CS399 Project Report

Using Semantic Networks for Question-Answering

Ido Milstein ido@cs.stanford.edu
Magnus Sandberg sandberg@cs.stanford.edu

1 Introduction

In this report, we describe the design of a question-answering system based on a structured semantic representation of the content of a text. Specifically, content is modeled using a semantic network, incorporating both logical/relational and probabilistic/associative information. The system converts sentences of natural language into sub-networks (semantic parse trees) and inserts them into the semantic network one by one. The network updates itself incrementally, matching new information with old and learning association strengths. Once the network has been fully constructed, questions about the text can be answered by localizing the appropriate information in the network and converting it back to a natural language representation.

2 Background

2.1 Question Answering

The task of traditional information retrieval (IR) systems is to find documents (in a database, on the Web, etc.) that contain some particular information needed by the user. Subject categories and/or keywords are frequently used to specify a query. The unit of answer is usually an entire document.

Question-answering systems, in contrast, allow queries in the form of questions posed in a natural language such as English. Moreover, a question-answering (QA) system is expected to generate a concise, well-formed answer, i.e. a word, phrase, or sentence containing the desired information. This task is, not surprisingly, considerably more difficult than traditional IR, and full-fledged QA is probably not feasible given the current state of natural language processing [Schwitter et al., 2000].

A somewhat easier task is *answer extraction* (AE). An system for answer extraction, unlike a QA system, does not generate it own answers, but rather attempts to retrieve the sentence or paragraph that is most relevant to answering the query. Hence, AE differs from traditional IR in that the unit of answer is much smaller.

It has been suggested that QA/AE type of systems can be evaluated using standardized reading comprehension (RC) tests, as used in schools to evaluate children's reading skills [Hirschman et al., 1999]. This is essentially a subset of the problem of *story understanding*, a long-time research focus of the AI community. As noted in [Riloff, 1999], much of the recent work in this area relies on fairly shallow information extraction (IE) techniques such as pattern matching and discourse analysis. Two recent, experimental AE systems, DeepRead [Hirschman et al., 1999] and Qanda [Breck, 2000], are essentially of this kind, not using any structured internal text representation. DeepRead, which uses bag-of-words techniques and some additional linguistic processing, is in fact presented as a baseline system to which other systems can be compared. We thought it would be interesting to see how well we could match DeepRead's results with a system that attempts to understand a story more in full, using a structured representation of the text it has been presented with.

Our goal has thus been to design a reading comprehension/question answering system capable of deep semantic analysis of short texts (but also of adequately dealing with larger amounts of data). The system should keep a rich enough representation of a text to make it possible, at least in theory, to answer arbitrary questions about its content that do not require inferential reasoning or access to information external to the text. While this is clearly an ambitious goal, we feel that the tools available for syntactic analysis today are good enough to make an attempt at improved semantic analysis feasible.

2.2 Modeling Semantics

It is fairly clear what the main components of a reasonably fine-grained semantic representation (at the symbolic level) should be. We need *symbols* representing things such as *concepts* (abstract objects), *object instances* (or *instances* for short), and *events*. We also want to represent *relations* between symbols, e.g. the relation of an instance I being the agent of an event E: $E \xrightarrow{\text{agent}} I$. The representation we have chosen is that of a *semantic network*, a directed graph with nodes representing symbols and edges representing relations. Throughout this paper, the terms symbol/node and relation/edge will be used interchangeably.

This part of the semantic model, the *structural* aspect, could be easily, though perhaps less intuitively, captured by some form of predicate logic (and indeed, logical models of semantics abound). However, our model also incorporates a probabilistic/associative aspect: nodes and relations are quantified by associating with each of them *activation levels* and *weights*, respectively. A semantic network lends itself in a natural way to an *association flow* mechanism, where symbols activate or inhibit each other depending on the weight and sign of their connecting relation. (In this context, relations will sometimes be referred to as *associations* and their weights as *association strengths*; relations with positive weights are *excitatory* associations, whereas those with negative weights are *inhibitory* associations.) This mechanism can be used to find information in the network and generalize on existing knowledge. In addition, letting node activation decay slowly creates a simple *attention* mechanism, whereby things recently discussed are treated as more likely to come up again in new sentences. Finally, the relation weights can, given enough data, learn to encode statistical information about the relationships between words/symbols. This learning forms the basis of a *relevance* mechanism for determining the relative importance of relations from a given node.

We believe that semantic networks are the natural framework in which to explore these mechanisms and investigate their utility. Semantic networks are, perhaps not surprisingly, frequently used to model language and semantics (for a review, see [Lehmann, 1992]); prominent examples include SNePS, MindNet, and WordNet. WordNet is actually more of a lexical model than a full semantic model, and so is not directly relevant to this discussion; however, as mentioned in section 3.4, we have found good use for it as a component of our system. SNePS, described in [Lehmann, 1992], uses a structural network model not unlike our own, also with story understanding as a key objective. In contrast to our system, however, the focus of SNePS is not on quantitative or "fuzzy" knowledge representation but rather on causal/logical representation and reasoning.

A more probabilistic approach is taken by the people behind Microsoft's MindNet. MindNet uses a rich and seemingly powerful semantic representation, and the system simultaneously learns new concepts/relations and gathers statistical information about them by parsing large amounts of dictionary text. Unfortunately, given the sparse documentation available (see for instance [Richardson et al., 1999]), it is hard to determine if and how the system could be used for story understanding, or indeed how it is to be used at all. Nevertheless, MindNet is probably the work that in spirit most resembles what we are trying to do.

It should also be mentioned that many connectionist network models exist for representing semantic information on a sub-symbolic level. This is however a very different game from ours, and the two approaches cannot be readily compared. All in all, we feel that our system takes a middle road between logical, statistical, and connectionist models; hopefully we are using the best from each world, and not the other way around.

3 Design

3.1 Design Overview

The basic operation of the system is as follows. For each sentence in a text, do:

1. Pre-process the sentence to convert it into a format acceptable by the parser.
2. Parse the sentence to obtain a parse tree of its phrasal constituents.
3. Post-process the parse tree, using additional information (if any) about the sentence.
4. Convert the syntactic parse tree into a semantic sub-network (subnet).

5. Identify symbols in the subnet with symbols in the network, if possible.

6. Insert those symbols into the network that were not there before.

7. Update the relevant association strengths.

For queries, the procedure is the same except that item 6 is replaced by 6.5: generate an answer.

The system is divided into two main components: *layer 1*, which is responsible for items 1 through 4 (and 6.5), and *layer 2*, which covers items 5 through 7. Thus, layer 1 converts NL sentences into our internal representation and back, whereas layer 2 is the knowledge base of the system, where information is learned, stored, and retrieved.

