1985

Inference in computer systems which understand natural language /

Robert J. Harwick
Lehigh University

INFERENCE IN COMPUTER SYSTEMS WHICH UNDERSTAND NATURAL LANGUAGE

by

Robert J. Harwick

A Thesis

Presented to the Graduate Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Computer Science

Lehigh University

1985

This thesis is accepted and approved in partial fulfillment of the requirements for the degree of Master of Science.

May 15, 1985

Gerhard Rayna
Professor in Charge

Donned J. Holman
Division Head

Eric D. Thompson
Department Chairperson

Acknowledgement

I wish to thank my advisor, Dr. Gerhard Rayna, whose introductory course in artificial intelligence inspired the writing of this paper, and whose criticisms and suggestions contributed immeasurably to its quality.

Table of Contents

Abstract

Inference is a major component in the understanding of natural language by computer. Whether deductive or non-deductive, the primary problem of inference is efficiency. Humans process language very quickly, and so must an intelligent machine. Procedural representation of propositional data improves the efficiency of deductive inference by providing direction to the theorem-proving process, but sacrifices the flexibility available in declarative representations. Much of the inference necessary for language understanding is not goal-directed in the way that formal deduction is, instead, predictions are made from the current input, and future input is examined to see how it fits the predictions. This provides an even greater efficiency problem. The prediction process must be controlled so that only the useful predictions are made. High-level knowledge structures such as frames and scripts add direction to the inferencing process, but again flexibility is sacrificed. Current research involves integration of high-level structures to provide control for all aspects of inference while maintaining an acceptable level of flexibility.

I. Introduction

When humans communicate in natural languages, much of the information they exchange is not explicit, but is inferred by the listener. For example, when we hear, "John is going to the store.", some of the things we infer are:

(1) John is not now at the store.
(2) John wants to buy something.
(3) He is using some mode of transportation.

A computer program which understands natural language may have to perform any of the above (and more) inferences in order to demonstrate that it has understood the sentence. For example, a future sentence may be "He is taking the bus." In order to know how this information fits the context, so that it can correctly answer the question, "How is he getting to the store?", the computer must infer (3).

Natural language understanding is usually divided into three areas: syntax (the grammatical relationship between words), semantics (the explicit meaning of words, phrases, and sentences), and inference (the discovery of implied meanings and relationships). Rieger, however, defines natural language comprehension as "the art of making explicit the meaning relationship among thoughts which are presumed to be meaningfully relatable." Note that this definition emphasizes

inference. To justify this emphasis on inference, consider the problem of pronoun resolution in: "John hit Bill. He went to the doctor." Syntactic clues for pronoun resolution are inadequate in this case, in fact, a purely syntactic rule might be "If a pronoun is the subject of a sentence, its referent is likely to be the subject of the previous sentence.", which is not the case here. Semantics offer no solution, since it is semantically valid for any human to go to the doctor. Only by inferring that someone who is hit is likely to be hurt, and that someone who is hurt is likely to see a doctor, can an understander correctly recognize "He" as "Bill".

There are two basic approaches to inference in computer natural language systems. The formal approach involves deduction and uses predicate logic, sometimes represented procedurally, as its primary knowledge representation. There are many manifestations of the informal approach, each of which uses some type of declarative knowledge structure. As may be inferred from the examples above, natural language inference requires knowledge of two types: real-world knowledge of such things as the consequences of actions and the goals of characters, and contextual knowledge which is continuously augmented throughout a story or

conversation.

The purposes of this paper are: 1) to examine the nature of the problem of inference in natural language communication, 2) to discuss some of the knowledge structures and how they are used for inference in natural language systems, and 3) to examine how the process of inferencing may be controlled to make it practical for computer systems. This will be done through an examination of the inference component of the following systems: SHRDLU [Winograd,1971], MARGIE [Schank and Rieger,1974], SAM [Cullingford,1981], GUS [Bobrow et. al.,1977], Ms. Malaprop [Charniak, 1978], PAM [Wilensky,1981], Rieger's commonsense algorithms [Rieger,1976], and BORIS [Lehnert et. al.,1983].

I. SHRDLU

In SHRDLU [Winograd, 1971], the user enters English commands and questions directed toward a robot whose job is to interpret the commands and perform them, and to answer the questions. The robot's domain is a table with a set of blocks of various colors and shapes; the commands involve movement of the blocks, and the questions are about the state of the blocks world. The system is a natural language system and not a robotics system. A real robot was never used in the demonstration of the system; instead, its movements were simulated on a video display. The main contributions of the system were its ability understand grammatical structures of almost unlimited complexity and the procedural representation of propositional data which gave its inference process more direction than previous deductive systems had. We are interested in the latter.

1. Knowledge Representation -- Procedures

In its limited domain (the blocks world), SHRDLU's inference is deductive (as opposed to predictive non-deductive inference systems which we shall discuss later). The principal method of deduction before SHRDLU was the general deductive approach. In this approach, propositional information was stored in a LISP representation of the first order predicate calculus.

Processing involved the Robinson resolution algorithm, which is based on proof by contradiction. First, the negation of the proposition to be proved is taken. Known propositions are stored as disjunctions of simple predicates. At each stage of the proof, the system searches the entire data base of known propositions for one which includes (as part of its disjunction) the negation of something included in the current proposition. When one is found, the negation and the portion of the current proposition cancel, and the resulting proposition is the new current proposition. This process is repeated until the cancellation leaves nothing, which means whatever might have come from assuming the proposition false has been contradicted by some known proposition, so the proposition is proved.

There are two advantages of such a general deductive system. First, if a proposition is provable from the known propositions, a proof will (eventually) be found. Second, the proof procedure is uniform and not dependent upon knowledge of proof techniques in a specific domain. However, these advantages can also be regarded as disadvantages in a practical natural language system. This is because the algorithm has no direction about how to go about proving the proposition, so it has to test all known propositions until it finds

one which is relevant. In doing inference in natural language understanding, we do not have the time to go through such an exhaustive process. The way we determine the truth of propositions is by having some heuristic knowledge about how our propositional knowledge may be used.

This is the idea behind the inferencing scheme used in SHRDLU. Propositions are stored as procedures in a language called PLANNER [Hewitt,1969]. Each procedure may include information on how to go about proving the proposition, such as which other procedures should be tried in an attempt to prove this one, and in what order. The propositions are written in a form similar to the predicate calculus, and as much or as little subject-dependent heuristic information can be added as the user desires. If he adds none, the system works with the full flexibility of a general deductive system; but with more information about how to prove it, the system is more efficient.

The calling of procedures by other procedures is goal-directed, that is, a general pattern-matcher looks at the goal (specified as part of the procedure) and chooses any procedure which satisfies the goal. In this way, it is not necessary to have all procedures know about the existence of others. However, it requires a

pattern-matching search similar to that of the resolution algorithm. If, on the other hand, specific procedures are recommended as the only ones to try in order to satisfy the goal of a given procedure, the exhaustive search is eliminated, but now the procedures have to know about other procedures. That is, the system involves a trade-off between efficiency and additivity.

Some of the features of PLANNER can be seen in the following example.

```
(DEFINE THEOREM EVALUATE
   (THCONSE (X Y)
      (THGOAL (#THESIS $?X))
      (THOR
         (THGOAL (#LONG $?X) (THUSE CONTENTS-CHECK
                                    COUNTPAGES))
         (THAND
            (THGOAL (#CONTAINS $?X $?Y))
            (THGOAL (#ARGUMENT $?Y))
            (THGOAL (#PERSUASIVE $?Y)
                    (THTBF THTRUE)) ))))
```

First, note the punctuation "#" and "$?". The "#" indicates that what follows is a predicate name; the "?$" indicates a variable. Another punctuation mark ":" is used to indicate an object, the value to which a variable might be bound.

This is the procedural representation of the theorem "A thesis is acceptable if it is either long or it contains a persuasive argument". The name of the theorem is EVALUATE. Its variables are X and Y; X is

the theorem to be tested; Y is used to hold something in the thesis which is tested to see if it a persuasive argument. THCONSE is the type of theorem -- this is a consequent theorem since we are trying to establish whether or not a given thesis is acceptable. There are two main subgoals. The first is to prove that X is a thesis; the second is to prove that either X is long or that X contains a persuasive argument. These are given as THGOAL. The first goal has the recommendation list CONTENTS-CHECK and COUNTPAGES, indicated by THUSE. This means that to establish the goal that the thesis is long, the procedures CONTENTS-CHECK (which presumably looks at the Table of Contents) and COUNTPAGES (which counts the pages) should be used in order. If this succeeds, it is not necessary to look for a persuasive argument, so THOR is an OR which does not check the second statement if the first one is true. If we do need to check for a persuasive argument, the theorem says we must first find something that the thesis contains, if this is found, see if it is an argument, and if so, try to prove that it is persuasive. Note that THAND only continues to the next statement if the previous one was true. The check for persuasiveness contains the recommendation (THTBF THTRUE). THTBF means try any theorem whose form satisfies the filter which

follows. THTRUE is the only filter used in SHRDLU, although a PLANNER user may define any filters he wishes. THTRUE is a predefined filter which any theorem matches.

We stated the theorem which this procedure represents in declarative form above. In imperative form, the procedure may be stated: "In order to evaluate X for acceptability as a thesis, first show that X is a thesis; then show that it is either long or that it contains a persuasive argument. To see if it is a thesis, look in the data base of simple assertions. To see if it is long, first check the table of contents, and if this fails, count the pages. If the thesis is not long, check to see if it has a persuasive argument. To do this, find something it contains, show that it is an argument, and show that it is persuasive using any theorem which might apply."

