WORKING PAPER 168

Story Understanding: the Beginning of a Consensus

David D. McDonald

June, 1978

ARTIFICIAL INTELLIGENCE LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Abstract

This paper was written for an Area Examination on the three papers: "*A Framed PAINTING: The Representation of a Common Sense Knowledge Fragment*" by Eugene Charniak, "*Reporter: An Intelligent Noticer*" by Steve Rosenberg, and "*Using Plans to Understand Natural Language*" by Robert Wilensky. Surprisingly, these papers share a common view of what it means to understand a story. The first part of this paper reviews the previous notions of "understanding", showing the progression to today's consensus. The content of the consensus and how the individual papers fit within it is then described. Finally, unsolved problems not adequately dealt with by any of the approaches are presented briefly.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defence under Office of Naval Research contract N00014-75-C-0643.

Introduction

Story understanding has its roots in the research of the late-sixties, early-seventies on natural language question-answering systems. Programs like SHRDLU and LUNAR pioneered the basic techniques in parsing and structure building that underlie all forms of text understanding. But what it meant for SHRDLU to understand a question turns out to be very different from what it is coming to mean to "understand" a story. This difference is instructive, and will serve to introduce the problems.

Query understanding

The people in this "earlier consensus" had a very clear conception of what it meant for their programs to understand a text. Their idea came from a strong tradition in logic, dating back at least to Tarski. It became a successful as an engineering technique - almost every modern question answering system uses it. And, it was (and is) believable - if you had asked: "but does SHRDLU "really" understand what it reads?", most everyone who worked on it would have said yes.

For them, the meaning of text was defined operationally. Understanding a question meant being able to answer it; understanding a command meant being able to carry it out; understanding a simple definition (e.g. "*call it superblock*") meant being able to use it as an alternate way to refer to something. Understanding was accomplished by the translation of the English sentence input into a formula in the internal language of the program, e.g. MICROPLANNER. Evaluating that expression would compute the answer to the question, or execute the action, or add the new definition to the data base.

The translation was straightforwardly compositional, under the control of the dictionary entries for the units analysed by the parser. The entire operation was done in strict isolation from the rest of the discourse with the user. The programs only took in a single sentence at a time, reacted to it, then (for the most part) forgot about it.

Stories are different

Stories do not have a natural operational semantics. Hearing a story changes one's knowledge state, but how the change will be realized in action is unpredictable. You cannot understand a story by translating it into a program unless you know what that program(s) should do.

But if meaning is not in action, where is it? The answer, as currently understood, is that the meaning of a text derives from the representation that the person - program builds when they read it. This means that there will still have to be a translation process of some sort, if only to remove the incidental stylistic variation in the English text from the final representation. The question is, what form should that representation now take?, what will it be used for?

If you want to know if a person has understood a story, you ask them questions about it. This same tack has been taken in testing the adequacy of story understanding computer programs, and it gives a focus to the design of the representations they use. (As of yet, no one has developed any alternative paradigms, e.g. reading instructions as a way to learn a skill.)

Story texts are translated into some declarative representation which can later be accessed to answer questions. Wilensky uses conceptual dependency; Rosenberg uses FRL; and Charniak uses a frames system of his own design. For present purposes, all of these formalisms are equivalent. They are verb centered, and use "word-sized" tokens and pointers (i.e. they are LISP based). Any expression in one of these representation styles could be reformulated as a predicate - argument assertion in the style of MICROPLANNER (Rosenberg has done exactly that in the 1978 revision of his paper).

The critical design question, common to all of these formalisms, is how delicately to represent a conceptualization. Schank holds, for example, that the representation for "*John ate some ice cream*" should explicitly represent the instrument that he used (defaulted as a "spoon") even though there was no mention of any instrument in the source text. At issue is the "vocabulary" from which formulas in the internal representation can be composed. Circa 1975, Schank believed that 14 terms would suffice to describe any action; Wilks held out for about seventy; Charniak believes that there should be roughly as many terms in the internal representation as words in English. Vocabulary size effects what is required to create and reason about "canonical forms" for the concepts that are mentioned in the source text. The identity of (canonical) form is what determines if two subformulas refer to the same thing. This is the "bread and butter operation" of all kinds of language understanding to date. The systems described in these three papers avoid having to settle the issue of what level of conceptualization to represent by working in small domains. This problem will be discussed further at the end of the paper.

Reading between the lines

Translating the input text into some form of "program-ese" is sufficient for understanding queries, commands, or short definitions. These speech-acts are complete in themselves - no additional text or pre-stored information is needed to understand them. Stories on the other hand, by their nature as complex, conventional objects, take their meaning only in the context of a great deal of general and specific knowledge that the speaker assumes his audience shares.

You can give a nine year old child a story to read, then ask them what happened in it, and typically they can give you an accurate, moment by moment account of who did what to whom. But for some children, if you were to then ask what were the character's motives, or what would probably happen next, they would be unable to tell you - and we would say that those children didn't "really" understand what they had read, not, at least, in the way that an adult would. This brings out a new, additional capability which story understanding requires but query understanding did not: *being able to read between the lines.*

Charniak's original program

Charniak's original work with children's stories (these were stories for, not by, children at the elementary reader level) was the first A.I. research on language understanding where the definition of "understand" had been expanded to include this ability to answer questions about motivations and to give explanations.

