Computers Math. Applic. Vol. 23, No. 2–5, pp. 51–73, 1992
Printed in Great Britain. All rights Reserved

0097-4943/92 $5.00 + 0.00
Copyright©1992 Pergamon Press plc

CONCEPTUAL DEPENDENCY AND ITS DESCENDANTS

STEVEN L. LYTINEN
Artificial Intelligence Laboratory
The University of Michigan
Ann Arbor, MI 48109, U.S.A.

Abstract — This paper surveys representation and processing theories arising out of conceptual dependency theory. One of the primary characteristics of conceptual dependency was the notion of a canonical form, built out of a small number of primitive representations. Although the notion of primitives has largely been lost in subsequent work, many other of the basic notions of CD have remained. In particular, the idea of building representations around inferential capabilities has prevailed in this family of research. The result is a set of representational structures, all of which are highly knowledge-intensive. The use of these structures in various processing theories has led to knowledge-based theories of language understanding, planning, reasoning and other tasks, which have contrasted sharply with the traditional search-oriented approaches used in other systems.

1. INTRODUCTION

This paper will outline a family of representational theories which have developed from the common ancestor of conceptual dependency (CD) theory created by Roger Schank [1,2]. In addition, we will discuss some of the programs that these representations have been used in, as well as the reasons why these representations have proven to be useful for these programs.

The original goal of conceptual dependency work was to develop a "representation of the conceptual base that underlies all natural languages." [1, p. 554] This rather lofty goal resulted in the proposal of a small set of primitive actions (10–12 or so), and a set of dependencies which connected the primitive actions with each other and with their actors, objects, instruments, etc. The claim was that this small set of representational elements could be used to produce a *canonical form* representation for English sentences (and other natural languages).

Almost twenty years later, no one seems to take seriously the notion that such a small set of representational elements covers the conceptual base that underlies all natural languages. Representation theories that have descended from CD, such as scripts [3], have had a virtually unlimited number of vocabulary items. The notion of a "primitive" seems to have disappeared. However, as we shall see, CD has certainly influenced these subsequent theories. One of the questions that this paper will address, then, is: What did we learn from conceptual dependency? What does conceptual dependency theory have in common with the descendant theories? As we examine these descendants and the tasks that they have been used for, we will try to discover the common threads that connect this family of research.

We will begin the paper by briefly reviewing conceptual dependency theory. For a more thorough review, see [2]. Then we will discuss subsequent theories, including scripts, plan/goal representations, Memory Organization Packets (MOPs), and Thematic Organization Packets (TOPs). Finally, we will examine some of the processing theories that have accompanied these representations. In particular, we will focus on the tasks of language understanding and reasoning. As we will see, the basic assumptions behind the representation theories have a large effect on the processing theories that use them.

2. CONCEPTUAL DEPENDENCY

2.1. A Brief Review of CD Notation

Conceptual dependency theory was based on two assumptions:

1. If two sentences have the same meaning, they should be represented the same, regardless of the particular words used.
2. Information implicitly stated in the sentence should be represented explicitly. That is, any information which can be inferred from what is explicitly stated should be included in the representation.

These assumptions have many implications for what a representation language should look like. The first assumption implies that representations must be *general*; that is, they must capture the similarities in meanings of synonyms. For example, since "get" and "receive" can be used synonymously in many contexts, their conceptual representations in these contexts ought to reflect this fact by consisting of similar, if not identical, predicates.

The assumption that representations ought to explicitly represent implicit information means that inferences must be made in order to produce complete representations. If a sentence implies information without explicitly stating it, then in order to include that information in the representation, machinery must exist which can infer it from what is explicitly stated. This means that representations must support inference. In order to do this efficiently, they must be *canonical*. Whenever a particular inference can be made, it would be desirable if the same inference rule could always be used to make it. Without a canonical representation, several rules would be required, one for each different possible representation form.

Given these representational requirements, then, the vocabulary for conceptual dependency consisted of the following:

- a set of *primitives*, used to represent actions in the world
- a set of *states*, used to represent preconditions and results of actions
- a set of *dependencies*, or possible conceptual relationships which could exist between primitives, states, and the objects involved.

Representations of English sentences could be constructed by piecing together these building blocks to form a *conceptual dependency graph*.

PTRANS: The transfer of location of an object
ATRANS: The transfer of ownership, possession, or control of an object
MTRANS: The transfer of mental information between agents
MBUILD: The construction of a thought or of new information by an agent
ATTEND: The act of focusing attention of a sense organ toward an object
GRASP: The grasping of an object by an actor so that it may be manipulated
PROPEL: The application of a physical force to an object
MOVE: The movement of a bodypart of an agent by that agent
INGEST: The taking in of an object (food, air, water, etc.) by an animal
EXPEL: The expulsion of an object by an animal
SPEAK: The act of producing sound, including non-communicative sounds

Figure 1. The conceptual dependency primitives.

The set of primitives varied somewhat during the course of conceptual dependency theory, but it consisted of approximately 10–12 predicates, each of which represented a type of action. The primitives are shown in Figure 1.

Each primitive had a set of *slots* associated with it, from the set of conceptual dependencies. Associated with each slot were restrictions as to what sorts of objects could appear in that slot. For example, the slots for PTRANS were the following:

ACTOR: a HUMAN (or animate object), that initiates the PTRANS
OBJECT: a PHYSICAL OBJECT, that is PTRANSed (moved)
FROM: a LOCATION, at which the PTRANS begins
TO: a LOCATION, at which the PTRANS ends.

In order to make explicit the information implicitly presented in the text, inference rules were written based on the primitives. For example, from the primitive PTRANS, the inference could be made that the OBJECT which was PTRANSed was initially in the FROM location, and after the PTRANS was in the TO location.