3.2 Pre-Processing

The parser we use, described in the next section, is not terribly restrictive when it comes to input sentence format, since it provides its own dictionary and lexer. This makes the job of the pre-processor much easier. However, there is still the issue of splitting a text into sentences, which is not entirely trivial. For the reading comprehension data described in section 4 this is done once for all by a Perl script, which, while being fairly general-purpose, is to some extent tuned to the precise nature of this data. In particular, it parses and extracts the information provided by XML tags in the texts.

3.3 Parsing

To parse sentences we use the Link Grammar Parser developed at CMU (see [Sleator and Temperley, 1991]). The Link Grammar Parser (LGP) produces two kinds of output: a *linkage* and a *constituent tree*. For example, the sentence "He bought a big estate in South Carolina" would come out as shown in figures 1-2. What is noticeable in figure 1 is that the parser usually tags only noun, verbs, and adjectives with their

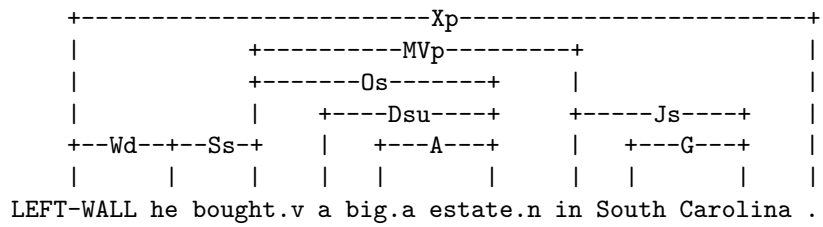

Figure 1: LGP linkage example

respective parts of speech (these are the subscripts appended to some of the words). On the other hand, the link structure provides a lot of information about the sentence. For example, the Ss and Os link tells us that "he" is the subject and "estate" the object of the sentence, both in singular, and that "bought" is the main verb. For complex sentences the parser usually generates more than one linkage (and corresponding parse trees), which are then scored according to a cost function. If it is unable to find a complete parse it will try to provide an incomplete one by eliminating words from the sentence. This feature has not been fully explored, but should prove useful in getting as much information out of a text as possible.

The plain text of a sentence together with a candidate parse is called a *literal*. A sentence can have any number of literal representations, just as it can have any number of semantic (subnet) representations. In both cases, the different representations can be weighted according to some probability measure. For a list of literals generated by the parser, these probabilities are calculated based on the cost assigned by the parser to each one of them.

3.4 The Cross-Reference Structure

A key function of layer 1 is to create and maintain links between symbols in the network and corresponding textual representation. The means for doing this are *cross-reference entries*. A cross-reference entry connects

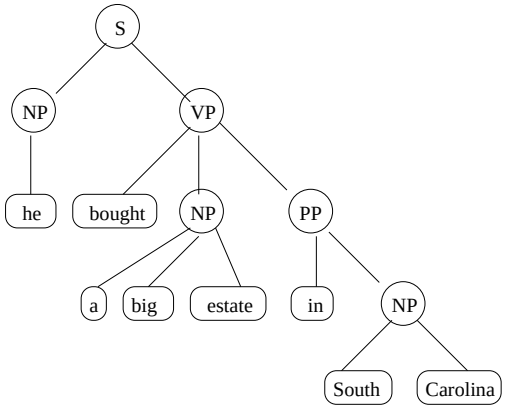

Figure 2: LGP parse tree example

one or several nodes in the network to one or several sub-parse-trees (i.e., nodes in parse trees), which are then in turn linked to sentence literals. In addition, a cross-reference connecting to network nodes that directly correspond to a word (*word nodes*) will point to a *dictionary entry* giving pertinent information about that word, such as its part of speech and various inflections. The dictionary grows continually as the system is used, using WordNet [Fellbaum, 1998] to find information about new words. A hash-based *word lookup table* is used to find the cross-reference entries of words encountered in a sentence, if such entries exist. The design is summarized in figure 3.

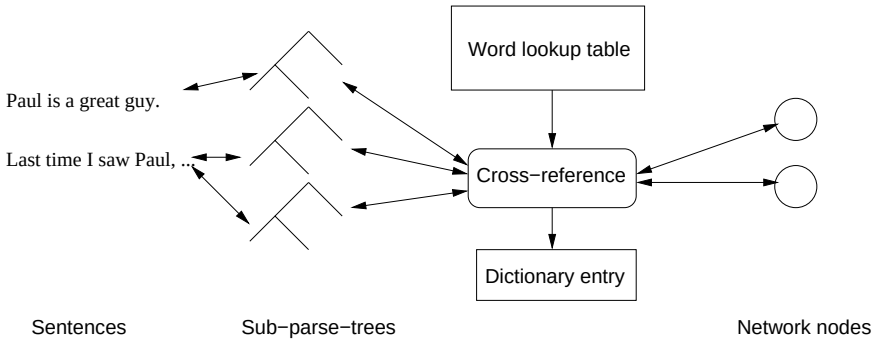

Figure 3: The cross-reference structure

It is perhaps obvious that a cross-reference will frequently refer to more than one literal, since content words tend to be repeated in a text. In addition, cross-references are used to keep track of coreferences within a text, e.g. pronouns referring to nouns (see section 3.5). However, it is sometimes also necessary that a cross-reference can refer to more than one node in the network. This occurs when two or more instance nodes represent objects with the same name, or objects that can be labeled in the same way. For example, in figure 3, we can imagine there being two individuals named "Paul", each represented by a separate node in the network, and each referred to from one or several places in the text.

Just as is the case with different representations of a sentence, the literal and semantic representations pointed to by a cross-reference can be probability-weighted. Suppose, for instance, that there are two Pauls in the network, and that it has been determined that 75% of the occurrences of the word "Paul" in the text refer to Paul 1. This would then be be reflected in the cross-reference entry, so that when querying the network for something Paul-related it can be done as two separate queries, activating either the Paul 1 or the Paul 2 node. The answers can then be weighted accordingly.

Regarding the word lookup table, it should be noted that rather than mapping from words to cross-

references it maps to *lists* of cross-references. The reason is of course that a word may have several senses. In its current form, the system can distinguish between different senses only inasmuch it is possible from inflections or parser output to infer that they are different parts of speech. There is nothing to prevent more advanced word-sense disambiguation to be added, however.

One final point is that the parse trees of individual sentences are not meant to be stored indefinitely. When dealing with large amounts of data, they will be periodically discarded. However, the dictionary entries will be kept, and so will any cross-reference that has a dictionary entry associated with it.

3.5 Post-Processing

Before a parse tree can be converted into a subnet, it must be modified slightly in order for the conversion to work properly.

First, the parses splits tokens such as "Paul's" and "he'll" into separate words – "Paul 's", "he 'll". In the first case, assuming this is a genitive and not "Paul is", the "'s" node will be removed from the parse tree and the "Paul" node marked as genitive. In the second case, "'ll" will be expanded into "will".

