COMPUTATIONAL UNDERSTANDING

Christopher K. Riesbeck

I. METHODOLOGICAL POSITION

The problem of computational understanding has often been broken into two sub-problems: how to syntactically analyze a natural language sentence and how to semantically interpret the results of the syntactic analysis. There are many reasons for this subdivision of the task, involving historical influences from American structural linguistics and the early "knowledge-free" approaches to Artificial Intelligence. The sub-division has remained basic to much work in the area because syntactic analysis seems to be much more amenable to computational methods than semantic interpretation does, and thus more workers have been attracted developing syntactic analyzers first.

It is my belief that this subdivision has hindered rather than helped workers in this area. It has led to much wasted effort on syntactic parsers as ends in themselves. It raises false issues, such as how much semantics should be done by the syntactic analyzer and how much syntactics should be done by the semantic interpreter. It leads researchers into all-or-none choices on language processing when they are trying to develop complete systems. Either the researcher tries to build a syntactic analyzer first, and usually gets no farther, or he ignores language processing altogether.

The point to realize is that these problems arise from an overemphasis on the syntax/semantics distinction. Certainly both syntactic knowledge and semantic knowledge are used in the process of comprehension. The false problems arise when the comprehension process itself is sectioned off into weakly communicating sub-processes, one of which does syntactic analysis and the other of which does semantic. Why should consideration of the meaning of a sentence have to depend upon the successful syntactic analysis of that sentence? This is certainly not a restriction that applies to people. Why should computer programs be more limited?

A better model of comprehension therefore is one that uses a coherent set of processes operating upon information of different varieties. When this is done it becomes clearer that the real problems of computational understanding involves questions like: what information is necessary for understanding a particular text, how does the text cue in this information, how is general information "tuned" to the current context, how is information removed from play, and so on. These questions must be asked for all the different kinds of information that are used.

Notice that these questions are the same ones that must be asked about ANY model of memory processes. The reason for this is obvious: COMPREHENSION IS A MEMORY PROCESS. This simple statement has several important implications about what a comprehension model should look like. Comprehension as a memory process implies a set of concerns very different from those that arose when natural language processing was looked at by linguistics. It implies that the answers involve the generation of simple mechanisms and large data bases. It implies that these mechanisms should either be or at least look like the mechanisms used for common-sense reasoning. It implies that the information in the data bases should be organized for usefulness -- i.e., so that textual cues lead to the RAPID retrieval of ALL the RELEVANT information -- rather than for uniformity -- e.g., syntax in one place, semantics in another.

The next section of this paper is concerned with a system of analysis mechanisms that I have been developing. While the discussion is limited primarily to the problem of computational understanding, I hope it will be clear that both the mechanisms and the organization of the data base given are part of a more general model of human memory.

II. ANALYSIS MECHANISMS

It has been recognized for some time now that understanding even apparently simple texts can involve the application of quite general world knowledge, that is, of knowledge that would not normally be considered part of one's knowledge of the language in which the text is written. The set of information that might be needed for understanding a text is therefore tremendous. Clearly an understanding system cannot be applying all it knows to everything it reads all the time. It must have mechanisms for guessing what information is likely to be needed in the near future. As long as its guesses are good, and the understander updates them in the light of new input, understanding can proceed at a reasonable rate.

In other words, the understander must be good at PREDICTING what it is likely to see. Further the data base must be organized so that coherent clusters of relevant information can be accessed quickly with these predictions. But since no finite static data base can have exactly the right information for every input, the understander must be able to prune and modify the information that the data base contains so that it applies more precisely to the situation at hand.

The analyzer which I developed in my thesis [Riesbeck, 1974] was based on the concept of "expectation". The analyzer program consisted of a fairly simple monitor program and a lexicon. The lexicon was a data base whose contents were organized under words and their roots. The information in the data base was in the form of pairs of predicates and programs, which were called EXPECTATIONS.

The analysis process consisted of the monitor reading sentences, one word at a time, from left to right. As each word was read, the monitor did two things. It looked up the word (or word root if no entry was found for the word) in the lexicon, and added the associated expectations (if any) to a master list of expectations. Then each element of this master list was checked. Those expectations with predicates that evaluated to true were "triggered" -- i.e., their programs were executed and the expectations were removed from the master list. Those expectations that were not triggered were left on the master list. When the end of the sentence was reached, the meaning of the sentence was that structure (if any) which the triggerings of the various expectations had built.