This ability is the keystone of story understanding, The striking thing about the consensus exhibited by these three papers, considering that they derive from such different research communities, is that they agree on how it should be done. It is useful to look closely, for a moment, at Charniak's early program and look at the technique he used.

As Charniak's 1971 program read a story, it accumulated facts or presumptions about the text in a MICROPLANNER data base which was unstructured except for the matching patterns of the assertions. When it was finished, the data base contained (1) the literal content of the story, for use in answering ordinary questions; and (2) the answers to just those potential "between the lines" questions that it thought it might be asked. The actual question-answering was done by making direct requests against the data base, just as in SHRDLU.

The program operated on the output of a (simulated) combined parser - "translator to canonical form" program. This output was a series of expressions, each translating roughly one clause of the text. One at a time, in sequence, the expressions were then "read". To read an expression was to enter it as an assertion in the data base, with the expectation that this would trigger antecedent theorems and that these would do the work of recognizing the facts needed for later questions and would do the bookkeeping that maintained the context.

There were two kinds of antecedent theorems: base routines and demons. The base routines represented the program's constant knowledge about the relations and objects in the stories it could read about. In effect, they constituted a lexicon, or encyclopedia. The demons did the "reading between the lines". Part of any base routine would be a set of demons that it would assert into the data base (activate) once that base routine was triggered by some phrase in the text. The trigger patterns on the demons were set to notice particular events (entered assertions) that might occur later in the text. When and if one of these events did occur, the appropriate demon would run and would enter into the data base the (presumed) reason why that event had occured, linking it with the earlier event that had triggered the base routine.

So for example we might have these two lines in a story:

"*Janet got her piggy bank*"

"*She used the money to buy ice cream*"

With the mention of "piggy bank" in the first line, the demon PB-FOR-MONEY (and many others) would be activated. Then when the next line is entered, the canonizer will have broken it into several assertions including "(have Janet money)" this is the pattern that PB-FOR-MONEY is "looking for" and it is triggered. What it does is to make the additional assertion to the effect that the reason why Janet now has some money is that she got it from her piggy bank.

Essentially all of the "common sense knowledge" in Charniak's program was in the form of demons like PB-FOR-MONEY - answers looking for a question to happen. This style of representation was more than a little precarious. Each demon was independent of all the others - any number of them might be "seeking" the same pattern simultaneously. Indeed, there was no global description of the program's expectations or current state whatsoever. This is (1) hard to debug, and (2) hard to extend, i.e. each new trigger pattern had to be coordinated those of related kinds of information and with the input translation routines.

Common sense inference rules

Charniak's use of demons to encode "common sense" was transformed, in suceeding years, into something less focused on answering specific questions but still very procedurally based. General purpose inference rules, encoded as antecedent theorems (or the equivalent) examined the input expressions in conjunction with the current state of the memory at large (the data base), and inferred new facts or adjusted annotations on facts already in memory.

Reiger [1974] is probably the best recorded example of this period. His position was to do the maximal amount of inferencing (i.e. hundreds of assertions per English clause). Most of these would be trivial and presumably ignored by the (never designed) higher level program which would recognize those important inferences that advanced the plot or the characterizations. However, the data base into which all these inferences were added had very little structure to it, making the searches that any potential high level program might make very cumbersome. No one ever succeeded in designing such a program and the problem of how to "winnow the chaff from the wheat" was never seriously explored - the new consensus won over people over to its paradigm and the common sense inferencing technique that Reiger typified was abandoned.

The new consensus

Modern day story understanding programs, as shown by these three papers, have done an end-run around the technical problems of coordinating demons and managing data bases by radically changing their use of data structures and, in the process, changing the definition of "understanding".

Before, the representation by which a program understood a story was constructed *de novo* with each separate story, through the action of the procedurally encoded common sense inference rules. Now, the programs have moved from doing construction to doing recognition. The representation that the events of the story will be mapped onto will already exist in the program before the story is begun.

Understanding a story becomes a process of "lighting up" the portions of the knowledge base that correspond to the text. The inferential links that connect events in the text do not have to be computed because they are already a part of the permanent structure. The problem becomes one of how to match expressions from the input text with expressions already embedded somewhere in the program's

knowledge structure.

Motivation

There are several reasons for this shift from constructed to pre-existing story representations. First is the general trend in A.I. away from the purely procedural style of encoding knowledge bases, typified by PLANNER and its derivatives, over to a highly structured, hybrid style based on frames or some similar schematism. This movement reached story understanding about 1975 with the creation of SCRIPTS by Schank and his group at Yale [1975], and at MIT a bit later with the Personal Assistant Project [Sidner 1978].

Then, once the technical capacity was available, people began to expand their conception of what a program should model. For the most part, stories are quickly and easily understood. A story understanding program should behave in the same way, in particular, it should not use its full scale deductive apparatus except in those places where people would also be puzzled (e.g. guessing who the murderer is). Passively recognizing that an event in the text was identical to a prefabricated virtual event in the program by using a search and matching process was taken to be a good model for "automatic" understanding.

More important in the long run is that in its "framish" form the knowledge base can serve multiple functions. Charniak makes the point explicitly that his frames language is designed to be used both for understanding stories about painting and for actually doing it. Rosenberg could make the same point; his program architecture is already used directly with Stansfield's [1977] wheat transactions model. Wilensky's PAM program uses the identical structures that Mehan [1977] uses to tell stories with. This is an instance of the general phenomena of using a single, schematic representation for knowledge which is then accessed and manipulated by different kinds of process interpreters according to the task at hand (see, for example, [McDonald 1978] where this same technique is used in language generation).

