A COMMON FACTS DATA BASE

William Crowther
BBN Systems and Technologies Corporation
10 Moulton Street
Cambridge, MA 02138

INTRODUCTION

This note is about a project whose goal has been to develop methods for converting large amounts of natural language text into machine-accessible data base relations. The easiest way to explain the project is to describe its historical development, pointing out along the way key ideas relating to the project's goal, its version of "meaning," its data representation, and how it treats errors.

PHASE 1 - THE GOAL

The origins of the project go back about ten years. At that time, I wanted to make a game that required a lot of information about animals - which ones were dangerous, which ones were good for eating, how big they were, etc. So I manually constructed a relational data base listing all the common animals and all their important properties. There were several thousand animals, and each had about 100 properties, but because of inheritance, the total number or relations was only about 20,000. I used a relational data base because it seemed, for example, that a structure such as (aardvark weight-in-pounds 140) captured the information in a natural way. Structuring the information as pairs - e.g., (rattlesnake - poisonous) was sometimes more natural, but it was easy to encode pairs as triples (rattlesnake is poisonous), and harder to encode triples as pairs.

This data base was the precursor of the current project, whose goal is simply to create a much larger version of the same thing, with many more nouns and properties, and tens or hundreds of millions of relations. We call the resulting data base a "common facts" data base to emphasize the notion that we are storing a large number of facts that are part of common knowledge. With such a goal, manual entry is prohibitive and some form of automatic acquisition is essential. Despite the change in scale, users will still ask prototypical questions such as "is x poisonous?" and "what is the weight of x?"

PHASE 2 - DICTIONARIES, AND THE MEANING OF "MEANING"

A few years ago, at a fairly low and intermittent level of effort, I started working with an on-line dictionary, initially just trying to find the head noun of each noun definition. This limitation had the virtue that it made constructing a parser plausible, and if it worked well I would have a complete inheritance hierarchy.

Finding the head noun was too simplistic. There were constructs like "body of water," "genus of rose" and "structural member" to deal with, as well as the general problem that words have more than one meaning. But a few general rules took care of such problems fairly well, for example "genus of x is a plant if x is a plant." These rules were stored in the data base as simple templates; e.g., see x, produce y. Surprisingly, a few hundred rules that I developed appeared to cover a large number of these constructs. I also needed to manually override a few hundred definitions - e.g., animal doesn't inherit much from kingdom, even though it was defined in terms of "animal kingdom." The result was a hierarchy incorporating about 50,000 nouns. There were errors, of course: some were due to the parser, some to the rules, and some could arguably be traced to ambiguity or omissions in the dictionary. Based on casual examination of samples of the dictionary, it was clear that a large percentage (maybe close to 90%) of the dictionary nouns were being correctly placed in the hierarchy.

The parser also found some modifiers. All the adjectives which preceded the head noun were available, and generally one modifier following the head noun (e.g., a prepositional phrase like "of Africa"). Beyond that, the parser would have to deal with prepositional attachment, which it was unable to do at the time. I didn't worry too much about how to organize the modifiers. For example, the program knew the relation (rattlesnake is poisonous), but really had no idea what it meant. However, as I shall argue below, the goal was to acquire and retrieve the information for retrieval applications, and meaning might not play a major role.

In retrospect, the idea that the program did not need to know the meaning of something in order to store and retrieve it has become a cornerstone of the system. As the system grew, it learned more and more about the relation "is poisonous." There were dozens of examples of poisonous things, a definition of poisonous containing a link to poison, which also has a definition, etc. There was a description of what poisons do, and what antidotes do, and how they are linked to poisons. Eventually, one is tempted to say that the system somehow "knows" the meaning of poisonous, but it really doesn't. From a pragmatic point of view, the system knows about definitions, modifiers, inheritance, and the like - all else is just interrelationships in the data. From this point on, I decided that dealing with meaning may not be necessary for the goals of this project. The system has no underlying model of the world, nor does it really try to acquire one from the data it reads. All it does is store and retrieve data, sometimes following relational links to do so.

The system was set up to accept a dictionary as input and produce as set of relations as output. Processing a 600-page dictionary through the system at this time produced 0.5 million relations. Very roughly, I estimated that this constituted about one-third the number of relations in the dictionary. Bearing in mind the difficulty of deciding on the correctness of relations, it was my judgement that as many as 80% of the 0.5 million relations were correct. (Note that storing this number of relations requires about 5-10 MB of memory. At this rate, one could imagine holding relations from several books in the main memory of a modern machine.)

Initially, we worried about the bootstrapping aspect of the project. The parser needed semantics from the data base to parse most things, and the data base couldn't be filled without a parser. Now we know that bootstrapping works well. What we can parse on syntax and hand entered semantics is enough to get started, and if necessary the same material can be read several times, each pass gleaning more information than the last.

PHASE 3 - TRANSFORMATION AND REPRESENTATION

At this point, a modest BBN internal research and development effort was started to support the project; however, it was still very much a part-time effort. There were now quite a few interesting new things to try - none of them particularly easy to do well, but many of the common cases could be handled with modest effort.

