ED 161 024 CS 204 270

AUTHOR Schank, Roger C.; And Others
TITLE Sam--A Story Understander. Research Report No. 43.
INSTITUTION Yale Univ., New Haven, Conn. Dept. of Computer Science.
SPONS AGENCY Advanced Research Projects Agency (DOD), Washington, D.C.; Office of Naval Research, Arlington, Va.
PUB DATE Aug 75
CONTRACT N00014-75-C-1111
NOTE 45p.

EDRS PRICE MF-$0.83 HC-$2.06 Plus Postage.
DESCRIPTORS Chinese; Cognitive Processes; *Computer Programs; *Conceptual Schemes; *Connected Discourse; *Data Processing; *Discourse Analysis; Language Patterns; Prose; *Research Projects

ABSTRACT

SAM (Script Applier Mechanism), a computer program designed to understand stories that rely heavily on scripts (typical sequences of events in particular contexts), is described in this report. Chapter one, which discusses SAM's background, shows how causal chaining was developed to connect events in stories, presents a typical script, and explains the general form for a script. The following chapter presents examples to show how SAM processes stories by creating a linked causal chain of conceptualizations that represent what took place and then generating the output back in English. Chapter three describes the following components of SAM: the English-to-conceptual dependency analyzer; the EXEC (executive program), which decides which script is required for each input; the script applier, which constructs a story representation from conceptual dependency input; the generator, which produces an English sentence as an output; and the Chinese generator, which can translate the output into Chinese. The chapter also explains how SAM creates paraphrases and summaries of processed stories and how it answers four types of questions that rely on information in a script. A brief concluding chapter notes that SAM's significance lies in its provision of a test for a theory of understanding based on scripts. (GW)

This work was supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored under the Office of Naval Research under contract N00014-75-C-1111.

The Yale A.I. project is composed of: Robert Abelson, Richard Cullingford, Gerald DeJong, Leila Habib, Wendy Lehnert, James Meehan, Richard Proudfoot, Chris Riesbeck, Roger Schank, Walter Stützman, and Robert Wilensky. All of the members of the A.I. project contributed to writing this paper, programming a piece of SAM, the ideas behind SAM, or all three.

SAM -- A Story Understander

Roger C. Schank and
the Yale A.I. Project

Research Report #43

August 1975

Yale University
Department of Computer S_ien

I. Background

In 1973 we designed and built the MARGIE system [Schank et al, 1975, and Schank, 1975]. MARGIE dealt with individual sentences in isolation for the most part. We built MARGIE primarily to test theories about the individual parts of MARGIE rather than because of any desire to create a useful system. We felt that MARGIE was successful because we found that we could parse directly into Conceptual Dependency from English, bypassing syntactic analysis per se [Riesbeck, 1975]. We learned a great deal about inference and memory and saw that we could use the primitive actions as the basis of an inference organization scheme [Rieger, 1975]. Finally we showed that it was possible to get out of Conceptual Dependency and into English again without loss of information [Goldman, 1975].

Two main problems were exemplified by MARGIE that we considered important issues for future research. One was the issue of the connectivity and interrelationship of sentences in text. It is not always possible to disambiguate sentences in isolation. Yet in context, such sentences often have only one obvious meaning. We were concerned with how to deal with this problem. Furthermore, parsing texts seemed to be more than just parsing the individual sentences that made up the texts. Just as there is implicit information within a sentence, so there is information implicit within the conjunction of two sentences that is not explicit in either of them. Paragraphs have a coherency to them just as sentences do. The fact that there can be nonsense paragraphs would indicate that there is an over-all organizational flow to paragraphs (and larger texts) that must be sought out in the parsing of those paragraphs.

The second problem was the seemingly endless expansion of the inference process. Rieger [1975] hypothesized that inference was an unconscious process of expansion based on the knowledge associated with an input conceptualization. But the number of inferences obtained from an input in the MARGIE system was just too large to work with. It seemed that there must be some method by which inferencing could be cut off or focussed such that the important inferences would be central and the unimportant ones would be ignored.

After MARGIE was completed we began to attack both of these problems. We started by looking at the problem of the representation of connected text. Schank [1973 and 1974] showed that the principal element in the solution of this problem was the causal chain. In order to know when some element must be inferred, it is necessary to know that there is a gap in the text. If we have "John was mowing the lawn. Suddenly he felt a pain in his toe," we must be able to figure out the connection between these two items. We invented a syntax of causality that said that actions can cause state changes and state changes can enable actions. We then applied a semantics of causality to relate specific actions and states. For the example above we know that there is an action and a state change. The semantics disallows "PROPELling something into grass" as a way of causing "PAIN in a toe." We are forced to hypothesize a physical contact between something in the story and the toe th[illegible] could cause pain This causes us to infer that "John pushed the la[illegible] across his toe." [illegible]s with all inferences, this particular one could be wrong. The general pri[illegible]le, however, is important. In order to make an inference about what events are implied by a story, it is crucial to understand that such events are missing and to be able to figure out the properties of these missing events.

We were able to use causal chains to connect events in entire stories, predicting resolutions of problems posed in a story and so on. Using these chains certain items got connected more frequently than others, and we created a paraphrase hypothesis that marked as important events linked in more than one chain in a story and marked as "forgettable" events that were without consequences.

With the principle of causal chaining established, we then became concerned with examples where the causal chain to be inferred was simply too long to be gotten from ACTs and states on either end of the gap. There comes a point where unless you have specific knowledge about the situation that you are in it is hard to understand the relationship between seemingly unrelated events. Our solution to this problem is what we labeled [Schank & Abelson, 1975] scripts.

A script is a preformed sequence of actions that constitutes the natural order of a piece of knowledge. For example, consider the sequence "John went to a restaurant. He found a table and ordered a hamburger. Later, he paid and left." Unless we have detailed knowledge about restaurants (the restaurant script) we cannot easily connect finding tables and ordering. Nor could we answer the question "What did John eat?" Any person who knows about restaurants could, however, do these things. Scripts, then, serve to fill in the gaps in a causal chain when they can't be inferred just by themselves. That is, scripts form the knowledge source that we can rely on in understanding. (Although the ideas were developed independently, scripts conform well to one part of Minsky's frame idea [Minsky, 1974].)

Scripts are intended to handle the range of events that are the most mundane. Thus we would expect a birthday party script, a restaurant script, an airplane traveling script, a going to the doctor script, and so on. Scripts will not account for things about which there is no specific detailed knowledge. We would expect that most people do not have a how to become president script, a what to do when the house burns down script, or a how to fix an oscillator script. On the other hand, some people do have such scripts.

