Intelligent Conversational Systems

6.863 Final Project

Emilie Kim, Alex Pesterev and Eugene Wu

18 May 2008
1 Introduction

If language is what truly separates humans from other animals, then good communication with language is what keeps humans together. All relationships are based on good communication, whether it be familial, friendly, romantic, or professional. Without the ability to share ideas or thoughts effectively, we as humans surely would not be able to function.

However, when it comes to our relationships with computers, we seem to make exceptions. According to a study conducted by independent research firm Kelton Research, 65% of consumers report spending more time with their computer than with their significant other [5]. Additionally, the report stated:

- The average consumer has experienced computer troubles eight times [...] over the last three years.
- The average American is wasting 12 hours per month [...] due to problems with their home computer.
- A majority of Americans (52%) describe their most recent experience with a computer problem as one of anger, sadness or alienation.

These findings illustrate the mind-boggling double standard we have created for computers. If any of the bulleted findings were about humans rather than computers, we would all immediately think the relationship problematic and unhealthy. Yet, that kind of frustration seems to be perfectly acceptable when it comes to computers.

Why don’t computers communicate as effectively as humans do? What would it take for computers to engage in the kind of natural dialogue that people have? Why is building intelligent conversational systems between humans and computers so difficult?

Our project seeks to investigate the components of a simple discourse system. This involves two major components. First, we will provide facilities for updating and maintaining the discourse system’s internal model of the world using the user’s input sentences. Second, the system will be able to process user questions about the world and respond with relevant answers.
To do this, we will extend the discourse package in NLTK by modifying the grammar and adding additional structures for modeling the world.

1.1 Background

Conversational systems can be thought of as the interaction between a human and a computer. At the most basic level, all human interaction with a computer can be thought of as a conversation, where the human is simply speaking entirely in the computer’s language. On the other hand, an intelligent conversational system is the dialogue between a human and a computer, but in the human’s language instead of the computer’s. The goal for intelligent conversational systems is for the human to feel as if they are speaking to another human in natural, unconstrained dialogue.

Intelligent conversational systems are not a new area of interest. People have been interested in having natural dialogue between humans and computers for decades, perhaps with one of the most prominent early examples being the famous ELIZA. Created by Joseph Weizenbaum in 1966, ELIZA was a program that pretended to be a therapist and is considered one of the first simulated human-human dialogue endeavours [6].

ELIZA then paved the way for other applications to begin to explore the nature of human-computer interaction. In 1976, Will Crowther wrote the first text-based adventure game, called Colossal Cave Adventure [2]. It was the first game that allowed the user to manipulate objects in the game world using textual commands as input. The game then responded to the user’s actions with textual output, describing the result or consequence of the user’s command.

Current applications that employ intelligent conversational systems might be more familiar, such as Clippy the helpful office assistant in Microsoft Office, or Google’s “did you mean” feature. Both of these applications try to engage the user in natural dialogue, but the conversation still feels very human-computer, instead of human-human.
1.2 Problem

Despite thirty years having passed since programmers first started trying to create applications that simulate human conversation, why do we continue to speak in the computer’s language to this day? There are many issues that make it difficult for conversational systems to engage in natural, human-like dialogue. All of these systems suffer from many of the same issues, for example, only allowing the user to use a small, constrained vocabulary or certain grammatical constructs. There is an extraordinary amount of knowledge about the language required to be able to fully understand and participate in natural dialogue, especially making and understanding inferences. Humans somehow pick this up while learning language, but computers are still far from being able to automatically learn this knowledge on their own.

We examine two aspects of language that make natural language processing difficult for computers. The first is that people are not precise about what they say [3]. We say things like “Who is she?” all the time and people understand what we mean, yet the question itself is actually quite vague. Another problem is that people often abbreviate what they say and speak in incomplete sentences. For example, consider the following dialog:

Alyssa (to Ben): "Hi, I’m Alyssa. What’s your name?"
Ben: "I’m Ben."
Alyssa (to Ben’s friend): "And you?"

Humans understand the dialogue that just took place and were able to infer that Alyssa asked both Ben and his friend what their names were. However, it is not as simple for a computer to understand partial sentences like “And you?” It is not a complete sentence, and almost all of the meaning is actually in the previous question asked by Alyssa.