The need to make inferences in order to explicitly represent implicit information suggested a criterion for what made a good primitive: it should support a cluster of reliable inferences. Thus, PTRANS constituted a good primitive because all PTRANS's shared several common inferences: from the representation one could infer the prior location of an object being PTRANSed, the location subsequent to the PTRANS, that the source and destination of the PTRANS must be locations, etc. These same inferences could be made no matter what type of PTRANS: flying, driving, walking, falling, and so on.

Conceptual dependency representations were written graphically as shown in Figure 2. The actor of a primitive action was connected to the primitive using a double arrow; the object appeared to the right with a single arrow connection, and the source and destination (TO and FROM) appeared to the right of the object. Thus, the sentence, "John gave Mary a book" would be represented as shown in Figure 3. John was explicitly represented as both the ACTOR and FROM slots of this action, since it was assumed that John had control/possession of the book originally.

Although individual primitives grouped together a set of similar actions, it was important that distinctions between these actions could also be captured in CD notation. For example, reading and talking are both types of MTRANS's, but there are obvious differences between them: reading involves using one's eyes while talking involves using one's mouth and ears; the source of information in reading is an inanimate object (such as a newspaper) while the source in talking is a human, etc. In order to express differences between similar actions, several primitive actions could be connected to each other, using one or more of the conceptual dependencies. For example, the CD graph in Figure 4 represents the action of reading. The "i" link in this graph stands for an *instrumental* connection between the MTRANS and the MOVE.

Talking, on the other hand, might be represented as shown in Figure 5. Again, an instrumental connection expresses the relationship between the MTRANS and John's MOVEment of his mouth and Mary's ATTENDing with her ears.

Sometimes connections also involved explicit mention of one or more intermediate states or actions. Consider this example, from [3]:

John cried because Mary said she loved Bill.

The causal connection between John crying and Mary loving Bill is not direct: most readers infer that John loves Mary, realizes that Mary does not love him back since she loves Bill, and therefore is sad. This might be represented as shown in Figure 6. In this graph, "I" stands for an *initiate* link, connecting together an event and a mental state; and "I/R" stands for *initiate/reason*, a complex connection between two mental events.

Representations could be arbitrarily complex, sometimes making explicit a whole series of inferences that could be made. These complex sets of primitive actions, states, and conceptual dependencies were called *causal chains*. For example, consider the following sentence:

John went to Sears and found a TV for $100.

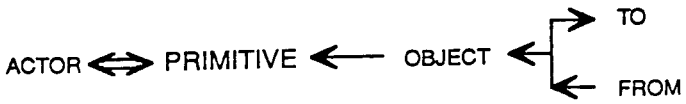


Figure 2. Basic form of a conceptual dependency graph.

Figure 3. Representation of "John gave Mary a book."

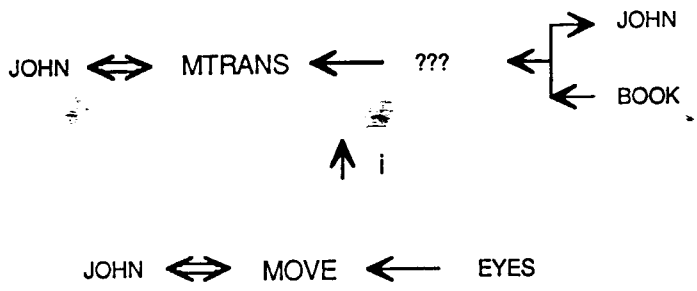


Figure 4. Representation of the act of reading.

Most readers infer that John bought the TV and paid $100 for it. This implies a rather complex chain of events, consisting of John going into the store, looking around, seeing a TV on sale for $100, taking it to the cashier, and paying $100 to the cashier. This causal chain might be represented as shown in Figure 7.

2.2. Conceptual Dependency and Semantic Nets

A distinction has been made in the literature between *content* and *structure* theories. In some ways, it is ironic that conceptual dependency and its derivatives are often grouped into the family of semantic net representations, because in general, semantic nets are a structure theory, whereas CD is a content theory.

The distinction between these two types of theories lies in their emphasis. Semantic net theory is about how knowledge should be organized: there will be nodes, with arcs connecting the nodes together. There is also some general (although, in the case of semantic nets, rarely formalized) notion of the structure's semantics; i.e., how to interpret a particular semantic net structure. Finally, there is usually the general notion of inheritence, etc. This is all structural information. That is, it says nothing about *what* will be represented, it simply says something about the (structural) form that the representation will take. Putting this another way, semantic nets tell us to use nodes and arcs, but they don't tell us what labels to put on the nodes, or what arcs to use and where. It is up to the user to decide these details. Without the details, however, semantic nets do not represent anything.

Conceptual dependency theory, on the other hand, was an attempt to enumerate the types of nodes and arcs which could be used to build representations. Rather than specifying the structure of representations, CD theory specified the *content*. True, the conventions used in drawing conceptual dependency graphs also specified structure, but this was not really the essence of the theory. The essence was the primitives, and the *names* of the dependencies which could be used to hook primitives together.

This distinction can be made clearer if one imagines trying to implement conceptual dependencies and semantic nets in first order predicate calculus. In the case of conceptual dependencies, it is not hard to imagine: the primitive acts, dependencies, and states would specify a set of predicates to use when writing predicate calculus statements to represent sentences.[1] We might have

[1] The sort of inferences proposed by those who have used CD are not typically of an exclusively deductive nature, as would be true in the predicate calculus, but this really has more to do with the *use* of the representations, rather than the representations themselves.