Thus, a script is a structure that describes an appropriate sequence of events in a particular context. A script is made up of slots and requirements about what can fill those slots. The structure is an interconnected whole, and what is in one slot affects what can be in another. Scripts handle stylized everyday situations. They are not subject to much change, nor do they provide the apparatus for handling novel situations.

For our purposes, a script is a predetermined, stereotyped sequence of actions that define a well-known situation. A script is, in effect, a very boring little story. Scripts allow for new references to objects within them just as if these objects had been previously mentioned; objects within a script may take "the" without explicit introduction because the script itself has already implicitly introduced them. (This can be found below, in the reference to "the waitress" in a restaurant, for example.) Stories can involve scripts in various ways. Usually a story is a script with some interesting deviations.

I. John went into the restaurant. He ordered a hamburger and a coke. He asked the waitress for the check and left.

II. John went to a restaurant. He ordered a hamburger. It was cold when the waitress brought it. He left her a very small tip.

III. Harriet went to a birthday party. She put on a green paper hat. Just when they sat down to eat the cake, a piece of plaster fell from the ceiling onto the table. She was lucky, because the dust didn't get all over her hair.

IV. Harriet went to Jack's birthday party. The cake tasted awful. Harriet left Jack's mother a very small tip.

Paragraph I is an unmodified script. It is dull. It would be even duller if all the events in the standard restaurant script (see below) were included.

Paragraph II is a restaurant script with a stock variation, a customer's typical reaction when things go wrong.

Paragraph III invokes the birthday party script, but something wholly outside the range of normal birthday parties occurs -- the plaster falls from the ceiling. This deviation from the script takes over the initiative in the narrative until the problem it raises is resolved, but the birthday script is still available in the indirect reference to the party hat and in the possibility that normal party activities be resumed later in the narrative. It seems natural for reference to be made to dust in the hair following the plaster's falling, which implies that there is a kind of script for falling plaster too. (This kind of script we call a vignette [Abelson, 1975].) Notice that "the ceiling" refers to an uninteresting "room" script that can be used for references to doors and windows that may occur. Thus it is possible to be in more than one script at a time.

Paragraph IV illustrates the kind of absurdity that arises when an action from one script is arbitrarily inserted into another. That one feels the absurdity is an indication that scripts are in inadmissable competition. It is conceivable that with adequate introduction the absurdity in paragraph IV could be eliminated.

With these examples, a number of issues have been raised. Let us at this point give a more extensive description of scripts. We have discussed previously [Schank, 1974] how paragraphs are represented in memory as causal chains. This work implies that, for a story to be understood, inferences must connect each input conceptualization to all the others in the story that relate to it. This connection process is facilitated tremendously by the use of scripts.

Each script has players who assume roles in the action. A script takes the point of view of one of these players, and it often changes when it is viewed from another player's point of view.

The following is a sketch of a script for a restaurant from the point of view of the customer. Actions ~~are~~ specified in terms of the primitive ACTs of Conceptual Dependency theory [Schank, 1973].

script: restaurant
roles: customer, waitress, chef, cashier
reason: to get food so as to go up in pleasure and down in hunger

scene 1: entering
- PTRANS self into restaurant
- ATTEND eyes to where empty tables are
- MBUILD where to sit
- PTRANS self to table
- MOVE sit down

scene 2: ordering
ATRANS receive menu
MTRANS read menu
MBUILD decide what self wants
MTRANS order to waitress

scene 3: eating
ATRANS receive food
INGEST food

scene 4: exiting
MTRANS ask for check
ATRANS receive check
ATRANS tip to waitress
PTRANS self to cashier
ATRANS money to cashier
PTRANS self out of restaurant

In this script, the instruments for performing an action might vary with circumstances. For example, in scene 2 the order might be spoken, or written down with predesignated numbers for each item, or even (in a foreign country with an unfamiliar language) indicated by pointing or gesturing.

Each act sequence uses the principle of causal chaining [Schank, 1973, and Abelson, 1973]. That is, each action results in conditions that enable the next to occur. To perform the next act in the sequence, the previous acts must be completed satisfactorily. If they cannot be, the hitches must be dealt with. Perhaps a new action not prescribed in the script will be generated in order to get things moving again. This "what-if" behavior is an important component of scripts. It is associated with many of the deviations in stories such as paragraph II.

In a text, new script information is interpreted in terms of its place in one of the causal chains within the script. Thus in paragraph I the first sentence describes the first action in scene 1 of the restaurant script. Sentence 2 refers to the last action of scene 2, and Sentence 3 to the first and last actions of scene 4. The final interpretation of paragraph I contains the entire restaurant script, with specific statements filled in and missing statements (that he sat down, for example) assumed.

In paragraph II, the first two sentences describe actions in scenes 1 and 2. Part of the third sentence is in the script as the first action of scene 3, but there is also the information that the hamburger is cold. The fourth sentence ("He left her a very small tip") is a modification of the third action of scene 4. The modifier "very small" is presumably related to the unexpected information about the "cold hamburger." Even a stupid processor, checking paragraph II against the standard restaurant script, could come up with the low-level hypothesis that the small size of the tip must have something to do with the temperature of the hamburger, since these two items of information are the only deviations from the script. They must be related deviations, because if they were unrelated the narrative would have no business ending with two such unexplained features.

Of course we do not want our processor to be stupid. In slightly more complex examples, adequate understanding requires attention to the nature of deviations from the script. A smarter processor can infer from a cold hamburger that the INGEST in scene 3 will then violate the pleasure goal for going to a restaurant. The concept of a very small tip can be stored with the restaurant script as a what-if associated with violations of the pleasure goal.

The general form for a script, then, is a set of paths joined at certain crucial points that define the script. For restaurants the crucial parts are the INGEST and the ATRANS of money. There are many normal ways to move from point to point. Ordering may be done by MTRANSing to a waiter or by selecting and taking what you like (in a cafeteria). Likewise the ATRANS of money may be done by going to the cashier, or paying the waitress, or saying; "Put it on my bill." There are also paths to take when situations don't go as planned. Paragraphs III and IV call up deviant paths in the birthday party script. All these variations indicate that a script is not a simple list of events but rather a linked causal chain; a script can branch into multiple possible paths that come together at crucial defining points.

To know when a script is appropriate, script headers are necessary. These headers define the circumstances under which a script is called into play. The headers for the restaurant script are concepts having to do with hunger, restaurants, and so on in the context of a plan of action for getting fed. Obviously contexts must be restricted to avoid calling up the restaurant script for sentences that use the word "restaurant" as a place ("Fuel oil was delivered to the restaurant").