Finding links

The older process of doing common sense inferencing would actively "glue" together the story events by creating links that described how they were related. One line in a story might establish a goal ("rescue Mary") and the next might be an action which was part of achieving the goal ("mount horse"). The two story lines would then be linked with a "goal-substep" assertion. A maxim of story understanding is that a program should make sufficient links, positing goals, implicit events, etc, to be

able to trace a path between any two events in the story.

Part of the consensus is an appreciation that there are two general ways of making these explanatory links, and that each is appropriate to a different kind of situation. The Yale group pioneered this distinction as SCRIPTS versus PLANS (cf. [Schank & Abelson 1977]). Other groups make the same distinction, though, of course, they use different names (the "not invented here" syndrome). *Charniak's PAINTING formalism is a kind of* SCRIPT, *and Rosenberg's NOTICING process does planning.*

The distinction is between the case where the links already exist in place as an intrinsic part of the knowledge base, and the case where they must be deduced (recognized for what they are). The first case is where SCRIPTS are used. A script is list of the actions that typically take place in some routine activity. The actions will be named by their role in the activity and possibly structured into alternative sequences and contingencies. To use a script to understand a story, the events in the text are matched against the actions recorded in the script, binding the particular actors and objects to the scripts variables - "instantiating" the script for this story. Very likely the story will not mention all of the actions that the script records, but those are assumed to have happened anyway since this is a routine activity and we would not expect the author to include them all. The motivating links between story lines can be read directly from the instantiated script.

When a story is not routine (the typical case in real life), the program writer can not hope to anticipate all the possible action sequences with pre-made scripts. Instead, it must be possible to compute the desired links "on the fly". This is done on the same principle as before, matching against precomputed knowledge structures, but now the structures must be annotated with abstract descriptions of what they can accomplish, what goals they imply for the actors, and so on.

For example, the first two lines of the story might be: (taken from [Schank & Ableson 1977])

"Wilma was hungry."

"She took out the Michelin guide"

After the first line, the planning system will have assigned Wilma the goal of finding something to eat. The second line has nothing obvious to do with eating, but because only the one goal is present (and, implicitly, because of the conventions of story telling), the system will assume that it somehow forwards her goal and will

proceed to guess what its connection with eating is. This requires two things: general knowledge about the objects in the story, such as that the Michelin ("red") Guide contains facts about where there are restaurants; and general purpose translation rules that can relate, e.g. the use of a guidebook with the "decide where to go" plan-step of the "get something to eat" plan.

In a PLAN-based system, when the relation between two lines of text is not immediately clear (i.e. when a script won't work), a heuristic search is instigated to determine what the link is. The pre-existing knowledge base defines the space through which this space takes place, and once a story is begun, the themes and expectations instantiated in the knowledge base by the text so far will narrow the space still further.

The problem for a PLAN based understander is to (1) design the structure of the planning space, and (2) design the heuristic interpreter that will walk that space trying to recognize where links can be instantiated. The two planning papers do not themselves try to characterize how the heuristic search is done. My own reading of the problem suggests that planning can be view as a process just like matching against scripts except that while a script is a structuring of specific, story level event types, "plans" are structurings of abstract, feature-oriented descriptions of event types, i.e. *a plan is just an abstract script.* To match the story level events against a plan, they must first be "expanded" by translating inferences into a spectrum of increasingly abstract descriptions (not unlike a "kind-of" // "is-a" hierarchy) which at some point will make contact with the level of description used the relevant plan(s).

Semantic paging

The matching process will always be sensitive to the size of the knowledge base with a tradeoff either in time or in storage size depending on what implementation is used. There will be considerable benefit to keeping the size of the space that the matcher must search as small as possible. The dependence upon a efficient matching process that will characterize any implementation of this new paradigm is offset, in large part by an ability to now design the knowledge base in terms of a small cache memory.

The term "semantic paging" is due to Goldstein in the 1976 MIT A.I. lab ARPA proposal. The idea is that now that the contents of the knowledge base are neatly segmented into conceptual units ("frames"). It should be possible to treat groups of frames the way pages of memory are treated in an operating system with virtual

memory. The matching algorithm is made to search the knowledge base in a particular order, always beginning with special "current context" (the cache). When the first call is made for a particular frame outside the current context (analogous to the first access to a particular page of memory), the option is there to also pull into the contextually related frames which we can assume may be needed shortly. The programs of all three papers allude to semantic paging and its usefulness but do not discuss how they do/would actually use it, nor, unfortunately, do they give sufficient implementation detail for the interested reader to figure it out on their own.

* * *

The remainder of this paper will review the three papers in turn and show how they fit into the new story understanding framework. Two problems general to all the papers will be discussed at the end.

Charniak

Charniak's *Cognitive Science* paper does not discuss his view of the story understanding process except implicitly through his choice of representation. That paper is the published form of a 1976 working paper done at the Fondazione Dalle Molle in Geneva, at which time he had not yet had a chance to actually implement his frame system as a runnable program or to design a story understander around it. Since his arrival at Yale, all this changed and his paper for IJCAI-77 reported on "Ms. Malaprop", his almost completed story understander.