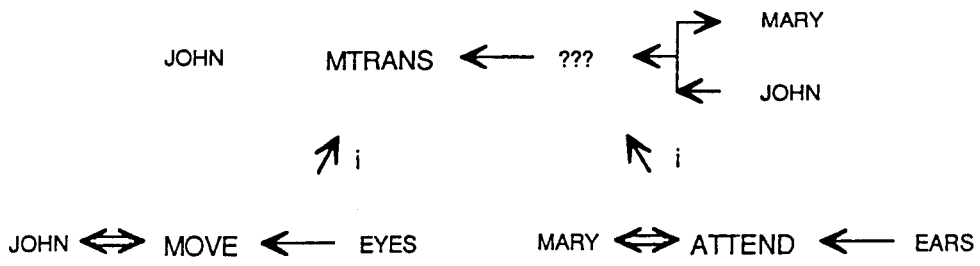


Figure 5. Representation of the act of talking.

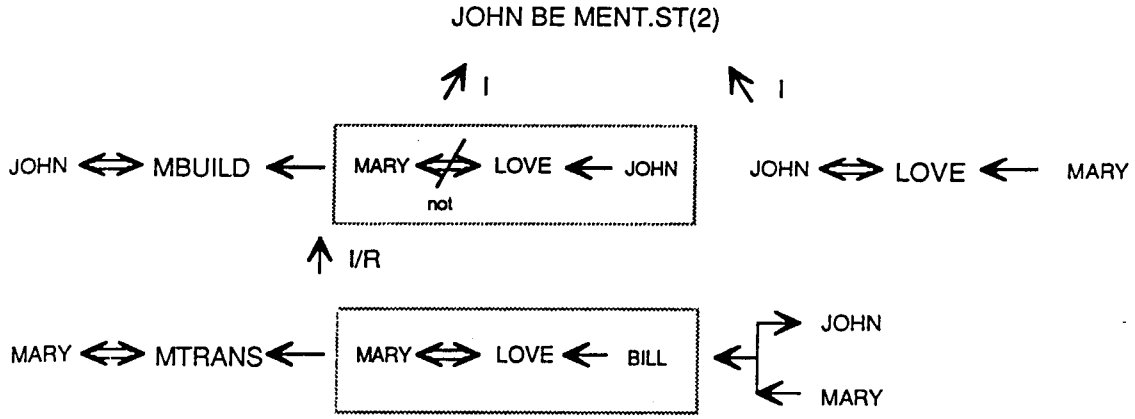


Figure 6. Representation of "John cried because Mary said she loved Bill."

to adopt some translation conventions in order to make all assertions first-order, but these would be quite straightforward. On the other hand, it does not make as much sense to "implement" semantic nets in predicate calculus. The two representations are in competition with each other: each provides a different syntax for distinguishing between predicates, arguments, and relations. There would be nothing left of semantic nets if we translated them to predicate calculus. Putting this another way, semantic nets would not add anything to first-order predicate calculus. This is in contrast to conceptual dependency, which adds the CD primitives as recommended predicates to be used.

2.3. Conceptual Dependency and Inferences

As we stated earlier, one motivation for representing the meaning of a text in a canonical form is to facilitate inferencing. Canonical form allows us to write inference rules as generally as possible: if representations did not always capture similarities in meaning, then rules about what can be inferred from a text would need to be duplicated, one rule required for every form of representation which had the same meaning relevant to the inference.

To illustrate that inferencing was facilitated by conceptual dependency, Rieger wrote the MEMORY program, which made inferences from text [4]. Inferences were used to build a "causal chain" to connect events in a story. For example:

> John hit Mary. Mary's mother took Mary to the hospital. Mary's mother called John's mother. John's mother spanked John.

MEMORY made several inferences from this story, including that John's mother spanked John because she was angry at him for hitting Mary, and that Mary went to the hospital because she was hurt.

Inferences were made in MEMORY by a set of rules, which were organized around inference categories. There were a total of 16 of these categories, some of which were:

1. Specification inferences, which filled in missing "slots" in a CD primitive, such as the ACTOR or INSTRUMENT of an action.