1.3 Goals

The goal of this project is to take a look at three of the problems that intelligent conversational systems suffer: lack of knowledge of the language, inability to make inferences, and imprecision in the input from the user. We will investigate and create a discourse model that addresses these issues, and if our solutions do not work, understand why they do not work and what
makes these problems difficult to solve.

2 Implementation

We created a system consisting of seven components, as illustrated in Figure 1. One of our design goals for the system was to make it modular and easily-extensible so that it would be easy to put new components in and pull other components out, creating a flexible framework to experiment with making a more robust and accurate system. In this section, we will describe each of these modules in detail.

2.1 User input

The user interacts with the system by entering text from a computer keyboard. To limit the scope of the project, the types of input are restricted to statements about the world such as “Mary is a girl”, and questions about the world such as “How tall is Mary?”. 
2.2 Query interface

The query interface is how the user interacts with the computer. Currently, it is implemented as an interactive Python loop. However, it would not be difficult to envision a richer interface such as a graphical user interface or a speech recognizer for speech input.

2.3 Parser

We used the Earley parser included in the NLTK distribution. Specifically, we used the feature parser that supports lambda expressions in the Context Free Grammar. We used the NLTK’s example “sem4” feature grammar as a starting point. The parser takes the user’s input and creates possible parse trees. Each parse tree also includes the semantic strings as defined by the lambda expressions. For our system, we primarily focused on processing the semantic strings, however, it is also possible to use the features embedded in the parse tree for a richer understanding of the user’s input.

We made three major modifications to the grammar to support the grammatical changes we wanted to implement. First, we added quantifiers for adjective phrases, such as “6” for the phrase “6 feet tall”. Adding numbers to the grammar proved to be challenging. We could not find a way for the grammar to arbitrarily parse numbers, and consequently resorted to hard coding the numbers 1 to 100 into the grammar. For example, the number 100 is represented as:

$$\lambda y \ P \ M. ((P \lambda Q.(Q M)) \ 100) \ and \ (y \ 100))$$

This complex expression is used because we wanted a clean way of inserting numbers into the final semantic expression. Here is “Mary is 10 feet tall”:

$$(\text{and} \ (\text{height_of} \ 10 \ Mary) \ (\text{foot} \ 10))$$

The first statement says that Mary’s height is 10, while the second statement indicates that 10 is in units of feet. The lambda expression above is also complex because it needs to work in conjunction with many other lambda expressions already present in the grammar. For example, the expression needs to accommodate proper nouns; Mary is represented as $$\lambda P.(P \ Mary)$$
instead of simply Mary.

Second, we added adjectives for question style input; the adjective “tall” needed a different representation depending on whether it was being used in the statement “He is tall” or the question “How tall is he?” The grammar has three different types of adjectives that are used depending on the context.

The different versions of an adjective are needed because the lambda expression is tweaked in different situations; in fact, the change represents semantic differences in what the words mean in the different cases. The first adjective is used in sentences like “Mary is tall”, where the lambda expression is simply tall, and the semantic expression is (tall Mary). “How tall is Mary?” needs an adjective that will query the tall property of Mary; this translates into the following lambda expression: \( \lambda P. \text{P (height_of x)} \). The following semantic expression is then \( \lambda x. (\text{height_of x Mary}) \). Finally, a third adjective type is needed to cover sentences like: “Mary is a tall girl.” This type of adjective is used to describe implicit conjunctions; the sentence “Mary is a tall girl” needs to be expanded out to “Mary is tall” and “Mary is a girl”. The lambda expression is: \( \lambda P Q. ((P Q) \text{and (tall Q)}) \) and the final semantic expression is (and (girl Mary) (tall Mary)).

Third, we added support for the conjunction (“and”) and disjunction (“or”). It proved tricky to expand phrases such as “Mary and Bob know Fido” into “Mary knows Fido” and “Bob knows Fido”. There are two forms of conjunctions. In addition to the first conjunction grammar rule, \( \lambda P Q R. ((P R) \text{and (Q R)}) \), the grammar needed massaging to parse the “and” and “or.” The second conjunction is used to bind sentences together, for example, “Mary is a girl or John is a boy.” The lambda expression for this rule is simply \( \lambda P Q. (P \text{and Q}) \).