A general idea of the way the analyzer worked can be obtained by following the flow of analysis of the simple sentence "John gave Mary a beating." The chart on the next page gives an outline of the basic sequence of events that takes place in the analyzer as the sentence is read, one word at a time, from left to right. The column headed "WORD READ" indicates where the analyzer is in the sentence when something occurs. The column headed "EXPECTATIONS WAITING" gives the predicate portion for all the activated but not yet triggered expectations. The column headed "EXPECTATIONS TRIGGERED" indicates, when a number is placed in that column, which expectation has just been triggered at that point in the analysis. The column headed "ACTIONS TAKEN" indicates what effects the triggered expectations had. INPUT refers to whatever has just been read or constructed from the input stream.

Step 0 is the initial state of the analyzer before the sentence is begun. The analyzer sets up one expectation which assumes that the first NP it sees is the subject of a verb that will come later.

In Step 1, the first word -- "John" -- is read. Because "John" is a proper name, it is treated as a noun phrase and thus Expectation 1 is triggered. The program for Expectation 1 chooses "John" to be the subject of whatever verb will follow. Expectation 1 is then removed from the set of active expectations. There were no expectations listed in the lexical entry for "John".

In Step 2, "gave" is read. The lexical entry for the root form "give" has three expectations listed and these are added to the set of active expectations. None of them are triggered.

In Step 3, "Mary" is read. "Mary" is a noun phrase referring to a human and so Expectation 2 is triggered. The program for Expectation 2 chooses "Mary" to be the recipient of the verb "give". Then Expectation 2 is removed. There were no expectatons in the lexical entry for "Mary".

In Step 4, "a" is read. There is one expectation in the lexicon for "a". This is Expectation 5 which has a predicate that is always true. That means that Expectation 5 is triggered immediately. The program for Expectation 4 is a complex one. It sets aside in a temporary storage area the current list of active expectations. In its place it puts Expectation 6, which will be triggered when something in the input stream indicates that the noun phrase begun by "a" is complete.

In Step 5, "beating" is read. There are no lexical entries and "beating" is not a word that finishes a noun phrase, so nothing happens.

In Step 6, the end of the sentence is seen. This does finish a noun phrase and so Expectation 6 is triggered. The program for Expectation 5 builds a noun phrase from the words that have been read since the "a" was seen. It places this back in the input stream and brings back the set of expectations that Expectation 5 had set aside.

In Step 7, the input "a beating" triggers Expectation 4. The program for Expectation 4 builds a conceptual structure representing the idea of someone hitting someone else repeatedly. It uses the subject "John" as the actor and the

recipient "Mary" as the object being hit. The final result therefore is a representation that says that John hit Mary repeatedly.

The program portions of the expectations therefore produced the meaning of a sentence. These programs were not limited in power. Not only could they build, modify and delete syntactic and conceptual structures, but they could add, modify and delete the list of expectations as well. This is why the analysis monitor was so simple. All the real work was done by the program portions of the expectations.

The predicates were predictions about likely situations that would be encountered in the processing of the sentence. Some of these predictions were about what words or word types would be seen. For example, one of the expectation pairs in the lexical entry for "a" contained a predicate that a noun would be seen soon. Elsewhere in the lexicon, there were expectations whose predicates were about the structures that other expectations had built or would build. There were also expectations with predicates that were true in all situations. In this case the programs were supposed to be executed whenever the word referencing them in the lexicon was read.

The predictive power of the predicates arose from the fact that the predicate did not look at all the things that an input might mean. Rather it asked if the input COULD mean some particular thing. If so the expectation was triggered. The predicate portions of expectations were the disambiguating component of the analyzer because they chose only those word meanings that the sentential context had use for.

To generalize this discription of the analyzer a bit more, the basic memory mechanism used was the expectation, which consisted of a prediction about a possible future situation and instructions on what to do if that situation occurred. The basic organization of memory was to have clusters of these expectations attached to words and word roots. The access to this memory was through the words seen in a sentence being understood.

The thrust of the work of the analyzer had been on the development of the expectation mechanism as a viable analysis tool. This meant defining what kinds of expectations were needed and how they could be easily retrieved. One of the major weaknesses of the analyzer was the lack of any satisfactory control over the set of currently active expectations. There was no real tuning of the set of expectations found in the lexicon to fit the situation at hand. The only interaction between expectations occurred when expectations were triggered and produced concrete structures. The only mechanism for removing untriggered expectations was the wholesale clearing of active memory at the end of a sentence.