Second, if there is separate information available about names appearing in the sentence (as is the case with the RC data), this will be used. For example, given the tag

```
<ENAMEX TYPE="PERSON">South Carolina</ENAMEX>,
```

the nodes for "South" and "Carolina" will be merged into one, marked as a location.

Third, any coreference information available will be used. For example, consider the following (simplified) markup of two sentences:

```
<COREF ID="70"><ENAMEX TYPE="PERSON">Christopher Robin</ENAMEX></COREF> is alive and well.
<COREF ID="78" REF="70">He</COREF> lives in England.
```

In this case, a cross-reference will be created between the "Christopher Robin" node in the first parse tree and the "he" node in the second. (Once the sentences have been inserted into the network, the cross-reference will also contain a link to the Christopher Robin instance node.)

3.6 Semantic Model: Network Structure

In spite of – or perhaps because of – endless discussions, the semantic model has so far been only incompletely and tentatively specified. While our ideas extend beyond what has already been implemented, it is likely that the final model will be changed many times yet, based on what we discover about the strengths and shortcomings of the query and localization mechanism. For this reason, it seems pointless at the moment to try to give a complete account of an incomplete model.

The following are the symbol types currently used:

concept An abstract, general entity; a class of objects.

instance A concrete, specific entity; an instance of a class.

event Something happening, something with a temporal span (which need not be momentary, but can be arbitrarily long).

action Something performed by an agent; a verb.

attribute A property or quality of something.

location Something with a spatial extent. We distinguish between an object (such as a house) and its role as a location.

modifier Something which modifies the meaning of something else.

In addition, concept and instance nodes are classified as being things, persons, places, organizations, etc. Moreover, nodes in a subnet (but not in the network) can be named *query nodes* (symbolized by a "?"), meaning that they represent information that is being requested in a query.

The following relation types are used:

isa Represents a subclass-superclass relation: *apple* $\xrightarrow{\text{isa}}$ *fruit*. Also used for attributes: *dark blue* $\xrightarrow{\text{isa}}$ *blue*.

instance The relation between an instance and its class: *the crazy cat* $\xrightarrow{\text{instance}}$ *cat*.

description A relation from a symbol to another that serves to describe the first one in some way: *the crazy cat* $\xrightarrow{\text{description}}$ *crazy*, *apple* $\xrightarrow{\text{description}}$ *round*, *brick house* $\xrightarrow{\text{description}}$ *brick*, *dark blue* $\xrightarrow{\text{description}}$ *dark*.

agent The relation between an event and its main actor.

action The relation between an event and the action taking place.

object A relation between an event and an object (in a grammatical sense) involved.

location A relation between an event and the place where it took place.

equivalent Signifies that two instances are really one and the same.

In addition, a relation can be *defining*, meaning that it represents a defining property of the symbol it extends from. For example, *apple* $\xrightarrow{\text{isa}}$ *fruit* would typically be a defining property, whereas *apple* $\xrightarrow{\text{isa}}$ *throwable-thing* would not. Of course, this depends to some extent on the context.

Negative relations, i.e. relations with a negative weight, fall outside this classification. A negative relation has no type; it has no meaningful interpretation, but serves only to prevent activation from spreading to irrelevant parts of the network (see section 3.8).

Using these building blocks, we can model for instance the question "Where did the big cat eat the tasty mouse?", as seen in figure 4. Needless to say there are also many things that can *not* be modeled, notably temporal and causal relations, ownership, and cardinality. We feel confident that there are ways to represent this kind of information as well; again, the problem is perhaps more that there are too many ways of doing it...

3.7 Converting to Semantic Representation

The core functionality of layer 1 is to convert a post-processed parse tree to a subnet adhering to the semantic model outlined in section 3.6. This is done in a recursive fashion, starting with the leaves of the parse tree and moving up toward the root.

A node in the parse tree may be represented by zero, one, or several nodes in the subnet. At the leaf level, as a rule of thumb, content words will have nodes created for them whereas function words will not. On the other hand, function words play an important role in determining what should happen at the next higher level in the tree, so they are kept around as "ghost nodes" for as long as they're needed. It is however also possible to entirely disregard a portion of the parse tree that is deemed to be irrelevant or unrecognizable.

As a side point, a subnet is not necessarily a tree. If a content word occurs several times in a sentence there will still be only one node created for it in the subnet.

The main steps in processing a leaf node are as follows:

1. Look up the word in the word lookup table (or, if it's not there, in WordNet), to obtain a) its base form and b) its cross-reference entry (if any).
2. Check if the subnet already contains a node for this word (its base form, really).
3. If not, determine if a node should be created and what its type should be. Create the node, if appropriate.
4. Determine additional context-sensitive information.

5. Check the cross-reference to see if there is a node for this word in the network.
6. If yes, add the node to the subnet's list of *known nodes* (i.e., nodes that have an identified counterpart in the network).
7. If no, add it to the list of *unknown nodes*.
8. Update the cross-reference and dictionary entries as needed.

For non-leaves, the course of action will be determined largely by what phrase type the parse tree node represents (noun phrase, verb phrase, and so on) and what children nodes have been created at the next-lower level. There are two typical situations, the first one being that the node is bypassed altogether, with no new subnet node being created, and the children simply passed on to the next level. This is for instance the default treatment for verb phrases, since we probably want to connect both the predicate and object parts to an event node at a higher level. The second situation is where a new subnet node *is* created and connected to all its (non-ghost) children with relations of the appropriate type. An example would be the event node that is typically created at the root level of the parse tree.

More complicated situations also arise. In some cases, we may not want to create a new node, but instead connect some of the children directly to one another. This holds for a sentence such as "Paul is a dog", where no event node will be created at the root level. In other cases, such as when dealing with compound nouns, we may want to create several new nodes and chain them with the children nodes. This latter case is currently not handled by the system.

The treatment of queries is almost identical to that of ordinary sentences (but special rules are of course needed to handle typical question syntax). For a typical "W" question an instance node will be created for the question word (who, what, where, ...), made into a query node, and then treated as any other instance node, playing the part of agent, object, or what not. This results in a subnet just like any other; again, figure 4 provides a good example.

Once the process is completed, what we have is a (hopefully) connected graph which is a semantic representation of the original sentence. The subnet will not be directly connected to the network, but there will be implicit links between the "known" nodes and nodes in the network. For unknown nodes, layer 1 will make an educated guess (well, ok, a pretty dumb guess) as to whether they can be expected to be in the network after all, or not, based on e.g. how "a something" and "the something" are used in English to signify new versus afore-mentioned objects. To actually decide the issue, however, and to find the relevant nodes in the network, is the job of layer 2, and is the story of the next few sections.