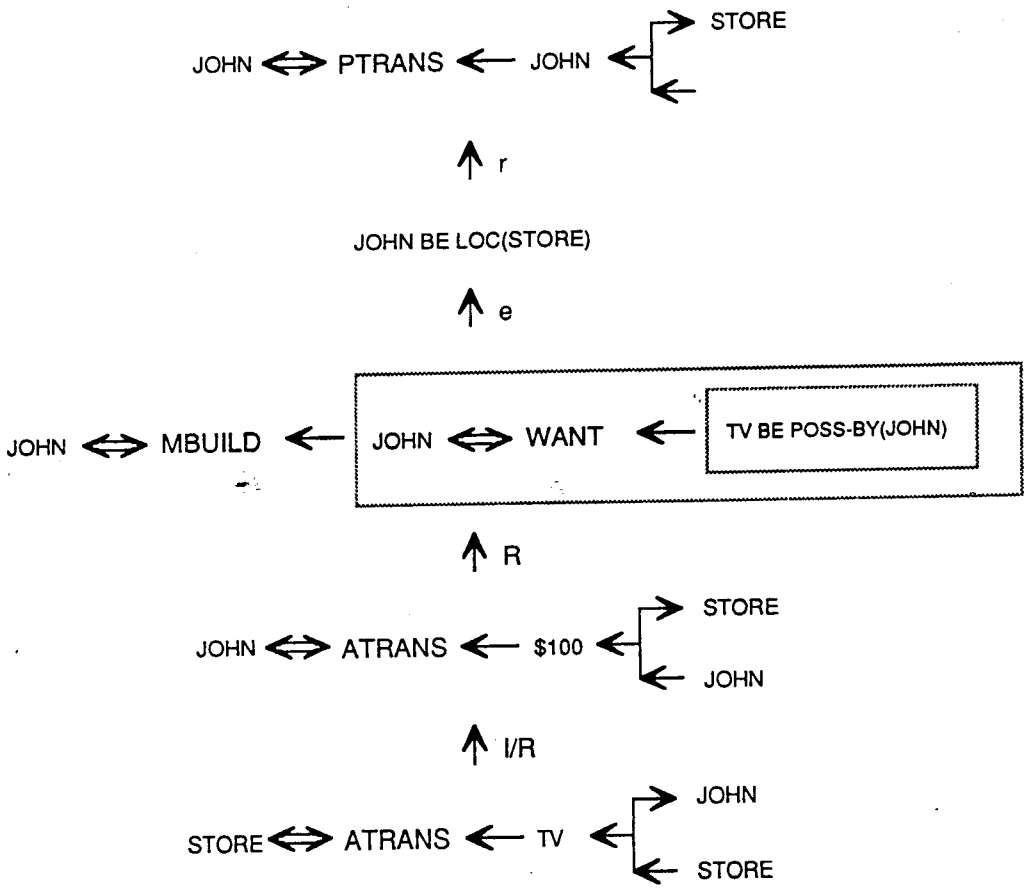


Figure 7. Representation of "John went to Sears and found a TV for $100."

2. Causative inferences, which hypothesized possible causes or preconditions of actions.
3. Resultative inferences, which inferred likely results of actions.
4. Function inferences, which inferred likely functions of objects.

Inference rules were applied in an *undirected* fashion. That is, when MEMORY read a sentence, its inference rules were automatically applied, without any particular goal in mind, such as building a causal chain to connect events together. This bridging often happened, but when it did it was fortuitous in some sense: inferences were applied to a new representation in an undirected fashion, sometimes resulting in the *confirmation* of another representation.

The undirectedness of MEMORY's inferences was meant to reflect the spontaneous nature of inferences that people make. These inferences seem uncontrollable for people: one cannot learn a new fact without inferring things about it. However, this undirected behavior led to problems: when processing a story, MEMORY did not know which inferences were most likely to lead to building a coherent causal chain to represent the story. This led to a combinatorial explosion in the number of inferences that the system had to consider in order to build causal chains. In some sense, Rieger's system was lacking common sense knowledge about what inferences were most likely to be relevant in a situation. For example:

John picked up the menu. He decided on the fish.

There are many conceivable inferences that can be made from someone picking up an object. People pick objects up for many reasons, as is illustrated by the following examples:

Use-as-instrument: John picked up the menu. He swatted a fly.

Subgoal-to-move: John picked up the menu. He found his fork.

Subgoal-to-read: John picked up the menu. He read it.

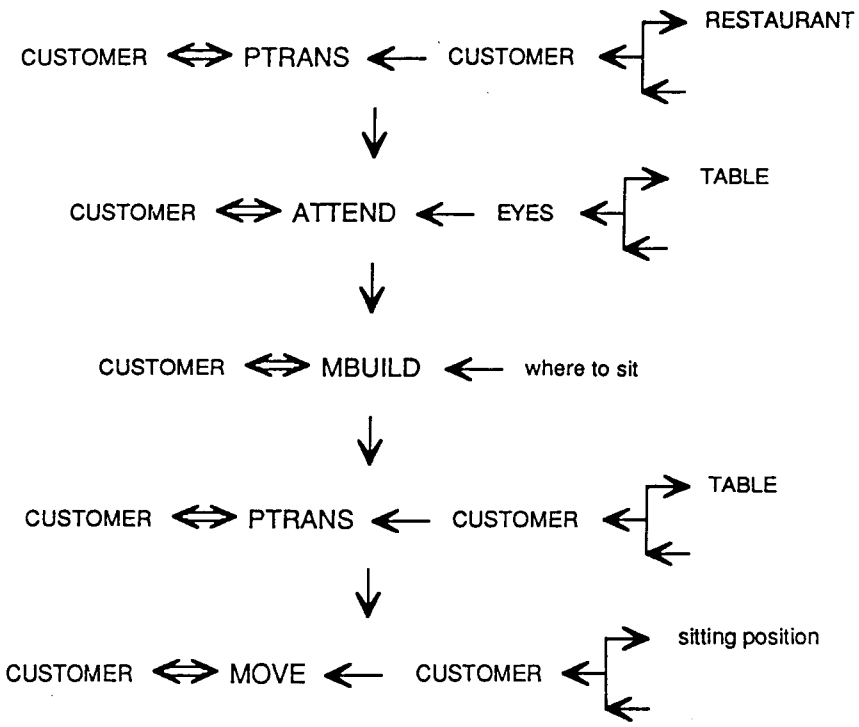


Figure 8. ENTER scene of the $RESTAURANT script.

These are certainly not the only possible reasons why John might pick up a menu. Yet, it is obvious what the most likely reason is why John picked up the menu. Usually people pick up a menu in order to read it, so that they can order food and eat. MEMORY, however, lacked this knowledge: there was no sense in which one inference was more common than others.

3. SCRIPTS AND MOPS

In order to solve the problems with undirected inferencing, larger knowledge structures, called *scripts*, were proposed [3]. Similar to frames [5] and schemas [6,7], scripts "precompiled" sets of likely inferences, packaged together so that they could be searched more efficiently than other, irrelevant inferences. Scripts represented stereotypical sequences of events, such as going to a restaurant. By using the script, the theory was that the understander had quick access to those events which always happen in a stereotypical event sequence, without having to think about other inferences which would most likely be irrelevant.

A script consisted of a set of *roles*, or participants involved in the script as well as common objects used; and *scenes*, each of which described the typical events in one portion of the script. For example, in the $RESTAURANT[2] script, some of the roles were the customer, the waiter/waitress, the restaurant, and the food. Scenes included ENTER, ORDER, EAT, PAY, and LEAVE. The details of each scene were represented as a sequence of conceptual dependency representations. For example, the ENTER scene in $RESTAURANT consisted of the causal chain in Figure 8.