Scripts organize new inputs in terms of previously stored knowledge. In paragraph I, many items that are part of the restaurant script are added to the final interpretation of the story. We don't need to say that a waitress took the customer's order or that he ate the hamburger. These ideas are firmly a part of the story because the restaurant script requires them. In understanding a story that calls up a script, the script becomes part of the story even when it is not spelled out. The answer to the question "Who served

John the hamburger?" seems obvious, because our world knowledge, as embodied in scripts, answers it.

II. SAM

SAM (Script Applier Mechanism) is a program running at Yale that was designed to understand stories that rely heavily on scripts. Below we present three stories, each of a different type. Story I makes references to a script and then stops the script in midstream. Story II is a standard boring story that adheres closely to script information. Story III calls up more than one script as well as having a complication arise in one script as a result of an odd occurrence in a previous one.

SAM understands these stories and others like them. By "understand" we mean SAM can create a linked causal chain of conceptualizations that represent what took place in each story. SAM parses the story into input conceptualizations that are fed to an executive program that looks for script applicability. When a script seems to be applicable, the script applier makes inferences about events that must have occurred between events it was specifically told about. When the applier finishes a script (i.e. when new inputs do not fit into it) it sends control back to the executive.

The final output is a gigantic Conceptual Dependency network. We could claim that this output indicates understanding, but as no one can read it (and for the more obvious reasons) we have developed programs that operate on the output of the understanding program. We have developed programs to generate the final output back in English. These programs constitute a paraphraser. The paraphrases obtained are longer than the original because inferences made by the script applier are retained. We also generate shorter paraphrases that are closer to the original and summaries that rely on measures of the relative importance of events within a script.

In addition, we have developed a program that can query the obtained representation so as to answer questions about the input story.

Since we have often claimed that Conceptual Dependency is interlingual and that generation in English is no harder for us than in any other language, we have also written a program to translate the stories we understand into Chinese. The translation program works by taking the output from the script applier and using Chinese data in conjunction with Goldman's program. Because we use the script applier output, our translation is longer than the original input in the same way that the long paraphrase expands on the story. It is a simple matter to make the translation conform more directly to the input, but we haven't bothered to do this. We feel that a translation that elaborates on an input text is a better indicator of understanding and the use of knowledge in translation than one that tries to reproduce faithfully the original text. We are trying even in this task to reflect human understanding processes.

Below we have some examples of input and the various outputs that SAM produces:

Input: John went to a restaurant. He sat down. He got mad. He left.

Long paraphrase:

John was hungry. He decided to go to a restaurant. He went to one. He sat down in a chair. A waiter did not go to the table. John became upset. He decided he was going to leave the restaurant. He left it.

Input: John went to a restaurant. The hostess seated John. The hostess gave John a menu. John ordered a lobster. He was served quickly. He left a large tip. He left the restaurant.

Long paraphrase:

John decided he was going to go to a restaurant. He went to one. He asked the hostess for a table. She told him he could go to it. He went to it. She went to it. He sat down in a chair. He got a menu from her. He read it. A waiter saw John was at the table. The waiter went to it. John ordered a lobster. A chef cooked it. The waiter got it from the chef. The waiter served it to John quickly. He ate it. He asked the waiter for a check. John got it from the waiter. John remembered the waiter had served the lobster to him quickly. John left the waiter a large tip. John paid the check. He left the restaurant.

Summary: John went to a restaurant and he ate a lobster.

In the stories above SAM works by analyzing each sentence into a Conceptual Dependency representation. If this representation calls a script, a script is brought into memory. Future inputs are analyzed and then matched against the script. If a new input matches, any necessary steps that are known to have been skipped between the first input and the second are inferred to have happened. This continues until there are no new inputs or until a new input does not match a part of the current script.

If the new input calls another script, then that script is brought in (see below for computer example of this). If the new input is unexpected we try to resolve its place in context. In the first story we do not expect John to get mad. Where we get that input we try to figure out why he got mad and look for the next expected event to see whether its non-occurrence might cause anger for a customer in a restaurant. Since it does, we make that inference.

Below are questions to SAM and answers that SAM produced after reading story II.

Q: Why did the hostess give John a menu?

A: So John could ask the waiter for the meal.

Q: Why did John go to a restaurant?

A: So John could eat a meal.

Q: Did the waiter give John a menu?

A: John got a menu from the hostess.

Q: Why did John leave the waiter a large tip?

A: Probably John remembered the waiter served a lobster to him quickly.

Q: Did John pay the bill?

A: John paid the check.

The Chinese translation differs from the paraphrase program at the last point, namely the mapping of the output of the paraphraser into a natural language. It was not very difficult to modify Goldman's program to do this. We would estimate that we can translate from any English story that we can parse.

Below is the translation SAM produced for Story II (Harvard Romanization):

John sheang [illegible] daw fangguar chiuh; ta daw fangguar chiuh le. Ta ching daytzuohde jiaw le [illegible] juotz, [illegible] jiaw John kee yi daw juotz nall chiuh. Ta daw juotz nall chiuh le. Daytzuohde daw juotz nall chiuh le. John tzuohshiah le. Ta tsorng daytzuohde nadaw i jyy tsaydann. John kann le tsaydann. Charfarng kann le John tzay ney ge juotz nall. Charfarng daw juotz nall chiuh le. John dean le i ge lungshia. Dahshyrfu ba lungshia yuh bei hao le. Charfarng tsorng dahshyrfu jiedaw lungshia. Charfarng geei le John lungshia geei de hen kuay. Ta chr le lungshia. Ta ching charfarng na i ge janqdhan lai. John tsorng charfarng nadaw janqdhan. John jihde charfarng geei ta ney ge lungshia geei de hen kuay. John geei charfarng lioushiah le hen duo sheaufei. John fu le chyan. Ta tsorng ney ge fangguar chuhchiuh le.

Story III was handled by SAM 2, a more advanced version of SAM 1 (written two months later). SAM 2 handles multiple scripts, unexpected events, and more complicated constructions in English.

Input: John went to New York by bus. On the bus he talked to an old lady. When he left the bus, he thanked the driver. He took the subway to Leone's. On the subway his pocket was picked. He got off the train and entered Leone's. He had some lasagna. When the check came, he discovered he couldn't pay. The management told him he would have to wash dishes. When he left, he caught a bus to New Haven.

Long paraphrase:

John went to a bus stop. He waited at it a few minutes. He entered a bus. The driver got the ticket from John. He went to a seat. He sat down in it. While John was on the bus an old lady and John talked. The driver took John to New York. He went to the driver. When getting off the bus John thanked the driver. John got off it.