3.8 Semantic Model: Network Dynamics

3.8.1 Probabilistic Model

The basic functionality of our semantic network is to receive from layer 1 sub-networks that are translations of natural language sentences and parts thereof, and find possible matches between the newly received sub-network, and the previously seen ones. Here is an example of such a process that will give us a closer look at the problem. Assume that we have heard that "The big cat ate a tasty mouse in the kitchen". At a later point in time we receive from layer 1 the sub-network that is shown in figure 4. This sub-network was produced from the sentence "Where did the big cat eat the tasty mouse?". The nodes "cat", "big", "eat","tasty", and "mouse" have been recognized by layer 1 to be the same as the ones we saw in the previous sentence.

One way to solve the problem of matching the missing nodes (the ones marked instance, event and '?'), is to first mark the first instance node as being "the big cat",and to mark the other instance node as "a tasty mouse". After this we can realize that "the big cat", "eat", and "a tasty mouse" form together the 'event of a big cat eating a tasty mouse. Finally we can go on to identify the location node "?" as the location in which "the big cat ate the tasty mouse" which corresponds to the previously seen "kitchen" node.

Now that we have seen an example of the problem of locating parts of sub-networks in other previously seen sub-networks, we would like to place it in a probabilistic framework.

We look at the space of all possible sub-networks that are translations of natural language sentences into semantic representations, as described in section 3.7. We assume that each node in these networks is

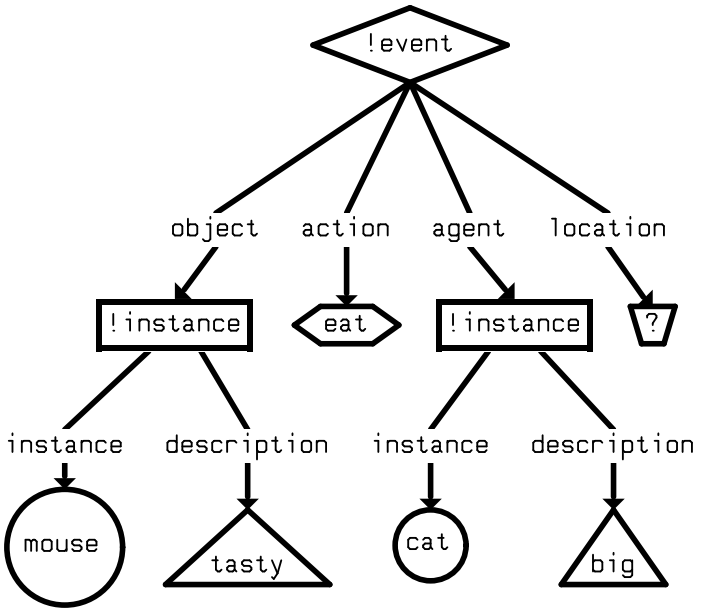

Figure 4: Where did the big cat eat the tasty mouse?

a reference to something (an object, an event, etc.) in a reference world (the "real world"), and will from now on use the terms node/reference interchangeably. We assume that these sub-networks (including the references of the nodes) appear according to some probability distribution.

In this context, the general problem is – given a sub-network structure, and a partial set of references from nodes to the reference world, find the most probable reference assignments to the rest of the nodes. The variant on this general problem in our case is – given a sub-network structure, and a partial set of references from nodes to the reference world, find the most probable reference assignments to the rest of the nodes from references that you have already encountered and from the possibility of having some references that were not previously encountered.

This problem requires us to hold a probability distribution for all nodes given all possible sub-networks that they may appear in, or at least do this for the nodes that have already been encountered, in the context of the sub-networks that the nodes have been encountered in. Trivial combinatoric calculations show that in order to store an arbitrary joint probability distribution of N random binary variables (for N nodes) we need $2^N - 1$ parameters (and this is for a single given sub-network). Yet, our network holds $O(N^2)$ parameters that are used for all given sub-networks (2 weights for each edge, a few parameters to store its type, and a few parameters to describe each node).

This last calculation shows that we are making some fairly strong assumptions that let us reduce the number of variables that we use to describe the distribution. The first, and most important assumption, is that when references appear in different sub-networks they appear in similar *local* contexts. These contexts are similar in both the other references that appear in them, and the relations that connect them. For example we assume that "the big cat" that appears in the sentence "Where did the big cat eat the tasty mouse?", as a reference to some specific cat, will appear commonly with the very same relations to the very same nodes (references) "cat" and "big" (which are referencing the concepts of cats and being big in the "real world"). This allows us to keep local probability distributions, instead of global joints, and to not need to keep separate distributions per sub-network that we have seen. We took this assumption to the extreme of maintaining for each node a distribution that depends only on its immediate neighbors in the sub-networks that we've seen, and, as will be described later, on its "step parents" – nodes that are connected to it via a path that has 2 intermediate nodes.

Our second assumption is that these local probability distributions can be approximated by a simple

model. If a node depends on N neighbors we would normally need $2^N - 1$ parameters to describe its probability distribution, and instead our model uses only N parameters.

In order to describe our local probability models, we would like to define another term. Given a sub-network S, with some missing references, and a reference A, we define a binary random variable with the same name A that has the value 1 iff A is a reference of a node whose reference in S is unknown. We use $P(A|S)$ as a short for $P(A = 1|S)$ and $P(\overline{A}|S)$ for $P(A = 0|S)$. So, in the above example, the reference for "the big cat" that appeared in the "The big cat ate..." sentence has a value of 1 for the sub-network of "Where did the big ...?", while the reference to "cat" that appeared in the same sentence has a value of 0 for the question sub-network, because it does not stand for any *unknown* reference.

Our local probability distribution for a node assumes that its neighbors are either excitatory or inhibitory in nature. So, if A is an excitatory node for X, B is an inhibitory node for X and Z is a set of other known nodes, then

$$\forall_Z P(X = 1|A = 1, Z) \geq P(X|Z) \quad \forall_Z P(X = 1|B = 1, Z) \leq P(X|Z)$$

Moreover, we treat each of these excitatory inputs as a cause, and each inhibitory input as a counter-cause so that if A_i are the excitatory inputs of X, and B_j are the inhibitory inputs, then

$$X = 1 \iff (\exists_i A_i = 1 \wedge \forall_j B_j = 0).$$

This seems to us like a reasonable assumption to make in the context of sub-networks of natural language, because nodes are based on an aggregation and decomposition idea, where every part of a whole makes the whole more probable (and vice versa), and each other non-related part makes it less probable (and again vice versa). For example having a reference to "cat" in a sub-network always makes the reference to "big cat" more likely to appear too, and having a reference to "elephant" makes it less likely. The way we treat it, for "elephant" to have an excitatory effect on "big cat" there would need to be some intermediate excitatory link that would overcome the immediate inhibitory effect of "elephant" on "big cat".

