Quagent Manipulation through Natural Language Understanding

Computer Science 242: Artificial Intelligence
April 16, 2004

Rob Van Dam and Greg Briggs

Abstract: Artificial intelligence algorithms designed to simulate an agent with the Quake II environment were controlled with natural language statements and commands. These natural language statements were parsed using versions of the XML-based OpenCCG grammar and lexicon specifically augmented for the understanding of the Quake agent. The parsed statements and command were interpreting semantically as short and long term goals for the agent to attempt. Short term goals were those that would be executed essentially upon receiving them and in the order received. Long term goals were those that would be executed over a period of time and often in parallel with other long term goals

Keywords: Quagent, Quake, AI Exploration, Natural Language, OpenCCG

Changes in this second version: Added appendix of grammar definition.

Background and Motivation

The Quagent concept is the use of the Quake II 3D video game engine for testing artificial intelligence algorithms. In this paradigm, an agent, or "bot," interacts with the Quake world under the direction of an external program. We design a bot that is defined and controlled by a learning algorithm and receives percepts in a way unique to the model of the "world" in which the quagent must maneuver. These learning algorithms use the rewards perceived from the environment to calculate appropriate actions for the quagent to perform from its given state to maximize reward. Controllers written in Java supply the quagent with the simple commands needed to maneuver within the world. It is logical to extend these controllers to enhance the quagent's inherent command syntax to accept, parse, and understand natural language statements and commands and respond to them appropriately.

Methods

Natural Language Understanding subdivides into four distinct tasks in order to allow the quagent to go from receiving a natural language sentence to fulfilling the implied goal. The first area is accepting natural language statements. This is a simple task completely within master controller using Java input. The second task is to parse this natural language statement (or command) into its syntactic and grammatical parts in a logical manner that can be used for further semantic parsing, which was done utilizing the OpenCCG framework and parser. Note that, before passing text to the parser, pre-parsing such as case-folding is performed. The third task is the semantic interpretation of these parses, to recognize specific instructions or questions. The fourth task is completing the tasks or goals as they were understood. A fifth task could be added that would entail generating full natural language responses to the user to allow for greater and more flexible communication and interaction between the user and the quagent.

As far as coding our agent in Java, Greg focused on the OpenCCG grammar, the parse scoring system, and resolving ambiguities, whereas Rob focused on taking the selected parse structure, figuring out what it means to the quagent, and actually getting the quagent to carry out the commands, ask questions when it is confused, and manage its list of requested and desired goals. Greg also implemented the "Where is the xx?"responses.

Pre-parsing

We performed several pre-processing steps to clean up the grammar

for the OpenCCG parser. These were case-folding, splitting of multiple sentences into separate sentences, and adding spaces before punctuation. This is because OpenCCG is case-sensative, our parse interpretation can only handle one sentence at a time, and because openCCG strictly uses spaces to denote tokens. (If you pass OpenCCG a word immediately followed by a period, it will consider that to be a single word.)

Parsing with OpenCCG

Parsing natural languages requires both a lexicon and a grammar to define the language (or the subset that is to be used). To parse, we used OpenCCG, a Java-based parser for a combinartory categorial grammar. A lexicon and a grammar based on the OpenCCG grammar written in XML were enhanced and altered to include the words and grammatical syntax necessary for the types of statements that would be expected as input to the quagent.

The result of an OpenCCG parse is a recursive tree-like structure. It defines actions and their named properties. The root of this tree is typically the verb in the sentence. The subject and direct object are considered to be properties (that is, child nodes) of the action.

Semantic Interpretation

The next task is the semantic interpretation of these parses into the goals that are implicit or explicit within the natural language sentence. This requires recognizing not only individual words for superficial understanding but how the structure of the sentence and the words used can have slightly different implications for what actions the quagent is to perform and when, where, and how it should do it.

Goal Completion

Once the agent understands what it is being told, it should attempt to act accordingly. This involves keeping track of current goals, erasing completed ones, and correctly altering the current set of goals when new ones are received before completing the current ones. These four basic tasks were implemented in Java by extending a previously existing structure of quagent controllers that take a learning algorithm (in this case Continuous Policy Iteration) and a world model (in this case specific both to the need of parsing natural language and maintaining the layout of the quake world in which the quagent moves) and manipulate the quagent within that world according to the policy calculated based on percepts received.

