The Use of Object-Specific Knowledge in Natural Language Processing

Mark H. Burstein
Department of Computer Science, Yale University

1. INTRODUCTION

It is widely recognized that the process of understanding natural language texts cannot be accomplished without accessing mundane knowledge about the world [2, 4, 6, 7]. That is, in order to resolve ambiguities, form expectations, and make causal connections between events, we must make use of all sorts of episodic, stereotypic and factual knowledge. In this paper, we are concerned with the way functional knowledge of objects, and associations between objects can be exploited in an understanding system.

Consider the sentence

(1) John opened the bottle so he could pour the wine.

Anyone reading this sentence makes assumptions about what happened which go far beyond what is stated. For example, we assume without hesitation that the wine being poured came from inside the bottle. Although this seems quite obvious, there are many other interpretations which are equally valid. John could be filling the bottle rather than emptying the wine out of it. In fact, it need not be true that the wine ever contacted the bottle. There may have been some other reason John had to open the bottle first. Yet, in the absence of a larger context, some causal inference mechanism forces us (as human understanders) to find the common interpretation in the process of connecting these two events causally.

In interpreting this sentence, we also rely on an understanding of what it means for a bottle to be "open". Only by using knowledge of what is possible when a bottle is open are able we understand why John had to open the bottle to pour the wine out of it. Strong associations are at work here helping us to make these connections. A sentence such as

(2) John closed the bottle and poured the wine.

appears to be self contradictory only because we assume that the wine was in the bottle before applying our knowledge of open and closed bottles to the situation. Only then do we realize that closing the bottle makes it impossible to pour the wine.

Now consider the sentence

(3) John turned on the faucet and filled his glass.

When reading this, we immediately assume that John filled his glass with water from the faucet. Yet, not only is water never mentioned in the sentence, there is nothing there to explicitly relate turning on the faucet and filling the glass. The glass could conceivably be filled with milk from a carton. However, in the absence of some greater context which forces a different interpretation on us, we immediately assume that the glass is being filled with water from the faucet.

Understanding each of these sentences requires that we make use of associations we have in memory between objects and actions commonly involving those objects, as well as relations between several different objects. This paper describes a computer program, OPUS (Object Primitive Understanding System) which constructs a representation of the meanings of sentences such as those above, including assumptions that a human understander would normally make, by accessing these types of associative memory structures. This stereotypic knowledge of physical objects is captured in OPUS using Object Primitives [5]. Object Primitives (OP) were designed to act in conjunction with Schank's conceptual dependency representational system [11]. The processes developed to perform conceptual analysis in OPUS involved the integration of a conceptual analyzer similar to Riesbeck's ELI [9] with demon-like procedures for memory interaction and the introduction of object-related inferences.

2. OBJECT PRIMITIVES

The primary focus in this research has been on the development of processes which utilize information provided by Object Primitives to facilitate the "comprehension" of natural language texts by computer. That is, we were primarily concerned with the introduction of stereotypic knowledge of objects into the conceptual analysis of text. By encoding information in OP descriptions, we were able to increase the interpretive power of the analyzer in order to handle sentences of the sort discussed earlier.

What follows is a brief description of the seven Object Primitives. A more thorough discussion can be found in [5]. For those unfamiliar with the primitive acts of Schank's conceptual dependency theory, discussions of which can be found in [10,11].

The Object Primitive CONNECTOR is used to indicate classes of actions (described in terms of Schank's primitives acts) which are normally enabled by the object being described. In particular, a CONNECTOR enables actions between two spatial regions. For example, a window and a door are both CONNECTORs which enable motion (PTRANS) of objects through them when they are open. In addition, a window is a CONNECTOR which enables the action ATTEND eyes (see) or MTRANS (acquisition of information) by the instrumental action ATTEND eyes. These actions are enabled regardless of whether the window is open or closed. That is, one can see through a window, and therefore read or observe things on the other side, even when the window is closed. In the examples discussed above, the open bottle is given a CONNECTOR description. This will be discussed further later.