He entered a station. He put a token into the turnstile. He went to the platform. He waited at it a minutes. He entered a subway car. A thief went to John. The thief picked John's pocket. He went. John went to the seat. He sat down in it. The driver took John to Leone's. He left the subway car. He left the station.

He entered Leone's. He looked around inside it. He saw he could go to a table. He went to it. He sat down in the seat. He ordered some lasagna. The waiter indicated to the chef John would like him to prepare something. The chef prepared the lasagna. The waiter got it from the chef. The waiter went to the table. He served the lasagna to John. He ate it. He became full.

He asked the waiter for the check. John got it from the waiter. John read the check. John discovered he was unable to pay the check. He indicated to the waiter he was unable to pay the check. The management told John he would have had to wash dishes. He entered the kitchen. He washed dishes. He left Leone's.

He went to the bus stop. He waited at it a few minutes. He entered the bus. The driver got the ticket from John. He went to the seat. He sat down in it. The driver took John to New Haven. He got off the bus.

[Paragraphing has been added to the computer output for ease of reading.]

III. SAM in More Detail

We will now describe in a little more detail the components that make up SAM.

A. The English Analysis Program: Riesbeck

The first program in the SAM system is the English-to-Conceptual-Dependency analyzer. It is the job of this program to take the input text and extract from it all the conceptual information conveyed by the linguistic elements of the text. Later programs in the system use the output of the anal[illegible] [illegible]nceptual Dependency and never deal wi[illegible] [illegible]eatures of the language. Only the [illegible]alyzer considers problems of word meaning, inflections, ordering relationships, and other idiosyncracies of linguistic expression.

The English analyzer is an extension of the one described in Riesbeck [1975]. That analyzer extracted the conceptual meaning from short texts of a few sentences each. The SAM project needed an analyzer capable of handling texts of normal paragraph length. This necessitated two areas of work:

1. Research into what made a text a unified structure rather than just a list of unrelated sentences.
2. Extension of the analyzer to allow it to combine the information contained in these larger structures with the knowledge it already had about English.

The earlier program was designed according to two basic considerations:

1. The important task for a language processing component in a large understanding system is the extraction of meaning from texts. It should do this in the most direct way possible, using tools such as syntactic analysis only where necessary.

2. The process of understanding at all levels, including the level of language processing, requires the ability to predict intelligently, based on what has already been understood, what things will be seen later in the text and what they will mean.

The earlier program worked by using the words in the input text to access routines — called expectations -- that predicted what conceptual and linguistic structures were likely to occur later in the text. The expectations also specified what additional meaning structures should be built (using the Conceptual Dependency representation system) if these structures were encountered.

The present analysis program combines the notion of frames, i.e. static structures organizing sequences of events, with this notion of the expectation routine. Frame structures are of various sizes, from the small CD descriptions of simple events to large scripts of event sequences. When SAM sees a reference to a frame in the text, it starts building an instantiated copy of the frame. Parts of the structure are already filled but other parts are not. The empty slots and the conditions on the values they will eventually have direct the course of analysis.

The conditions associated with an empty slot specify what sorts of structures might fill this slot. When the expectation routines are accessed, the structures they are capable of building are compared with these assumptions. Each expectation that builds a structure satisfying the conditions placed on some empty slot is tied to that slot. An expectation is kept active until either it is triggered or the slot to which it is tied is filled by some other expectation.

By associating the expectation routines with slots to be filled, the analyzer controls the expectations, combining those that serve the same function, removing those that are no longer necessary, and handling in a uniform way not only expectations that fill out small CD templates but also those that fill out larger event sequences — i.e. scripts. This allows the structures predicted by an expectation to be refined by the higher-level assumptions placed on the slot that the expectation fills.

Consider again Story III:

John went to New York by bus. On the bus he talked to an old lady. When he left the bus, he thanked the driver. He took the subway to Leone's. On the subway his pocket was picked. He got off the train and entered Leone's. He had some lasagna. When the check came, he discovered he couldn't pay. The management told him he would have to wash dishes. When he left, he caught a bus to New Haven.

In this story there are instances where the meaning of a verb depends on the objects attached to it — "took" in "took the subway," "had" in "had some cheesecake," "came" in "the check came," etc. There are the various structures of clauses and phrases that communicate time relationships between events — "on the subway," "when the check came," "he would have to," etc. Of greater theoretical interest, however, are those places where the SAM system required more than a knowledge of English in order to assign a meaning to a piece of text. For example, to realize that the phrase "the check came" means that the waiter (probably) brought the check to John required knowing who does what in restaurants and that this particular text is about John's going to a restaurant.

The structure "when X, Y" is interesting in that it can express either "while X,Y" or "after X,Y." In the example paragraph both uses of "when" occur — "when [while] he left the bus, he thanked the driver" and "when [after] he left, he caught a bus to New Haven." In order to assign the likeliest time relationship, SAM needed to know where the driver of the bus is when people are leaving and that buses normally do not pass through restaurants.

Besides allowing knowledge from various sources to interact, the expectation approach makes long texts manageable because word senses are decided on as they are seen. Meanings for very ambiguous words, such as prepositions, are set up in advance by expectations attached to the verb and other elements of the sentence. The approach used in some purely syntactic systems of keeping all possible analyses leads to generation of an awkward number of possibilities with simple sentences and becomes unworkable for texts of paragraph length, where the sentences themselves may be quite lengthy. This is because each ambiguity multiplies the number of possible interpretations that must be kept. A text analyzer must be able to make intelligent assumptions about word meanings as it goes along if it is to avoid combinatorial explosion. By embedding expectation routines within CD forms, which are in turn embedded in larger script structures, the current analysis program is able to use general world knowledge such as scripts together with language-specific knowledge about English to make intelligent guesses about the meaning of a text in a straightforward one-pass manner.

The new version of the analyzer is written in MLISP and runs on the PDP-10 computer at Yale. In interpreted form it takes approximately 40K of core to do texts of several sentences and 50K to do the longest texts that the

SAM system has tackled. Sentences take between 5 and 10 seconds to be analyzed, not including garbage-collecting overhead in the LISP system (between 0 and 10 seconds).

B. Overview of the EXEC: Meehan and Proudfoot

When stories contain more than one script it is necessary to decide when a script is to be called in and when it is finished. SAM has an executive program (EXEC) that decides which script is required for each input from the parser. The applier mechanism works in one "script context" at a time; when it is running, it is not "aware" of the other scripts. One of the chief functions of the EXEC is to set up the correct script context before calling the applier. (This means that the applier's control structure is equivalent to a set of coroutines.)