In order to account for the variability of stereotypical events, scripts often contained different *tracks*. Each track reflected one possible alternative sequence that could be followed in the script. For example, the scenes of $RESTAURANT do not always take place in the same order: in a sit-down restaurant, usually one sits at a table, orders, then eats, then pays. However, in fast food restaurants, paying precedes sitting, which is then followed by eating. Other variations take place in other specific restaurants or types of restaurants. The complete $RESTAURANT, then, is shown in Figure 9.

[2]Schank and Abelson preceded script names with a $.

Script: RESTAURANT
Track: Coffee Shop
Props: Tables
Menu
F-Food
Check
Money

Roles: S-Customer
W-Waiter
C-Cook
M-Cashier
O-Owner

Entry conditions: S is hungry.
S has money.

Results: S has less money
O has more money
S is not hungry
S is pleased (optional)

Scene 1: Entering

S **PTRANS** S into restaurant
S **ATTEND** eyes **to** tables
S **MBUILD** where to sit
S **PTRANS** S **to** table
S **MOVE** S **to** sitting position

Scene 2: Ordering

(menu on table)
S **PTRANS** menu **to** S

(W brings menu)
W **PTRANS** W **to** table
W **ATRANS** menu **to** S

(S asks for menu)
S **MTRANS** signal **to** W
W **PTRANS** W **to** table
S **MTRANS** 'need menu' **to** W
W **PTRANS** W **to** menu

S **MTRANS** food list **to** **CP**(S)
S **MBUILD** choice of F
S **MTRANS** signal **to** W
W **PTRANS** W **to** table
S **MTRANS** 'I want F' **to** W

W **PTRANS** W **to** C
W **MTRANS** (**ATRANS** F) **to** C

C **MTRANS** 'no F' **to** W
W **PTRANS** W **to** S
W **MTRANS** 'no F' **to** S
(go back to *) or
(go to Scene 4 at no pay path)

C **DO** (prepare F script)
to Scene 3

Figure 9. The restaurant script [3, p. 43].

Again, one of the points of proposing scripts was to facilitate common inferences that took place in the events that they represented. Consider our earlier example:

John picked up the menu. He decided on the fish.

If $RESTAURANT is inferred from the first sentence, then it is not difficult to understand how the second sentence connects to it. Making the connecting inferences is simply a matter of matching the second event to some other event in $RESTAURANT, and picking out the precompiled causal chain from the script which connects the two events.

In addition to being a powerful processing structure, scripts were also proposed as a memory structure. Since a script contains all of the standard inferences which are made about a stereotypical event sequence, the theory was that there should be no need to duplicate these inferences in the record of a particular episode. This predicted that various confusions should occur: confusions about which particular scenes in a scriptal episode were mentioned in a story should occur, and memories across episodes using the same script might also get confused.

Bower, Black and Turner [8] conducted a study in which they found that recall confusions did in fact occur. Subjects were shown stories about trips to the doctor's and dentist's office. Scenes, such as waiting in the waiting room, being examined, and checking out, were either included in or omitted from the stories. During subsequent testing, subjects had difficulties recalling which scenes were explicitly mentioned. They also could not recall which particular scenes went together in the same episode. This seemed to provide evidence for scripts or some sort of schema-based memory of the stories.

3.1. Scripts and Generality

Charniak [9] observed that, for efficiency reasons, causal knowledge in scripts or frames should not be duplicated when that knowledge comes from more general causal laws. Thus, in his frame-style representational system, which represented mundane knowledge about painting, he distinguished between two types of frames: *simple events*, which corresponded to common sense causal laws; and *complex events*, which referred to the simple events for their causal explanations. The PAINTING frame was a complex event, which consisted of sequences of actions such as "get paint on the painting instrument," and "bring instrument in contact with object," along with pointers to simple events, like STICK (i.e., "sticks to"), which provided the causal rules explaining why events within the PAINTING frame proceeded in that order. STICK consisted of a causal rule, explaining why bringing the painting instrument in contact with the object to be painted would cause paint to stick on the object. These simple events, like STICK, could be shared between many complex events, like PAINTING, so that knowledge common to more than one situation would not have to be duplicated in the frames used to represent those situations.

For different reasons, a similar sharing of knowledge was proposed in [10], as a modification of script theory. Schank discussed the use of scripts for learning. This new task raised some problems with scripts. In script theory, although similar scenes appeared in different scripts, the representation of these scenes did not capture these similarities. For instance, the scripts $RESTAURANT and $DEPARTMENT-STORE both contained a scene having to do with ordering. In the case of $RESTAURANT, this scene was ORDER-FOOD, and in $DEPARTMENT-STORE, the scene was ORDER-MERCHANDISE. However, these two scenes were totally separate entities, with no representation of the fact that they were similar in very important ways. There was no more general concept ORDER to which they could refer which was an abstraction of the common entities of both scenes. This presented a problem for learning. If a program were to learn scripts like $RESTAURANT and $DEPARTMENT-STORE, knowledge common to the ordering in a restaurant and the ordering in a department store would have to be stored in two different places, since the corresponding scenes in $RESTAURANT and $DEPARTMENT-STORE did not share information. This meant that knowledge learned in one domain could not be accessed in the other domain. So, for example, if one learned that one should be polite in order to get good service in a restaurant, that information could not be accessed by $DEPARTMENT-STORE in order to realize that one should be polite to the catalog clerks in order to get good service at a department store, also.