A SEPARATOR disenables a transfer between two spatial regions. A closed door and a closed window are both SEPARATORs which disenable the motion between the spatial regions they adjoin. In addition, a closed door is a SEPARATOR which disenables the acts MTRANS by ATTEND eyes (unless the door is transparent) or ears. That is, one is normally prevented from seeing or hearing through a closed door. Similarly, a closed window is a SEPARATOR which disenables MTRANS with instrument ATTEND ears, although, as mentioned above, one can still see through a closed window to the other side. A closed bottle is another example of an object with a SEPARATOR description.

It should be clear by now that objects described using Object Primitives are not generally described by a single primitive. In fact, not one but several sets of

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

use CG to generate

primitive descriptions may be required. This is illustrated above by the combination of CONNECTOR and SEPARATOR descriptions required for a closed window, while a somewhat different set is required for an open window. These sets of descriptions form a small set of "states" which the object may be in, each state corresponding to a set of inferences and associations approriate to the object in that condition.

A SOURCE description indicates that a major function of the object described is to provide the user of that object with some other object. Thus a faucet is a SOURCE of water, a wine bottle is a SOURCE of wine, and a lamp is a SOURCE of the phenomenon called light. SOURCEs often require some sort of activation. Faucets must be turned on, wine bottles must be opened, and lamps are either turned on or lit depending on whether or not they are electric.

The Object Primitive CONSUMER is used to describe objects whose primary function is to consume other objects. A trash can is a CONSUMER of waste paper, a drain is a CONSUMER of liquids, and a mailbox is a CONSUMER of mail. Some objects are both SOURCEs and CONSUMERs. A pipe is a CONSUMER of tobacco and a SOURCE of smoke. An ice cube tray is a CONSUMER of water and a SOURCE of ice cubes.

Many objects can be described in part by relationships that they assume with some other objects. These relations are described using the Object Primitive RELATIONAL. Containers, such as bottles, rooms, cars, etc., have as part of their descriptions a containment relation, which may specify defaults for the type of object contained. Objects, such as tables and chairs, which are commonly used to support other objects will be described with a support relation.

Objects such as buildings, cars, airplanes, stores, etc., are all things which can contain people. As such, they are often distinguished by the activities which people in those places engage in. One important way of encoding those activities is by referring to the scripts which describe them. The Object Primitive SETTING is used to capture the associations between a place and any script-like activities that normally occur there. It can also be used to indicate other, related SETTINGs which the object may be a part of. For example, a dining car has a SETTING description with a link both to the restaurant script and to the SETTING for passenger train. This information is important for the establisnment of relevant contexts, giving access to many domain specific expectations which will subsequently be available to guide processing both during conceptual analysis of lexical input and when making inferences at higher levels of cognitive processing.

The final Object Primitive, GESTALT, is used to cnaracterize objects which have recognizable, and separable, subparts. Trains, hi-fi systems, and kitchens, all evoke images of objects characterizable by describing their subparts, and the way that those subparts relate to form the whole. The Object Primitive GESTALT is used to capture this type of description.

Using this set of primitives as the foundation for a memory representation, we can construct a more general bi-directional associative memory by introducing some associative links external to object primitive decompositions. For example, the conceptual description of a wine bottle will include a SOURCE description for a bottle where the SOURCE output is specified as wine. This amounts to an associative link from the concept of a wine bottle to the concept of wine. But how can we construct an associative link from wine back to wine bottles? Wine does not have an object primitive decomposition which involves wine bottles, so we must resort to some construction which is external to object primitive decompositions.

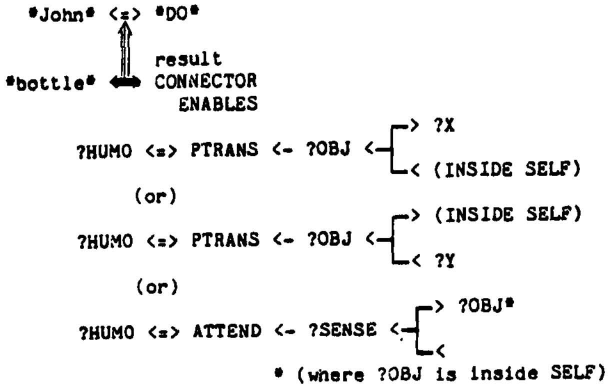
Four associative links have been proposed [5], each of which points to a particular object primitive description. For the problem of wine and wine bottles, an associative OUTPUTFROM link is directed from wine to the SOURCE description of a wine bottle. This external link provides us with an associative link from wine to wine bottles.