If we were to draw our node model, it would look like this:

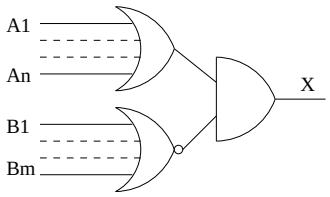

Figure 5: A schematic representation of a node

One way to take the same idea into a continuous range of values is to have

$$P(X = 1|A_1, \ldots, A_n, B_1, \ldots, B_m) =$$

$$= [1 - \prod_{i=1}^{n}(1 - P(X|A_i = 1, 0, \ldots, 0)I_{[A_i=1]})] \prod_{j=1}^{m}(1 - P(\overline{X}|B_j = 1, 0, \ldots, 0)I_{[B_j=1]})$$

What this means is that each excitatory input has a chance $P(X|A_i = 1, 0, \ldots, 0)$ to excite X, and each inhibitory input has a chance $P(\overline{X}|B_j = 1, 0, \ldots, 0)$ to inhibit it, and X is on if at least one excitatory input has excited it, and no inhibitory input has inhibited it. Trivially, if these probabilities are all one we return to the deterministic gate depicted above. This can also be looked at as a noisy OR and a noisy NOR combined with a deterministic AND to produce the value of X. Notice that this already requires us to use all of our edge parameters (remember that every edge has 2 parameters, one for each direction). Another thing we should point out is that if A excites X then X excites A, and the same is true for inhibitors (all this is very simple to prove). So we can refer to the edges themselves as inhibitory or excitatory in nature, and the strengths of these $A \to X$ edges would represent $P(X|A_i = 1, 0, \ldots, 0)$ for excitatory links or $P(\overline{X}|B_j = 1, 0, \ldots, 0)$ for inhibitory links.

One may immediately point out that OR functions do not cover the basic AND function that we use when deciding if a "bell" + a "boy" is a "bell boy". The effects of the disjoint words "bell" and "boy" on the compound noun "bell boy" is not a disjoint effect. This compromise between AND and OR is a compromise that we have to make only if we wish to keep the number of variables at that, but one can always pick a more complicated node model, and keep around more variables to support it. One such variation we had in mind was to add as inputs to the ORs selectively picked ANDs as additional excitatory input for compound nouns and for events. This would add one additional parameter of the sort $(1 - P(X|A_{i_1} = 1, A_{i_2}, 0, \ldots, 0)I_{[A_{i_1}=1, A_{i_2}=1]})$ to the first product, and would make the learning rules that will soon be described just a little more complicated.

A slightly different approach we intended to implement and compare with our current node model is as follows. Let each edge hold 2 weights per direction, and let the strengths of the edges represent $P(X|A_i = 1)$ and $P(X|A_i = 0)$ rather than $P(X|A_i = 1, 0, \ldots, 0)$. We can then look at each input edge as an "expert", and keep for every node a (learned) function that represents the current best estimate for the value of the node, given the state of the "experts". For example one simple function of this sort can be a weighted average between the experts so that

$$P(X = 1|A_1, \ldots, A_n) = g(\theta + \sum_{i=1}^{n} \alpha_i (P(X|A_i = 1)I_{[A_i=1]} + P(X|A_i = 0)I_{[A_i=0]}))$$

(note that in this formulation excitatory and inhibitory relations differ just by the relative sizes of $P(X|A_i = 1)$ and $P(X|A_i = 0)$.)

3.8.2 Propagation

Now that we have described our network model and how it represents local probability distributions, we have to describe how we use the information about known references that are given with a sub-network, and the sub-network structure, in order to condition our probabilities on them. This section will focus on using the known references. Section 3.9 describes how the structural information is used.

Instead of solving the problem of conditioning the whole joint probability of possible node (reference) assignments on the given nodes, we decided to take a naive approach, calculating the node probabilities independently. We had three main ideas in mind. The first idea was to either sample the network (or a local area within it) several times, and group the results into main trajectories or just take their average (and by this get a whole joint). The second idea was to give the nodes some random instantiation, and then repeatedly select nodes and re-sample them given all the other nodes. This would be something like the Gibbs algorithm, and would give us a likely trajectory of the joint. The third approach, which is the one we selected, was to keep for each node the probability that it is "on", and to use a local naive assumption when calculating the probability of a new node. This means that each node looks at the probabilities of each of its neighbors, and assumes that they are all independent random variables. We then repeatedly calculate the probabilities of the separate nodes given their neighbors. This translates simply to

$$P(X^{t+1} = 1|Z) =$$

$$= [1 - \prod_{i=1}^{n}(1 - P(X^{t+1}|A_i^t = 1, 0, \ldots, 0)P(A_i^t = 1|Z))] \prod_{j=1}^{m}(1 - P(\overline{X^{t+1}}|B_j^t = 1, 0, \ldots, 0)P(B_j^t = 1|Z)) =$$

$$= [1 - \prod_{i=1}^{n}(1 - W(A_i \to X)P(A_i^t = 1|Z))] \prod_{j=1}^{m}(1 - W(B_j \to X)P(B_j^t = 1|Z)) =$$

where, as usual, A_i are the excitatory neighbors of X, and B_j are the inhibitory neighbors of X.

One problem that one quickly runs into is the problem of positive feedback loops. We are interested in undirected loop-less paths from the given nodes T to the other nodes. However, even when we have only 2 nodes interconnected by an excitatory link propagating back and forth between them would result in a loop. We are avoiding it in the 2 node case by propagating from X to A $P(X^{t+1} = 1|Z, A^t = 0)$, so the only part that is propagated to A is the part that A was not responsible for. This number will also turn out to be

useful in the section 3.8.3 about weight learning. This does not solve the problem of cycles of length 3 or more, however, and careful analysis of these will have to be performed on large networks. We should point out that we expect most weights to be significantly smaller than one; assuming we have 3 weights of strength 0.5 the total cycle is of weight 0.5^3, and we get a total increase over an infinite amount of propagations of

$$\sum_{i=1}^{\infty}(\frac{1}{8})^i = \frac{1}{7}$$

We also introduced a decay factor of 0.98 for every propagation to further decrease these effects.

The selection of the nodes to propagate into is done by maintaining a queue of expected change in $P(X = 1|Z)$. Whenever we propagate into a node A we calculate the value $P(A = 1|Z)$, and after that propagate into each relation $X \leftrightarrow A$ the value of $P(A = 1|Z, X = 0)$ as described above. The sum of the absolute differences in $P(A = 1|Z, X = 0)$ for all the relations of X is used as an indicator for the expected change in $P(X = 1|Z)$.

The advantage of the method we chose over the two other methods we considered is that it is fairly simple, and, more importantly, given that the huge distribution is represented by a fairly small number of variables, almost all trajectories of the other approaches would be inconsistent with the given sub-network. Consistent trajectories can only have an amount of "on" nodes that is smaller or equal to the amount of unknowns in the sub-network. Our approach lets us get a comparison measure between nodes that used with other comparison mechanisms described later can let us select and direct the assignment of nodes.