Responding to Messages

Once an incoming message is understood, the quagent will respond in a way that clearly indicates what it did or did not understand. Additionally, it may need to respond to answer questions. The grammar used to respond was not based on the input grammar. Rather, each response was programmed using a fixed string with variables being filled in. A response is actually generated for each possible parse, but only the best parse's response is returned to the user.

For questions, each response was individually programmed. The bot knows how to list what it is doing, and how to tell the locations of nearby objects.

Test Results

Using Grammar for disambiguation of Singular vs. Plural:

I say “Pick up the tofu.”. Both interpretations are returned by the parser:

(pickup ^ <tense>pres ^ <Goal>(tofu ^ <det>the ^ <num>pl))
(pickup ^ <tense>pres ^ <Goal>(tofu ^ <det>the ^ <num>sg))

But, if I say “Pick up a tofu.”, we only get the singular interpretation:

(pickup ^ <tense>pres ^ <Goal>(tofu ^ <num>sg))

Using context for disambiguation of Singular vs. Plural:

In Quake world, “data” is used as both singular and plural. If I say “Get the data”, while there is more than one data in sight, then the evaluations are:

(get ^ <tense>pres ^ <Goal>(data ^ <det>the ^ <num>pl))
The above parse had likelihood score 0.5: "I will try to get some of those."

(get ^ <tense>pres ^ <Goal>(d1 ^ data ^ <det>the ^ <num>sg))
The above parse had likelihood score 0.49: "I will try to get one data."

As you can see, the plural interpretation is favored, to agree with the context of having more than one data. The opposite happens when only one data is in sight.

Scoring different meanings (infinitive or command) of the same verb, if I say “Walk to him.”

(walk ^ <tense>pres ^ <Goal>(p1 ^ pro3m ^ <num>sg))
The above parse had likelihood score 0.4: "So I guess I should walk (to) the pro3m. I don't know what a pro3m is."

<GenRel>(walk ^<tense>pres ^ <Goal>(pro3m ^ <num>sg))
The above parse had likelihood score -1.0: "What is/am/are walk doing? (I didn't hear any verbs.)"

The first parse considers “walk to him” a command, although it is not able to resolve the third person pronoun. The second parse, on the other hand, considers the idea of walking, such as “To walk”, the infinitive.

Pronoun resolution, demonstrated:

I say “Pick up the tofu.” Quagent response: “So I guess I should pickup the tofu.” Then I say “Drop it.” Quagent response: “So I guess I should drop the tofu.”

I say “Where is the data?” The response, “You want to know where a data is. The nearest data is to the right, a short distance away.” I say, “Get it.” The quagent responds, “I will try to get one data.”

Using context to resolve pronouns:

I say, as my first statement to the Quagent, “pick up it,” while it is standing within view of a data CD. It replies: “I will try to pickup one data.”

Additional examples of comprehension:

I say “Walk.” (or “You walk.”)
It says “So I guess I should walk. I'll take a few steps.”

I say “I walk”
It says “So I guess you should walk.”

I say “I walked”
It says “I do not remember you walking.”

I say “Turn left. Walk. Turn right.”
It says “So I guess I should turn. So I guess I should walk. I'll take a few steps. So I guess I should turn.”
As you can see, it can handle multiple sentences at once. However, it does not join its responses together into a single “So I guess I should turn left, walk, and then turn right.”

I say “What are you doing?”
It says “I don't have any current short term goals.
My long term goals are to:
* find all the tofu objects.

* find one of the head objects."

I say "Where are the data?" It says "You want to know where every data is. The nearest data is to the right, a short distance away."

Discussion

Syntactic Parsing

The OpenCCG parser returns a list of possible parses. Some of these parses arrange the sentence in ways that don't make sense, others use words as having a different meaning, and some are correct. The OpenCCG grammar is combinatorial, which means that it considers all possible parses, and categorical, which means that words are not only divided into their precise parts of speech, but they are also divided into a categories. For example, one might have a subset of nouns that are "objects in quake world" (tofu, gold, data, box, etc.). Then, one can define a verb such as pick up to make sense when it's direct object is a "pickupable" noun.