The easiest thing to try was to process definitions other than nouns. Here it became apparent that a dictionary was really not a repository of facts about the world, but rather of facts about words. The definition of an adjective rarely said anything more than "this word is equivalent to this phrase," leaving the meaning still opaque. Still, we were able to capture these equivalences without much change to the parser. Verb definitions were harder, because they often resembled clauses. But the simple cases were tractable, and again related a word to an expression. In retrospect, the noun definitions which had seemed to contain real information really amounted to an assertion that, among all the possible things one could describe, some had names. Again, we see the concept of "meaning" becoming weakened to equivalence between expressions.

Still, the processing of these equivalences was the start of something which has become more and more important as the project has gone on. Going back to the idea of a game for a moment, suppose the client wants to ask whether a particular animal is "poisonous." This is easy when the source text is cooperative and uses the same word the client chose. But suppose the text used the word "venomous." The information to discover the relation between poisonous and venomous is readily available - in this case venomous has poisonous as a direct synonym, and one need not even bother to process anything beyond synonyms. So, we implemented a synonym link, and also incorporated actual definitions from the dictionary, which expressed equivalence between a word and a phrase instead of between a word and a word. In this manner, we had started down a path which would eventually become another cornerstone of the project: while we might not know what something "meant," we did know how to transform it into another expression which meant roughly the same thing.

Meanwhile, it was important to improve the parser again to parse more than the head noun and the first phrase. Part of this involved adding new and more complex syntax - even some simple clauses. It was tempting to try to use one of the existing well-developed parsers at BBN. But for a number of reasons, I was unable to use any of these parsers in my system; for example, none of them was geared to process tens of thousands of sentences per hour.

A main thrust in parser improvement was the idea of using the data base to resolve parse ambiguities. The way the parser worked was to process a sentence until it discovered one of a set of ambiguities, which it tried to resolve using information already in the data base. The parser recognized about eight different classes of ambiguities, of which a typical example was the ambiguous conjunction: (x or y z); the parser must decide whether the expression means ((x or y) z), or (x or (yz)). For example, "iron or stone fence" would be grouped one way, while "gate or stone fence" the other. For this type of ambiguity, the data base itself often has the information needed to resolve the ambiguity. In this example, for instance, iron and stone are both materials while gate and fence are man-made structures. The program is able to resolve these types of ambiguities from the similarities discovered by questioning the data base.

Prepositional attachment is another type of ambiguity that could sometimes be resolved because the program could readily distinguish categories like places, times, objects, and actions. In this way, the parser was not only a front end processor, it also used the data base explicitly in doing its work in a bootstrapping fashion.

With these improvements in place, we estimate that we will be able to acquire about two-thirds of the relations from the dictionary, though we have not as yet processed the whole dictionary.

Continuing the attempt to acquire more relations, we processed through the system encyclopedia articles instead of dictionary entries. In the encyclopedia, the problems are much harder. Some of the difficulty comes from simple things, like the fact that our parser is incomplete about three-way conjunctions. Part of the difficulty comes from much harder things, like figuring out what pronouns are referring to. (We do match pronouns to a crude context list, which works well for the most common cases.) The principal difficulty with encyclopedias was that the information we were gathering was much more complex than for dictionaries. The representation of such information became a real issue, which we discuss next.

Representation

Originally our representation was straightforward: animals had properties like *COLOR* , *WEIGHT* , *HABITAT* , *PREY* , *PREDATORS* , *FOOD-VALUE* , and *ADJECTIVAL-MODIFIER* ; and our goal was simply to fill in the values of those properties with words like *BLACK* , 140, *GRASSLAND* , ... *CARNIVOROUS* . There was a need to extend the representation to cover more complex constructions, as detailed below.

The representation was first extended to include multi-word constructs and relations, such as "automatic weapons factory" and "body temperature in winter." The program uses a mapping between data base relations and sentence structure operators. The method used to implement the mapping, while developed independently, seems to resemble the work of Katz[1] at MIT.

The next extension included representations for synonyms and equivalent expressions. With this extension, the system was able to relate *POISONOUS* to *VENOMOUS* , and even "*COLOR RED* " to "*LOOKED RED*," for example. However, we didn't want to make these transformations on input, because generally the system had not acquired enough information to make all of these transformations by then. For example, upon encountering "rattlesnake", the system had not seen "venomous." Since it didn't matter when the transformation was made, and the system was already making simple inheritance transformations on retrieval, we chose to make all transformations at retrieval time. This choice turned out to be fortuitous, even from a performance point of view, since there were many more input sentences than queries.