3.8.3 Learning the Weights

Learning probabilities from sparse data is a difficult task. One way to address it is to simply reduce the number of free parameters in the PDF, and by this make the data less sparse in proportion to the number of parameters. This was one of the main goals of the model design that has been described above, and it works in two ways.

First, the total number of learned parameters is reduced to no more than $N*(N-1)$ parameters. Second, and not less important than this, instead of learning probabilities for whole sub-networks the learning focuses on the small parts (i.e. nodes) that are the building blocks of the sub-networks. Phrases like "The big cat ate the small mouse" are not commonly seen in natural language texts, and so estimating the probability of a whole phrase like this is not simple. However, this phrase can teach us about the probability of phrases that deal with cats that eat mice, and about the probability that cats are big, and the probability that mice are small.

Now for the actual process. With no reason to assume otherwise, we assume that a current value of a node is the correct value, and that any other node that is responsible for changing it should pay. Recall that we calculate $P(X^{t+1})$ by

$$P(X^{t+1}) = [1 - \prod_{i=1}^{n}(1 - P(X|A_i = 1, 0, \ldots, 0)P(A_i^t = 1))]\prod_{j=1}^{m}(1 - P(\overline{X}|B_j = 1, 0, \ldots, 0)P(B_j^t = 1))$$

We define the error to be

$$Err \stackrel{def}{=} |P(X^{t+1}) - Val(X^t)|$$

For excitatory relations we get:

$$\frac{\partial \frac{1}{2}Err^2}{\partial P(X^{t+1}|A_k = 1, 0, \ldots, 0)} = Err * \frac{\partial P(X^{t+1})}{\partial P(X^{t+1}|A_k = 1, 0, \ldots, 0)}$$

$$= Err * (-P(A_k^t = 1) * \prod_{i \neq k}(1 - P(X|A_i = 1, 0, \ldots, 0)P(A_i^t = 1))]\prod_{j=1}^{m}(1 - P(\overline{X}|B_j = 1, 0, \ldots, 0)P(B_j^t = 1))$$

$$= -Err * P(A_k^t = 1) * (\prod_{j=1}^{m}(1 - P(\overline{X}|B_j = 1, 0, \ldots, 0)P(B_j^t = 1)) - P(X^{t+1}|A_k^t = 0))$$

For inhibitory relations we get:

$$\frac{\partial \frac{1}{2} Err^2}{\partial P(X^{t+1}|B_k = 1, 0, \ldots, 0)} = Err * \frac{\partial P(X^{t+1})}{\partial P(X^{t+1}|B_k = 1, 0, \ldots, 0)}$$

$$= Err * (-P(B_k^t = 1) * \prod_{i=1}^{n} (1 - P(X|A_i = 1, 0, \ldots, 0) P(A_i^t = 1))] \prod_{j \neq k} (1 - P(\overline{X}|B_j = 1, 0, \ldots, 0) P(B_j^t = 1))$$

$$= -Err * P(A_k^t = 1) * P(X^{t+1}|B_k^t = 0)$$

This is convenient, because we keep around both the $P(X^{t+1}|B_k^t = 0)$ terms and both products (i.e. the $\prod$s) for propagation purposes.

Since then values of nodes tend to be more correct towards the end of the propagation, because by then more knowledge is accumulated, we take bigger steps against the direction of the gradient near the end of the propagation phase, but we only make the actual weight change at the end of the propagation, in order to keep it stable. In addition, the step size gets smaller as a relation matures.

Notice that we assume the value of X at time t to be independent of the weights for the purpose of calculating the gradients. One can also propagate back the errors that are due to the $P(A^t)$ values, but we do not do that.

3.9 Updating and Querying the Network

As seen in section 3.7, the subnets created by layer 1 will contain a list of nodes, some of them known, some of them unknown. The task of layer 2 is to try to move nodes from the "unknown" list to the "known" list by identifying them with existing nodes in the network. Of course, this requires that the list of known nodes is non-empty to begin with, so that there is a starting point for the localization process. If there are no known nodes, the subnet will be left unconnected to the network for the time being.

Updating and querying the network is done almost exactly in the same way. The difference is only that when querying the network we do not copy into the network the parts that we couldn't find in it, and that we may use different priors when locating nodes. The process of locating sub-networks inside the whole network is done as described above on a node by node basis. It consists of two main parts: Finding possible matches between known nodes and unknown nodes, and evaluating and comparing the possible matches to produce a 'match' or a 'no-match' answer and produce a confidence measure for the decision.

The actual steps of the localization process are as follows: We dissect the sub-network that we are trying to locate into 2 separated parts, an inner part and an outer part. The outer part includes all unknown nodes that are connected (via paths that include only unknown nodes) to a single known node, or to zero known nodes. The inner part is made of all other unknown nodes. In the sub-network in figure 4, assuming that the given nodes are "big", "cat", "eat", "tasty", and "mouse", the outer part consists of the singleton "?" and the inner part consists of the two instance nodes and the event node.

We start propagating through the network beginning at the given nodes. Each other node that we propagate into carries also the name of the node that originated the propagation into it. When there is more than one origin we pick the one with the highest activation, and register the node as a possible match for the origin nodes. We then go on propagating for a while waiting to find other possible matches between the same origins. At this point we locate in the sub-network all possible nodes in which a match of the same origin nodes can occur. Now we score each possible pair, and take the best matching pair. If the score of this best matching pair is above some predefined threshold we declare this as a match, and add the nodes to the list of given nodes. A confidence measure for the selection is made of 2 numbers: the score of the best match, and the relative size of it compared to the sum of the scores.

This procedure is repeated until all inner nodes are found, or a maximum number of propagations have passed since the last match was found. All remaining unknown nodes (inner and outer) are joined together for a second location attempt. As possible matches for each one of these nodes we consider nodes in the network that have the same neighbors as the unknown node, connected by the same type of a relation (we allow some variance here but any variance cost is represented in the match score). Each possible match is given a score, and again we pick for each node a best fit, check if it is above some threshold and produce a

confidence measure. We repeat this process until no new good matches are found, or until we have matches for all unknown nodes.

The score for a match between nodes X (in the network) and X' (in sub-network S given nodes Z) would ideally be

$$P(X = X'|S, Z) = P(X = \text{some node } A' \in S|S, Z) * P(A' = X'|S, Z, X) = P(X|S, Z) * P(A' = X'|S, Z, X)$$

We approximate $P(X|S, Z)$ by $P(X|Z)$ (this is a like saying that we approximate the probability of "the big cat", given the template in figure 4 and given the words, with the probability of "the big cat" given only the words "big" "cat", ...). Note that this already includes knowledge from recently seen sentences, as this affects the activation of all nodes.