Among PLANNER's other features are THGO, THAMONG, THERASE, THASSERT, THANTE, and THFIND. THGO is a GOTO statement. THAMONG is like LISP MEMQ, succeeding if the value of the given variable is in the given list. THERASE removes assertions (which are declaratively represented simple predicates) from the data base; THASSERT adds assertions to the data base. THANTE is for declaring an ANTECEDENT theorem, which is used to

indicate that certain assertions should be added when certain facts are determined. For example, when we find that X drove a car somewhere, we may wish to assert that X is human, because this may be useful in proving other theorems. To do this, we could define an ANTECEDENT theorem as follows:

```
(DEFPROP DRIVETHEOREM
   (THANTE X  (DROVE $?X $?Y)
              (THASSERT (HUMAN $?X))) THEOREM).
```

THFIND is used to find objects or assertions satisfying a given condition. This is useful in the blocks world, where the system might, for example, wish to find all the red blocks. It could use the following:

```
(THFIND ALL $?X (X)
    (THGOAL (BLOCK $?X))
    (THGOAL (COLOR $?X RED))).
```

2. Processing

SHRDLU, as noted above, receives commands and questions in English. The syntactic and semantic portions of the system process the commands or questions into PLANNER statements. For example, the command "Pick up the block and put it into the box." could be expressed in PLANNER as:

```
(THAND (THGOAL (#PICKUP :BLOCK23))
       (THGOAL (#PUTIN :BLOCK23 :BOX7)))
```

assuming that semantic analysis has correctly identified "the block" as :BLOCK23 and "the box" as :BOX7. :PICKUP

and :PUTIN are procedures which indicate the steps to be taken to perform the actions. An example of PLANNER representation of a question is:

```
(THGOAL (#ACCEPTABLE :SAM-THESIS)
                (THUSE EVALUATE)),
```

which is the representation of the question "Is Sam's thesis acceptable?". Since PLANNER is actually as interpreter of the PLANNER language, processing consists of evaluating, in the sense of LISP EVAL, such PLANNER statements.

PLANNER has a more sophisticated backup facility than LISP, which is very useful in theorem-proving. PLANNER has the ability to do backup in case of failure, and that backup always goes to the last place where a decision of any kind was made. Thus if it is attempting to find an object in the data base which has two properties, it would attempt to use the statement

```
(THAND (X) (THGOAL (PROPERTY1 $?X))
           (THGOAL (PROPERTY2 $?X))).
```

Upon finding an object which has property 1, it will then see if that object has property 2. If not, it will back up to the first THGOAL and look for another object which has property 1, rather than returning NIL for the entire AND and resuming at the next statement the way LISP does. Similarly, backup is effective in THOR in the expected way. If the first member of the THOR

succeeds, the second is not tried, but if there is a failure further along on this path, PLANNER can back up to the THOR and try the second.

The system can keep track of events and states in the changing blocks world through its imperative representation. Since assertions may be true at one time and become false later, or vice versa, it is necessary to be able to remove and add them to the data base of assertions in order to keep the state of the blocks world up to date. This is handled very naturally by the use of ANTECEDENT theorems which may contain the imperative statements THERASE or THASSERT.

3. Discussion

SHRDLU understood a much larger subset of English grammar than did previous systems. Complicated embeddings of relative clauses used to identify a particular block in the world were correctly understood. It also represented a significant advance in inferencing. By representing propositional data in the form of procedures, the proof process had knowledge of both the available facts and how to use them for a proof. This, of course, was much more efficient than the resolution algorithm, which had to search, with no heuristic direction, the entire set of available propositions for one which might be applicable.

SHRDLU's advantage was in another sense a disadvantage if the system was to be extended to a domain wider than the blocks world. Its procedural representation for propositional data, in providing information on how the knowledge could be used, also limited the use of that knowledge to those purposes embodied in the procedures. In wider domains, the ways knowledge must be used are not so limited. For example, human understanders use knowledge not only to verify or discover facts about the world, but also to predict what kind of information or events may follow in a story. The procedural representation, while efficient and useful for formal deductive inference, was too limited for the type of inference that must be done in understanding stories or conversation.

III. MARGIE

Most current natural language systems do not rely on formal deduction as the primary method of inference, indeed, many systems use no formal deduction at all. Instead, they use knowledge structures and inferencing techniques which allow faster, shallower inferences than deductive systems.

[Schank and Rieger,1974] outline the differences between the kind of inference used in natural language understanding and the formal deduction which is more appropriate for problem-solving systems:

1. Unlike deduction, inference generation is a reflex response to the input. It is not only done as needed, but is a constant process of making predictions and looking for their fulfillment. This prediction/fulfillment model is a useful one. When we read a sentence, by predicting what we might see next, we can then see how the information in the following sentences fulfills or contradicts our prediction. In so doing, we have made explicit the meaning relationship among the sentences.

2. Inferences are not necessarily logically valid. When we know that a certain action causes a certain state, we can infer from the presence of that state that the action has occurred (and change our mind later if

new facts contradict the inference) but we can not deduce it.

3. As a result, inferences can be made which are not accurate. An inferencing system must allow for the possibility of erroneous inferences by providing a means of handling contradictions to facts it has inferred.

4. Inference, unlike deduction, is not goal-directed. With deduction, we try to determine if a given proposition is true; with inference, we are just looking to see what we can see and use those inferences as predictions of what may follow, so that the following information can be understood in terms of how it fits the prediction.

This is all very different from the deduction in SHRDLU, where proofs for propositions were sought as needed through a strictly deductive theorem prover.

5. With inference, we need to know why something is thought to be true.

1. Knowledge representation -- Conceptual Dependency

MARGIE's knowledge is represented in Conceptual Dependency. Conceptual Dependency is a structure for representing the meaning of a sentence which based on the theory that meaning can be represented in a language-free form which is indicates the concept conveyed by the sentence. Unlike the PLANNER

representation used in SHRDLU, CD is a strictly declarative way of representing knowledge.

The basis of CD is its representation of events. Every event has four slots, which may or may not be filled at any given time. The slots are: an ACTOR, an ACT, an OBJECT of the action, and the DIRECTION in which the action is performed.

These are twelve primitive ACTs:

INGEST -- an animal actor takes something internally
PROPEL -- a physical force is applied to an object
MOVE -- a body part moves
GRASP -- an actor takes hold of an object
ATRANS -- transfer of an abstract relationship such as possession, ownership, or control
PTRANS -- a physical change in the location of an object
EXPEL -- expulsion of an object from an animal to the outside world
CONC -- an actor thinks about an object
MBUILD -- an actor builds new information from old
ATTEND -- an animal directs sense organs toward an object
SPEAK -- an animal produces sounds from its mouth
MTRANS -- transfer of mental information

Each ACT may have any or all of four cases: OBJECTIVE, RECIPIENT, DIRECTIVE, and INSTRUMENTAL.

Real world objects are called PPs (picture producers). In addition to PPs, CD also has concepts representing times, locations, attributes of objects (PAs) and attributes of actions (AAs).

In addition to events, CD may also represent the following relationships among ACTs, PPs, PAs, AAs,

times, and locations: a PP has a PA, a PP is an actor or an object of an action, two PPs represent recipient and donor within an action, direction of object within an action, causality, state change of objects, and possession of one PP by another PP. Each of these is represented in a CD network graph by a different symbol linking the cases involved.

The most important feature of CD is that it is language-free: "John consumed the cake.", "John ate the cake.", and "The cake was eaten by John." all have the same CD representation. If a natural language sentence is first analyzed into CD (by a semantics-directed parser), the knowledge structure created can be matched against permanent or contextual knowledge structures in memory, also stored in CD, in order to use the knowledge for inference.

To build a theory of inference, Schank and Rieger list the types of inferences that an understander must employ within the framework of the twelve primitive ACTs of Conceptual Dependency theory. By basing the inference types on a small set of primitives, they avoid the problem of needing a different inference type for each different verb. The inference types are:

1. Linguistic Inference

A word or syntactic construction implies the presence of

some unmentioned object. For example, the presence of a certain ACT may imply the presence of various participants in various case roles for the ACT.

2. ACT Inference

An actor and an object may occur with no ACT to connect them, but our knowledge indicates that the object has a normal function, so we infer that the object was used for that function.

3. TRANS-enable Inference

A TRANS ACT involving an object and its recipient enables another ACT to take place.

4. Result Inference

We can infer the usual result of a TRANS ACT.

5. Object-affect Inference

A physical ACT can be inferred to have a certain effect on an object.

6. Belief-pattern Inference

An ACT together with its inferred results (from 4. and 5.) often fit a belief pattern involving the usual reason for the ACT.

7. Instrumental Inference

Each ACT has instrumental ACTs associated with it, that is, ACTs which are involved in its performance:

INGEST -- PTRANS
PROPEL -- MOVE, GRASP, PROPEL
PTRANS -- MOVE, PROPEL
ATRANS -- PTRANS, MTRANS, MOVE
CONC -- MTRANS
MTRANS -- MBUILD, SPEAK, ATTEND, MOVE
MBUILD -- MTRANS
EXPEL -- MOVE, PROPEL
GRASP, SPEAK -- MOVE
ATTEND -- sometimes MOVE, but usually none
MOVE -- none

8. Property Inference

We can infer certain properties of objects (e.g., their existence) from their presence in a sentence and their performance of a given ACT.