The extension of the concept of expectations to make them more controllable without destroying their generality has been the core of the work that I have been doing since the thesis. Programming is going on right now to incorporate the extensions into a second version of the analyzer.

The first basic extension to the predicate-program format of the expectations was the addition of explicit information about the purposes of various expectations. That is, an expectation was made and -- more importantly -- kept around because there was some need that the triggering of this expectation would fulfill. For example, the verb "give"had listed in its lexical entry several expectations which could fill the recipient slot for that verb if triggered. There was one which looked for the next noun phrase referring to a human. This expectation, activated as soon as "give" was seen, would fill the recipient slot in sentences like "John gave Mary a book." A separate expectation, activated at the same time, looked for the preposition "to" followed by a noun phrase referring to something that was at least a physical object. This expectation if triggered would fill the recipient of "give" with the object of the "to", as in sentences like "John gave the book to Mary."

Both of these expectations have the same purpose: to fill the recipient case of the verb "give". As long as no recipient is found there is a reason for keeping both expectations active. And this implies that when the recipient case is finally filled, either by one of the expectations set up by "give" or by some expectation set up by some later word, then there is no longer any reason for keeping any of these expectations and they should all be removed.

If the monitoring program is to be capable of both loading and removing the various expectations, it must know what the purposes of the expectations are. Unfortunately, there are no constraints on what sorts of functions can appear as predicates and programs in an expectation, which makes such a capability impossible. However it is not necessary for the monitor to recognize purposes for ALL expectations. It is sufficient for it to know about just those expectations that fill empty conceptual or syntactic slots when they are triggered. The two expectation examples given above for filling the recipient case of the verb "give" are of this type. We can specify the purposes of such expectations by simply specifying what slot they fill if triggered. The monitor can tell with these expectations when they should be kept and when they should be removed. The monitor leaves alone actions -- such as those that manipulate other expectations -- which are not linkable to simple purposes.

While this was the first important extension to the expectation format it was not the last. Almost immediately it was realized that many expectations are dependent upon others in the sense that they cannot possibly be triggered until the other ones are. For example, suppose we have an expectation whose predicate looks at the

syntactic object slot of the verb "give" and whose program builds some conceptual structure using this information. Further suppose we have another expectation active at the same time whose predicate looks for a noun phrase in the input stream and whose program will fill in the syntactic object slot for "give" with that noun phrase. Then clearly the former expectation must wait for the latter to be triggered first before it has a chance of being triggered itself.

This kind of dependency relationship between expectations is not just an interesting observation. Remember that the predicate portion of an expectation was a PREDICTION about what might be seen. This means that the first expectation -- the one whose predicate looks at the syntactic object of "give" when it is finally filled -- is not only waiting for the second expectation to be triggered but in fact is making a prediction about what the second expectation will produce. This has two implications.

First, if the second expectation cannot produce a structure that will satisfy the predicate of the first expectation, but there is an expectation that can, then the second expectation is less preferable to this third one, which means that the third one would be checked first when new input arrives. A dynamic ordering has been induced on the set of active expectations.

Second, structure building expectations often build from pieces of structures that other expectations build. If we have a prediction about what an expectation should produce, we can then make predictions about the sub-structures that the expectation builds with. These new predictions can then influence the expectations producing those sub-structures, and so on.

For example, consider the two expectations for "give" that were given above. Suppose the predicate of first expectation looks for a syntactic object referring to an action -- such as "a sock" in one interpretation of the sentence "John gave Mary a sock." Since the second expectation is the one that fills in the syntactic object slot of "give", there is now a prediction that the second expectation will produce a noun phrase referring to an action. Since the second expectation fills the syntactic object of "give" with a noun phrase that it finds in the input stream, the monitor can predict that a noun phrase referring to an action will appear in the input stream. The effect of this prediction is that when words are seen in the input, the first thing that is looked for is to see if they can refer to an action. If so, then that sense of the word is taken immediately. Thus a word like "sock" is disambiguated immediately as a result of an expectation originally made about the syntactic object of "give".