How does the EXEC know what script should handle a given input? Sometimes the parser has explicitly specified the name of the script, as in the representation of "John went hunting" or "while John was on the bus." But at other times the EXEC must inquire of each script whether it can handle the present input. Part of the context of each script is a list of expected inputs, called the "search queue." A pattern match is done with each element of the search queue. If the match succeeds, the applier is called in the context of that script. Initially, the search queue of a script contains those events that could reasonably be assumed to "introduce" the script, such as "John went to a restaurant."

There are two sets of problems that the EXEC must handle. The first set includes actions to be taken when all or part of a sentence is "weird" --

that is, not understood by any script. A weird sentence is marked as such and is otherwise ignored by the EXEC. Future versions of the EXEC will include programs to make inferences from weird inputs. (The applier makes the inferences for the non-weird inputs.) In a story in which John gets his pocket picked and later has to wash dishes to pay for a meal, the applier, working in the context of the restaurant script, will want to know whether the concept of John's having no money has been seen before. That would be an inference from the "weird" pocket-picking event.

A weird part of a non-weird sentence might be a reference to a character outside the active script, and since the EXEC has access to all the scripts it can resolve such references. For example, if John is eating in a restaurant, the restaurant script is active. But if during the meal John feels ill and gets the waitress to bring him a glass of water for his pills, then the sentence "The waitress brought John a glass of water" has a weird part from the perspective of the illness script. The fact that someone brings John water makes sense in terms of that script. What's weird is "the waitress" since there's no waitress in the illness script. So the applier asks the EXEC whether it knows who the waitress is. The EXEC looks at the script contexts of all the scripts and finds "waitress" mentioned in the restaurant script, so it says yes.

The second set of problems for the EXEC is the interface between scripts: How do they start and stop? When is a script finished as opposed to being interrupted? In theory, there are (at least) three kinds of script interfaces: sequential ("John took a bus to town and went shopping"), nested ("John made a phone call from the restaurant"), and parallel ("John and Bill swapped old stories over a l ng lunch"). The current EXEC can handle some examples of all three cases, but more work remains to be done in developing the theory of script interfaces.

C. Script Applier: Cullingford

Construction of a story representation from CD input supplied by the parser is the job of the script applier portion of SAM. Under control of the EXEC, the applier locates each new input in its data base of situational scripts, links it up with what has gone before, and updates its predictions about what is likely to happen next. Since the SAM system as a whole is intended to model human understanding of simple, script-like stories, the applier organizes its output into a form suitable for later summary, paraphrase, and question-answering processing

Situational scripts: As implemented in SAM, a situational script [Schank & Abelson, 1975] is a network of CD patterns describing the major paths and turning points of a common situation. These patterns are of two general types: events, which we will construe broadly as including states and state-changes as well as mental and physical acts; and causal relations among these events [Schank, 1974a]. Patterns are used in the script not only because of the variety of possible fillers for the roles in the script but also to provide the minimum amount of information needed to understand a story input. Thus, for example, the applier uses a pattern like:

```
((ACTOR (&X) <=> (*PTRANS*) OBJECT (&X) TO
          (*INSIDE* PART (&RESTAURANT)))
```

to identify inputs like:

John went to Lindy's.
John walked into Lindy's.
John came into Lindy's from the subway.

(&X and &RESTAURANT are dummy variables.) This allows the applier to ignore inessential features of an input (like the Instrument of the underlying ACT or the place John came from in the examples given above) and thus provides a crude beginning for a theory of forgetting.

At the present time, SAM possesses three "regular" scripts, one for riding on a bus, one for riding on a subway, and one for going to a restaurant. These scripts have been simplified in various ways. For example, all of them assume that there is only a single main actor. The bus script has been restricted to a single "track" for a long-distance bus ride. The restaurant does not have a "McDonald's" track or a "Le Pavillon" track. This was done primarily to have a data base capable of handling several specific stories of interest available in a reasonable time, secondarily to limit the amount of storage needed. Nevertheless, the scripts presently implemented are a reasonable first pass at the dual problems of creating and managing this type of data structure.

Goals, predictions, and roles in scripts: Each situational script supplies a default goal statement that, in the absence of planning input, is assumed to be what the script is about. It may be the case that two people go to a restaurant to discuss business and only incidentally to eat, but the script assumes that the INGEST is the central act nonetheless. Related to the goal statement is the implied sequence of mutual obligations that most scripts seem to entail. Invoking the bus script, for example, implies the contract between the bus management and the rider of a PTRANS to the desired location in return for the ATRANS of the fare. While this obligation network is not explicitly built into SAM's scripts, it has a powerful influence on the predictions the applier makes about new input. In the restaurant context, for example, the applier does not initially expect to hear about an input beyond ordering, or perhaps eating, the initial statement of obligation, although it will eventually identify a story sequence like: "John went to a diner. He left a large tip." Having heard about ordering, its horizons widen to expect input about preparing, serving, eating, paying, but not, initially, about leaving, since the other half of the obligation has not been fulfilled.

The bindings of nominals in the story input to appropriate fillers in the script templates is accomplished in SAM by means of script variables with associated features. The script variables are used in conjunction with a pattern-matcher. In the rather crude system of features currently used, each script variable is assigned a superset membership class; certain variables are also assigned to roles. The former property would provide the distinction between "The waiter brought Mary a hamburger" and "The waiter brought Mary a check." The latter identifies important roles in script contexts, primarily those to which it is possible to refer with a "the," like "the driver," "the cook," or "the check."

Each script used by SAM is organized in a top-down manner as follows: into tracks, consisting of scenes, which in turn consist of subscenes. Each track of a script corresponds to a manifestation of the situation differing in minor but important features of the script roles or in a different ordering of the scenes. So for example, eating in an expensive restaurant and in McDonald's share recognizable seating, ordering, paying, etc. activities but contrast in the price of the food, the type of food served, the number of restaurant personnel, the sequence of ordering and seating, and the like. Script scenes are organized around the main top-level acts, occurring in some definite sequence, that characterize a scriptal situation. In general, sub-scenes are organized around acts more or less closely related to the main act of the scene, either contributing a precondition for the main act, as walking to a table precedes sitting down, or resulting from the main act, as arriving at the desired location follows from the driver's act of driving the bus. All paths through a scene go through the main act (except abort paths, discussed below), and only a few events are at scene edges. For example, in the restaurant's ordering scene, the main act of ordering has many paths through it; at the boundary between being seated and ordering, the main actor can either know what he wants, read the menu at the table, or ask the hostess for a menu.