One possible way to approximate $P(A' = X'|S, Z, X)$ is to say that

$$P(A' = X'|S, Z, X) = \frac{1}{\#\text{unknown nodes in S}}$$

But this is too crude. We approximate it by saying

$$P(A' = X'|S, Z, X) \simeq P(A' = X'|S, Z, Template(X))$$

$Template(X)$ is defined by the structural and type information of X, its relations, and its neighbors. The information that we are discarding in making this approximation is that of the specific instantiation of the template of X. So, for example, the difference as far as template goes between "the big cat" and "the tasty mouse" is minor, and is only represented by the fact that the former is connected to the event by an "agent" type relation, and the later is connected to the event by an "object" type relation. This means that if we were to consider both nodes as possible matches for the real "the big cat" node in the network we wouldn't know which one to pick. However, we are discarding the "tasty mouse" possibility at an earlier stage, when we look for possible nodes that can be reached at the same time by a propagation from "big" and "cat". Another way to look at it is that the actual approximation is

$$P(A' = X'|S, Z, X) = P(A' = X'|S, Z, Template(X), Semantics(X)) \simeq$$
$$\simeq P(A' = X'|S, Z, Template(X)) * I_{[X' \text{ has similar semantics to } X]}$$

The last indicator function could be (but is not currently) replaced by a continuous measure of similarity that is based on the relative location of X to the origin nodes in the network, and the relative location of X' to the same nodes but in the sub-network, and on the proportion of $P(X|Z)$ that is attributed to the different origins.

Unfortunately we don't have a good measure for $P(A' = X'|S, Z, Template(X))$, and not even an automatic way to get it. Instead, we count the number of exact correspondences between the relations of the nodes X and X' (such as having a node A both in the network and the sub-network that connects to both nodes using the same relation type), we check that the nodes themselves are of types that may substitute each other, and we use our personal prior knowledge to give a score to the similarity between the template of X and that of X', based on their node types and the relations they have. One could (and should) simply learn these numbers automatically using some training data. The basic forming blocks of templates are very repetitive, and so it seems to us that it wouldn't require a large training set to be able to approximate them right. One example of such a number would be the probability that the same event is referred to using different action verbs.

One final component that we use to calculate the score of a match is the prior belief passed from layer 1 that a node in the sub-network actually appears in the network. Let us summarize, then, the six different components that are used to calculate a score for a match between nodes X and X'.

- An indicator (0 or 1) that X is of a type that may substitute X' (an event node may for example not substitute an instance node).
- An indicator (0 or 1) that X and X' may have the same origins. This is what distinguishes "the big cat" from "the tasty mouse".

- The current activation of X marks the probability that X is referred to by the sub-network.
- The number of matching relations that X and X' have.
- Prior probabilities that are assigned to the non- matching relations of X and X'.
- Prior knowledge about the probability that X' matches a node in the network.

The final score is 0 if any of the indicators is 0. Out of the other 4 numbers three are probabilities, and one (the number of matching relations) is not. Ideally it would be combined with the fifth number in an automatic way, as suggested above, to form another probability. Simply multiplying all 4 numbers is probably not a good idea because they don't share the same units, and more importantly these different sources of information have different credibilities. So instead we use 4 weights, one for each number, and consider each number to the power of its weight. These weights too could and should be learned automatically, and we use our own knowledge to give them fixed numbers instead. One last additional thing that did about these weights is to assume that the activation rates of nodes get more accurate as the node "matures" (meaning that when we see more sentences that refer to the node we get its probabilities better), and so its weight increases with time.

This is the place to emphasize the importance of the negative, so called *inhibitory* relations, as one of the key participants in directing the search through the network. Channeling the search into areas of high probability requires us not only to activate the correct paths, but also to inhibit the incorrect paths so as not to distract our "attention". Our idea was to use lateral inhibition, so that an active path will inhibit all other parallel paths. This was done by adding an inhibitory relation between every node and its "step parents" – if we look at the network as an undirected graph then these "step parent" nodes inhibited by node B are:

$$Inhibited(B) \stackrel{def}{=} \{node X \neq B | \exists_{C \neq X, D \neq B}\ B \overset{excite}{\longleftrightarrow} C \overset{excite}{\longleftrightarrow} D \overset{excite}{\longleftrightarrow} X, B \overset{excite}{\not\longleftrightarrow} X\}$$

Let's examine the effects of node B on a node X which it inhibits, and let's look at a specific quadruple "B C D X" that satisfies the conditions above. One example for such a quadruple can be the nodes "cat(B)", "the big cat(C)", "big(D)", and an additional node "the big elephant (X)". When only "cat" is instantiated it may either excite or inhibit "elephant", because it has one positive path and one negative path (the actual strengths of the relations will decide). The interesting case is when the node "big" is instantiated. If "cat" is not instantiated at the same time, then both "the big elephant", and "the big cat" get excited. But when "cat" is also excited then "cat" only inhibits "the big elephant" because its excitatory effect is blocked by the instantiation of "big" on the way. So we end up having the effects of "big" and "cat" adding up to activate "the big cat", and destroying each other to keep "the big elephant" disactivated.

3.10 Answer Formation

The first part of answering a query, as described in previous sections, consists of setting up a subnet with a query node and then trying to identify it with an existing node in the network. Given such a node N, the second part is to generate a natural language response based on it. There are two principal ways in which this can be done. One way is to trace a subtree in the network, rooted at N, which adequately describes the symbol represented by N. For example, if N is an instance node, we will want to look for defining attributes of this object instance. The leaves of the subtree can be assigned words as given by the cross-reference/dictionary information, words which can then be assembled into an answer phrase.

While this would certainly be the "best" way to solve the problem, and the only one feasible if a large network is to be built, it is far from trivial in practice to find the right subtree, add proper word inflections, and formulate a grammatically correct answer. For the present system we have chosen a different approach. We know that through the cross-reference structure, N will be linked to at least one sub-parse-tree (if there are several we choose the largest one, on the assumption that it is the most descriptive). Moreover, it is easy to generate an answer from this subtree: simply traverse the tree and print the leaves in left-to-right order. For example, given the sentence "The mouse killed the mad cat" and the query "Who did the mouse kill?",

the system will respond "the mad cat" since the subtree corresponding to the cat instance node is precisely that noun phrase.

This solution should be adequate in most cases for reasonably straightforward questions in a reading comprehension setting.

4 Test Data and Evaluation

This section will be rather short, since the system is currently not able to process the test data it was originally intended to be used with. Suffice it to say that the prepared data consists of roughly 100 short stories accompanied by five questions each plus answer keys. In addition, the stories have been tagged (though not by us) with name and coreference information, as well as with information on what sentences are most relevant to answer the various questions. This data was used by Hirschman et al. to evaluate the DeepRead system, and our intention has been (and still is) to test our system on the same data and with the same performance measures. A scoring mechanism is in place for computing precision and recall, possibly averaged over several competing answers. Some care has been taken to use, for example, the same list of stop words as that used by Hirschman et al. Now if we only had some results...