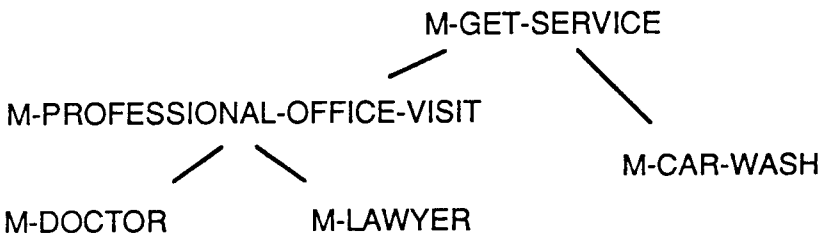
Script representation

$DOCTOR:
HAVE-MEDICAL-PROBLEM + MAKE-APPT + GO +
DOCTOR-WAITING-ROOM + TREATMENT + PAY

$LAWYER:
HAVE-LEGAL-PROBLEM + MAKE-APPT + GO +
LAWYER-WAITING-ROOM + LEGAL-CONSULTATION + PAY

$CAR-WASH:
HAVE-DIRTY-CAR + GO + WAIT-IN-LINE +
GET-CAR-WASHED + PAY

MOP representation



M-GET-SERVICE:
(NEED-SERVICE) + GO + (WAIT) + (GET-SERVICE) + PAY

M-PROFESSIONAL-OFFICE-VISIT:
HAVE-PROBELM + MAKE-APPT + (GO) + WAITING-ROOM +
(GET-SERVICE) + (PAY)

M-CAR-WASH:
HAVE-DIRTY-CAR + (GO) + WAIT-IN-LINE + GET-CAR-WASHED + (PAY)

M-DOCTOR:
HAVE-MEDICAL-PROBLEM + (MAKE-APPT) + (GO) +
DOCTOR-WAITING-ROOM + TREATMENT + (PAY)

M-LAWYER:
HAVE-LEGAL-PROBLEM + (MAKE-APPT) + (GO) +
LAWYER-WAITING-ROOM + LEGAL-CONSULTATION + (PAY)

Figure 10. Script vs. MOP representation of various events.

There was another problem with scripts, etc., which became evident from the Bower *et al.* experiment. In addition to the within-script memory confusions discussed earlier, recognition confusions were also found to occur between stories about visits to the dentist and visits to the doctor. Intuitively, this result was not surprising, since most people have experienced such confusions. But how could they be explained by scripts? Should we posit a "visit to a health care professional" script to explain it? Clearly, this would be beyond the initial conception of what a script was.

To accommodate solutions to these problems, a modification of script theory was proposed in [10] that introduced a new processing structure, called a MOP (Memory Organization Packet). The general idea behind MOPs was to store knowledge which is common to many different situations in only one processing structure, and then to make this processing structure available in all the different situations in which it applies.

To see how MOPs differ from scripts, let us compare the script and MOP representations of several events as presented in Figure 10. In script theory, all the scenes of the doctor, lawyer, or car wash episode were provided by one structure, $DOCTOR, $LAWYER, or $CAR-WASH. However, although similar scripts had many similar scenes, such as HAVE-MEDICAL-PROBLEM and HAVE-LEGAL-PROBLEM in $DOCTOR and $LAWYER, there was no connection between such scenes. In MOP theory, however, all the common elements shared among specific scenes of different contexts are abstracted together into a more general scene. Thus, the common features of doctor visits and lawyer visits are abstracted into a more general structure, M-PROFESSIONAL-OFFICE-VISIT (or M-POV for short). M-POV has scenes, WAITING-ROOM, GET-SERVICE, etc., which are abstractions of the scenes DOCTOR-WAITING-ROOM, LAWYER-WAITING-ROOM, and GET-TREATMENT, LEGAL-CONSULTATION. Then, features unique to doctors' offices are provided by the more specific scene DOCTOR-WAITING-ROOM, but features shared by other professional office visits exist in the generalized scene WAITING-ROOM. Similarly, the sequential information shared by these scripts is abstracted together also into M-POV. Thus, a great deal of information that was in $DOCTOR in script theory is not in M-DOCTOR in MOP theory. Rather, much of this information comes from more general MOPs.

Similarly, information which is shared between professional office visits and other types of getting service, such as getting your car washed, is abstracted even higher, into M-GET-SERVICE. All get-service actions have things in common, such as first having a problem, waiting for the service, and paying. This common information is abstracted to as general a structure as possible, into even more general scenes such as NEED-SERVICE and WAIT.

MOPs and scenes, then, are arranged hierarchically. M-POV is a generalization of the common elements in M-DOCTOR and M-LAWYER, M-GET-SERVICE is a generalization of M-POV and other types of service MOPs, WAITING ROOM is a generalization of DOCTOR-WAITING-ROOM and LAWYER-WAITING-ROOM, WAIT is a generalization of WAITING-ROOM and WAIT-IN-LINE, etc. Knowledge is stored at as general a level as is possible.

4. PLANS, GOALS, AND TOPS

Another line of work which proceeded from conceptual dependency was the notion of high-level, goal-based representations. Since CD primitives were mainly about physical actions, they did not seem to capture well the concepts useful to express knowledge about goal-based behavior. For example:

Willa was hungry. She picked up the Michelin guide.

The causal chain which connects these events is shown in Figure 11. Given the complexity of this causal chain, again it seems necessary for a program to have some heuristics about how to search the space of possible inferences in order to construct it. However, this time, it does not seem likely that a script would be available to guide the process. Somehow, $LOOKUP-RESTAURANT-IN-GUIDE seems to be too uncommon an event to warrant a script.