His frames are scripts but not entirely

Charniak's frame system is a kind of SCRIPT. His complex events are sequences of actions achieving some goal, with their explanatory links already in place. Input sentences instantiate a line(s) in a frame(s) and thus are understood. Inferences that go through because of form translations effected by general rules ("all X's are also Y's") are already precomputed in the representation of a frame.

However, Charniak's notion of a SCRIPT differs critically from scripts at Yale on the question of redundant representation in the knowledge base. Charniak abstracts out from his complex event frames all common references to causes, physical laws, states, subevents, etc. and gives them an equal status as independent frames. Whereas in Yale's scripts, all the data given by Charniak's interframe pointers would be given redundantly, right with each script. (Specifically, Charniak has frames for "simple

events", single cause and effect pairs that are typically part of the model of the physical world; for "states" which are the repositories of antecedent rules that describe what follows from being in that state; and for "objects" which he says nothing about.) If Charniak were to have a restaurant script, for example, it would share a "payment for service" script with his supermarket script and his department store script, and his college bursar script, etc..

This difference reflects a fundamental difference in philosophy between Charniak and Schank. The motivation for Charniak's style is very familiar to psychologists, linguists (who have greatly influenced Charniak's thinking), and anyone who has had to program a knowledge base with very little memory available. It is a general concern to avoid redundancy and a desire to use the program as a psychological model - having only one representation of, e.g., fluid containment will insure that everything that refers to it is truely refering to the same thing. Schank, on the other hand, genuinely believes that there is no general inheritance mechanism in the human mind and reflects this belief in his redundant model.

Charniak's style of processing

As in his original program, the story text is first parsed and put into a canonical form, i.e. a sequence of assertions with predicate-oriented syntax - [<predicate> <arg1> <arg2> ...]. Because it is intended to be possible to "match" translated input assertions with pre-existing frame statements in the knowledge base, the canonization process is very critical. It is what guarantees that the two sources will speak the same language. Charniak has nothing to say in this paper about how this guarantee can be made, except to point out that it is a problem. His thesis discussed possible tradeoffs between expanding the representation of the conceptualizations in the text to a very general level and thereby "exposing" them to general purpose inferences, and keeping the representation closer to the original "conceptual clumping" of the English, thereby requiring more demons to cover the same inferences but also avoiding inadvertent inferences due to "confusion" among demons with a general trigger pattern applying where they weren't intended to. In his present paper, he has opted decidedly for remaining close to English.

Once the translation is made to canonical form, a general procedure scans the statements of the "active" frame complexes, finds a statement with a "matching pattern" and "instantiates" the statement. These terms have an intuitive meaning to programmers in A.I. and Charniak appeals to that intuition because he wants to

avoid being more precise, as in fact he himself had no more precise a design at the time the paper was written.

The matching is to be done after the manner of MICROPLANNER, including the use of type restrictions on pattern variables. To instantiate a line of a frame is to create a binding environment for its variables, bind them to the values introduced with the story assertion and (presumably, since he hints but does not actually say) to then propagate those binding through the rest of the frame and out the LEADS-TO and COMES-FROM links as far as makes sense. This binding environment (presumably the entire frame) is then added to the current context.

In his IJCAI paper, Charniak alludes to the use of a list of "context frames" - those frames that had been mentioned in the story so far - which is to be used to cut down the search space. But he says essentially nothing about how much is to be moved in or out of that context a one time, e.g. is it just the affected frame statement?, the frame that contains it?, that frame plus the next level of frames pointed to?, etc.. (Semantic paging would suggest moving the entire painting complex at once.) Details of this sort become of extreme importance as the size of the knowledge base grows. While Charniak gives no answers in these papers, it is clear from his discussion of variable binding that he is concerned about the problem.

Can these frames plan?

Charniak does not speak to the "already known" vs. "must be deduced" dicotomy in conceptual links that I have taken to be a considerable part of the new consensus. However, by looking at his use of "inferences", we can decide for him that *his frames are good scripts but bad plans.*

The inferences Charniak discusses are involved with the purpose links between frames. The links often connect a complex event with general purpose rules or sub-events. Just because these are general purpose frames, their use of variables and the form of their goal statements may have been designed for descriptions which are not quite those that a given complex event uses itself. Since the matching process is not itself knowledge driven, making a cross-link instantiation will require bridging that descriptive gap by some kind of inferencing.

The interframe links would be used by an interpreter (say a program that was actually going to use them to tell it how to paint) to do consequent reasoning - breaking down a complex action into finer, more primitive actions, or to make predictions about "what would happen if". Alternatively, successive lines of a story

might "light up" statements in frames on either side of one of these links. And the story interpreter would then have to determine how the two were connected.

Charniak spends considerable time in the PAINTING paper discussing the "impedance matching" inferences which effect the bridging of variables between frames. These turn out to be independently motivatable rules of physical causation or logic. In principle, the pathway through these rules from one frame to the next could be computed on the fly - perhaps via means-ends analysis. That would make the frames PLANS of a sort. However, Charniak himself says that they should be pre-computed and entered directly in the event frame as part of the link structure, and does not provide any structure to the space of rules which make a heuristic planning process plausible.