9. Sequential Inference

Sentences in sequence may share a subject or proposition. When we read "John wants to join the army" and later "John is a pacifist", we may infer that the second sentence refutes the first.

10. Causality Inference

Sentences in sequence or connected with "and", together with our real-world knowledge that the first may possibly cause the second, may allow us to conclude causality.

11. Backward Inference

When an ACT occurs which normally requires a prerequisite ACT, we may infer the occurrence of the prerequisite ACT.

12. Intention Inference

From the performance of an ACT by an actor, we may infer that the actor intended for the results (from 4. and 5.) to occur and is pleased that they have occurred.

2. Processing

The inferencing component of MARGIE does all of the above kinds of inferences. The MARGIE system consists of three processes: conceptual analysis, memory, and generation. Inferencing is done in the first two; the third is for generating responses.

Conceptual analysis has two phases. First, a Conceptual Dependency graph of the explicit meaning of the sentence is produced. Second, the analyzer initiates inferencing which extends the graph to include implicit information which is not dependent upon the context, but only upon the sentence itself.

The function of memory is to take the conceptual analysis and generate probabilistic information about how it relates to knowledge previously stored. The predictions are of three forms: 1) predictions about causes, 2) predictions about results, and 3) predictions about future and past actions of characters.

Predictions are stored as propositions in list positional form, with the predicate first, then the case slots. Each simple concept has an occurrence set, which

is a set of pointers to the propositions which contain instances of that concept. Propositions also have occurrence sets, so that propositions can be embedded in other propositions. Propositions also have the following characteristics:

STRENGTH -- the credibility of the proposition, stored as value between Ø and 1. This is the probabilistic component of a proposition.

MODE -- negations have mode = false.

TRUTH -- current truth value of the proposition.

REASONS -- other propositions used to infer this proposition. This gives the system the ability to explain its reasoning.

OFFSPRING -- other propositions inferred in part from this proposition.

RECENCY -- time of last access of this proposition.

Memory performs the following inferencing tasks:

1.) Establish referents of all concepts in the Conceptual Dependency graph.

2.) Serve as a data bank and access mechanism for answering questions and processing proof requests.

3.) Store the analyzed contents of the sentence.

4.) Perform appropriateness checking on the implications of the input.

5.) Generate inferences. Completatory inferences supply a candidate for missing information. Causal inferences relate the input to belief patterns in memory

in order to explain the reason for the input. Result inferences establish possible outcomes of actions given in the input. Some of these inferences may be elevated to predictions, which are inferences focussed on as noteworthy.

6.) Maintain a record of inferencing and prediction, and discuss reasoning. This includes modifying STRENGTHS and MODES.

7.) Answer "wh-" questions about the conceptualizations and inferences which it receives from the conceptual analyzer.

For each new concept, memory receives a descriptive set, which is the set of propositions about that concept. For example, the conceptual analyzer, after building the conceptual dependency graph and doing its inferencing, sends the following to memory for the sentence "John hit Mary.":

```
((CAUSE ((PROPEL C1: {(ISA - #PERSON) (NAME -"JOHN")}
                C2: {(ISA - #HAND) (PART - C1)}
                C1
                C3: {(ISA - #PERSON) (NAME -"MARY")}
                                                    ))
        ((PHYSCONT C2 C3)) )
        (TIME -C4: {ISA - #TIME) (BEFORE - #NOW)}) )
```

The concepts in the graph are John, Mary, John's hand, and the time. After memory establishes the referents to the concepts, we have:

```
((CAUSE ((PROPEL #JOHN #C0001 #JOHN #MARY))
     ((PHYSCONT #C0001 #MARY))
     (TIME - #C0002)))
```

where C0001 represents John's hand and C0002 is the time of the event. The main proposition of this sentence is thus understood as "John propelled something (his hand) at Mary causing it to make physical contact with Mary."

Before doing its inferencing, memory generates subpropositions, which are units of information conveyed directly by the conceptualization produced in conceptual analysis after referents are established. Subpropositions are of three types:

1.) Explicit focussed -- that the main event of the sentence occurred. In our example, this is the CAUSE proposition above.

2.) Explicit peripheral -- that other stated events in the sentence occurred. In our example, the PROPEL and PHYSCONT propositions are peripheral.

3.) Implicit -- that events not stated, but normally true, occurred. "A hand was moved." is and example of an implicit subproposition.

Inferencing starts by assigning the main proposition (CAUSE in our example) a STRENGTH = 1, TRUTH = T, MODE = T, REASONS = T (means not inferred, but stated), and putting it on the inference list. Then the PROPEL and PHYSCONT inferences are given STRENGTH = 1,

TRUTH = T, MODE = T, and REASONS = the CAUSE proposition, and added to the inference list. Before inferring anything from PROPEL and PHYSCONT, memory infers more about CAUSE. Thus inferencing is done breadth-first.

Inferred from the CAUSE ... PROPEL ... PHYSCONT proposition is the possibility of a NEGCHANGE in the health of the person being hit. This is an example of a Result inference. This is done by using a pattern in memory which states that such a combination CANCAUSE a negative change in health. Another inference is that John intended that result, which is an Intention inference:

```
((MLOC ((CANCAUSE ((NEGCHANGE #MARY #PSTATE))
                  ((POSCHANGE #JOHN #JOY)) ))
       C0001))
```

where C0001 is John's long-term memory.
Memory always infers that actions are volitional unless this is contradicted. The meaning of this pattern is that it was in John's long-term memory (MLOC) that hitting Mary CANCAUSE her pain and thus him joy.

From the NEGCHANGE proposition, memory tries to determine the cause. In this case, the REASONS for the NEGCHANGE give the cause. If the cause was not present, memory would make a prediction that information about the cause will follow. Another inference from NEGCHANGE

involves a belief pattern that someone who has a NEGCHANGE will seek a POSCHANGE (remedy). This leads to several predictions about what Mary will do next (go to a doctor, take medicine, hit John back).

From the inference about John's volition, we try to infer a cause -- why it caused John joy for Mary to be hurt. A belief pattern in memory associates with this pattern the belief that such a person must have been angry. Thus this is a belief-pattern inference. It is further inferred that the cause of anger is something that Mary did to John. When no cause for this is found in memory, the system generates the response, "What did Mary do to John?", thus indicating the depth of its understanding.

Notice that the inferencing need not be particularly deep (only three levels from the stated fact) in order to understand the sentence, even in terms of the goals of the actors. Shallow inference is a characteristic of natural language understanding. Time constraints prevent such systems from making the deep inferences needed in problem-solving systems. One expects the computer to take some time to solve a problem, but the response to "John hit Mary." is expected to be almost instantaneous.

3. Discussion

MARGIE represented a significant improvement over previous systems. First, it embodied a theory of non-deductive inference which more accurately models human understanding. Second, it demonstrated the usefulness of conceptual dependency as a language-free meaning representation language which could be applied to inference. Finally, it demonstrated that understanding is not so much syntax-based as inference-based. For example, syntactically incorrect sentences can be understood if the proper inferences are made. Consider "John his dog the bone gave.". While this sentence is incorrect syntactically, and thus syntax gives no clue as to who did the giving, we understand that John gave the bone to his dog, because we know that dogs like to get bones and people like to give bones to dogs.

MARGIE's main shortcoming was its uncontrolled process of inferencing. Although the implementation of the system did place constraints on the depth of inferencing, in theory the inferences for each sentence could go on and on. When new sentences arrive, they cause more inferences, resulting in a combinatorial explosion. The problem is that MARGIE employed a very primitive theory of context -- the context of a new input sentence was the (possibly huge) set of inferences

generated by previous sentences. This was a direct result of the belief that inferences, unlike deductions, are not always goal-directed. MARGIE, however, produced inferences as though they were never goal-directed, that is, all inferences were made bottom-up from the input. The systems which follow represent attempts to add some control and direction to the inference process.

IV. SAM

There have been two basic approaches to solving MARGIE's problem of uncontrolled inferences, both of which involve developing a theory of context among sentences which MARGIE did not have. The first approach incorporates high-level knowledge structures which control the inferencing process by specifying a context and providing predictions about how subsequent input might fit the context. The second focuses the inferencing on plans and goals which can explain the relationships between actions described in sentences.

1. Knowledge Representation -- Scripts

The first high-level knowledge structures used for inference control were scripts [Schank and Abelson,1977]. Scripts describe knowledge of everyday situations which human understanders often use to comprehend stories. This is especially useful in situations in which causal relationships between events are not stated explicitly because of the speaker's assumption that the understander has knowledge of the stereotypical relationships in the given situation.

Scripts consist of a sequence of causally related events (and the causal relationships between them) which describe a well-known situation. Examples of such situations are going to a restaurant, which consists of

the events of finding a table, ordering, eating, and paying; and riding a bus, which consists of the events of boarding, paying, finding a seat, riding, and getting off. The characters in a script are said to fill various stereotyped roles in the script; the objects used by the characters are called props; and the places that occur in a script are called settings. Together, the roles, props, and settings of a script make up the script's variables.