3. THE PROGRAM

I will now describe the processing of two sentences very similar to those discussed earlier. The computer program (OPUS) which performs the following analyses was developed using a conceptual analyzer written by Larry Birnbaum [1]. OPUS was then extended to include a capacity for setting up and firing "demons" or "triggers" as they are called in KRL [3]. The functioning of these demons will be illustrated below.

3.1 THE INITIAL ANALYSIS

In the processing of the sentence "John opened the bottle so he could pour the wine," the phrase "John opened the bottle," is analyzed to produce the following representation:


John <=> *DO*
result
bottle CONNECTOR
ENABLES
?HUMO <=> PTRANS <- ?OBJ <
> ?X
< (INSIDE SELF)
(or)
?HUMO <=> PTRANS <- ?OBJ <
> (INSIDE SELF)
< ?Y
(or)
?HUMO <=> ATTEND <- ?SENSE <
> ?OBJ*
<
* (where ?OBJ is inside SELF)

Here SELF refers to the object being described (the bottle) and ?--- indicates an unfilled slot. *John* here stands for the internal memory representation for a person with the name John. Memory tokens for John and the bottle are constructed by a general demon which is triggered during conceptual analysis whenever a PP (the internal representation for an object) is introduced. OP descriptions are attached to each object token.

This diagram represents the assertion that John did something which caused the bottle to assume a state where its CONNECTOR description applied. The CONNECTOR description indicates that something can be removed from the bottle, put into the bottle, or its contents can be smelled, looked at, or generally examined by some sense modality. This CONNECTOR description is not part of the definition of the word 'open'. It is specific knowledge that people have about what it means to say that a bottle is open.

In arriving at the above representation, the program must retrieve from memory this OP description of what it means for a bottle to be open. This information is stored beneath its prototype for bottles. Presumably, there is also script-like information about the different methods for opening bottles, the different types of caps (corks, twist-off, ...), and which method is appropriate for which cap. However, for the purpose of understanding a text which does not refer to a specific type of bottle, cap, or opening procedure, what is important is the information about how the bottle can then be used once it is opened. This is the kind of knowledge that Object Primitives were designed to capture.

When the analyzer builds the state description of the bottle, a general demon associated with new state descriptions is triggered. This demon is responsible for updating memory by adding the new state information to the token in the ACTOR slot of the state description. Thus the bottle token is updated to include the given CONNECTOR description. For the purposes of this program, the bottle is then considered to be an "open" bottle. A second function of this demon is to set up explicit expectations for future actions based on the new information. In this case, templates for three actions the program might expect to see described can be constructed from the three partially specified conceptualizations shown above in the CONNECTOR description of the open bottle. These templates are attached to the state description as possible consequences of that state, for use when attempting to infer the causal connections between events.

3.2 CONCEPT DRIVEN INFERENCES

The phrase "so he could pour the wine." is analyzed as

```
           ⇑  enable                   ┌> ?X
*John* <≡> PTRANS <- *wine* <─┤
                               └< (INSIDE ?CONTAINER)
```

When this representation is built by the analyzer, we do not know that the the wine being poured came from the previously mentioned bottle. This inference is made in the program by a slot-filling demon called the CONTAINER-FINDER, attached to the primitive act PTRANS. The demon, triggered when a PTRANS from inside an unspecified container is built, looks on the list of active tokens (a part of short term memory) for any containers that might be expected to contain the substance moved, in this case wine. This is done by applying two tests to the objects in short term memory. The first, the DEFAULT-CONTAINMENT test, looks for objects described by the RELATIONAL primitive, indicating that they are containers (link = INSIDE) with default object contained being wine. The second, the COMMON-SOURCE test, looks for known SOURCEs of wine by following the associative OUTPUTFROM link from wine. If either of these tests succeed, then the object found is inferred to be the container poured from.