Since the intent of OpenCCG is to be able to interpret all aspects of a natural language, it comes with a grammar which predefines numerous parts of speech. (We downloaded the latest grammar from the OpenCCG CVS repository.) It is a very complex grammar. For example, there are 17 parts of speech just for verbs. This complexity made it fairly impossible to correct any problems in the definitions for the parts of speech. For example, the word "if" cannot be parsed in several common uses, and it would take great time to broaden the grammar to fix this problem.

Semantic Interpretation

There are really two halves to the issue of achieving a correct semantic interpretation of any given natural language sentence. First and foremost it is imperative that the syntactic parse is as correct as possible. However, the act of determining this level of "correctness" is in fact the act of interpreting it semantically. Since the controller has no way of knowing what is "correct" it must instead assign a probability to each syntactic parse and then attempt to find a semantic understanding of the parse with the highest likelihood of being "correct".

Each of the parses is individually submitted to a module (called "ConsiderSentenceUnit") which examines the parse. (If there are multiple possible pronoun resolutions, it may be submitted once for each possibility.) It returns a likelihood score (which combines ability to understand as well as context- and sensory-dependent information), a response message for the quagent to say, a list of new goals that the quagent should try to

accomplish, and a list of new possible pronoun references. Additionally, nearby objects are considered for resolving third person pronouns like “it” and “them”.

The “ConsiderSentenceUnit” module understands a subset of the possible parse trees that it might receive. When the parse tree does not make sense to it, it returns a low likelihood score, in hopes that an alternative parse which it can understand is later submitted. For parse trees where it understands a complete meaning, a high likelihood score is given. Similarly, if it understands a partial meaning, a intermediate score is returned. These scores are then combined with a score for sensory information. When possible, singular and plural verbs are checked against the surrounding to see whether singular or plural is intended. Since “tofu”, “data”, and “kryptonite” are all ambiguous as to whether they are singular or plural, this helps resolve the problem. Note that, if usage in the sentence makes it clear that only singular or plural was intended, then OpenCCG's parser will only return the grammatically correct interpretation, so there is no sensory comparison needed in those cases, even though the score for the single parse will still be altered by the sensory data.

Additionally, if the quagent recently asked a question, and the parse can be interpreted as an answer, then it will receive a very high likelihood score. This allows the user to answer questions, or to say something else instead of answering the question.

Once a particular syntactic parse has been chosen then comes the task of creating a semantic understanding of the statement that the quagent can later understand and comply with or respond to. A single sentence could contain various parts that need to be separated out and fulfilled or complied with in distinct ways. A simple sentence could contain a single command but a more complicated sentence could contain a list of explicit commands, could imply a necessary task or could contain a conditional statement. It is necessary then to extract any and all tasks given within a single statement, determine whether those tasks must be completed serially or in parallel (generally the idea of short-term versus long-term goals).

If a sentence contains a conditional statement such as “If you have x Health, avoid the Kryptonite.” requires that the quagent both be able to determine if current conditions comply with the conditional part of the sentence and make sure that the task is only completed once the condition has been fulfilled. Depending on the context, this could translate into maintaining a list of tasks to complete whenever a certain event occurs, such as its Health dropping below a certain threshold. The issue of whether the user meant for the quagent for condition to hold indefinitely or only meant “If this is true NOW” is a difficult semantic ambiguity.

Goal Completion

In order to complete the tasks that it interprets from its communication with the user, the quagent requires some sort of simple storage mechanism in order to maintain a list of current tasks and their level of completion and other useful information. Even though this information is contained within the same natural language sentences that were initially inputted, it is not efficient for the quagent to reprocess every sentence every couple seconds to see if it should be doing something different.

Two classes were implemented for the use of storing and retrieving these tasks. One was a simple Goal container that allowed for the creation of Goals based on an action (generally the verb in the command), and various parameters or objectives necessary for the completion of that action. In particular a given goal could include an action, some immediate parameter for that action (such as how many degrees to turn or how far to walk), the object (i.e. Tofu, Data, Head, etc) to which the action should be applied and how many times it should be performed. There was also an option to perform the action on any and all objects of the specified type. This allows for general commands such as “Find all the heads.” Each goal also maintained the origin of the quagent when it received the command. Although they were not included in the lexicon, this could be used to better define such words as “here”, “there”, “behind”, “in front”, “far”, “near”, etc. Each goal could also contain a list of points to visit in order to complete goals that would require multiple steps, such as “Bring a head here”.

