with a very limited subset of the language – for example, using a small lexicon and simple grammar rules. Systems of this kind can perform well in demos, but are often baffled by a naïve user attempting to communicate with them naturally. Other systems simply start at the semantic level, requiring their input to be in the system's own semantic representation.

We propose a system that communicates with the user in Lojban, an artificial language. Lojban is meant for use by humans, has as much expressive power as a natural language, and has an active community of speakers, but its rules are well-defined enough that a computer can have full command of the language. Using Lojban simplifies the translation from the user input level to the semantic level. This creates a platform that can connect semantics-based tools, such as the Semantic Web (Berners-Lee et al., 2001) or OpenCyc (Cycorp, 2002), to an interface that a user can converse with.

Using our project, the user "meets the computer halfway"; the computer has to translate Lojban into a semantic representation, which is easier to do than with a natural language, and the user has to learn a new human language, which is easier to do than learning the system's programming language. In this way, we hope to make progress in areas of conversational NLP in a way that is currently not possible using English and pave the way for later work.

One major use of this system would be to understand how users interact with a system which is fluent in a language in which they speak and does not require any training. This would aid system developers understanding of how people naturally interact with computers and what behavior expectations people would have with such a system. This would aid in developing interfaces to better fit people's natural computer interaction patterns.

2. Background

Lojban is derived from Loglan, a logical language created by James Cooke Brown in 1960. Like Loglan before it, Lojban is based on predicate logic; a basic sentence consists of a predicate (which acts as the verb) and a number of arguments (noun phrases); many of these noun phrases are themselves derived from predicates.

For example, *blanu* is a predicate meaning "x is blue". As a complete utterance, *blanu* means "something is blue". *mi blanu* means "I am blue", and *le blanu* is an argument referring to "the thing that is blue". *viska* is another predicate meaning "x sees y", so that *mi viska le blanu* means "I see the blue thing".

The grammar of Lojban is unambiguous; a parser can use the formal grammar for Lojban to parse any grammatical Lojban sentence into a unique parse tree (Cowan, 1997). This is quite advantageous, as it means that the work in parsing Lojban is already done for us.

One of the design goals of Lojban is to facilitate communication between humans and computers. This project is a step towards accomplishing that goal.

3. System Overview

Our system, named JIMPE (Lojban for "understand"), is a Python program composed of a number of modules that run concurrently and communicate using messages on a central "whiteboard" object. The interface to a module consists of event-based methods that specify what to do when a given type of message is received. Using this modular structure,

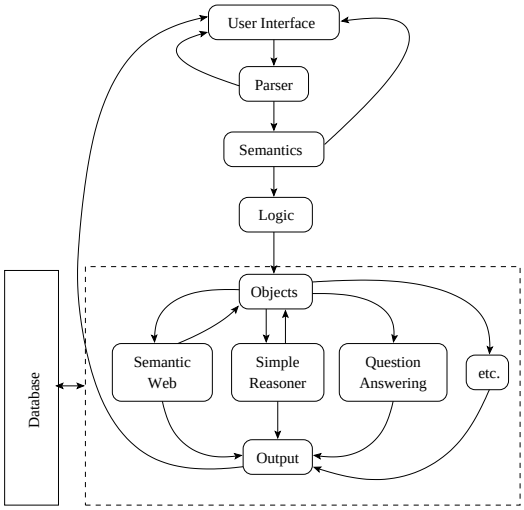

Figure 1. Interactions between some of the system's modules.

we hope to make it easy to extend the system.

Figure 1 shows the interactions between some of the modules in JIMPE.

3.1 User Interface

The user interface module takes input from various modules and displays it to the user. When the user types a line of input, it sends this input as a message to the parser.

3.2 Parser

The formal grammar for Lojban that is the easiest to use is in the form of a Parsing Expression Grammar (PEG), a kind of hierarchical grammar that is well-suited to unambiguous languages. PEGs can be parsed in linear time by a packrat parser (Ford, 2004). So for the parsing step, JIMPE passes the text through *Rats!*, a packrat parser for Java, and builds a parse tree from its output. If the user's input does not parse, it sends a message to the user interface saying so.

3.3 Semantics

The *semantics* module turns a parse tree into an intermediate semantic representation, in which Lojban structures such as predicates and arguments are represented by Python objects.

Sentences that mean the same thing with different word order are reduced to the same representation here – an analogue in English would be to use the same representation for "John kissed Mary" as for "Mary was kissed by John". This intermediate representation is used to keep track of the discourse and to find antecedents of pronouns.

The system so far only has semantic rules for the basic grammar of Lojban; its grasp of Lojban is roughly equivalent to that of a novice user of the language. The semantic rules that the system can handle cover roughly a quarter of the Lojban Reference Grammar (Cowan, 1997).