Miscellaneous observations

Like everyone else, Charniak uses the term "frame" with his own intended meaning - one closer to Bartlett's original "schema" than to Minsky's "frame". Two central aspects of Minsky's frames which do not appear in Charniak's system are (1) the existence of "slots" with default values, and (2) frame transfer systems. The actions in Charniak's complex event frames do not occupy named slots (perhaps his variables do? -given that they are categorically described). The notion of a frame system with attendant transformations to take the "home" frame through different points of view (e.g. looking around a room) does not seem to have any counterpart in story understand at the level people can currently study. Possibly when programs can read real novels this will become relevant.

As thorough as Charniak's paper is in describing his representation, one still would like to know more. In particular the interpreter that is going to breath life into his frames is not discussed at all. Without the specifications of how the frame data structures can be manipulated, it is impossible to make any hard observations about what his system can and cannot do. As another AI programmer, one can "guess" at how he would design an interpreter, but there are a great many degrees of freedom with no guide as to what choices Charniak would make.

Wilensky

Wilensky may be said to have invented the notion of PLANS, or rather to have been the first to relate the planning process to the style of SCRIPTS. The kinds of inferences that his PAM program makes are akin to those made by Reiger or other

"commonsense inference" designs. What Wilensky did was to move that inferencing process over to the style of the new recognition paradigm of story understanding.

Wilensky's 1976 ACM paper is regrettably shallow in contrast with Rosenberg's paper, which is a technical working paper. Wilensky's paper was designed to introduce the concept of PLANS, and to convince a general audience that something like planning must go on when we understand unrestricted stories. This explains the lack of detail in the descriptions of the heuristic search process. Unfortunately, no more extensive documentation on PAM exists or can be expected before the end of this summer when he finishes his thesis (at Yale). As a result, the design of major parts of PAM, critical to a proper evaluation and comparison, is unavailable and can only be conjectured.

The structure of the knowledge base

As with all the projects in the Yale A.I. group, PAM is grounded in the conceptual dependency representation. This is the formalism that story sentences are converted into, and this is the formalism that the plan structures with which those sentences will match are composed from. However, the "primitives" of conceptual dependency (CD) have been considerably extended for plans and plan related research, so much so that is no longer possible to make categorical statements about CD's representational power or its utility as a programming formalism. There does remain a potentially serious question about how much the implemented heuristic search or matching process depends on using CD - or rather depend on manipulating concepts at just that level of delicacy. If that level had to be changed, could the processes accomodate? But at the moment, the system is not sufficiently specified to permit speculations.

On top of CD, the PLANS formalism has created: "themes", "goals", and "(named-)plans", and gives a special status to "raw" CD structures that denote "states" and "actions". In the story understanding process, the CD translations of story lines will only directly match states or actions. However, because of the structure of the knowledge base, the matches can be taken as evidence for the involvement in the story of particular "higher level", complex structures.

For Wilensky, story understanding revolves around infering the goals of the actors in the story and interpreting their actions in terms of those goals. PAM expects that a story will begin by giving some facts about its actors - a source of motivation for the plan-projection that PAM will have to make later. State expressions involving actors will be examined for evidence about "themes" that can be associated with them.

Themes are structured packages of goals. Therefore, once an actor has been associated with a given theme, then we can impute to them any of the goals that theme is associated with.

A "goal" is a structured package of plans, each of which might be a good way for achieving the goal under certain circumstances, i.e. goals may be annotated with prerequisites (and presumably other kinds of conditions). The plans are named and, of course, are shared among goals.

A "plan" is a structured package of events, ordered sequentially, which would accomplish the goal. The paper is not entirely clear about the distinction, but it would appear that an event can be either a CD structure describing some action or state change that should happen, or it may be an embedded goal, which would represent the possibility of there being many different ways ("sub-"plans) to achieve that plan-step.

Now exactly what a "structured package" (my phrase) is, we are not in a position to say because it is not publicly documented. We do not know how they are indexed in the knowledge base; we do not know how much semantic paging is done; we do not know how "salient" the different parts of their substructure are, and therefore do not know what operations the heuristic search routines would find difficult and which easy; we do not know how that relative difficulty would change with minor changes in the package structure or with more extensive cross-indexing.

The representation that results from understanding a story has two parts. The "backbone" of the representation is a sequence of CD statements corresponding to the "direct" translations of the lines of the story into CD. Overlaying and annotating that part are instantiations of the themes, goals, named-plans, and plan-steps with PAM linked to the story statements as it read. This compound structure is what is accessed in order to answer questions about the story. Nothing is said in the paper about how the answers are actually found but presumably the same technique as is used with SCRIPTS is used.

Modes of heuristic search

PAM operates in two modes, depending on the kind of story line being processed at the moment and the current state of the knowledge base. The first is reminiscent of Charniak's original work. Previous lines of the story will have caused themes to be instantiated for the some of the actors, or given evidence that the actors were in the midst of carrying out particular named-plans. These will have registered

"expectations" with the parser. . Expectations are demonic processes looking for matches with subsequent lines of text. When found, the match will trigger some action. The example actions in the paper were all to add explanatory links to the story data base. Presumably there would also be expectations with "meta-"actions - bookkeeping actions - which activated and deactivated say all the expectations associated with a given goal now found to be fulfilled.

The expectations operate via a matching process ("PLANMATCH") which is not described in the paper. How the matching is done is important to evaluating the system but very hard to "second guess". For example, we would like to know whether the matcher is allowed to search through the knowledge base, perhaps trying to chain goals to plans to actions to "unearth" statements that would match, or whether it is restricted to only already instantiated structures. We would like to know whether there is an indexing system that automatically "lights up" a matching statement anywhere in a specified part of the knowledge base, or whether some kind of "top-down" traversal of heuristically chosen structures is made.