2. Processing

The system SAM [Cullingford,1981] uses scripts to understand stories about everyday situations. SAM has three basic modules. The first is conceptual analysis, which is the translation of English language input sentences into Conceptual Dependency representation. The second resolves references to actors in a sentence by identifying them with roles in a script to which they were bound because of previous sentences, or if no role was previously bound to the actor, to determine into which role they fit. The third is the script applier, which locates new input in the data base of scripts, sets up predictions about what input is likely to follow, and instantiates the appropriate parts of scripts. The script applier is therefore the principal inferencing component of the system.

Since the script applier deals with the language-free CD representation of the input, the events in SAM's scripts are also in CD form. The basic idea behind the script applier is to match the CD patterns found in the input with CD patterns found in the scripts in order to determine which script applies and which event in the script is being referred to by the input. Once this is done, the variables can be bound according to pattern-matching criteria involving the known characteristics of the actor, object, or place compared to the expected characteristics of the role, prop, or setting.

Thus SAM, like MARGIE, uses predictions of what may follow in the story matched against what does follow in order to understand the meaning relationships between the thoughts. The difference is that SAM's predictions are embodied in the structure of the script, which contains information about what is how events fit together. Thus there need not be the kind of shot-in-the-dark inferences generated by MARGIE in the hope that some may be useful predictions.

The script applier first introduces the most inclusive script referred to in the story by matching a story event against script headers which are included in scripts to determine what story patterns should cause their instantiation. Subsequent inputs are recognized

as events in the script, again by pattern matching. As these inputs are recognized, the script applier makes predictions about what information is likely to follow. When an input is not recognized in the script or by a prediction, the applier assumes that a new script is to be introduced, and again the most inclusive script is introduced, starting the cycle all over again. The instantiation of a new script does not usually mean the removal of an existing one, since the information in both may be useful in understanding. This requires that when a new script is brought in, roles and props must be matched with those of the old script. Thus related scripts have interfaces which relate their variables by function.

As implied above, pattern matching is done in phases. First, the constant parts of the conceptualization for the input are matched against the constant parts of the pattern found in the script. Script patterns may represent events in an already instantiated script, or if the pattern matching is being done in order to determine which script is to be instantiated, the pattern used will be a script header. This first phase is called the backbone match. The backbone match has four basic rules: 1) "literal" roles and fillers specified in the pattern must appear in the

input; 2) any extra roles and fillers in the input are ignored; 3) a dummy which appears more than once in a pattern must match the same thing from the input every time it appears; and 4) an empty slot in the input matches anything, unless the pattern uses the EXPLICIT tag to demand that a filler be present.

In the second phase, a process called Rolefit receives from the first phase the candidates for variable bindings, and checks them for reasonableness. This is done by including with each variable a set of conceptual categories into which an object must fall in order to be bound to that variable. In addition, another process Rolemerge is used to determine whether values previously bound to variables may be bound to new variables. This process uses knowledge attached to the variables to see, for example, if a given actor may have two roles in a script, or whether these roles must be filled by two different actors. In the latter case, Rolemerge would return a matching failure if an attempt was made to bind an actor to a role when that actor was previously bound to an exclusive role.

In order to do prediction, the events of a SAM script are grouped into episodes. For example, the episode "finding a seat" on a bus would consist of the events "see an empty seat", "go to it", and "sit down".

The prediction phase clears from active memory those episodes which occurred before the currently-matched event pattern, and brings into active memory those episodes containing event patterns predicted by the present one. This, of course, reduces the combinatorial explosion of inferences which plagued MARGIE.

Another important inferencing task performed by SAM involves implied preconditions for scripts, episodes, and events. Whenever SAM finds that an event has taken place, or a script is to be instantiated, the preconditions for the event or the script are assumed to be true unless otherwise indicated in memory. For example, upon finding that John went into the library, SAM assumes that the library is open.

The event patterns in SAM scripts are the usual CD events, typically involving a primitive CD ACT with slots for case roles appropriate to the ACT. The script header patterns, which are used for script instantiation are somewhat different. Attached to each script header is a set of predictions about what should happen first in that script. Script header patterns are of four types, each of which is a complete conceptualization, not just a reference to an object. Thus scripts are instantiated by complete events, so that "John walked into a restaurant." will cause the RESTAURANT script to

be introduced, but "John walked past a restaurant." will not. Precondition headers involve a weak prediction that a script should be introduced because one of the preconditions for the script is mentioned in the story. For example, "John was hungry." is a PH for the RESTAURANT script. Instrumental headers commonly occur when the input refers to two or more scripts, one of which may be an instrument for the others. For example, the BUS script is instrumental to many scripts which require a change of locale. Locale headers cause scripts to be instantiated when an actor is found to be present in a given locale. For example, John's presence in a restaurant represents a strong prediction that RESTAURANT is to be instantiated. Direct headers have the strongest predictive value, and are thus the first patterns to be checked in a context. In a DH, the script situation (not just the locale) is mentioned directly in the story.

3. Discussion

SAM successfully understood stories dealing with stereotyped situations, thus demonstrating the appropriateness of scripts for this purpose. The script model proved to be an especially useful one in understanding newspaper stories, which usually involve situations in which the information provided fits into

well-defined categories. Scripts provided control of the prediction component necessary to successful non-deductive inferencing, which led to a reduction in the number of useless inferences when compared to MARGIE's bottom-up "shot-in-the-dark" inference generation.

In the next two sections, we will discuss systems which use frames. Frames are high-level knowledge structures which afford the same efficiency in inference control as scripts.

V. GUS

Another high-level knowledge structure used for inference in natural language systems is the frame [Minsky,1975]. Frames are of many forms, all of which involve two main ideas. First, a frame represents a form whose structure is known by its user in terms of slots to be filled. Second, a frame is instantiated only when needed, and may be discarded when no longer relevant.

A system which uses frames to represent knowledge for inference is GUS [Bobrow et. al.,1977]. Unlike many of the other systems mentioned in this paper, which read and understand stories and then answer questions about them, GUS is a dialog system --it carries on a conversation with the human user. GUS acts like a travel agent. It asks questions of the client in order to schedule an airline flight. If the client needs flight information in order to make a decision, GUS will provide that information upon request. Thus although GUS attempts to maintain control of the dialog, there are times when the client may take over control.

1. Knowledge representation -- Frames

A frame instance in GUS consists of three parts: the name of the frame, which is only a mnemonic for programmers and is not used in processing; a reference

to a prototype frame; and its slots. Slots have slot-names, fillers, and possibly attached procedures. An example prototype frame is the TRIP-SPECIFICATION frame, which contains slots for all the characteristics of a trip, such as departure time, arrival time, and flight number. In addition to the usual values, slot fillers may be other frames, or, for a prototype frame, a description constraining the fillers for the slots of any instance. ISA is used in a frame instance to specify what prototype it is an instance of.

The procedures attached to the slots are for finding fillers and other types of reasoning. There are two types of attached procedures -- servants and demons. Servants are activated on demand. They are indicated in the slot by the directive TOFILL followed by a procedure name. Thus the procedure is being suggested by the frame as a possible way of filling the slot. Two standard servants are ASKCLIENT, which causes GUS to ask the client for information on how to fill the slot, and CREATEINSTANCE, which indicates that a new instance of a specified prototype should be created to fill the slot.

Demons are activated automatically. Some slots have the directive WHENFILLED, followed by a procedure name. Thus this procedure is run when the slot is filled. For example, the DATE frame contains a slot for

DAYOFWEEK, which has a demon WHENFILLED COMPUTEDATE. This directs GUS, upon receiving the information "next Tuesday", to compute the month, day, and year from contextual knowledge (e.g. knowledge of the current date and day).

GUS uses several permanent knowledge structures. A stem dictionary contains a list of stems and idioms; together with morphological rules, it is used to determine word meanings. A transition network grammar is used for syntactic analysis. This is unlike MARGIE and SAM, whose parsing phase is semantics-driven rather than syntax-driven. A case-frame dictionary relates to each verb a set of semantic cases. These slots are filled in during case-frame analysis. Thus frames are used in semantic analysis as well as in inference. Information about conversational patterns is stored in domain-specific frame forms. These relate words and phrases commonly used in conversation to the domain of travel. For example, in "I want to go to Florida", the phrase "want to go" is interpreted as a plan to take a trip, causing TRIP-SPECIFICATION to be instantiated; the agent of go is interpreted as the TRAVELLER in the TRIP-SPECIFICATION; and FLORIDA is the DESTINATION. A dialog query map, which is a set of templates for questions the system may ask, is used in generating GUS's questions to

the client, and a flight description template, together with a data base containing the Official Airline Guide, is used to generate responses to the client's questions. Finally, of course, frames and attached procedures are used for the inference component of the system, and to direct the dialog.

2. Processing

GUS uses frames to direct the dialog by instantiating a top-level dialog frame. The system then goes through the slots of this frame in order, trying to find fillers. When a slot is filled by a new frame instance, the system immediately tries to fill the slots of the new instance. Thus slot-filling is ordinarily done depth-first through the hierarchy of frames. Since GUS is trying to fill slots, if ASKCLIENT is the servant for a slot, GUS maintains control of the conversation. Slots may be filled out of this depth-first sequence if the client volunteers information, or if attached procedures are called.

VI. Ms. Malaprop

Another system which uses frames is Ms. Malaprop [Charniak,1978]. Ms. Malaprop is a system which reads stories about painting (walls and houses, not pictures) and answers questions about them. Real-world knowledge in the domain of painting is crucial to understanding. The purpose of the system is to demonstrate a knowledge representation of events and causes in a specific domain, and to use that knowledge in making inferences in language understanding.