These Goals were then kept in lists that allowed the quagent to continuous check them to see what it should be doing in each cycle. Short term goals were in a FIFO style queue so that they would be executed essentially in the order that they were received (this could be altered slightly if the commands were given in association with a condition). Long term goals, ones that could not be completed in a single action, (such as picking up all the Tofu) were stored in a list in which all the goals could be processed simultaneously if possible. It was predetermined which types of long term goals could be processed in tandem. These goals generally involved altering the Transition-Reward Model of the world to make the quagent move around the map in a certain way, such as visiting specific places.

If a Goal received from the user requires the quagent to find certain objects, it is possible that by that point the quagent has explored enough territory to have at least some of those objects in its memory in order to go back and find them but most generally the Goal at hand will involve finding new objects of the specified type. The quagent must then thoroughly

explore the map. The style in which the quagent explores is based on a continuous policy iteration style learning algorithm that includes a look ahead factor that alters how much emphasis is placed on prior knowledge versus new percepts. If the look ahead is low, the quagent tends to meander more and explore more unexplored territory. If the look ahead is high, the quagent will move more quickly along the path with the best reward values. This is very useful for retrieving certain objects since the state which contains a desired object can be assigned a higher reward value and the look ahead raised so that the quagent quickly moves to that place. Once the object is retrieved, the quagent continues exploring but still at the high look ahead rate until the task is completed. If necessary, the quagent can also be told to stop exploring, at which point new unvisited states are no longer assigned a reward value, basically halting the quagent (unless he still has a specific long term Goal to achieve). This can be used to make the quagent follow a set of specific directions in the order given without the risk of the quagent moving to another state for exploration while performing other assigned short term Goals.

Response Generation

Since the responses were generated based on fixed template sentences, there were often exceptions that needed to be specially handled. Problems we encountered during implementation included subject-verb agreement, selection of prepositions to use with verbs (such as “walk to” versus “search for”), and removing the second “I” from “I don't remember I walking,” (as opposed to “I don't remember you walking.”)

Conclusion

Any attempt at true Natural Language Understanding will always be heavily limited by any limitations in the lexicon and grammar provided. Although it is possible to learn new words, this is a very complicated process and would not yield many results in an NLU application such as this that already starts with a very small and more easily defined lexicon.

Our quagent has a lexicon that is fairly simple but covers many of the basic tasks that a quagent might be expected to understand. It successfully handles low complexity sentences and meanings. It is limited of course by its ability to correctly interpret the input into appropriate goals. However, when the semantic interpretation is successfully and the goal is achievable, our quagent is very quick and thorough in its completion.

Complicated sentences combined in strange ways are beyond the scope of the quagent's abilities. One of the more complicated types of

sentences to parse and interpret are questions, since they have a different syntactic structure to that of regular statements and even commands. However, the quagent is capable of understanding and responding to some basic types of questions that were anticipated.

Though not capable of the most complex natural language understanding, our quagent has proved to be quite usable in the basic quake scenario.

Appendix: Excerpts from the Grammar Definition

The following was added to the existing OpenCCG grammar. The OpenCCG grammar already included all English pronouns and several propositions and articles, as well as a lexicon with several parts of speech.