The second mode is more of a pure heuristic search, in that there are not expectations to act as guides. Here we see the use of abstraction and what look to be "kind-of" hierarchies. When the second sentence of the story is read (*"One day, a dragon stole Mary from the castle"*), PAM finds that it can no longer work top-down because it has no expectations about this new character, the dragon. What it must do is, essentially, to generate speculations, bottom up, about what this dragon will do in the hope that at least one of these will make a connection ("match") with the active themes or goals of the earlier characters.

An operation of the same sort takes place when descriptions of states or state changes are noticed in the story - that is, a general search is required to see which of the active structures this description refers to. Similarly, actions in the story are to be interpreted as forwarding known or plausible goals. This requires search.

Again, Wilensky gives no details in this paper about how this heuristic search is done. But here we must be particularly skeptical and cannot be satisfied with the benefit of the doubt. What, exactly, is the program doing when it observes that a dragon is an "evil" character. What else does it know about dragons? Does that other information "get in the way of the search"? Is this translational inference not part of a general mechanism at all but patched directly into the program? This kind of search - translation to new terms and then looking for matches in the new form - is

the most interesting of all the kinds of reasoning that PAM is advertized to be able to do. But it is also the hardest, since the A.I. community at large has had the least experience with it.

Rosenberg

Rosenberg's research, as evidenced by his working paper and a recent revision of it to appear at the Canadian A.I. Conference this summer, is the most forward-looking of the three. He anticipates the time when programs will want to read "real" stories or news articles, where the plot will unfold gradually and assumptions will need to be updated, and where the programs themselves will come to the texts with a non-trivial knowledge of the domain and will be able to make independent evaluations of the assertions the texts make.

However, Rosenberg is hampered by a basic limitation of the formalism he uses, Roberts' and Goldstein's FRL, which makes it impossible for him to cover the same phenomena that are the forte of PAM, namely recognizing sequences of actions. This is for the simple reason that FRL has no primitives for sequences - they must be added as ad-hoc LISP code.

Rosenberg's research instead focuses on a problem that neither Wilensky or Charniak consider, though their mechanisms could be adapted to it. The problem is very specific: to find support for some hypothesis, where the hypothesis - assertion - may either have been asserted in the input text or (presumably) internally generated. The support may be either from facts given earlier in the text, facts already stored in the knowledge base, or by conjectures made (recursively) on the basis of general knowledge and or other facts.

To handle this problem, he has developed the clearest heuristic search mechanism of any of the papers. He has also developed clear mechanisms for reasoning with incomplete information and for continually monitoring changes so as to maintain the consistency of the assumed support relations - "dynamic updating". That kind of maintenance operation will be critical when programs begin to read material with "real life" complexity. Charniak's original work also did this kind of data base maintenance, but at the cost of considerable added complexity because he did not have the present mechanisms ("frames") for clumping the data into manageable pieces. I will not discuss these mechanisms further here except to say that they do "the sensible thing" given the current, informal opinions (at MIT at

least) on how monitoring operations should be implemented in planner-style data bases.

Heuristics for NOTICING

NOTICING is the process of discovering a "complex event" from amongst a large, diverse knowledge base of general and specific facts and events. The event consists of an assemblage of facts and events which are found to support the existence - correctness - of some "target event". The process of noticing adds annotating links to relevant datums which describe how they support the target event.

Everything in Rosenberg's system is a "frame", which is to say that each is an object with a name and a list of properties. There are three types of frames: "things", "rules", and "links". Each type has a expected set of properties ("slots"). Target events will be "things", and the rules will be used to find supported things (facts or events) already in the knowledge base or themselves independently justifiable which would count as evidence for the target event. If supporting facts are found, a link frame is instantiated to embody the support relation in a way that can be dynamically updated.

The NOTICER has one, general purpose procedure for trying to support a fact. The "heuristic" aspect of the process is recorded as data on the frames that the general purpose procedure manipulates. The target event corresponds to a "thing" frame. (Presumably this correspondence is instantiated by an earlier parsing - translation to canonical form process which Rosenberg does not discuss.) Every thing in the knowledge base includes a property ("slot") which is a list of rules which, if they applied in that situation, would support it.

The general procedure accesses this "rule" slot and applies the rules in it. If one of them "agrees" to support the target event then that is enough, if none of them can support it then there is a reason to "dis-believe" the event, and if no decision can be reached, the support decision is defered until more information becomes available.

Every rule has a test - a predicate against the current state of the knowledge base. The tests refer to specific data that may not be available at the time. If that occurs, then either of two things can be done. (Though the dynamic updating system will keep track of the "expectation" that the missing data implicitly created, and should the "expected" fact later become known, it will cause the system to go back and retry the test. All such processing is done at an interrupt level and (presumably) does not conflict with the directed process of looking for supports.)

One possible action is to look at the rule's "alternate" slot. It will contain rules that are weaker versions of the original rule, in that they will require less information for their test. However, when the precision of a rule's test goes down, the possibility that the rule will be missapplied - used in a situation where it isn't valid - will go up. This is protected against by the inclusion of "caveats" with the rules. These are tests which check that the assumption under which the rule are intended to apply do in fact hold. If a caveat turns up true, then its rule becomes inapplicable. (Note that when the general procedure goes to check out an alternative rule, it is applying itself recursively.)