1. Knowledge Representation -- Frames

The goals of the knowledge representation are:

1.) Modularity -- if a process can occur in several domains (e.g. evaporation can occur in painting, but also in such domains as boiling and distillation), there should be a separate, single frame representation for that process rather than incorporating slightly different versions of the frame for the process into each of the domain frames.

2.) Separation of worldly knowledge from control knowledge -- systems like PLANNER, which attach control knowledge to worldly knowledge, have the problem that the worldly knowledge can only be used in the way that the control knowledge directs.

3.) Cleanliness -- frames should be precisely

defined with no procedural embedding.

4.) Adaptability to problem-solving -- in a complex, problem-oriented domain (such as painting), language understanding may involve some problem-solving.

The basic type of knowledge represented in Ms. Malaprop is how-to-do-it knowledge. The statements express states to be achieved rather than actions to perform.

Objects used in painting are represented as variables, which are declared in frame form with value restrictions. Absolute restrictions specify conditions the values must meet. For example, the instrument used in painting must be solid. Normal restrictions specify defaults. For example, the instrument is normally a paint brush.

The process of painting itself is represented as a complex event. Complex events are similar to scripts. They have three main parts. The first part is the variable list, described above. The second part specifies the goal of the event and a name given to the goal. For example, the goal of painting is called PAINTING-GOAL, which is (PAINT coats OBJECT), where OBJECT is one of the variables. The third part is the event, which specifies the temporal sequence of states to be achieved in reaching the goal. The event may

contain selection and repetition of states.

In addition, complex events have two types of links. The COMES-FROM link, attached to one of the states in the event part, is used to specify statements in other structures which indicate how the state may come to be achieved. For example, the state called PAINTING1, which is (OBJECT is clean), in the PAINTING complex event, has a COMES-FROM link to the goal of the complex event WASH. This goal is called WASH-GOAL and is (WASH-OBJECT is clean), where WASH-OBJECT is a variable which can be matched to the variable OBJECT in PAINTING. This, of course, limits the additivity of the knowledge representation, since in order to add a new complex event frame which uses other frame modules to achieve its states, one would have to know the names of the statements used inside the other frames.

The LEADS-TO link is used to find a cause-effect relationship which can help in inferring the reasons for actions. In PAINTING, we have the state PAINTING8, which is (NOT (STICKY-ON SOME-PAINT INSTRUMENT)), which indicates the necessity of cleaning the brush after painting. This state is linked to a simple event called PAINT-DRY via a LEADS-TO link. A simple event expresses a single cause-effect relationship of the form (pre-condition CAUSES condition) as its event part. A pre-

condition of PAINT-DRY is PAINT-DRY1, which is (STICKY-ON PAINT OBJECT). A resulting condition is PAINT-DRY4, which is (NOT (ABSORBENT OBJECT)). Note that PAINT-DRY1 matches the negation of PAINTING8. The LEADS-TO link thus allows the system to answer the question, "Why did he clean the brush?" with "Because if he didn't, it would no longer be absorbent.".

State frames describe relations between given state and other states. For example, the LIQUID-IN state frame indicates that if an object is in liquid, then it is not in contact with the atmosphere. This helps in answering questions about why the painter might leave the brush in the paint when he takes a break.

Another type of frame is the object frame. These describe physical objects. The object frame for PAINT-BRUSH has heading PAINT-BRUSH OBJECT and contains variables BRUSH, BRISTLES, and HANDLE. The description part of the frame contains conditions on the variables which make it a paint brush. Examples are (SOLID HANDLE) and (PART-OF HANDLE BRUSH). There is also a LEADS-TO link in the PAINT-BRUSH frame to PAINTING-BRUSH, which is (PAINT-BRUSH INSTRUMENT). This is one of the normal restrictions on the INSTRUMENT variable in the PAINTING complex event frame, and since it matches the heading of the PAINT-BRUSH frame, we infer that an

instrument which satisfies these conditions is a normal instrument for painting.

Adjunct frames are used for specific examples of complex events. For example, the PAINTING complex event contains nothing which would make it specific to brush or roller painting. Instead, there are adjunct frames for each which fill in the specifics. Thus the ROLLER-PAINTING adjunct frame has variables INSTRUMENT and TRAY. In this case, INSTRUMENT is normally a roller (something which satisfies the description in the ROLLER object frame). The MASTER for this adjunct frame is the PAINTING complex event. The event portion of the adjunct frame refers to statements in the master. For example, the statement

(DURING PAINTING3 ROLLER-PAINTING1)

with

ROLLER-PAINTING1 (FLUID-CONTAINMENT TRAY PAINT)

indicates that in order to achieve PAINTING3, which is:

```
(LOOP
  PAINTING4 (PAINT on INSTRUMENT)
  PAINTING5 (INSTRUMENT in contact with OBJECT) ) ,
```

if we are roller painting, the tray must contain fluid. Also

PAINTING4 COMES-FROM: (ROLL3 (ROLL INSTRUMENT TRAY))

indicates that the way to achieve PAINTING4 (that there is some paint on the instrument), if we are roller

painting, is by rolling the instrument in the tray.

2. Processing

Internally, a frame is stored as an atom with a property list of properties such as VARS, EVENT. Frame statements are stored as atoms also, with properties such as BODY and COMES-FROM. Frame statements are indexed according to predicates in which they appear, thus affording two-way linking of knowledge structures.

As is implied by the word "frame", a complex event frame becomes active when it is mentioned in the story. Since stories are short, frames do not become inactive later, that is, unlike SAM, the system does not forget. Removing useless frames is something that would be necessary if stories were long, or if they changed situations often. The list of active complex event frames is called the context list.

Likeliness is used when information indicating how a variable might be bound is received. This is done by classifying bindings as GOOD, BAD, SATISFACTORY, or POSSIBLE. A GOOD binding indicates that the variable had been previously bound to the object, or that none of the other three are applicable. A BAD binding is used when the variable had been previously bound to something else, or when none of the normal conditions on the variable are matched by the object. A SATISFACTORY

binding is given if there are no normal conditions and all of the absolute conditions are satisfied. A POSSIBLE binding is used if the only unsatisfied normal conditions deal with properties the object should have rather than with the type of object it should be.

Ms. Malaprop does deductive read-time inferences of three types. Consistency inferences involve resolving contradictions. Whenever a new fact is added to the data base, the system checks to see if it is the negation of some fact already in the data base. If so, the system resolves the inconsistency by deleting either the original fact or the negation. If the original fact is deleted, the facts inferred from it may be incorrect. So Ms. Malaprop adds their negations to the data base in order to induce a new round of consistency inferencing. This process is repeated until all inconsistencies are removed. This is facilitated by the two-way linking of inferences -- when a fact is inferred from another, this is recorded on both sides.

Unexpected situation inferences are made when something unexpected, according to the active frames, takes place. That is, the story gives information which does not fit into any slot. Since these are likely to lead to important facts, the system attempts to infer the consequences.

Now we will see an example of the third type of inference, a needed-fact inference. In order to prove the assertion (PHYS-OB PAINT-BRUSH1), the system looks at the state frame for PHYS-OB. This frame contains the relation

```
(PHYS-OB1 (PHYS-OB OB)
IFF (OR
      PHYS-OB2 (SOLID OB)
      PHYS-OB3 (LIQUID OB)
      PHYS-OB4 (GAS OB)).
```

Recall that PHYS-OB1, PHYS-OB2, PHYS-OB3, and PHYS-OB4 are just names for the statements that follow them. Thus this relation states that something is a physical object iff it is a solid, liquid, or gas. The state frame for PHYS-OB also contains the line IF-NEEDED: ((INFER-FROM PHYS-OB1)). This directs the system to use the rule PHYS-OB1 to prove that something is a physical object. The system will now use the SOLID state frame (then the LIQUID and GAS state frames, if necessary) to try to prove (SOLID OB). The SOLID state frame has the variable SOL and the line IF-NEEDED: ((ISA-LINK SOL)). This directs the system to use the ISA link for the variable SOL, which is bound to OB, which is bound to PAINT-BRUSH1. Since PAINT-BRUSH1 ISA PAINT-BRUSH and the PAINT-BRUSH object frame describes a paint brush as a solid, the assertion is proven. ISA is used here not as a predicate, but as a pointer to the most restricted

object type of the given object, so in the case of PAINT-BRUSH1, the ISA pointer stores PAINT-BRUSH.

3. Discussion of Frames

Frames make "explicit the meaning relationship among thoughts" by identifying thought as fillers of slots in frames which were instantiated by other thoughts. Charniak lists three reasons why frames are popular knowledge structures for inference in natural language understanding:

1. Frames are a natural way to partition knowledge. By only bringing in the relevant frames, the combinatorial explosion of inferences is eliminated.

2. Frames provide for the broad, shallow inferencing useful in understanding language. To understand language, we often need a broad knowledge of the aspects of a situation, but we seldom have to do the very deep sort of reasoning necessary in problem-solving. For example, in understanding a story about a restaurant, we need to know about ordering from a menu, paying, tipping, seating, and a variety of other aspects, but we seldom have to make a deep inference about the motives of a waiter in wanting a tip (he needs a new car, or wants to take a vacation). Frames ordinarily contain the former knowledge, but not the latter.

3. The representation is more natural than in deductive methods such as predicate calculus. For example, it is much easier to express type constraints on variables.