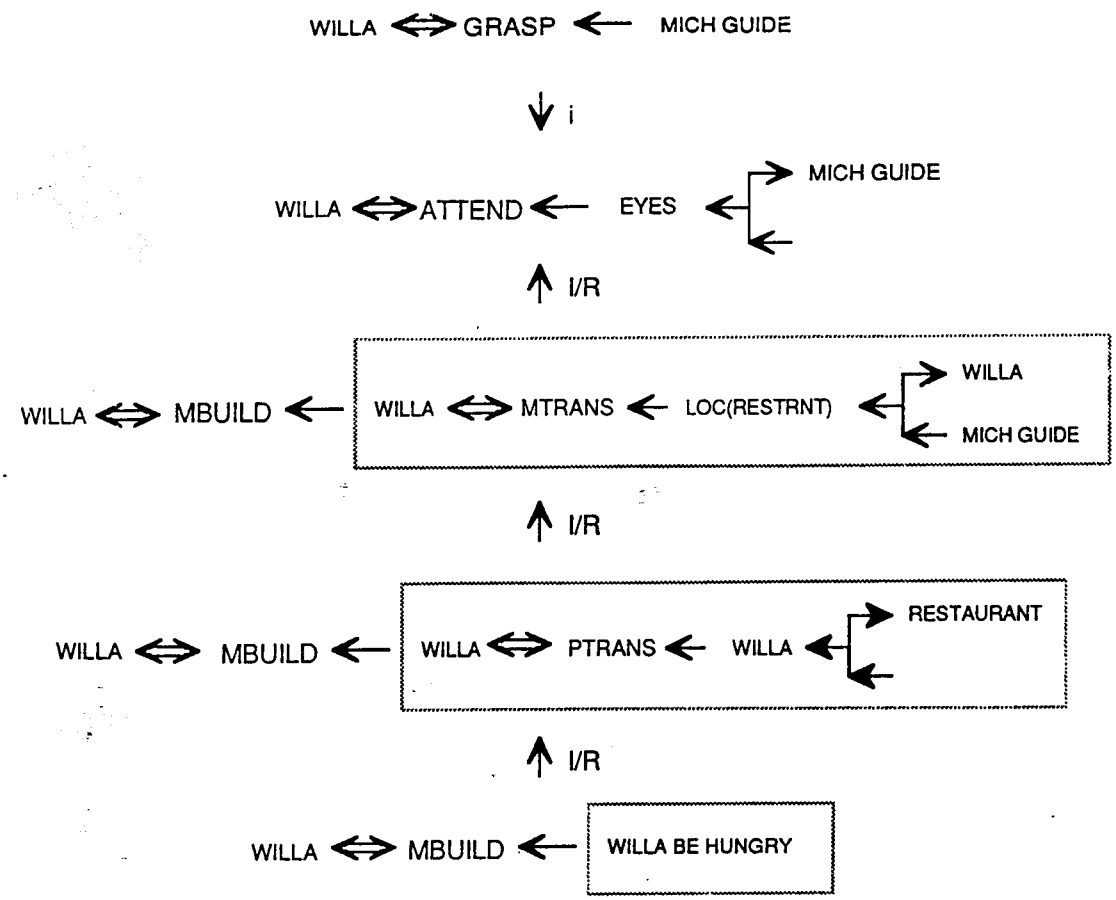


Figure 11. Representation of "Willa was hungry. She picked up the Michelin guide."

To account for the ability to infer complex causal chains which represented plans constructed by an agent in order to solve a problem, Schank and Abelson proposed representations for *goals* and *plans*. Several categories of goals were proposed, including:

S-goals: *Satisfaction* goals which recur regularly, such as S-HUNGER, S-SEX, and S-SLEEP

E-goals: *Enjoyment* goals, such as E-TRAVEL, E-EXERCISE

A-goals: *Achievement* goals, such as A-SKILL or A-POSSESSIONS

I-goals: *Instrumental* goals, or subgoals which arise during pursuit of other goals

D-goals: *Delta* goals, similar to I-goals, but with standard plans associated with them.

Just as with CD primitives, goals were categorized because of inferences that could be made based on the category. For example, D-goals, which included D-KNOW (the goal to know a particular fact), D-CONT (the goal of getting control of an object), and D-PROX (the goal of moving to a location), were associated with a common *planbox*, or set of plans that could be used to achieve them. The D-goal planbox contained plans such as ASK, THREATEN, and INVOKE-THEME (persuading someone to help because of a thematic relationship, such as parent-child). Each plan could be used to satisfy any D-goal in the right circumstances.

Schank and Abelson also proposed *named plans*, which were schematic structures, but at a much more general level than scripts. Named plans consisted of sequences of planning events which led to the satisfaction of a goal. An example is the named plan **USE**, which consisted of the following steps:

$$\text{USE(x)} = \text{D-KNOW} + \text{D-PROX} + \text{D-CONT} + \text{I-PREP} + \text{DO}$$

For example, the goal of using a book would be realized by first knowing where it is, moving to it, achieving control of it (by GRASPing it, for example), preparing to use it in some way (for example, one might have to put on reading glasses), and finally reading the book.

Just as with scripts, this knowledge of common sequential actions could be used to facilitate story understanding. In our earlier example:

> Willa was hungry. She picked up the Michelin guide.

The connection between the events is made as follows: Willa has the goal S-HUNGER. She selects the $RESTAURANT script to satisfy this goal, but before she can execute USE(RESTAURANT), she must KNOW(LOC(RESTAURANT)) (from the named plan). This generates the steps of the plan as subgoals, including the subgoal D-KNOW, which in this case will be achieved by an MTRANS from the Michelin guide. Again, to USE(Michelin-guide), the named plan suggests several steps, one of which is D-CONT, which Willa achieves through a GRASP ("picked up").

4.1. More Complex Reasoning about Goals

Schank and Abelson's work provided the vocabulary needed to do elementary reasoning about single-goal situations. Wilensky extended this work to more complex goal situations, involving goal interactions and competitions [11]. Wilensky proposed rules for plan recognition based on goal configurations, as well as key characteristics such as resource limitations, recurrence of goals, etc. For example:

> John wanted to watch the football game. Mary wanted to watch the Bolshoi ballet. John got the old black-and-white TV out of the attic.