Appendix A has a list of sample sentence that our grammar can parse.

2.4 Background knowledge

This module is one of the two main modules dealing with conversational context and discourse representation and is comprised of two parts: the hierarchical class model and the natural language processor (NLP) manager. The hierarchical class model allows the system to represent and apply at-
Figure 2: Hierarchical class model. STUDENT inherits attributes from PERSON.

tribute inheritance, providing the system with inferential power, and the NLP manager provides the system with knowledge of language.

2.4.1 Hierarchical class model

To understand how the hierarchical class model works, take the example statement, “Mary is a student.” If one person says this statement to another person, the other person automatically knows that Mary also has two feet, has a student ID and countless other properties. We can make these kinds of logical jumps because we know so much about the knowledge encoded into language, namely that a student is a kind of person, and most people have two feet, so therefore if Mary is a student, Mary also has two feet. Computers do not come with this knowledge, so we include this information as background knowledge so the discourse system has an adequate model of the world and uses it to make intelligent inferences.

Our model borrows heavily from the class model in object-oriented programming. An example of what the class model looks like is illustrated in Figure 2. The Person class has the attributes name, height, and feet, where feet = 2. The Student class has student-specific attributes student_id and year, but also inherits from the Person class. Similarly to how object-
oriented class models work, Student inheriting from Person means that all attributes of a Person are automatically inherited by Student.

Without this hierarchical class model, if a user wanted to tell the computer that “Mary is a student,” the user would have to also specify “Mary is a person,” so that the computer would also know that not only does Mary have a student ID and a school year, but that she also has a name, a height, and two feet. The hierarchical class model allows the user to simply state that “Mary is a student,” and then the system can automatically infer that Mary is also a person.

Additionally, we create object instances for each unique subject, such as “Mary” or “Fido” that the system recognizes. When an instance is instantiated, the system specifies the set of classes the object inherits from, which populates the object with a set of default attributes. The instances are stored in a global data structure so that they can be referenced in future parts of the discourse. For example, after the first utterance of “Mary”, the system creates an object to model “Mary”, and appends a pointer to the object to all future occurrences of “Mary”.

Currently the class model is static, in the sense that the discourse system does not add, remove or modify properties or classes in the model. However this limitation is because of time constraints that prevent us from adding the necessary functionality to the grammar and discourse system rather than a limitation in the model. With more time, we would support general statements such as “Every person has two ears”, and the system would add an ears attribute to the \verb Person+ class with the default value of 2, or more specific statements such as “Mary has brown hair”, which modifies a single object instance’s attributes.

2.4.2 Natural language processing manager

The NLP manager controls two sub-managers: the alias manager and the verb manager. The system uses these managers to store background knowledge that does not change and does not get “learned” by observing input over time, at least by our system.
The alias manager is in charge of managing words that are similar enough in meaning so that the parser should treat them equivalently. For example, the words “tall” and “height” have similar meanings, so the system needs to know that these seemingly different words relate to the same concept. An area for improvement for the alias manager would be to look at using WordNet, a database that groups nouns, verbs, and adjectives together based on their conceptual meaning, much like the alias manager does [7]. Unfortunately, there was not enough time to sufficiently investigate using WordNet for our system, but would be an excellent area to start future work.

The verb manager associates verb conjugations with their respective infinitives, the number of arguments the verb takes, and the index of the subject. Again, using WordNet would be something to look at, but another area of improvement would also be to explore dealing with the different conjugations in the grammar, before it reaches the background knowledge module. This way the grammatical rules of conjugation are taken care of in the grammar, and the background knowledge module wouldn’t have to know about verb conjugation.

2.5 Environment frames

This module is the second of the two main modules that make up the intelligence of the system’s conversational model. Similar to context environments described in [1], environment frames allow the system to keep track of the conversation, both what the current subjects are as well as the history of the conversation. Figure 3 illustrates the environment frames.