Frames and scripts are based on the frame comprehension hypothesis: that a major part of understanding is the matching of incoming information against framed knowledge of what normally occurs. The problem with frames is that they are inadequate for understanding tasks in which the understander does not know what normally occurs, that is, when he is asked to understand a situation about which he knows nothing, and cannot therefore make reliable predictions.

VII. PAM

PAM [Wilensky,1981] is was written to correct the inadequacy of the frame hypothesis for understanding situations about which the reader has no prior stereotypical knowledge. Human understanders realize that an event in a story is part of a plan that an actor is carrying out in order to meet a goal, even if they have never had such a goal or used such a plan. In addition to the plan/goal theory in [Schank and Abelson, 1977], PAM is able to deal with plan/goal situations involving the following:

1.) Goal subsumption -- A recurring goal may lead an actor to plan for all occurrences at the same time.

2.) Goal conflict -- A character may have several goals which are in conflict with one another.

3.) Goal competition -- Several characters may have goals which are in conflict with one another.

4.) Goal concord -- Characters may have goals which coincide.

1. Knowledge Representation -- Goal/plan rules

PAM's knowledge base consists of knowledge about planning. PAM does not do planning, but it needs this knowledge to generate explanations of actions which appear to be goal-based. The planning knowledge is encoded in rules, which are stored as LISP atoms with

properties including conditions, actions, and suggestions.

Conditions are predicates in CD form. These predicates indicate the conditions which must be found in the story in order for the rule to be applied. For example, the condition for HUNGER-RULE is that the CD representation of an event in the story matches the CD representation for "X is hungry.", where X is some actor in the story.

Actions of a rule are used to indicate what should be done when the rule is applied. The actions for HUNGER-RULE direct the system to add a goal, a source of the goal, and a plan for attaining the goal to the knowledge. The goal role is filled with the "Satisfy-hunger" goal for some planner. The source of the goal is the theme of having the hunger drive, and the plan role is left empty until input indicating how the actor intends to achieve the goal is found.

Gaps in rule structures are filled by appealing to the suggestions for the rule. Suggestions indicate the target gap to be filled, the place to look to fill it, and a possible rule to apply. Thus HUNGER-RULE contains a suggestion whose target is the plan role. This suggestion directs the system to focus on INPUT (the story, or information inferred from the story) to see

what the plan for satisfying hunger might be. The rule to apply to fill the gap is called a request. The request rule for this suggestion in HUNGER-RULE is SUITABLE-PLAN-RULE. The condition of SUITABLE-PLAN-RULE is SUITABLE-PLAN-PREDICATE, which examines a list of plan rules associated with a goal. Among the plan rules associated with the goal of hunger satisfaction is a rule called DO-RESTAURANT-PLAN-RULE. Thus if it is learned that an actor who is known to have a goal of satisfying hunger subsequently enters a restaurant, the system is able to identify this as part of a plan to attain a goal, and has thus explained the connection between the sentences.

2. Processing

PAM's inferencing component combines top-down predictive ability with bottom-up inferencing. The latter is necessary because PAM does not rely on the availability of a frame or script, since it attempts to understand new situations. We may note here that MARGIE's inferencing was primarily bottom-up, in that inferences were generated from the input to form a context. On the other hand, frame and script systems have inferencing which is primarily top-down, in that once a script or frame is chosen, predictions are made from that knowledge structure to form a context, and the

input is then matched against the predictions. PAM does inferencing in both directions.

PAM processes an input according to the following algorithm. First, the predictive component determines if there is a prediction which is confirmed by the input. If there is, the prediction explains the input. If not, the bottom-up component takes over. This component attempts to draw inferences from the input, which are then checked against the predictions. When an inference is found that matches a prediction, the input and inferences made in explaining it are added to the story representation by a third component called the incorporation component.

The interaction of these components, and how they use the knowledge structures described above can be seen by examining how PAM processes the story "John was hungry. He ate at a restaurant." When the inferencer receives the Conceptual Dependency representation of the first sentence, there are not yet any predictions which explain the input, so the predictive component turns over control to the bottom-up component. The CD form of the sentence matches the condition for HUNGER-RULE, i.e. that someone is hungry. The HUNGER-RULE action structure is then built in memory, with the roles PLANNER and PLAN left as gaps. The focus of the

suggestion for filling the PLANNER gap is the ACTOR of the CD form of the sentence. The request is FOCUS-REQ, which moves the focus into the target if the focus is not empty. The focus is already filled with JOHN, and so this gap is filled immediately.

The bottom-up component then passes the structure back to the predictive component. Finding fulfillment of a prediction is implemented as determining if there is a request focussed on the input structure whose condition is met by the structure. There are some very general predictions in PAM before a story even starts so that the processing of the first sentence may terminate (inferencing terminates when a prediction has been confirmed). One such prediction is a request whose condition looks for a structure in the input that has a theme in it. The structure for HUNGER-RULE satisfies this condition, so this prediction is fulfilled, and the incorporation component attaches this structure to the story representation.

The second sentence does not immediately fulfill the prediction of the first, i.e. that a plan for satisfy-hunger will be found, since the second sentence merely states an action. The bottom-up component finds a rule called DO-RESTAURANT-PLAN-RULE whose condition, that a person enters a restaurant script, is met by the

input. This rule has an action which creates a structure for a plan, DO-RESTAURANT-PLAN, rather than for a goal as the first sentence does. The gaps of this plan are the PLANNER, which is filled in by using the request FOCUS-REQ with the focus being the CUSTOMER of the RESTAURANT script as it appears in the input CD; the RESTAURANT, which is filled similarly, and the ACTIONS, which is used to specify the actual events and is thus filled with the entire CD input action. Now the predictive component finds that the request for the PLAN in HUNGER-RULE is focussed on input, where the structure for the restaurant plan is located. Furthermore, the request, SUITABLE-PLAN-RULE, has its condition met by this plan, since it is one of the plans for satisfying hunger. Thus the incorporation component can attach the restaurant plan to the PLAN gap of the HUNGER-RULE, which means that the system has recognized the second sentence as a plan for attaining the goal of satisfying the condition stated in the first.

3. Discussion

PAM was able to understand a variety of goal-based stories, even when the situation was one for which it had no framed knowledge. This was due to its knowledge of plans and goals embodied in its rules, which can be applied in the absence of stereotypical situations.

The problem with PAM was that its bottom-up inferencing, while restricted to inferences involving plans and goals, still led to a problem of uncontrolled, irrelevant inferences. The theory behind PAM was that the plans and goals of all actors should be constantly monitored. [Schank,1979] gives an example of a sentence in which PAM would, with a full body of planning knowledge, generate irrelevant inferences which a human understander would probably not bother to think about: "A small twin-engine airplane carrying federal marshals and a convicted murderer who was being transported to Leavenworth crashed during an emergency landing at O'Hare Airport yesterday." PAM would generate goal/plan inferences about the marshals, the murderer, the pilot, and maybe even the implicitly mentioned air controllers. The goal of the marshals is to transport the convict to prison, which a subsumption of the higher level goal of keeping a job, which they do to get money, and so on. These inferences are not really necessary to the understanding of the sentence, but are generated in an attempt to keep track of all goals and plans.

VIII. Commonsense Algorithmic Knowledge

Like Ms. Malaprop, a system by [Rieger,1976] uses knowledge structures and processes which are adaptable to both problem-solving and language comprehension. Since our topic is not problem-solving, we will discuss only the knowledge structures and how they are used in language comprehension.

1. Knowledge Representation -- Commonsense Algorithms

The basis of the knowledge structure is commonsense algorithmic knowledge, which represents dynamic knowledge relating to actions, states, causality, and enablement. The events are classified as actions, states, statechanges, tendencies, and wants. An action is simply something an actor can do, such as grasp or strike. A state is an actorless world condition, such as (LOCATION JOHN HOME(JOHN)) which indicates that John is at his home. An example of a statechange is then (LOCATION JOHN HOME(JOHN) OFFICE(JOHN)) which indicates that John's location changes from his home to his office. A tendency is an actorless action which occurs whenever enabling conditions are satisfied, such as gravity. A want is a state or statechange which an actor desires.

The commonsense algorithms also contain links between the events. Each of the links has restrictions

on the type of events it may connect. The links may represent causality, enablement, concurrency, iteration, gating, or intent.

Causality links connect actions or tendencies to states or statechanges. A causality link may be one-shot or continuous, indicating whether the action or tendency needs to be applied continuously in order to produce the state or statechange. Causality links of either type may also be gated, that is, there may be a set of states associated with the link which represent conditions which must be satisfied in order for the state or statechange to occur. By-product links may also be one-shot or continuous and gated or not gated. They are used to represent states or statechanges which do not occur directly as a result of an action, but occur nonetheless. Thus the direct statechange a person experiences when reading is an increase in knowledge or pleasure, the by-product statechange may be tired eyes.

State-coupling links express equivalence of states or statechanges with unspecified causality. These may also be gated. For example, in the presence of the states (ATTACHED W Z) and (MOVEABLE Z), the statechange (LOCATION W X Y) will be equivalent to the statechange (LOCATION Z R S), where R and S depend upon X and Z. This allows the representation of the fact that when

John moves his arm, he also moves his hand.

Threshold links express good continuation of a statechange to some desired level. Thus in the presence of the gating condition (FACING X Z), the action (WALK X) produces a statechange (LOCATION X LOC(X) Z). This is linked by a threshold link to the state (LOCATION X Z), showing that the continuation of the statechange will eventually produce the resulting state.