The discussion above should indicate that certain events in a script are distinguished: Scripts, their tracks, scenes, and subscenes all have maincons, for the main event occurring in the associated entity; entrycons, for the first events; and exitcons, for the final events. Scripts and tracks also have associated summaries, which correspond to inputs that summarize a script or track.

In general, there is only one path through a subscene. In SAM scripts, these paths are given a "value" to indicate roughly their "normality" in the script context. Several pathvalues have been found useful in setting up applier output. At the lower end of the normality range is "default," which designates the path the applier takes through a scene when the input does not explicitly refer to it. For example, the input sequence "John went to Consiglio's. He ordered lasagna" makes no mention of John's sitting down, which would commonly be assumed in this situation. The applier, using the default path, would fill in that John probably looked around inside the restaurant, saw an empty table, walked over to it, etc. Next on the normality scale is "nominal," designating paths that are usual, not involving errors or obstructions in the normal flow of the script. An example of a nominal path would be one involving the waiter's coming to the table in a restaurant during the ordering scene. Finally, there are the "interference/resolution" paths in a script. These are invoked when an event occurs that blocks the normal functioning of the script. In a restaurant, for example, having to wait for a table is an example of a mild interference; its resolution occurs when one becomes available. More serious because it interferes directly with the goal/obligation structure of the restaurant script is the main actor's discovery that he has no money to pay the bill. This is resolved in the current script by his doing dishes. An extreme example of an interference in this context is the main actor's becoming irritated when a waiter fails to take his order, followed by his leaving the restaurant. When this happens, the script is said to have taken an "abort" path.

In addition to the paths above, certain incomplete paths, i.e. paths having no important consequences within the script, have been included in the SAM data base. The most important of these partial paths are the inferences from and preconditions of the events in the direct causal paths. Lumped under the pathvalue "inference," these subsidiary events identify crucial resultative and enabling links that are useful in particular for question-answering [Lehnert, 1975]. For example, the main path event "John entered the train" has attached the precondition that the train must have arrived at the platform, which in turn is given as the result of the driver's bringing the train to the station. Similarly, a result of the main event "John paid the bill" is that he possesses less money than he did previously. Both of these types of path amount to a selection among the vast number of inferences that could be made from the main path event by an inferencing mechanism such as the conceptual memory program of Rieger [1975].

D. Paraphrase, Summary, and Question-Answering: Lehnert

Expansion paraphrase: When people communicate, it is natural to omit expression of any actions or states that can readily be inferred. When a narrative refers to a common script-type activity, the majority of script-related actions go unmentioned because they are easily inferred from the context of the script. In fact, the only script-related actions that are likely to be stated explicitly are those that describe variations within the script or unusual departures from the script. It is enough to say, "John went to a restaurant and had a hamburger," to convey the standard restaurant script activities involved. When a narrative spells out standard script-based inferences, it sounds boring: "John went to a restaurant and sat down at a table. A waitress came over to him and he ordered a hamburger. The waitress gave the order to the cook and the cook prepared the hamburger. Then the waitress served it to John. After John finished the hamburger, he paid the check and left the restaurant." This sounds tedious and uninteresting because nothing is said that couldn't have been inferred from the context of a restaurant script.

The expansion paraphrase expands the input story by inserting those script-related actions that would normally be inferred. The paraphraser takes as input the causal chain generated by the script applier. It then deletes from this sequence of states and acts those states that follow from preceding acts. What remains is a sequence of events describing (in glorious detail) the activity of the story; e.g. part of the causal chain might be:

> The waitress walks to the table.
> The waitress is at the table.
> The waitress gives John a menu.
> John has the manu.
> John reads the menu.
> :

The paraphraser would return from this the first, the third, and the fifth conceptualizations, so the paraphraser outputs an expanded event list that fills in the inferred actions of the script(s) involved. This list of conceptualizations is passed to the generator.

Short paraphrase: When a story is processed, the EXEC keeps track of what scripts are triggered and what kind of time relations exist among the scripts activated. A record is kept of sequential and nested script occurrences. This record is used to generate a short paraphrase of the story. For each script that is activated, the script applier generates a summarization of the script activity. The short paraphrase is constructed from those script summaries, combining them according to the sequential or nested relationships.

Summary: The summary program uses the script applier output as well as output from the EXEC. In a story where just one script is triggered, the summary is a script summary, as in short paraphrase. In stories where more than one script occurs, the program builds a summary based on plot components.

Plot components are key conceptualizations that are recognized by the script applier and the EXEC. Basic plot components recognized by the EXEC are the maingoal, unusual occurrences, and immediate consequences of unusual occurrences. The script applier recognizes pairs of interference/resolution conceptualizations. The summary program is basically a discrimination net with nodes that test for the occurrence of various plot components. The net terminates at various generation templates that combine the plot components with conjunctions and punctuation. The appropriate template is instantiated with the plot component conceptualizations and then passed to the generator.

Question-answering: The question-answering techniques designed for SAM are oriented to script-type data bases. Therefore the SAM system can answer only those questions that rely on information in a script. Given this restriction on content, SAM processes four types of questions. For a more detailed discussion of the processing and theory involved, see Lehnert [1975].

1. Fill-in-the-blank questions

These are questions like "What did John eat?" or "Who gave John a menu?" SAM searches the script applier output for the relevant conceptualization and returns the answer in one of two possible modes. The long answer mode returns an entire conceptualization, such as "John ate a hamburger" or "The waitress gave John a menu." The short answer mode returns only the missing information, as in "A hamburger" or "The waitress."

2. What-happened-when questions

These are questions like "What happened when John ordered a hamburger?" In this case SAM examines the causal chain generated by the script applier and extracts that portion of the chain that begins with the question concept (John's ordering a hamburger) and ends with the next conceptualization that was explicitly mentioned in the input story. SAM then deletes uninteresting states from this subchain and passes to the generator the remaining list of actions. Once the subchain is extracted, the processing is the same as in the expansion paraphrase program.

3. Why questions

While there are many ways to answer a why question reasonably; the response most natural in a script context is a goal-oriented answer. All script-related activities exist in a hierarchical structure of script sub-goals. Here is the goal structure for the restaurant script:

```
[1]                                                      >eat meal
[2] go to restaurant   sit down     order                        pay check ⟶ leave
[3] .......look for table....ask for menu...serve meal..ask for check..........
```

(Not all third-level sub-goals are shown here.) Once the question concept is found in the goal structure, SAM returns the first goal found to the right of the question concept on a higher level. If no such goal exists, SAM takes the goal immediately to the right of the question concept on the same level.