Dictionary

```
<!--Adverbs->
<entry stem="left" pos="Adv">
  <member-of family="Adverb"/>
</entry>
<entry stem="right" pos="Adv">
  <member-of family="Adverb"/>
</entry>

<!-- Prepositions, Temporal Adverbials -->
<entry stem="near" pos="Prep"> <!-- not yet semantically understood -->
  <member-of family="Prep-Nom"/>
  <member-of family="Prep-Time"/>
</entry>

<!-- Verbs -->
<entry stem="walk" pos="V">
  <member-of family="IV"/> <!-- Intransitive Verb -->
  <word form="walk" macros="@base"/>
  <word form="walking" macros="@ng"/>
  <word form="walk" macros="@pres @sg-agr @1st-agr"/>
  <word form="walk" macros="@pres @sg-agr @2nd-agr"/>
  <word form="walks" macros="@pres @sg-agr @3rd-agr"/>
  <word form="walk" macros="@pres @pl-agr"/>
  <word form="walked" macros="@past"/>
</entry>
<entry stem="bump" pos="V">
  <member-of family="IV"/>
  <word form="bump" macros="@base"/>
```

```
  <word form="bumping" macros="@ng"/>
  <word form="bump" macros="@pres @sg-agr @1st-agr"/>
  <word form="bump" macros="@pres @sg-agr @2nd-agr"/>
  <word form="bumps" macros="@pres @sg-agr @3rd-agr"/>
  <word form="bump" macros="@pres @pl-agr"/>
  <word form="bumped" macros="@past"/>
</entry>
<entry stem="talk" pos="V">
  <member-of family="IV"/>
  <word form="talk" macros="@base"/>
  <word form="talking" macros="@ng"/>
  <word form="talk" macros="@pres @sg-agr @1st-agr"/>
  <word form="talk" macros="@pres @sg-agr @2nd-agr"/>
  <word form="talks" macros="@pres @sg-agr @3rd-agr"/>
  <word form="talk" macros="@pres @pl-agr"/>
  <word form="talked" macros="@past"/>
</entry>
<entry stem="explore" pos="V">
  <member-of family="IV"/>
  <word form="explore" macros="@base"/>
  <word form="exploring" macros="@ng"/>
  <word form="explore" macros="@pres @sg-agr @1st-agr"/>
  <word form="explore" macros="@pres @sg-agr @2nd-agr"/>
  <word form="explores" macros="@pres @sg-agr @3rd-agr"/>
  <word form="explore" macros="@pres @pl-agr"/>
  <word form="explored" macros="@past"/>
</entry>
<entry stem="step" pos="V">
  <member-of family="IV"/>
  <word form="step" macros="@base"/>
  <word form="stepping" macros="@ng"/>
  <word form="step" macros="@pres @sg-agr @1st-agr"/>
  <word form="step" macros="@pres @sg-agr @2nd-agr"/>
  <word form="steps" macros="@pres @sg-agr @3rd-agr"/>
  <word form="step" macros="@pres @pl-agr"/>
  <word form="stepped" macros="@past"/>
</entry>
<entry stem="go" pos="V">
  <member-of family="IV"/>
  <word form="go" macros="@base"/>
  <word form="going" macros="@ng"/>
  <word form="go" macros="@pres @sg-agr @1st-agr"/>
  <word form="go" macros="@pres @sg-agr @2nd-agr"/>
  <word form="goes" macros="@pres @sg-agr @3rd-agr"/>
  <word form="go" macros="@pres @pl-agr"/>
  <word form="went" macros="@past"/>
```

```
</entry>
<entry stem="turn" pos="V">
  <member-of family="IV"/>
  <word form="turn" macros="@base"/>
  <word form="turning" macros="@ng"/>
  <word form="turn" macros="@pres @sg-agr @1st-agr"/>
  <word form="turn" macros="@pres @sg-agr @2nd-agr"/>
  <word form="turns" macros="@pres @sg-agr @3rd-agr"/>
  <word form="turn" macros="@pres @pl-agr"/>
  <word form="turned" macros="@past"/>
</entry>
<entry stem="pickup" pos="V">
  <member-of family="PickupDropSeeSense"/>
                          <!-- Special type of transitive verb -->
  <word form="pickup" macros="@base"/>
  <word form="pickingup" macros="@ng"/>
  <word form="pickup" macros="@pres @sg-agr @1st-agr"/>
  <word form="pickup" macros="@pres @sg-agr @2nd-agr"/>
  <word form="picksup" macros="@pres @sg-agr @3rd-agr"/>
  <word form="pickup" macros="@pres @pl-agr"/>
  <word form="pickedup" macros="@past"/>
</entry>
<entry stem="get" pos="V">
  <member-of family="PickupDropSeeSense"/>
  <word form="get" macros="@base"/>
  <word form="getting" macros="@ng"/>
  <word form="get" macros="@pres @sg-agr @1st-agr"/>
  <word form="get" macros="@pres @sg-agr @2nd-agr"/>
  <word form="gets" macros="@pres @sg-agr @3rd-agr"/>
  <word form="get" macros="@pres @pl-agr"/>
  <word form="got" macros="@past"/>
</entry>
<entry stem="open" pos="V"><!-- not yet semantically understood -->
  <member-of family="PickupDropSeeSense"/>
  <word form="open" macros="@base"/>