Enablement links represent connection between actions or entire algorithms which may be viewed as primitive for some purpose, and states which are preconditions for those actions or algorithms.

For language comprehension (as well as for problem-solving), commonsense algorithms are organized in networks called causal selection networks. The representation is explicit rather than embedded in procedures so that causality and enablement can proceed in both directions. Each network represents one state or statechange concept. Each node of the network, organized as a tree, is a test for checking real-world knowledge or contextual knowledge in order to decide how to proceed.

At the leaves of the tree are commonsense algorithm patterns for achieving the state or statechange which the network represents. These algorithms are called

approaches, and are of three types: abstract algorithms, mechanism descriptions, and sequential abstract algorithms.

Abstract algorithms are themselves divided into forms. The first is a causality link connecting an action to a statechange. The second consists of a tendency, with enablement links from states, connected by a causality link to a state or statechange. The last form of abstract algorithm is a state-coupling link. The state or statechange at the bottom of each represents the goal state for the network.

Mechanism descriptions represent internal cause-effect relationships between events in a mechanism. A sequential abstract algorithm is a linear list of abstract algorithms stored when a plan is found which solves a problem, and recalled when the problem arises again.

The gate conditions of the approaches may have recommendations, which are pointers to other abstract algorithms which contain those states or statechanges. The first time an algorithm is reached, a gating condition has no recommendations, so the network which has it as a goal must be traversed. Once this process has found the abstract algorithm for the gating condition, the two are linked, so that if the gating

condition is encountered again, the abstract algorithm producing it can be found without traversing the network again.

Networks or portions of networks can be bypassed in another way. Once a test is performed in one network, its result can be stored, allowing the test to be bypassed if it occurs again with no relevant statechanges between occurrences. This is done by adding a path from the test which precedes the given test to the test which is performed afterward whenever that result occurs. Since tests are shared among networks, this is done in every network in which that test occurs. The system then seeks the transitive closure of all these links, allowing it possibly to skip entire networks.

2. Processing

Causal selection networks are used for language understanding according to the prediction/fulfillment model of comprehension. The model can be made explicit as follows: given a context C(T1), elucidate the relationship between it and the next thought T2. Call this the interpretation of T2 in this context, and denote it I(T2,C(T1)). T1 is used to generate expectations (predictions) of actions and see how the subsequent thought T2 fits in. One way to do this is to

start with goals, which are expressed as wants of states or statechanges. Predicting the goals then amounts to identifying the tops of the causal selection networks as representing likely goals. The approaches at the bottom of the networks, together with their subgoals and recommended actions, then represent the realm of predicted actions.

In order to do this, the idea of a causal selection network is adapted by defining two new types of networks: inducement networks and prediction networks. Inducement networks are used to discover what states some action or state might induce in a potential actor. These are similar to causal selection networks, except that at the leaves there is a set of internal states which might be induced. Thus each network represents and inducing action or state; traversing the network amounts to discovering what it might induce. As each thought enters, the appropriate network is traversed for each potential actor. The states that are discovered represent the input to the prediction networks.

Prediction networks are used to discover the goals some internal state might cause an actor to have. At the leaves is the set of possible goals. Thus each prediction network represents a certain state in an actor; traversing the network amounts to discovering the

predicted goals of the actor. These goals are then input into causal selection networks, which discover actions from knowledge of goals.

To bypass prediction networks and causal selection networks, inducement networks may contain recommendations of actions. This has the advantage of using not only the internal state of an actor for prediction (which is all that a prediction network has available), but also the action or state (available to an inducement network) which induced the internal state. Clearly, an actor's response to an internal state might well depend on what caused that state. To produce higher order inferences, the predicted goal states may be fed back into inducement networks to see what states they might cause.

Having finished the prediction phase, which depends only upon the context, the system will then see how the input thought fulfills the predictions. Each input thought is matched with an abstract algorithm which may occur at the bottom of a causal selection network. The system then searches up the causal selection network hoping to find a goal in the prediction set. If one is found, I(T2,C(T1)) has been determined.

This upward traversal is done as follows. At each node, pose the test. If the test would have directed

the system to the node it came from below, keep going. If this does not happen several times in a path, try another path. When the top of a network is reached, if the goal represented by the network is in the prediction set, the system has understood how the new thought fits into the context. If not, the goal is considered a higher-level goal. The system looks for occurrences of it in abstract algorithms at the leaves of other networks, and then traverses up these, looking for something in the prediction set.

This process stops when the number of networks traversed reaches a cutoff value. Anything beyond this level would be both a remote interpretation and expensive. If there is more than one interpretation, choose the one with the fewest networks traversed. In case of a tie, choose the one which fared better on the tests at each node.

Bypassing tests may occur in two ways. First, recommendations are preferred over networks in climbing up. Second, bypasses may be implanted by climbing down again along the interpretation path.

3. Discussion

Abstract algorithms and causal selection networks are an efficient way of organizing knowledge for problem-solving and language comprehension. In language

comprehension, they are especially useful for stories involving goals and plans, in that the input is used to infer a state in an actor, which is in turn used to predict a goal the actor might have, which is then used to predict his actions by identifying them as part of a plan to achieve the predicted goal.

As in PAM and MARGIE, much of the inferencing consists of bottom-up prediction-making. While this gives the system the ability to work in situations in which it has no stereotypical knowledge, it also produces the problem of uncontrolled inferencing. The network system artificially controls the inferencing for efficiency purposes by restricting the number of networks traversed, just as MARGIE has artificial controls on the depth of its inferencing. The problem is that although there is probably a negative correlation between depth and relevance, that correlation is not necessarily -1, that is, there may be relevant inferences which can be obtained only by deeper inferencing than can many irrelevant ones. Thus in order to guarantee that all relevant inferences are generated for prediction, the depth may have to be set in such a way as to also generate the irrelevant ones.

IX. BORIS

A system that uses a variety of knowledge structures for inference is BORIS. The story below, which BORIS understands in depth, is included as a source of examples for the following discussion of the system's knowledge structures and processes.

> Richard had not heard from his college roommate Paul for years. Richard had borrowed money from Paul which was never paid back. But now he had no idea to find his old friend. When a letter arrived from San Francisco, Richard was anxious to find out how Paul was.
>
> Unfortunately, the news was not good. Paul's wife Sarah wanted a divorce. She also wanted the car, the house, the children, and alimony. Paul wanted the divorce, but he did not want to see Sarah walk off with everything he had. His salary from the state school system was very small. Not knowing who to turn to, he was hoping for a favor from the only lawyer he knew. Paul gave his home phone number in case Richard could help.
>
> Richard eagerly picked up the phone and dialed. After a brief conversation, Paul agreed to have lunch with him the next day. He sounded extremely relieved and grateful.
>
> The next day, as Richard was driving to the restaurant, he barely avoided hitting an old man on the street. He felt extremely upset by the incident, and had three drinks at the restaurant. When Paul arrived, Richard was fairly drunk. After the food came, Richard spilled a cup of coffee on Paul. Paul seemed very annoyed by this, so Richard offered to drive him home for a change of clothes.
>
> When Paul walked into the bedroom and found Sarah with another man, he nearly had a heart attack. Then he realized what a blessing it was. With Richard there as a witness, Sarah's divorce case was shot. Richard congratulated Paul and suggested that they celebrate at dinner. Paul was eager to comply.

In order to understand this story in depth, BORIS deals with the following problems, which are not dealt with as effectively in the other systems reported in this paper:

1. Inferences must often be made from the absence of a fact or event. For example, from "John walked into the room and Mary was not there.", the system should infer that John wanted to see Mary, not just that Mary is somewhere other than in the room.

2. Inference of motives of characters may need to use knowledge of the relationships between characters. Thus Richard's motive for helping Paul is understood by BORIS to be the desire to return Paul's favor of lending him money.

3. Knowledge of human situations is needed for in-depth inference. Examples of such situations in the story are divorce, borrowing and repayment, writing and receiving letters.

4. Figures of speech need to be understood. When we read that Paul did not want to "see" his wife "walk off" with everything he had, we understand that "see" and "walk off" are not to be taken literally.

5. Pronoun reference often requires knowledge and inference in addition to syntactic and semantic rules for resolution.

1. Knowledge representation -- MOPs and TAUs

The main permanent knowledge structure which the parsing module examines is the dictionary. Each lexical item in the dictionary has associated knowledge structures and demons. Demons represent procedural knowledge; their purpose is to search and construct episodic memory. Demons may spawn other demons. Demons may be used to fill slots in knowledge structures, to determine character's plans and goals, to deal with settings, to deal with events, or to handle prediction fulfillment or violation.

BORIS also uses a variety of knowledge structures which we have already seen, such as scripts, settings, and events, and two we have not seen, which are known as Memory Organization Packets (MOPs) and Thematic Affect Units (TAUs). The knowledge structures are associated with lexical items in the dictionary for instantiation, and are integrated by a set of links, as discussed below.

A MOP is a configuration of Conceptual Dependency graphs formed into a discrete knowledge structure by a standard set of links. MOPs differ from scripts in that they focus on goals and intentions. For example, the MOP for borrowing and lending involves two objects, the BORROWER and the LENDER. The concepts in the MOP are:

WANT-OBJECT (BORROWER)
ASK-FOR-OBJECT (BORROWER to LENDER)
CONVINCED-TO-LEND (LENDER)
GIVE-OBJECT (LENDER to BORROWER)
WANT-TO-RETURN (BORROWER)
WANT-RETURNED (LENDER)
GIVE-OBJECT-BACK (BORROWER to LENDER).