Q. Why did John ask for a menu?
A. So he could order.

Q. Why did John pay the check?
A. So he could leave.

Notice that these goals are so standard that such goal-oriented answers make sense even when asked without reference to a specific story. The only exception to this approach occurs when the question concept is the causal result of a script variation. Then the answer should be motive-oriented. Suppose we had the following story:

John went to a restaurant. The host seated him and gave him a menu. John ordered a hamburger but the waitress said that they didn't have any. So John ordered a hot dog instead. The waitress brought him the hot dog. John ate and left the restaurant.

Q. Why did John go to a restaurant?
A. So he could eat a meal. [goal-oriented]

Q. Why did the host give him a menu?
A. So he could order. [goal-oriented]

Q. Why did John order a hot dog?
A. Because the waitress told him they didn't have any hamburgers. [motive-oriented]

4. Did questions

These are yes-or-no type questions like "Did John pay the check?" The interesting thing about yes-or-no questions is that they are often answered with more than a yes or a no. Suppose we had the story:

John went to a restaurant. The host gave him a menu and he ordered a hamburger. But the hamburger was so burnt that John left without paying the check.

Q. Did the waitress give John a menu?

A. No, the host gave John a menu.

Q. Did John pay the check?

A. No, because the hamburger was burnt.

The elaborations in these answers are script-dependent responses, which SAM can handle. If an initial search of the script applier output returns the answer no, then SAM examines the question concept to see whether it is a script constant or contains a script variable.

A script constant is an expected act of the script that cannot embody any variations. The patron's going to the restaurant, the patron's eating, the patron's paying the check are examples of constants in the restaurant script. If any of these fails to occur, our expectations have been violated and we try to account for the deviation by asking why that constant didn't happen. So when "No" is returned for "Did John pay the check?" we then go on to ask "Why didn't John pay the check?" This is a motive-oriented why question, which is processed as in (3) to arrive at the elaboration "because the hamburger was burnt."

Some expected acts of a script have room for variations. In the restaurant script we know that the patron is going to get a menu. But there is a variable involved because John may get a menu from a waitress, or from the host, or he may just pick it up himself. Similarly the patron will get a check but it can come from the waitress or maybe the host. When an expected script act containing a given variable does not occur, we look for the expected act with some other value in the variable component. The variable in "Did the waitress give John a menu?" is the waitress. When the initial search of the script applier output returns no, we identify the variable component and search the script applier again. This time we look for the act without trying to match the specific variable component "waitress." We return whatever conceptualization matches the remaining concept: "The host gave John a menu."

E. The Generator: DeJong and Stutzman

Goldman's generator [1975] from the MARGIE system has been incorporated in SAM. Goldman's program (BABEL) handled input of Conceptual Dependency and produced an English sentence as output. Since SAM deals with more complicated sentences, the generator had to be modified in certain ways. In addition, the use of scripts presents some lexical problems. The basic modifications were:

1. Intersentence pronominalization: BABEL originally had a facility for pronominalizing successive occurrences of a syntax node within a sentence. We added a routine to handle cross-sentence pronominalization. The decision to realize a given noun phrase as a pronoun was based on identity with the last-mentioned NP carrying the relevant feature. The controlling features were masculine, feminine, or neuter gender or plural number, indicated by conjoined nouns derived from *GROUP* actors in a conceptualization.
2. Time atoms: BABEL was modified to accept time-role fillers of a relative nature such as "after" and "quick." This was done so as to be able to

generate adverbs such as "quickly" and time relations such as "After entering the restaurant, John went to the table."

3. Script words: We observed that English has "canned" expressions for expressing concurrent ACTs, one of which is a script. For example, we have "While in the restaurant, John ate a lobster," as opposed to "While on the subway John sat down." The choice of preposition is dependent on a lexical item associated with a script name. We modified the routine that resolved conceptualizations to verbs to select appropriate phrasal expressions.
4. Adjectives: A routine to express REL links as adjectives was written. "An old lady" is derived from

   (*LADY* REL ((ACTOR (*LADY* IS (*AGE* VAL (6))) REF (DEF)).

5. Increased capabilities for discrimination nets: The routine that selects verbs by evaluating discrimination nets was modified to accept a new terminal-node structure. Terminal nodes may now contain names of routines as well as pointers to the concexicon ("verb dictionary"). These routines may return concexicon pointers or set global variables for later use in the generation. It is this latter function that permits selection of phrasal expressions for script acts.
6. Optionality of syntax-frames: The routine that matches syntactic case-frames with syntax-net nodes was altered to allow frames with no corresponding node to be disregarded. For example,

   ((ACTOR (*MARY*)) <=> (*PTRANS*) OBJECT (*MARY*) TO
   (*NEW-YORK*)

is realized as "Mary went to New York" while

```
((ACTOR (*MARY*) = (*PTRANS*) OBJECT (*MARY*) TO
(*NEW-YORK*) INST ((ACTOR (*MARY*) = (*SDO*) OBJECT
($BUS)))))
```

is realized as "Mary went to New York by bus." Only a single concexicon entry, with optional instrumental frame, is required. These examples also give another example of a phrasal expression for a script act. In this case, the script-name "$BUS" leads us to choose the expression "by bus" instead of "by taking a bus."

7. Dependence on scripts to choose words: Scripts have associated nouns and verbs. MTRANSing that receiving food would lead to increased happiness is "ordering" in restaurants and "asking for" elsewhere. A new predicate was added to the discrimination net repertory that allowed interrogation of the script. This extension works only for stories in which a single script is active. A high-priority extension to the generator is building an interface to the script applier to allow determination of the script and scene for any conceptualization.

F. Generation of Chinese: Stutzman

The Chinese generator is a modified version of the BABEL program described by Goldman [1975]. The modifications fell into three major categories, each of which will be discussed in turn.

The first group of changes enabled the generator to express multiple sentences as connected discourse. Changes made to the English generator for this purpose were easily adapted for this program, and vice versa. For example, the alterations to the discrimination-net applier were originally made for the Chinese generator. This routine was then used to implement selection of phrasal expressions in English. The optional frame-handler, the new time-role evaluator, the script interrogation predicate and pronominalization scheme were written first for the English generator. The first three changes were incorporated directly into the Chinese program, while the pronominalization routine required minor alterations.