5 Results

Except for what has already been demonstrated live, we don't really have any results to show. Given that the system is not yet in a state where it can deal with sentences of normal complexity reasonably well, it is hardly possible to do any meaningful large-scale testing. Unfortunately, the absence of quantitative results means that we have been unable to accomplish what we set out to do: analyze and evaluate the usefulness of our network model. On a more positive note, we have not encountered anything in the course of the project that has lead us to believe that our approach to semantics is impractical or impossible.

Being confined to a very limited set of "convertible" sentences, and being thus forced to write our own test sentences for the network, put a tight limit to the amount of testing we could do on the network. As far as locating small sub-networks in a small network, it seems that our system was able to make the correct matches most of the time. One nice example of this is that after being taught the sentences "Bully is a dog" and "the dog is big" it recognized that "Bully is big". Most cases where it failed were cases that we didn't finish implementing solutions for (like being able to make simple inferences from concepts to instances). Of course, this nice example is quite meaningless considering the goals we set for the system. Moreover, when the nodes in the network are all young, the network makes decisions that are based mainly on our hand crafted priors, described in section 3.9, which represent the structural information that is known rather than the association strengths. The largest networks that we built were made of approximately 20 simple sentences, and did not allow for any inspection of the weight learning and use, other than letting us see that the inhibitory relations seemed to effectively shut off irrelevant paths. The main interesting question about the network – how it will behave given hundreds or thousands of sentences – still remains.

6 Discussion

Since the project is still unfinished, and considering also the great improvements it made with every additional day of coding, we feel that a discussion of the results (what results?) is too early at this point. It seems more meaningful to discuss the possible problems that we may run into in the future, and what improvements will have be made on the basic design.

The next step that we need to pass, in order to be able to deal with the given set of reading comprehension tests, is clearly to extend the number of possible syntactic structures that layer 1 can handle, and extend the semantic model as necessary to deal with more complicated relationships. This requires work in the subnet conversion module as well as adjustment of network priors to the new possibilities that will arise. Both processes can probably be made simpler by building a parser to automatically read conversion rules and associated priors. The system could also be made more modular, and less dependent on any particular

parser, by building a stand-alone "semantic parser" that accepts as input syntactic parse trees in some general format – XML, say – and generates semantic sub-networks in the same format.

While converting parse trees to semantic subnets turned out to be a more intricate problem than we originally envisioned, it appears to be essentially solvable, given an enough powerful set of conversion rules. By essentially we mean that the ambiguity and vagueness inherent in natural language would sometimes make it hard or impossible to obtain an exact semantic representation of a sentence, regardless of how many rules we had at our disposal. However, it seems likely that it would more often than not be possible to get at least a half-descent semantic parse. Also, one benefit of our network model, as we see it, is that it should be less vulnerable to anomalies in the data than a typical logical model.

Once this goal is achieved we should be able to try the system on the RC stories and get an initial performance evaluation. We should be able to get an evaluation of the system's absolute performance, using common measures of precision and recall, but also be able to examine how the system's performance improves, hopefully, as it sees more data and learns more on the basic probabilities of English phrases. This should also enable us to look more closely at the way the weight learning and propagation mechanisms work on medium-scale data, and possibly try out some of the other approaches for weight learning and propagation that were discussed before. It would also be important, at that point, to examine which of the simplifying assumptions that we made regarding the probabilistic model are reasonable and which ones are not.

If the network model holds up, it would then be necessary to expand the capabilities of layer 1 in order to be able to work with more general types of data. For instance, instead of relying on tagged data the system would need to do its own name identification and pronoun resolution. There are of course existing tools for this; however, especially in the case of pronoun resolution, some kind of interaction between the layers would probably be helpful. After all, resolving cross-references is at the heart of the network localization problem. Other possibilities include using statistical tools for word-sense disambiguation (again, layer 2 knowledge could come in handy) and attachment ambiguity, and using WordNet for looking up e.g. synonyms, antonyms (useful for creating important negative relations), and verb frames.

Our next goal would be to feed into the system regular WWW and/or encyclopedia information. This would require work on several levels. We would need to be able to deal with less "clean" data, and would need to add considerably more pre-processing before handing a text to the parser. Standard IE techniques may be useful for identifying relevant portions of, for instance, a Web page. Other challenges we expect from this part is making sure that the weight mechanisms work the way we expect them to, finding robust and large-scale mechanisms for testing the system, and dealing with contrasting data from different sources.

Finally we could insert into the system more "real world" information by means of an enhanced structural model. This may include building mechanisms to deal with specific times of events, precise knowledge about relativeness of prepositions (if A is on B and B is on C, then A is on C), knowledge of cause and effect, and so on. We could build and improve mechanisms for language production from network representation, and think of some specific applications for which the system could be used. There is really no end to what we think can be accomplished by the system, and we're pretty sure that it can all be done in one quarter.

References

[Breck, 2000] Breck, E., Burger, J., Ferro, L., House, D., Light, M., and Mani, I.: "A Sys Called Qanda". MITRE Technical Report, http://www.mitre.org/support/papers/tech_papers99_00/breck_qanda/, 2000.

[Fellbaum, 1998] Fellbaum, C.: *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[Hirschman et al., 1999] Hirschman, L., Light, M., Breck, E., and Burger, J. D.: "Deep Read: A Reading Comprehension System". In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pp. 325-332. College Park, Maryland, 1999.

[Lehmann, 1992] Lehmann, F., ed.: *Semantic Networks in Artificial Intelligence*. Pergamon Press, Oxford, 1992.

[Richardson et al., 1999] Richardson, S. D, Dolan, W. B, and Vanderwende, L.: "MindNet: acquiring and structuring semantic information from text". Microsoft Technical Report, http://research.microsoft.com/nlp/nlppubs.asp, 1999.

[Riloff, 1999] Riloff, E.: "Information Extraction as a Stepping Stone toward Story Understanding". In Ram, A. and Moorman, K., eds., *Computational Models of Reading and Understanding.* MIT Press, 1999.

[Schwitter et al., 2000] Schwitter, R., Mollá, D., Fournier, R., and Hess, M.: "Answer Extraction: Towards better Evaluations of NLP Systems". In *Workshop on Reading Comprehension Tests as Evaluation for Computer-Based Language Understanding Systems* (ANLP-NAACL'00), pp. 20-27. Seattle, Washington, 2000.

[Sleator and Temperley, 1991] Sleator, D. D. and Temperley, D.: "Parsing English with a Link grammar". Carnegie Mellon University Computer Science technical report CMU-CS-91-196, 1991.