At different times, either the DEFAULT-CONTAINMENT test or the COMMON-SOURCE test may be necessary in order to establish probable containment. For example, it is reasonable to expect a vase to contain water since the RELATIONAL description of a vase has default containment slots for water and flowers. But we do not always expect water to come from vases since there is no OUTPUTFROM link from water to a SOURCE description of a vase. If we heard "Water spilled when John bumped the vase," containment would be established by the DEFAULT-CONTAINMENT test. Associative links are not always bi-directional (vase ---> water, but water -/-> vase) and we need separate mechanisms to trace links with different orientations. In our wine example, the COMMON-SOURCE test is responsible for establishing containment, since wine is known to be OUTPUTFROM bottles but bottles are not always assumed to hold wine.

Another inference made during the initial analysis finds the contents of the bottle mentioned in the first clause of the sentence. This expectation was set up by a demon called the CONTENTS-FINDER when the description of the open bottle, a SOURCE with unspecified output, was built. The demon causes a search of STM for an object which could be OUTPUT-FROM a bottle, and the token for this particular bottle is then marked as being a SOURCE of that object. The description of this particular bottle as a SOURCE of wine is equivalent, in Object Primitive terms, to saying that the bottle is a wine bottle.

3.3 CAUSAL VERIFICATION

Once the requests trying to fill slots not filled during the initial analysis have been considered, the process which attempts to find causal connections between conceptualizations is activated. In this particular case, the analyzer has already indicated that the appropriate causal link is enablement. In general, however, the lexical information which caused the analyzer to build this causal link is only an indication that some enabling relation exists between the two actions (opening the bottle and pouring the wine). In fact, a long causal chain may be required to connect the two acts, with an enablement link being only one link in that chain. Furthermore, one cannot always rely on the text to indicate where causal relationships exist. The sentence "John opened the bottle and poured the wine." must ultimately be interpreted as virtually synonymous with (1) above.

The causal verification process first looks for a match between the conceptual representation of the enabled action (pouring the wine), and one of the potentially enabled acts derived earlier from the OP description of the opened bottle. In this example, a match is immediately found between the action of pouring from the bottle and the expected action generated from the CONNECTOR description of the open bottle (PTRANS FROM (INSIDE PART SELF)). Other Object Primitives may also lead to expectations for actions, as we shall see later.

When a match is found, further conceptual checks are made on the enabled act to ensure that the action described "makes sense" with the particular objects currently filling the slots in that acts description. When the match is based on expectations derived from the CONNECTOR description of a container, the check is a "container/contents check," which attempts to ensure that the object found in the container may reasonably be expected to be found there. The sentence "John opened the bottle so he could pull out the elephant", is peculiar because we no associations exist which would lead us to expect that elephants are ever found in bottles. The strangeness of this sentence can only be explained by the application of stereotypic knowledge about what we expect and don't expect to find inside a bottle.

The container/contents check is similar to the test described above in connection with the CONTAINER-FINDER demon. That is, the bottle is checked by both the DEFAULT-CONTAINMENT test and the COMMON-SOURCE test for known links relating wine and botles. When this check succeeds, the enable link has been verified by matching an expected action, and by checking restrictions on related objects appearing inthe slots of that action. The two CD acts that matched are then merged.

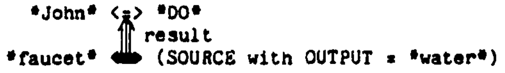
The merging process accomplishes several things. First, it completes the linking of the causal chain between the events described in the sentence. Second, it causes the filling of empty slots appearing in either the enabled act or in the enabling act, wherever one left a slot unspecified, and the other had that slot filled. These newly filled slots can propagate back along the causal chain, as we shall see in the example of the next section.

3.4 CAUSAL CHAIN CONSTRUCTION

In processing the sentence

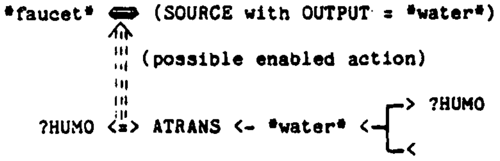
(4) John turned on the faucet so he could drink.

the causal chain cannot be built by a direct match with an expected event. Additional inferences must be made to complete the chain between the actions described in the sentence. The representation produced by the conceptual analyzer for "John turned on the faucet," is



As with the bottle in the previous example, the description of the faucet as an active SOURCE of water is based on information found beneath the prototype for faucet, describing the "on" state for that object. The principle expectation for SOURCE objects is that the person who "turned on" the SOURCE object wants to take control of (and ultimately make use of) whatever it is that is output from that SOURCE. In CD, this is expressed by a template for an ATRANS (abstract transfer) of the output object, in this case, water. An important side effect of the construction of this expectation is that a token for some water is created, which can be used by a slot-filling inference later.