To pass the information from one expectation to the next about what an expectation would like to see, we need to know where the expectation is looking. That is we need to know what the predicate of the expectation is applied to. This information can be specified in the same way that the purpose of the expectation was: by giving a conceptual or syntactic slot. In this case, instead of giving the slot that the expectation fills if triggered, we specify the slot that the predicate of the expectation is applied to. Then by knowing what slot an expectation looks at, we know what expectaions this expectation depends on. It depends on those expectations that fill this slot -- i.e., that have a "purpose slot" equal to the "look at slot" of the expectation.

Let me summarize this discussion by giving the current format for specifying expectations:

(NEED FOCUS TEST ACTION SIDE-EFFECTS)
where

NEED is the slot the expectation fills if triggered,
FOCUS is the slot the expectation looks at,
TEST is the predicate portion of the expectation,
ACTION is the structure building portion of the expectation,
SIDE-EFFECTS are those programs that act upon other expectations and are not -- at the moment -- incorporated into the network of dependencies and predictions.

The analysis monitor is fairly content-independent. Its job is to take input, use it to access clusters of expectations, keep active those expectations that might fill slots that are still empty in partially-built structures, and keep track of the predictions/preferences that are induced by the dependency relationships between expectations. The actual knowledge about language and the world is still contained in the expectations, as was true in the original analyzer.

This encoding of knowledge into small pieces of programs that have both procedural and declarative aspects is of both practical and theoretical importance. In terms of implementing an AI model, I have found it much easier to specify procedural knowledge in small units of "in situation X do Y". Further it is much easier, as a programmer, to extend and modify procedures written in this form. It is also easier for a program to manipulate knowledge in this way.

Theoretically, the expectation format seems to me to be a viable memory representation for highly procedural knowledge. With it we can design explicitly a theory of computational understanding that does not have the forced division between syntactic and semantic analysis. Individual expectations are usually concerned with syntactic or conceptual structures, but all of the expectations are maintained in one large set. This allows for those important expectations that convert information about syntactic structures in semantic information and vice-versa. Thus information that originally started as an abstract conceptual

prediction can be quickly disseminated throughout a dependency network of expectations and lead eventually to predictions about things like word senses.

For example, my thesis describes how the interpretation of the text "John was mad at Mary. He gave her a sock," uses a conceptual prediction that "John wants something bad to happen to Mary," which follows from the first sentence, to choose the appropriate sense of the word "sock" in the second sentence the first time the word is seen. This can be done because the general conceptual prediction in interaction with the expectations in the lexical entry for "give" led to predictions about the nature of the syntactic object of "give", which in turn led to predictions about the words that would be seen in the input stream.

In other words, the analysis system -- both the original one and the new version -- as an approach to the computational understanding problem, exemplifies the general points made in the methodological portion of this paper. It demonstrates the feasibility of doing understanding using very simple mechanisms for manipulating small but flexible units of knowledge, without forcing the development of independent syntactic analyzers or semantic interpreters. These simple mechansisms allow for a direct attack on such problems as what information is absolutely necessary for understanding, how it is called for, and how a workably sized set of active information can be maintained.

REFERENCE

Riesbeck, C. "Computational Understanding: Analysis of Sentences and Context," Ph.D. Thesis, Computer Science Dept., Stanford University, Stanford, CA. 1974.

STEP	WORD READ	EXPECTATIONS ACTIVE	EXPECTATIONS TRIGGERED	ACTION TAKEN
0	none	1 - is INPUT a NP?	none	none
1	John	1 - is INPUT a NP?	1	choose "John to be the subject of the verb to come
2	gave	2 - does INPUT refer to a human? 3 - does INPUT refer to a physical object? 4 - does INPUT refer to an action?	none	none
3	Mary	2 - does INPUT refer to a human? 3 - does INPUT refer to a physical object? 4 - does INPUT refer to an action?	2	choose "Mary" to be the recipient of "give"
4	a	3 - does INPUT refer to a physical object? 4 - does INPUT refer to an action? 5 - true	5	save the current set of expectations and replace it with: 6 - does INPUT end a NP?
5	beating	6 - does INPUT end a NP?	none	none
6	period	6 - does INPUT end a NP?	6	set INPUT to the NP "a beating" and reset the expectation set
7	none	3 - does INPUT refer to a physical object? 4 - does INPUT refer to an action?	4	set the main action of the interpretation to the action named by INPUT; set the actor to the subject (John) and set the object to the recipient (Mary)