It then became apparent that the concept of "synonym" was an extreme form of the concept "overlapping information." Is a *DANGEROUS* snake *POISONOUS* ? Probably so, but perhaps not! In this case, we couldn't make the transformation on input; *DANGEROUS* is really different from *POISONOUS* , and we didn't want to corrupt the data base with possibly wrong inferences, particularly when we might suddenly acquire the correct information. However, in the absence of any other information, we wanted to say yes to the question or at least to say that the statement was likely. In this fashion, we have extended our representation to include inferencing. This extension resulted in a representation that was inherently ambiguous, since the same information was likely to be in the data base in different forms. As an example, there were literally thousands of

[1] Katz, B. 1988. Using English for Indexing and Retrieving. In *Proceedings of the Conference on User-oriented Content-based Text and Image Handling, RIAO '88*, Vol. 1, 314 - 332. MIT, Cambridge, MA.

ways to store the information "*RATTLESNAKE IS POISONOUS* ," all of which would yield "YES" to the question "is a rattlesnake poisonous?" In retrospect, we saw that we had been storing information in more than one way from the beginning. Simple inheritance - "birds can fly" - gives us an alternate way to store the fact that "ravens can fly." Only now we had a situation in which things were not black and white. Instead of answering yes or no, we were answering "probably." Of course, things were never really black and white - not all birds can fly, just typical birds, yet our source of data was very prone to saying things like "birds fly," without qualifiers.

Confidence in Retrieval

Not all the different ways to store information have the same confidence level, which leads us to the next topic, confidence of retrieved information. With an ambiguous representation, we were forced to look for answers in a variety of places. Usually the answer was found in several places, and often the answers did not agree. In a long search, one would frequently encounter a broken entry, roughly equivalent to "true is false," and then prove just about anything, so dealing with errors and likelihood of error became very important. Our view is that the program establishes a path from query to answer by finding a number of inference links, and that each link carries its own measure of disbelief. The goodness of a path is measured by the sum of the disbelief acquired over the links. The disbelief may be as simple as the fact that inheritance sometimes fails because a relation expresses "typical" truth, while the child may not be typical. Or the disbelief may be of a completely different kind: "the encyclopedia says x" can be quite different from "Himalayan tribesmen say x." In our world, every statement has a source, which has an associated level of disbelief. Recognizing this, we can now choose the most believable answer as our reply. Or we can report conflicting answers if the belief is roughly equal.

Inexact inferences can lead to errors in information retrieval from the data base. We were originally tempted to ascribe the errors to some flaw in our system - perhaps if we relied on better sources, or understood them better, we could operate without errors. Many existing systems do assume an underlying perfect world model in which everything can be precisely expressed in terms of a set of fundamental properties, from which all other properties can be derived. Instead, our underlying model is inherently ambiguous and incomplete. We believe that both the number of underlying facts and the number of possible properties are infinite, and that we will never know more than a small fraction of them. In addition, we know that relations and concepts are themselves related, usually in imprecise and ambiguous ways. Therefore, we will always be trying to infer some relation we do not know from some other relation we do know, and that inference itself will always be suspect. We consider this ambiguity and uncertainty in our model to be one of the strengths of our system, because it seems to mirror the ambiguity in the data we are acquiring.

One might think we have taken an enormous performance penalty by doing inferencing on all our queries, since general inferencing is notoriously slow. However, the system does not do general inferencing, and there are a number of tricks which make special types of inferences rather speedy. For example, we have the whole of the data base available, and we never add an inference unless the data base has entries which might match it . The space of real data is much smaller than the space of possible inferences. Also, any inference which amounts to a simple substitution of one word or phrase for another is handled by a special and efficient algorithm. Furthermore, because each inference introduces disbelief, we simply refuse to follow long chains of inference; they are not believable. And, of course, we rely on the fact that the data base is small enough to fit in real memory, where access is relatively cheap.

PHASE 4 - THE FUTURE

Because of recent DARPA interest in this project, our focus has shifted more toward possible applications of the work. It becomes important at this point, therefore, to determine what can and cannot be done with this approach. Crucial to such determination is the development of methods to evaluate the performance of the system in a rigorous manner, which will require the addition of certain capabilities to the system before such evaluation can meaningfully take place. Our primary focus now is on measuring what fraction of the available information the approach can recover, trying to determine both what we can do now and what we could do with a little work (or a lot of work). We also may be able to distinguish between retrievable and non-retrievable information.

We are also trying to determine possible applications for such a system. While it is designed to retrieve specific information, it is just as well suited to retrieving references, and could be used to find articles about a particular topic, or even find articles about rather specific interrelationships of topics, since that is the kind of information it manipulates. It might also serve as a memory component of some other system. Surely many parsers would benefit from having this sort of information available, and systems which model parts of the real world could perhaps find some useful information here.

We are working toward a massive processing of an encyclopedia - as much as we can fit into memory of the biggest machine we can access. Sheer size has always been important to this project, and we need to know how things will scale in the presence of larger amounts of data. There is always the worry that irrelevant clutter and wrong information will choke off growth, but we have the hope that redundant information will allow elimination of errors and promote growth. As the information becomes more complex, issues of representation, meaning, and belief become more and more urgent.