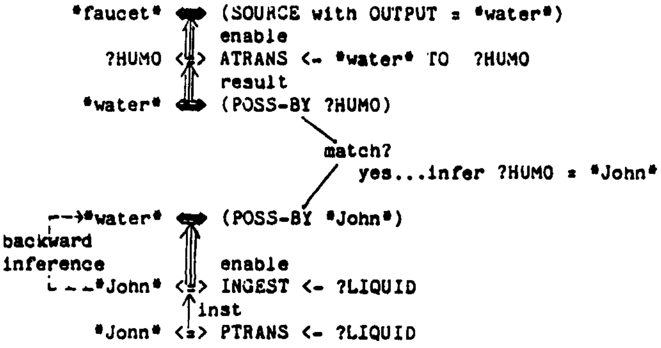
The representation for "he could drink" is partially described by an INGEST with an unspecified liquid in the OBJECT slot. A special request to look for the missing liquid is set up by a demon on the act INGEST, similar to the one on the PTRANS in the previous example. This request finds the token for water placed in the short term memory when the expectation that someone would ATRANS control of some water was generated.


faucet (SOURCE with OUTPUT = *water*)
(possible enabled action)
?HUMO
?HUMO <=> ATRANS <- *water* <
<

The causal chain completion that occurs for this sentence is somewhat more complicated than it was for the previous case. As we have seen, the only expectation set up by the SOURCE description of the faucet was for an ATRANS of water from the faucet. However, the action that is described here is an INGEST with instrumental PTRANS. When the chain connector fails to find a match between the ATRANS and either the INGEST or its instrumental PTRANS, inference procedures are called to generate any obvious intermediate states that might connect these two acts.

The first inference rule that is applied is the resultative inference [8] that an ATRANS of an object TO someone results in a state where the object is possessed by (POSS-BY) that person. Once this state has been generated, it is matched against the INGEST in the same way the ATRANS was. When this match fails, no further forward inferences are generated, since possession of water can lead to a wide range of new actions, no one of which is strongly expected.

The backward chaining inferencer is then called to generate any known preconditions for the act INGEST. The primary precondition (causative inference) for drinking is that the person doing the drinking has the liquid which he or she is about to drink. This inferred enabling state is then found to match the state (someone possesses water) inferred from the expected ATRANS. The match completes the causal chain, causing the merging of the matched concepts. In this case, the merging process causes the program to infer that it was probably John who took (ATRANSed) the water from the faucet, in addition to turning it on. Had the sentence read "John turned on the faucet so Mary could drink.", the program would infer that Mary took the water from the faucet.


faucet (SOURCE with OUTPUT = *water*)
enable
?HUMO <=> ATRANS <- *water* TO ?HUMO
result
water (POSS-BY ?HUMO)
match?
yes...infer ?HUMO = *John*
water (POSS-BY *John*)
backward
inference
enable
John <=> INGEST <- ?LIQUID
inst
Jonn <=> PTRANS <- ?LIQUID

One should note here that the additional inferences used to complete the causal chain were very basic. The primary connections came directly from object-specific expectations derived from the Object Primitive descriptions of the objects involved.

4. CONCLUSIONS

It is important to understand how OPUS differs from previous inference strategies in natural language processing. To emphasize the original contributions of OPUS we will compare it to Rieger's early work on inference and causal chain construction. Since Rieger's research is closely related to OPUS, a comparison of this system to Rieger's program will illustrate which aspects of OPUS are novel, and which aspects have been inherited.

There is a great deal of similarity between the types of inferences used in OPUS and those used by Rieger in his description of MEMORY [8]. The causative and resultative inferences used to complete the causal chain in our last example came directly from that work. In addition, the demons used by OPUS are similar in flavor to the forward inferences and specification (slot-filling) inferences described by Rieger. Expectations are explicitly represented here as they were there, allowing them to be used in more than one way, as in the case where water is inferred to be the INGESTed liquid solely from its presence in a previous expectation.