Each environment frame contains a reference to the previous environment frame as well as all of the subjects in the user’s input. The subjects are pointers to the globally instantiated objects described in the hierarchical class model. The decision to only store subjects was made for simplicity and proof of concept. In the future, the frames could be augmented with additional information. For example, the topic of the sentence could be stored, as well as all of the verbs and attributes in the sentence. Modeling of the user’s input, such as attention or intention, could also be stored, allowing the system to respond more intelligently to user queries [3].

One of the functions the environment frames enable is pronoun disam-
biguation, which is essential for understanding human language. When people speak, often they are vague about who or what they refer. Being able to determine the antecedent gives the conversational system the ability to understand the user.

For this project, we focused on supporting gender-based pronouns, such as “he” or “she” by encoding gender into the pronouns and proper nouns of the system's grammar. For example, the proper noun “Mary” would have an attribute $\texttt{sex=f}$, and the proper noun “Bob” would have an attribute $\texttt{sex=m}$. Likewise, the pronoun “he” would have an attribute $\texttt{sex=m}$ and “she” would have the attribute $\texttt{sex=f}$. This new attribute of the grammar allows the system to match pronouns to proper nouns based on gender.

The system finds pronoun antecedents by looking up through the previous environment frames and finding an already-mentioned subject that fits the gender of the pronoun in question. An example of how these environment frames look is illustrated in Figure 3. In frame 2, “She” is the pronoun, and we look up to see that “Mary” is the only female subject that can match the pronoun, so we associate “She” to “Mary.” Likewise, in frame 3, the previous subjects from the conversation are “Mary” and “Fido”, but because

Figure 3: Environment frames.
“He” has a male attribute, “Mary” is eliminated as a possible subject choice, and “Fido” becomes the antecedent. If there are more than one possible subject that matches on gender, the system then sorts the possible subjects temporally, so the most recent matching subject becomes the antecedent.

If the user’s input is a question, the environment frames also are able to determine if the system is able to answer the question, and if so, what the answer is. The objects that are saved in the environment frames are the objects from the class models, so all of the objects know all of their own attributes and can answer questions.

2.6 Answer interpreter

If the user poses a question to the system, then the answer interpreter is responsible for turning the unformatted answer returned from the environment frames into a human-readable response that the user can understand.

Currently, the answer interpreter simply returns unformatted, factual answers; if the user asks “What colour is Mary’s hair,” our system would return \{\texttt{x}: \texttt{brown}\}, where \texttt{x} is the variable representation of \texttt{hair\_colour}. The very basic step above this would be to return a human-readable response, such as “brown”. An area for further improvement would be to make the answer interpreter return a more conversational answer, such as “Mary’s hair is brown”, or a more inferential answer, such as “Mary is a brunette”.

2.7 Computer output

The last module in the system represents how the computer communicates back to the user. Currently the system simply displays text to the user, but future work could extend this to include speech synthesis or other modalities to express the output.

3 Evaluation

We decided to give our system background knowledge of some well-known people. Below is a sample conversation with the system. The conversation
was continuous; the breaks in the conversation are to provide an interpretation of the most recent conversation turn.

-> Berwick knows Winston and Fodor
Berwick.know = Winston
Berwick.know = Fodor

Both Winston and Fodor are now in the list of people that Berwick knows.

-> He is tenured
Berwick.tenured = True

Because the subject of the previous frame was Berwick, the environment frames were able to determine that “He” maps to “Berwick” and then tenured Berwick.

-> Fodor and Winston are tenured
Fodor.tenured = True
Winston.tenured = True

This is an example of applying an attribute to multiple subjects connected by an “and” in the same sentence.

-> Who is tenured
{'x': 'Winston'}
{'x': 'Berwick'}
{'x': 'Fodor'}

The system goes through all the subjects and sees if any of them are tenured.

-> Fodor is 100 years old
Fodor.age = 100

The system sets the value 100 for the age attribute of Fodor.

-> How old is he
{'x': '100'}
Here, \( x \) is the representation for age, just in unevaluated variable form. The system also correctly identified “he” as “Fodor” because Fodor was the most recent subject that fit the “he” pronoun.

\[
\text{-> Berwick is 6 feet tall} \\
\text{Berwick.height = 6}
\]

The system sets the value 6 for the height attribute of Berwick.

\[
\text{-> How tall is Berwick} \\
\{'x': '6'}
\]

Here, \( x \) is the representation for height, just in unevaluated variable form.