The other alternative is look at the "support" slot. This will name some fact (or possibly an event) which, if it were true (supported), would be evidence that the rule went through. The general procedure will see if it can find a "rules" property ("slot") which is a list of rules which, if they applied in that situation, would support it.

The general procedure accesses this "rule" slot and applies the rules in it. If one of them "agrees" to support the target event then that is enough, if none of them can support it then there is a reason to "dis-believe" the event, and if no decision can be reached, the support decision is defered until more information becomes available.

Every rule has a test - a predicate against the current state of the knowledge base. The tests refer to specific data that may not be available at the time. If that occurs, then either of two things can be done. (Thought the dynamic updating system will keep track of the "expectation" that the missing data implicitly created, and should the "expected" fact later become known, it will cause the system to go back and retry the test. All such processing is done at an interrupt level and (presumably) does not conflict with the directed process of looking for supports.)

One possible action is to look at the rule's "alternate" slot. It will contain rules that are weaker versions of the original rule. In that they will less information in their test. However, when the precision of its test goes down, the possibility that a rule will be missapplied - used in a situation where it isn't valid - will go up. This is protected against by the inclusion of "caveats" with the rules. These are tests which check that the assumption under which the rule are intended to apply do in fact hold. If a caveat turns up true, then its rule becomes inapplicable. (Note that when the general procedure goes to check out an alternative rule, it is applying itself recursively.)

The other alternative is look at the "support" slot. This will name some fact (or possibly an event) which, if it were true (supported), would be evidence that the rule went through. The general procedure will see if it can "assume" the supporting fact by calling itself recursively with the fact as the new target event.

This procedure looks good. It is clear where it accesses heuristics; it is clear how its operations could be monitored and regulated. At this point the only way to further evaluate the system would be to actually write and debug programs for it. Charniak describes this situation clearly when he says: "the PAINTING complex has now become sufficiently elaborate that its computational implications are no longer very clear to me. Hence the need for a program".

Wishes yet to be fulfilled

The bulk of the capabilities which Rosenberg discusses in the first part of his paper have yet to be designed or implemented. The are a "wish list". This does not seriously detract from the impressiveness of the other parts of his system which do in fact exist. On the contrary, they are evidence that he is designing his system with these harder, eventually needed capabilities in mind.

Among his wishes (and everyone else's) are: -Being able to mediate conflicts between contradictory assumptions made at different times while exploring possible chains of support. -Being able to write stategies describing what rules to look at first under various conditions. -Having "meta-"rules which act to change those strategies dynamically. -Having evaluation metrics which describe how effective or reliable a given chain of support is. -Being able to (dynamically?) adjust those metrics with experience.

NOTICING as part of a complete story understander

In his paper, Rosenberg writes as though the NOTICING process were all that story understanding amounted to. He is probably correct that it is ultimately the most important part, or the most interesting, but he does not speak to the fact that his system is unable to appreciate sequences of events as instances of knowledge base structures. Every line in the text is supposed to be taken as a target event to be supported by the facts given by the earlier text or by general knowledge, as were the two example lines. But it is clear that this is not possible - that is, unless I greatly misunderstood Rosenberg's system. (E.g. what kind of a target event corresponds to a restaurant script?)

The story data base consists of assertions that are supported or not supported in various degrees. The notion of "support" is a close relative of the notion of "truth". Rosenberg is, in fact, dealing with data bases that are natural extensions of the truth-based possible worlds of logicians. Established arguments in the linguistics literature conclude that the notion of truth has very little to say about entire classes of utterances - by extension, these arguments apply to Rosenberg's story data bases as well. What would it mean, for example, to support a question? -or a command? -or a description of a sequence of actions?

Rosenberg's NOTICER must be augmented by PAM-style mechanisms to handle the kinds of understanding that searching for support relations is not applicable to. The NOTICER can only reach conclusions if a target event is first selected for it. In the example of the ground being wet, some undiscussed oracle must make the "guess" that maybe it has just rained, before the NOTICER mechanism can search out the linking rules that will justify the guess.

One necessary extension will be routines that decide when it would be useful to process some part of a text as a target event. The support-finding and maintaining apparatus should be very effective at performing the kind of heuristic searches that PLANS call for, if properly phrased target events can be constructed out of the the text.

For example, we might have: (from Rosenberg's 1976 paper)

"*John wished to be chairman of the department*"

"*He went out and bought a gun*"

In terms of PLANS, this is a perfect case of needing to find the goal that the gun-buying action is a part of, given that it should somehow be connected with "becoming department chairman". The technique Rosenberg uses of annotating "thing" frames with the names of rules that would support their existence could be directly applied here - backward chaining through the rules for "becoming chairman" and forward chaining from "what could you do with a gun". Some kind of marking operation, a la Quillian for example, could signal when a connection was made.

It is a fair question to ask if what the NOTICER does can really be called story understanding, given that it is as yet incapable of the basic common sense inferences of PAM. The answer, I believe, is yes - that what the NOTICER does in fact reflects the most advanced kind of story understanding. The NOTICER is the first program that really reads between the lines, making conjectures, arguing out justifications (it could

easily disagree with the author's opinions for example), expecting the author to soon give justify his assumptions and being prepared to be surprised at them, etc. These are the phenomena of which real reading is constituted.