There are, however, two ways in which OPUS departs from the inference strategies of MEMORY in significant ways. (1) On one the level of computer implementation there is a reorganization of process control in OPUS, and (2) on a theoretical level OPUS exploits an additional representational system which allows inference generation to be more strongly directed and controlled.

In terms of implementation, OPUS integrates the processes of conceptual analysis and memory-based inference processing. By using demons, inferences can be made during conceptual analysis, as the conceptual memory representations are generated. This eliminates much of the need for an inference discrimination procedure acting on completely pre-analyzed conceptualizations produced by a separate program module. In MEMORY, the processes of conceptual analysis and inference generation were sharply modularized for reasons which were more pragmatic than theoretical. Enough is known about the interactions of analysis and inference at this time for us to approach the two as concurrent processes which share control and contribute to each other in a very dynamic manner. Ideas from KRL [3] were instrumental in designing an integration of previously separate processing modules.

On a more theoretical level, the inference processes used for causal chain completion in OPUS are more highly constrained than was possible in Rieger's system. In MEMORY, all possible inferences were made for each new conceptualization which was input to the program. Initially, input consisted of concepts coming from the parser. MEMORY then attempted to make inferences from the conceptualizations which it itself had produced, repeating this cycle until no new inferences could be generated. Causal chains were connected when matches were found between inferred concepts and concepts already stored in its memory. However, the inference mechanisms used were in no way directed specifically to the task of making connections between concepts found in its input text. This lead to a combinatorial explosion in the number of inferences made from each new input.

In OPUS, forward expectations are based on specific associations from the objects mentioned, and only when the objects in the text are described in a manner that indicates they are being used functionally. In addition, no more than one or two levels of forward or backward inferences are made before the procedure is exhausted. The system stops once a match is made or it runs out of highly probable inferences to make. Thus, there is no chance for the kinds of combinatorial explosion Rieger experienced. By strengthening the representation, and exploiting an integrated processing strategy, the combinatorial explosion problem can be eliminated.

OPUS makes use of a well structured set of memory associations for objects, the Object Primitives, to encode information which can be used in a variety of Rieger's general inference classes. Because this information is directly associated with memory representations for the objects, rather than being embodied in disconnected inference rules elsewhere, appropriate inferences for the objects mentioned can be found directly. By using this extended representational system, we can begin to examine the kinds of associative memory required to produce what appeared from Rieger's model to be the "tremendous amount of 'hidden' computation" necessary for the processing of any natural language text.

REFERENCES

[1] Birnbaum, L., and Selfridge M. (1978). On Conceptual Analysis. (unpublished) Yale University, New Haven, CT.

[2] 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, No. 1.

[3] Bobrow, D. G., and Winograd, T. (1977). An overview of KRL, a knowledge representation language. *Cognitive Science 1*, no. 1

[4] Charniak, E. (1972). Toward a model of childrens story comprehension. AITR-266, Artificial Intelligence Laboratory, MIT, Cambridge, MA.

[5] Lehnert, W. G. (1978). Representing physical objects in memory. Technical Report #131. Dept. of Computer Science, Yale University, New Haven, CT.

[6] Minsky, M. (1975). A framework for representing knowledge. In Winston, P. H., ed., *The Psychology of Computer Vision*, McGraw-Hill, New York, NY.

[7] Norman, D. A., and Rumelhart, D. E., and the LNR Research Group (1975) *Explorations in Cognition*. W. H. Freeman and Co., San Fransisco.

[8] Rieger, C. (1975). Conceptual memory. In R. C. Schank, ed., *Conceptual Information Processing*. North Holland, Amsterdam.

[9] Riesbeck, C. and Schank, R. C. (1976). Comprehension by computer: expectation-based analysis of sentences in context. Technical Report #78. Dept. of Computer Science, Yale University, New Haven, CT.

[10] Schank, R. C., (1975). Conceptual Dependency Theory. in Schank, R. C.(ed.), *Conceptual Information Processing*. North Holland, Amsterdam.

[11] Schank, R. C. and Abelson, R. P. (1977). *Scripts, Plans, Goals, and Understanding*. Lawrence Erlbaum Press, Hillsdale, NJ.