The concepts are connected by links representing intention (WANT-OBJECT and ASK-FOR-OBJECT), motivation (GIVE-OBJECT and WANT-RETURNED), or achievement (WANT-OBJECT and GIVE-OBJECT or WANT-RETURNED and GIVE-OBJECT-BACK).

MOPs may be linked to other MOPs by MOP links. Examples of MOP links involving the borrowing and lending MOP would be links to FAVOR and BUSINESS-CONTRACT. The links indicate under what conditions the MOP linked to should be instantiated. In this example, FAVOR is instantiated if the BORROWER and LENDER are friends, otherwise, BUSINESS-CONTRACT is used. The MOP links are necessarily uni-directional --the BORROWING and LENDING MOP is brought in when borrowing or lending is mentioned in the story, and since it is usually important to know the reason for the transaction, FAVOR may be brought in. However, when the story mentions a favor one character does for another, it is not necessary to bring in BORROWING and LENDING and all the other MOPs which may be interpreted as possibly involving favors. Indeed, in the story, Richard's

motivation for doing something for Paul is best interpreted in terms of FAVOR rather than BORROWING and LENDING. Even if the money had been paid back, in which case all of the motives and intentions represented in the BORROWING and LENDING MOP would have been achieved, Richard may still have considered the loan a favor. The FAVOR MOP would indicate that the normal desire to return a favor had not yet been satisfied, thus explaining Richard's motivation for doing something for Paul.

The modularity of MOPs is also important. It would have been erroneous to represent the idea of favor directly in BORROWING and LENDING, because a loan could be a business contract having nothing to do with favors. Thus the idea of favor is brought in only if it is determined that the loan was probably done as a favor. Thus modularity helps prevent erroneous conclusions. Modularity also improves efficiency. For example, the DIVORCE MOP does not contain information about the legal aspects of divorce, instead, it has a MOP link to LEGAL-DISPUTE. Furthermore, LEGAL-DISPUTE will not always be instantiated in the presence of DIVORCE. Additional information about the divorce, such as the presence of a lawyer, or an indication of a conflict in the goals of the husband and wife, is needed for LEGAL-DISPUTE to be

used with DIVORCE.

When MOPs are linked, BORIS stores a specification of which component of one knowledge structure is equivalent to a component in another structure. For example, the LAWYER MOP contains a PETITION component, indicating that the lawyer petitions on someone's behalf. In our story, since Richard's agreement to petition on Paul's behalf is an attempt to return Paul's favor, this PETITION component is linked to the DO-RETURN-FAVOR component of the FAVOR MOP.

This again improves modularity. The PROF-SERVICE MOP, for example does not need to include information about legal representation or psychiatric analysis; instead, the LAWYER or PSYCHIATRIST MOP is linked to the PROF-SERVICE MOP with the corresponding components specified. This also allows several perspectives: for example, the system can view Richard's representation of Paul as a professional service and as a returned favor.

TAUs are used to represent knowledge that deviates from a normal goal/plan model. TAUs have the following features:

1. They are thematic -- they capture knowledge which people often represent in adages.

2. They are affective -- they can represent such emotions as anxiety, upset, and shock.

3. They can explain variations from goal/plan situations: for example, they can explain surprising events that may cause a plan to go wrong.

4. They are sensitive to points-of-view of different characters in a thematic situation.

An example of a TAU used in the story about divorce is DIRE-STRAITS, which contains knowledge about how people react in a crisis.

BORIS's episodic memory is therefore a collection of instantiations of high-level knowledge structures and the links between them.

BORIS uses a single parsing module for reading the text and building episodic memory. Episodic memory contains the knowledge structures built up from the input so far. Therefore this module may also search previously-produced episodic memory to aid in the parsing task. This same module is used in question-answering. Thus episodic memory units are created as well as searched while parsing questions.

2. Processing

As is the case with Conceptual-Dependency-based systems, parsing in BORIS is semantics-directed rather than syntax-directed. Syntactic information about each word is stored in the lexicon, along with conceptual structures and demons, and is consulted only when needed

for understanding.

Episodic memory is searched during parsing to help to solve the following problems:

1. Inferring roles -- BORIS can infer that in the sentence "The money was never paid back.", it means that it was not paid back by Richard to Paul. This inference is possible because of the recent instantiation of the BORROW MOP with Richard as borrower and Paul as lender.

2. Pronoun resolution -- BORIS can infer that "He" in "He sounded extremely relieved and grateful." means Paul, because of the instantiation of the DIRE-STRAITS TAU.

3. Disambiguation of words -- BORIS can infer that "hitting a man on the street" means hitting with his car rather than getting into fight, since ACCIDENT is available in episodic memory.

3. Discussion

As in PAM, knowledge may be applied bottom-up or top-down. One type of bottom-up application is the episodic memory creation from the input which is described above. Bottom-up application is also used in goal/plan processing, in which rules are used for monitoring goal conflicts, competition, and achievement. The conditions for these rules are events and situations (or non-events and non-situations), and the actions

involve filling slots in goal-oriented MOPs. A third type of bottom-up processing involves scenario-mapping. BORIS follows characters as they proceed from one location to another in a story. This is done because an actor's location may explain his action, or if his action is unusual in that location, this may explain another actor's surprise.

Top-down knowledge application consists of fulfilling predictions by filling slots in high-level knowledge structures. Note the difference between BORIS and PAM in goal/plan processing. PAM creates goal/plan structures bottom-up from the input (by attaching plan rules to the PLAN component of goal rules), which, as [Schank,1979] points out, forces PAM to generate irrelevant inferences about the goals and plans of actors in the story, even if there is little chance that those inferences are needed. BORIS has pre-existing high-level plan/goal knowledge structures (MOPs) with rules that are used to fill their slots. With high-level structures, BORIS controls goal/plan inferences by increasing the role of the predictive top-down component, allowing a corresponding decrease in the less controlled bottom-up inferencing. This eliminates many of the useless inferences generated by PAM, without sacrificing the ability to understand goals and plans.

Bibliography

Bobrow, D.G., Kaplan, R.M., Kay, M., Norman, D.A., Thompson H., and Winograd, T. (1977). "GUS, a Frame-Driven Dialog System" Artificial Intelligence, Vol. 8, pp.155-174.

Charniak, E. (1978). "On the Use of Framed Knowledge in Language Comprehension" Artificial Intelligence, Vol. 11, pp. 225-265.

Charniak, E. (1981). "A Common Representation for Problem-Solving and Language Comprehension Information" Artificial Intelligence, Vol. 16, pp. 225-256.

Cullingford, R. (1981). "SAM". In R.C. Schank and C.K. Riesbeck (Ed.), Inside Computer Understanding: Five Programs Plus Miniatures. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Hewitt, C. (1969). "PLANNER: A Language for Proving Theorems in Robots" Proceedings of the International Joint Conference on Artificial Intelligence, pp. 295-301.

Lehnert, W., Dyer, M.G., Johnson, P.N., Yang, C.J., and Harley S. (1983). "BORIS --An Experiment in In-Depth Understanding of Narratives" Artificial Intelligence, Vol. 20 (1983), pp. 15-62.

Minsky, M. (1975). "A Framework for Representing Knowledge". In P.H. Winston (Ed.), The Psychology of Computer Vision. New York: McGraw-Hill.

Rieger, C. (1976). "On Organization of Knowledge for Problem Solving and Language Comprehension" Artificial Intelligence, Vol. 7, pp. 89-128.

Schank, R.C. (1979). "Interestingness: Controlling Inferences" Artificial Intelligence, Vol. 12, pp. 273-298.

Schank, R.C., and Abelson, R. P. (1977). Scripts, Plans, Goals, and Understanding. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Schank, R.C., and Rieger, C.J. (1974). "Inference and the Computer Understanding of Natural Language" Artificial Intelligence, Vol. 5, pp. 373-412.

Wilensky, R. (1981). "PAM". In R.C. Schank and C.K. Riesbeck (Ed.), Inside Computer Understanding: Five Programs Plus Miniatures. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Winograd, T. (1971) "Procedures as a Representation for Data in a Computer Program for Understanding Natural Language". Ph. D. Dissertation. Cambridge, Massachusetts: Massachusetts Institute of Technology Project MAC.

Vita

Robert J. Harwick, born August 26, 1953 in Emmaus, Pennsylvania, graduated from Emmaus High School in 1971. After graduating summa cum laude from Ursinus College as valedictorian with a Bachelor of Science in mathematics in 1975, he entered graduate school at Lehigh University. Despite a (thankfully brief) interruption in his studies, during which he taught mathematics at the secondary level, he received a Master of Science degree in mathematics in 1978.

This was followed by a one-year appointment as a mathematics instructor at Lafayette College, after which he returned to Lehigh to study computer science. While working as a systems programmer for Rapidata, Inc. in 1981, he was offered a teaching position at Allentown College of St. Francis De Sales, which allowed him to continue his studies of computer science at Lehigh, while teaching mathematics and computer science.

Mr. Harwick married Wendy Richards of Bethelehem, Pennsylvania in August, 1982. Their daughter Rebecca was born in November, 1983. Together, they are members of Holy Cross Evangelical Lutheran Church in Bethlehem.