  <word form="opening" macros="@ng"/>
  <word form="open" macros="@pres @sg-agr @1st-agr"/>
  <word form="open" macros="@pres @sg-agr @2nd-agr"/>
  <word form="opens" macros="@pres @sg-agr @3rd-agr"/>
  <word form="open" macros="@pres @pl-agr"/>
  <word form="opened" macros="@past"/>
</entry>
<entry stem="hit" pos="V"><!-- not yet semantically understood -->
  <member-of family="PickupDropSeeSense"/>
  <word form="hit" macros="@base"/>
  <word form="hitting" macros="@ng"/>
```

```
  <word form="hit" macros="@pres @sg-agr @1st-agr"/>
  <word form="hit" macros="@pres @sg-agr @2nd-agr"/>
  <word form="hits" macros="@pres @sg-agr @3rd-agr"/>
  <word form="hit" macros="@pres @pl-agr"/>
  <word form="hit" macros="@past"/>
</entry>
<entry stem="drop" pos="V">
  <member-of family="PickupDropSeeSense"/>
  <word form="drop" macros="@base"/>
  <word form="droping" macros="@ng"/>
  <word form="drop" macros="@pres @sg-agr @1st-agr"/>
  <word form="drop" macros="@pres @sg-agr @2nd-agr"/>
  <word form="drops" macros="@pres @sg-agr @3rd-agr"/>
  <word form="drop" macros="@pres @pl-agr"/>
  <word form="droped" macros="@past"/>
</entry>
<entry stem="see" pos="V"><!-- not yet semantically understood -->
  <member-of family="PickupDropSeeSense"/>
  <word form="see" macros="@base"/>
  <word form="seeing" macros="@ng"/>
  <word form="see" macros="@pres @sg-agr @1st-agr"/>
  <word form="see" macros="@pres @sg-agr @2nd-agr"/>
  <word form="sees" macros="@pres @sg-agr @3rd-agr"/>
  <word form="see" macros="@pres @pl-agr"/>
  <word form="saw" macros="@past"/>
</entry>
<entry stem="sense" pos="V"><!-- not yet semantically understood -->
  <member-of family="PickupDropSeeSense"/>
  <word form="sense" macros="@base"/>
  <word form="sensing" macros="@ng"/>
  <word form="sense" macros="@pres @sg-agr @1st-agr"/>
  <word form="sense" macros="@pres @sg-agr @2nd-agr"/>
  <word form="senses" macros="@pres @sg-agr @3rd-agr"/>
  <word form="sense" macros="@pres @pl-agr"/>
  <word form="sensed" macros="@past"/>
</entry>
<entry stem="find" pos="V">
  <member-of family="PickupDropSeeSense"/>
  <word form="find" macros="@base"/>
  <word form="finding" macros="@ng"/>
  <word form="find" macros="@pres @sg-agr @1st-agr"/>
  <word form="find" macros="@pres @sg-agr @2nd-agr"/>
  <word form="finds" macros="@pres @sg-agr @3rd-agr"/>
  <word form="find" macros="@pres @pl-agr"/>
  <word form="found" macros="@past"/>
</entry>
```

```
<entry stem="search" pos="V">
  <member-of family="TVPred"/>
  <word form="search" macros="@base"/>
  <word form="searching" macros="@ng"/>
  <word form="search" macros="@pres @sg-agr @1st-agr"/>
  <word form="search" macros="@pres @sg-agr @2nd-agr"/>
  <word form="searches" macros="@pres @sg-agr @3rd-agr"/>
  <word form="search" macros="@pres @pl-agr"/>
  <word form="searched" macros="@past"/>
</entry>
<entry stem="tell" pos="V"><!-- not yet semantically understood -->
  <member-of family="DTV"/>
  <word form="tell" macros="@base"/>
  <word form="telling" macros="@ng"/>
  <word form="tell" macros="@pres @sg-agr @1st-agr"/>
  <word form="tell" macros="@pres @sg-agr @2nd-agr"/>
  <word form="tells" macros="@pres @sg-agr @3rd-agr"/>
  <word form="tell" macros="@pres @pl-agr"/>
  <word form="told" macros="@past"/>
</entry>

<!-- Nouns -->
<entry stem="head" pos="N">
  <word form="head" macros="@sg"/>
  <word form="heads" macros="@pl"/>
</entry>
<entry stem="tofu" pos="N">
  <word form="tofu" macros="@sg"/>
  <word form="tofu" macros="@pl"/>
</entry>
<entry stem="data" pos="N">
  <word form="data" macros="@sg"/>
  <word form="data" macros="@pl"/>
</entry>
<entry stem="box" pos="N">
  <word form="box" macros="@sg"/>
  <word form="boxes" macros="@pl"/>
</entry>
<entry stem="kryptonite" pos="N">
  <word form="kryptonite" macros="@sg"/>
  <word form="kryptonite" macros="@pl"/>
</entry>
<entry stem="gold" pos="N">
  <word form="gold" macros="@sg"/>
  <word form="gold" macros="@pl"/>
</entry>
```

```
  <entry stem="battery" pos="N">
    <word form="battery" macros="@sg"/>
    <word form="batteries" macros="@pl"/>
  </entry>

<!-- Wh words -->
  <entry stem="what" pos="WhNP">
    <member-of family="WhNP"/>
  </entry>
```