4 Discussion

We were fairly ambitious in what we thought our system would be able to do. Certain parts, such as the answer interpreter, ended up being harder than we had anticipated. While our discourse only works for a limited set of phrase types, our system provides a good structural starting point for improving NLTK’s discourse module. In this section, we propose a few areas of future work that would improve the general discourse system, as well as enumerate our contributions.

4.1 Future work

Given the expansive area of topics covered by the scope of this project and the constrained time frame of a class project, our system has many possible avenues to take for future work. Several of these improvements are targeted towards building a robust dialogue system rather than an input-only discourse system.

- **Flexible background knowledge store.** Learning new attributes and relations from the discourse and adding them to the class models and NLP Manager would dramatically improve the system’s model of the real world.
• **Incorporate WordNet.** WordNet seems like it does exactly what the alias and verb managers in our system were intended to do, as well as much more. Given more time, we definitely would take a look at using WordNet to help handle the daunting task of providing the system with knowledge about language.

• **Increase information stored in environment frames.** We currently only store sentence subjects in the environment frames. Adding more information to the environment frames only helps with later disambiguation, filtering, or determining an appropriate response for the user.

• **Improve the answer interpreter.** Although not directly related to natural language processing, we feel that this is an integral part of any discourse system. The simple solution would be to use sentence templates for the system’s responses to questions, however, a more sophisticated system may use the data from WordNet to build a more robust sentence generator.

• **Provide natural, human-like error messages.** Currently if the system fails to understand some aspect of the user’s input, it outputs a generic error message simply saying it couldn’t handle the input and on which word it failed. These error messages could also be made more conversational. For example, if the user asked, “How tall is Bob?”, and the system did not have a height attribute set for Bob, it could respond with “I don’t know how tall Bob is,” which is what a human might say but is difficult for a computer to generate.

4.2 Contributions

Over the course of this class project, we learned a lot from our investigation and implemented a modest set of contributions to the NLTK discourse package.

• **Developed natural language processing manager to draw on knowledge of language.** The alias manager and the verb manager, both part of the natural language processing manager, were used to store immutable information about language. The system used this information to intelligently draw inferences between seemingly different words that shared conceptual meaning.
• **Populated background knowledge module.** This was a crucial part of creating a system that would not only contain background knowledge about a domain, but also able to infer based on that knowledge. We were able to get our system to utilise inferential power to draw conclusions based on statements not explicitly stated by the user.

• **Designed environment frames.** An environment frame stores all relevant information about a particular sentence. Our current system uses the frames to keep track of known subjects in each sentence. These are used for our simple pronoun disambiguation algorithm, which filters by pronoun gender, then picks the temporally closest subject. Richer algorithms can be implemented by embedding more information in the environment frames.

• **Implemented flexible and modular system of conversation understanding.** We have built a modular and easily-extensible system for future developers to implement and plug in their own modules. This flexibility provides an experimental environment that will help these developers make significant contributions to the field of discourse processing.
References


Appendix A

This appendix contains sample sentences that our grammar can parse. The first set of sentences shows the type of quantifiers that can be used in our grammar.

- Fido is the dog
- Fido is a dog
- Fido is every dog
- Fido is no dog
- Fido is some dog
- What is the dog
- How is the dog
- Mary has a dog
- Mary has Fido
- the dog is happy
- the dog is not happy
- a dog is not happy

Next, the sentence below show how pronouns can be used in substitute for proper nouns.

- she chases Mary
- Mary chases her
- she chases her
- John chased Fido
- Who chases Fido
- who chases who
Below are example sentences using “and” and “or”:

- Mary and John chase Fido
- Mary or John chase Fido
- Mary chases Fido and John
- Mary chases Fido or John
- Mary walks with Fido and John
- Mary and John walk with Fido and Bob
- Mary is a girl and John is a boy
- Mary is a girl or John is a boy

The following sentences demonstrate adjective and quantifiers.

- Mary is 10 years old
- How old is she
- Mary is tall
- How tall is Mary
- How tall is she