Outstanding problems

The vocabulary problem

With the shift to a non-procedural, recognition paradigm, certain problems become magnified which before could be swept under the rug of clever encoding. None of the three authors discuss the process that converts texts into canonical form, and only Rosenberg specifies the matcher that then relates them to the forms in the knowledge base. (i.e. it is trivial - rules are "matched" with facts by reading out a pre-computed list of candidates from the "rules" slot.)

The trade-off between how delicately a text is broken down into "canonical" conceptualizations and how much power the matcher is given to generalize has been mentioned throughout this paper. We cannot tell what a understanding system is really capable of without knowing in detail how it plays this tradeoff.

The choice of vocabulary for the canonical descriptions of the story text plays a crucial role in the potential modularity of the system. For a rule to be noticed, the matcher (or rather the heuristic search mechanism as a whole) must be able to appreciate that it is applicable to the current situation. If, after a system is intially developed, someone wants to add a new rule, they must insure that the description of the rule uses the same descriptive vocabulary as the facts or events that the rule is intended to refer to. This is in large part a matter of human-engineering: how much of the detail of the existing system must a person know to be able to insure that they will be able to select effective descriptions for the facts and rules that they want to add to it.

To date, the story domains that programs have been designed to understand have been quite limited. It has been possible to design their knowledge bases by "brute force", i.e. each person learns the content of the entire system. This will obviously not be possible in the future unless some sort of abbreviation capability is introduced. This is the direction that I personally believe the canonical description languages will have to go. When a person defines a new term of the vocabulary, they assign it a position in a common network of abstract categories and features. When the matcher manipulates a formula that includes the term, it is prepared to

make associations on the basis of these "type" descriptors as well as directly with the new term. SCRIPTS *would be rules, themes, etc. which were written in specific terms.* PLANS *would be rules, themes, etc. which were written in the common language of categories and features.*

Noticing the discourse structure

None of these papers even alluded to the possibility that it might be useful to include as part of the representation of stories a description of the "discourse structures" by which the content of the story was conveyed in English. There is no sign, for example, that PAM paid heed to the *but* in "*John loved Mary but she didn't want to marry him*". Charniak's system did not use the explicit introductory participle in: "*John had to paint a chair. Finding he had no paintbrush, he proceeded to dip his hand into the paint and ...*", which could have served to short-circuit the search process that deduced that his hand was now serving as a "painting instrument".

Of course, considering that only a small amount of work has been done to date in taxonomizing and formalizing discourse structures, it is perhaps understandable that the authors of these papers do not make use of them in their representation. Also, the additional information that one would would then get through the recognition of discourse structures will likely be higher-order facts which existing reasoning systems would not know how to use. That is, discourse structure will identify different ordering strategies in the presentation of items in the text, or notice that certain function words (like *but*) have or have not been used. These differing stategies apparently make a great deal of difference to how easily a human being will understand the message in a text. But no one at the moment understands the understanding process that humans actually use well enough to know how it is that these strategies are an aid. Therefore, except for explicitly experimental systems, there would be no use to having a description of the discourse structure available to a story understanding program.

However, there are far more "mundane" phenomena, such as determining the references of definite noun phrases, which, it is becoming clear, depend crucially on recovering a discourse structure of some sort. For example, the PAM story refers to "*the castle*". It is impossible to properly understand that reference without already having constructed a context that specifies that the story takes place in "days of olde", that Mary is a princess or otherwise someone who you would expect to be living in a castle, and so on. These systems have avoided using this kind of discourse structure

either by ignoring the references, or by restricting the domain and wording of the stories so that they never arise. This cannot be kept up much longer.

References

Charniak [1972] Toward a Model of Children's Story Comprehension, Technical report #266, MIT A.I. lab.

_____ [1977] *"A Framed PAINTING: the Representation of a Common Sense Knowledge Fragment*, Cognitive Science, 1, 4.

_____ [1977] Ms. MALAPROP, A Language Comprehension Program, in the proceedings of IJCAI-77, August 1977, MIT.

McDonald [1978] A Simultaneously Procedural and Declarative Data Structure and its Use in Language Generation, to appear in the proceedings of Canadian A.I. Conference, July 1978, Toronto.

Mehan [1976] TALESPIN: A Program for Generating Stories, Doctoral Dissertation, Yale University.

Reiger [1974] CONCEPTUAL MEMORY: A Theory and Computer Program for Processing the Meaning Content of Natural Language Utterances, memo AIM-233, Stanford A.I. lab.

Rosenberg [1976] Discourse Structure, MIT A.I. working paper 130.

_____ [1977] *Reporter: An Intelligent Noticer*, MIT A.I. Lab. Working Paper 156.

_____ [1978] SLEUTH: An Intelligent Noticer, memo 475, MIT A.I. lab.

Schank & Abelson [1977] Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures, Lawrence Erlbaum Assoc., Hillsdale N.J.

Schank & the Yale A.I. Project [1975] SAM -- A Story Understander, Research report #43, Dept. of Computer Science, Yale University.

Sidner (formerly Bullwinkle) [1978] A Progress Report on the Discourse and Reference Components of PAL, MIT A.I. memo 468.

Stansfield [1977] COMEX: A Support System for a Commodities Analysist, MIT A.I. memo 423.

Robert Wilensky [1976] *Using Plans to Understand Natural Language*, Proceedings of the ACM, Houston, July 1976.