Rewriting the discrimination nets was the second step in the modification. Some nets are virtually identical to their English counterparts (i.e. INGEST) while others differ significantly. For example, the ATRANS of the lobster to the waiter and to John are both expressed by the English "received." In Chinese, two separate verbs, "jie" and "na," are required. The choice is currently based on the relationship of donor and recipient: John is the consumer, while the waiter is part of the preparer-server-consumer chain. With a more sophisticated interface to memory, the actual difference could be utilized. This difference is based on the instrument, now absent from the conceptualization. In the case of the chef-waiter ATRANS, the transfer is indirect. The chef is assumed to leave the lobster on the counter, where the waiter will later pick it up (verb = "jie"). In the case of the waiter-John transfer, John is assumed to be present at the table to receive the food (verb = "na"). If he had stepped away from the table, "na" would be used. Thus, a revised version of the executive, able to produce inferences about instruments, would be necessary to select the correct verb.

An interesting point-of-view problem was encountered. Some verbs realizing PTRANS acts require a complement indicating motion relative to the speaker. Thus, the conceptualization

```
(ACTOR (*JOHN*) <=> (*PTRANS*) OBJECT (*JOHN*) FROM
(*INSIDE* PART (*LINDYS*)
```

will be realized with the verb "chuh" + directional complement. If the narrator is assumed to be inside the restaurant, the complement "chiuh" ("go") is selected. Expressing this conceptualization from the point of view of one outside the restaurant requires the "come" ("lai") complement. The English verb "leave" is neutral with respect to point of view. The phrases "went out" and "came out" parallel the "chiuh"-"lai" distinction. The correct solution to this problem rests with a future addition to the generator, the ability to generate texts from an arbitrary point of view.

The Chinese generator uses discrimination nets to select the proper realization for some nouns. Money ATRANSed to a waiter in the context of the restaurant script is a tip, while money ATRANSed to the management is realized as the object "chyan" (money) in the verb-object compound "fu-chyan" ("pay a bill"). Chinese requires some verbs derived from PTRANS acts to follow locative NP with a directional complement. This complement is realized as zero for certain nouns, essentially places, like restaurants and cities. Thus, the Chinese generator has a discrimination (sub-) tree for "PROX." Chinese differentiates between express (= long distance) and local buses. In the current system, the memory interface is bypassed and the correct lexeme for bus chosen by evaluation of predicates constructed to be sensitive to a particular conceptualization.

The modifications to the surface generator were the simplest part of the project. The optional syntax-frame modification allowed a simple treatment of coverbs. Any syntactic frame could specify a coverb by means of a "special action." Other special actions include routines to insert prepositions and make a literal the value of a given frame. Every concexicon entry specified the coverb syntax relation but this frame was processed for only those entries with an object for which a coverb was specified. This eliminated having to define several new frames, the only feature of which would be the presence of a coverb.

The discrimination net input routine was redesigned for the Chinese program. Nets are retrieved on a sentence-by-sentence basis, instead of loading the entire collection. This modification permits the Chinese generator to run in approximately 40K words of storage, representing a 15K savings over the current English generator. A similar modification is planned to permit dynamic accession of concexicon entries.

Perhaps the most important observation made was that very little of the original BABEL design was changed. The basic algorithm of applying discrimination nets to conceptualizations to obtain the verb, from which the dependent cases were linearized, remains intact. Apart from the rewriting of the discrimination nets according to the Chinese pattern of expression and syntactic reformulations, generation of Chinese looks essentially like generation of English.

IV. Significance

Why have we done what we've done? SAM represents, in our opinion, an important advance in the area of computer understanding of natural language. SAM understands more than MARGIE because it knows more than MARGIE. It knows about certain situations as well as knowing about how events relate to each other.

But, as always, one of our principal motivations in this work remains psychological. SAM is important because it provides a test for a theory of understanding based on scripts.

Of course, SAM is just a beginning. It is important to point out just where we feel the problems ahead lie. SAM handles boring little stories. Theory must be developed to detect the point of a story; to determine when a problem has been created and to look for its resolution. It is necessary to establish an understanding of the individual characters in a story so as to know when they can be expected to do what. That is, it is necessary to determine characters' goals and motivations and to understand how a given action on their part fits in terms of a plan to achieve a given goal. We still need to account for non-scriptlike knowledge application. Often in understanding we need to bring in a rule about why people do what they do that is more general than any particular situation. What these rules are and how they are applied is something we have just begun to work on. One of the most important problems ahead is a good theory of forgetting. Just what people choose to remember of a novel they read is significant towards telling us what is most important about a text and what can always be filled in later. Scripts obviously provide the key to some of that. All that need be remembered when a script occurs is that it occurred. From then on the script can be retraced fairly accurately as long as the weird deviations or highlights of the scriptlike event are remembered separately. Thus in story III we could remember just "bus script, subway script, robbery, restaurant script with no-pay default path, bus script." But much more comes into play in forgetting and we need to determine that too.

What we can say, then, is that SAM represents a step past MARGIE on the road to understanding.

References

Abelson 1973
R. P. Abelson. The structure of belief systems. In R. C. Schank and K. M. Colby, editors, Computer Models of Thought and Language. Freeman, 1973.

Abelson 1975
R. P. Abelson. Concepts for representing mundane reality in plans. In D. Bobrow and A. Collins, eeditors, Representation and Understanding: Studies in Cognitive Science. Academic Press, 1975.

Goldman 1975
N. Goldman. Conceptual generation. In R. Schank, editor, Conceptual Information Processing. North Holland, 1975.

Lehnert 1975
W. Lehnert. What makes SAM run? Script-based techniques for question answering. Proceedings of the conference on Theoretical Issues in Natural Language Processing, edited by R. Schank and B. Nash-Webber, 1975.

Minsky 1974
M. Minsky. Frame-systems. MIT AI Memo, 1974.

Rieger 1975
C. Rieger. Conceptual memory. In R. Schank, editor, Conceptual Information Processing. North Holland, 1975.

Riesbeck 1975
C. Riesbeck. Conceptual analysis. In R. Schank, editor, Conceptual Information Processing. North Holland, 1975.

Schank 1973
R. C. Schank. Causality and reasoning. Technical Report #1, Instituto per gli studi semantici e cognitivi, Castagnola, Switzerland, 1973.

Schank 1974
R. C. Schank. Understanding paragraphs. Technical Report #6, Istituto per gli studi semantici e cognitivi, Castagnola, Switzerland, 1974.

Schank 1975
R. C. Schank, editor. Conceptual Information Processing. North Holland, 1975.

Schank & Abelson 1975
R. C. Schank and R. P. Abelson. Scripts, plans, and knowledge. Proceedings of the Fourth International Joint Conference on Artificial Intelligence, Tbilisi, USSR, 1975.

Schank et al. 1975
R. C. Schank, N. Goldman, C. Rieger, and C. Riesbeck. Inference and paraphrase by computer. Journal of the ACM, 1975.