Lexicon Base: Indicates how words can be used together in sentences, a-well as how the parts of the sentence should be returned in a tree. Poor documentation from OpenCCG made this difficult to work with. The below verb forms were added.

```
<!--Intransitive verb -->
  <xsl:variable name="E.Default">
    <lf>
      <satop nomvar="E">
        <prop name="[*DEFAULT*]"/>
      </satop>
    </lf>
  </xsl:variable>
  <family name="IV" pos="V">
    <entry name="IV">
      <xsl:call-template name="extend">
        <xsl:with-param name="elt" select="xalan:nodeset($iv)/*"/>
        <xsl:with-param name="ext" select="$E.Default"/>
      </xsl:call-template>
    </entry>
  </family>

<!-- Transitive verb + Goal -->
  <xsl:variable name="E.Default.Goal.Y">
    <lf>
      <satop nomvar="E">
        <prop name="[*DEFAULT*]"/>
        <diamond mode="Goal"><nomvar name="Y"/></diamond>
      </satop>
    </lf>
  </xsl:variable>
  <family name="PickupDropSeeSense" pos="V" closed="true">
    <entry name="TV">
      <xsl:call-template name="extend">
        <xsl:with-param name="elt" select="xalan:nodeset($tv)/*"/>
        <xsl:with-param name="ext" select="$E.Default.Goal.Y"/>
```

```
      </xsl:call-template>
    </entry>
  </family>

<!-- Transitive verb + For + Goal -->
  <xsl:variable name="E.Default.For.Goal.Y">
    <lf>
      <satop nomvar="E">
        <prop name="[*DEFAULT*]"/>
        <diamond mode="Goal"><nomvar name="Y"/></diamond>
      </satop>
    </lf>
  </xsl:variable>
  <family name="TVPred" pos="V" closed="true">
    <entry name="TVPred">
      <xsl:call-template name="extend">
        <xsl:with-param name="elt" select="xalan:nodeset($tv.pred.Y)/*"/>
        <xsl:with-param name="ext" select="$E.Default.For.Goal.Y"/>
      </xsl:call-template>
    </entry>
  </family>

<!-- DiTransitive verb + Beneficiary + Goal -->
  <xsl:variable name="E.Default.Goal.Y.Beneficiary.Z">
    <lf>
      <satop nomvar="E">
        <prop name="[*DEFAULT*]"/>
        <diamond mode="Goal"><nomvar name="Y"/></diamond>
        <diamond mode="Beneficiary"><nomvar name="Z"/></diamond>
      </satop>
    </lf>
  </xsl:variable>
  <family name="DTV" pos="V" closed="true">
    <entry name="DTV">
      <xsl:call-template name="extend">
        <xsl:with-param name="elt" select="xalan:nodeset($dtv)/*"/>
        <xsl:with-param name="ext"
select="$E.Default.Goal.Y.Beneficiary.Z"/>
      </xsl:call-template>
    </entry>
  </family>
```

References

White, Michael. OpenCCG. Online at http://openccg.sourceforge.net/

Russell, Stuart and Norvig, Peter. Artificial Intelligence, a Modern Approach, Second Edition. Prentice Hall, 2003.