3.4 Logic

The *logic* module breaks down semantic objects into logical expressions (such as predicates, conjunctions, disjunctions, and negations) that describe the relations between a network of objects.

These logical expressions can have probabilities attached to them. Having probabilistic expressions in the system accomplishes two main purposes. First, though the grammar of Lojban is unambiguous, it is quite possible (and expected) for speakers to say things that are semantically vague, and expect the listener to fill in the meaning from context. This way, the system can make probabilistic "guesses" at the semantic value of expressions that are unclear. Secondly, the system can also make its own guesses about the world and try them out, such as in the *equality* module described below.

3.5 Objects

The *objects* module takes the objects described by *logic* and stores them in a database, which represents JIMPE's model of the world. Starting at this point in the module chain, modules can read and modify the database.

3.6 The database

The modules after *logic* in the module chain can interact with the database. The database stores, for each object it knows about, a list of logical expressions that the object is part of; these logical expressions thus describe the object's relation to other objects.

It also stores a *cardinality*, describing how many things the object represents. A single description might refer to a number of objects, like "two dogs" (*re gerku*); the cardinality can also be a fuzzy quantity such as "many".

Finally, it stores pointers to *children* and *parents*, describing which objects are specific instances of other objects; for example, "two dogs" (*pa gerku*) are an instance of "all dogs" (*ro gerku*). These pointers form a directed acyclic graph describing the hierarchy of objects.

We also store predicates as objects themselves, representing the event or state of that predicate being true. These events store a reference to a lexical object representing the "verb" of the predicate. So the sentence *le gerku cu blanu* ("the dog is blue") is stored as an event object, and as its verb it stores a reference to the lexical object "blanu".

These lexical objects store nothing in themselves. The only way that our system knows about the meaning of "blanu" is its relation to various objects. This means that the system does not start out knowing, for example, that blue things are not red, but as an advantage it means that the system does not need a pre-programmed lexicon of predicates.

3.7 Basic inference engine

JIMPE currently uses a simple inference engine, which draws conclusions from the logical expressions in the database using basic rules of logic. Here is a simple example of the system in action, using the inference engine:

`coi mi'e jimpe`	*Hello, I'm* JIMPE.
`> ganai la bab crino`	If Bob is green,
`  gi la erik blanu`	then Eric is blue.
`je'e`	*Okay.*
`> la erik na blanu`	Eric is not blue.
`je'e`	*Okay.*
`i ua la bab na crino`	*Aha! Bob is not green.*

Note that the system does not need to be told beforehand that *blanu* and *crino* are predicates or that *erik* and *bab* are names – it can determine these from the rules of Lojban. This is another reason that the system does not need a lexicon of predicates.

3.8 Question answering

JIMPE can also answer questions based on facts it has learned or deduced previously. When given a question, it looks in the database for objects matching the descriptions given in the question, and finds whether the question matches the facts it knows about those objects. This "matching" can include discovering that the question is asking about a special case of a more general statement that it knows, as in this example:

`> lo cukta na viska`	Books don't see.
`je'e`	*Okay.*
`> xu lo cukta cu viska mi`	Does a book see me?
`na go'i`	*No.*

3.9 Equality

It is often necessary to conclude that two objects that are described independently are the same object. One very common case is, for example, mentioning "a dog", and then soon after referring to "the dog" (*pa gerku ... le gerku*). Since they are not necessarily the same object, these phrases will refer to two different objects in the database. The job of the *equality* module is to conclude that "le gerku" is likely to be the same object as "pa gerku", and to add a predicate to the database saying that they are equal with a certain probability.

4. Future Work

The representation that our system uses for objects and their relations can easily be converted to the representation used for the Semantic Web. In the future, we plan to add a module that uses Semantic Web inference engines to draw conclusions. By supplying translation rules mapping Lojban words to existing Semantic Web concepts, we could take advantage of the information available on the Semantic Web, including WordNet (Melnik & Decker, 2001).

We will keep programming grammatical and semantic rules until the system can understand almost any grammatical text in Lojban. We can then add some more sophisticated modules for output, which would allow the system to use more of the language to express its own thoughts in a way that is clear to the user.

Finally, in the future, when there is a more developed system for natural language understanding in English, we hope that the progress made using our project will be available to that system as well.

References

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic Web. *Scientific American, 284*, 34–43.

Cowan, J. W. (1997). *The complete Lojban language*. Fairfax, Virginia: The Logical Language Group.

Cycorp (2002). The OpenCyc project. `http://www.opencyc.org`.

Ford, B. (2004). Parsing expression grammars: A recognition-based syntactic foundation. *Symposium on Principles of Programming Languages*.

Melnik, S., & Decker, S. (2001). Wordnet RDF representation. `http://www.semanticweb.org/library/`.