Wilensky proposed the following rule for understanding this story:

> If a set of characters have goals that compete due to a shortage of a non-consumable functional object that the characters need to use simultaneously, and one character tries to acquire additional functional objects of the same kind, then that action is part of a plan to ease the goal competition. [11, p. 204]

In both Schank and Abelson's work and Wilensky's extensions, goal-level representations were quite separate from the content level, such as CD primitives or scripts. However, it seems that often it is the case that content can have an effect on the goal/plan level. Particular facts about the context in which a goal arises can play a key role in determining the particular plan to select in order to achieve the goal.

Evidence for rather specific structures which combine goals/plans and contextual information can also be seen in remindings which people have about these situations. Schank [10] presents a large number of these remindings. For example:

> X was talking about how there was no marijuana around for a month or two. Then, all of a sudden, everyone was able to get as much as they wanted. But the price had gone up 25 percent. This reminded X of the oil situation that cleared up as soon as the price had risen a significant amount. [10, p. 120]

The two situations in the example have many things in common. First, they are about D-CONT goals (possession of marijuana and gasoline). The D-CONT goals are recurrent, since both desired objects are used up regularly and must be bought over and over. They also involve a situation in which the selected plan is BARGAIN-OBJECT (i.e., exchange money for the desired object), but the object is in short supply. Also, perhaps a common feature is that the desired object is controlled by unethical people (drug pushers, and the OPEC cartel). If we view this as a planning structure, the structure provides a rule something like, "If you want to D-CONT an object using money, the goal recurs regularly, the object is in short supply, and you're dealing

with unethical people, then try waiting a little while, and you'll be able to buy it then, but at a higher price."

Note the difference between this thematic structure (or TOP, as Schank called them) and the goal/planning structures we discussed earlier. The TOP has goal features, such as D-CONT and BARGAIN-OBJECT. There are also very specific contextual features, such as the object being a usable commodity and in short supply; and a very specific suggested plan, to wait a while and buy at a higher price.

Schank gives many examples of TOPs, based on evidence of cross-contextual reminding such as the example above. All of them suggest a combination of goal/plan information and arbitrary sets of conditions, specifying the context in which the structure applies. Each TOP can then provide a specific prediction or set of predictions about what is likely to occur in the situation, or what the planner can do to achieve his/her goal.

Seifert and her colleagues found evidence for thematic remindings in human subjects, providing psychological evidence for thematic structures such as TOPs. In [12], subjects were presented with sets of stories which included TOP-like similarities. Adages were selected as thematic patterns that would be familiar to subjects and culturally shared, minimizing variability in the structures used for encoding by subjects. For example, the adage "closing the barn door after the horse is gone" can be characterized as a planning failure where X knows a plan to prevent goal failure, but delays execution to avoid the cost until the goal is failing; then, the plan is executed, but fails because it is not a recovery plan but a prevention plan [13]. Here are two stories used in the experiments that represent this thematic pattern:

> Story 1: Academia
>
> Dr. Popoff knew that his graduate student Mike was unhappy with the research facilities available in his department. Mike had requested new equipment on several occasions, but Dr. Popoff always denied Mike's requests. One day, Dr. Popoff found out that Mike had been accepted to study at a rival university. Not wanting to lose a good student, Dr. Popoff hurriedly offered Mike lots of new research equipment. But by then, Mike had already decided to transfer.
>
> Story 2: Wedding Bells
>
> Phil was in love with his secretary and was well aware that she wanted to marry him. However, Phil was afraid of responsibility, so he kept dating others and made up excuses to postpone the wedding. Finally, his secretary got fed up, began dating, and fell in love with an accountant. When Phil found out, he went to her and proposed marriage, showing her the ring he had bought. But by that time, his secretary was already planning her honeymoon with the accountant.

Two stories were presented in sequence, that either shared the same theme or did not, followed by a test list of items to respond to. Since both stories should be encoded with the same theme (Thematic Abstract Unit, from [13]), understanding the same-theme stories was predicted to result in a connected memory representation. If such a characteristic reminding structure is set up during the understanding process, will reference to the academia story result in activation of the wedding-bells story? To measure activation, subjects responded to a series of test sentences about the two stories, including negatives ("Mike decided to buy his own research equipment") and the two items of interest:

> by then, Mike had already decided to transfer
>
> his secretary fell in love with an accountant

If the two stories are connected in memory, responding to an item from one should speed the time to respond to an item from the other. This same-theme response time was compared to one where the preceding item refers to a story that did not share the same theme, and therefore should have no connection in memory between the two stories. For example, if one story was based on the "pot calling the kettle black" theme, and the other on the barn door theme, no priming of the response would be expected.

The results showed that the expected same theme result did not significantly affect response time to the test items. The shared knowledge structure did not affect the ease of access of the episodes in memory, so the strong hypothesis that connection alone would result in automatic activation of related episodes was not supported. However, if the process is not automatic, it may still occur under strategic conditions, when subjects attempt to utilize effort or take cues from the task specifications.

In a second experiment, subjects were instructed to think about the theme as they read, and to rate the similarity of the story pair after the test list. Otherwise, this experiment was identical to the first. However, the results were different: this time, there was a significant effect of thematic similarity: mean response time for stories sharing the same theme was significantly faster than for different themes. This indicates that responses were faster for test items when the story pair shared the same thematic organizing structure.

The only difference between the two experiments was the instruction to attend to similarity. It seems, then, that accessing the same schema does not automatically connect two episodes in memory; instead, some strategic purpose is necessary to promote activation of prior episodes. Note that the comparison alone is not the cause of the effect, since both conditions required comparing story similarity. When the task includes a strategic purpose for the reminding, such as in a straightforward memory test for access to prior thematically-related exemplars, robust evidence for thematic reminding has been found.

5. PROCESSING THEORIES

We have examined many types of representations that have grown out of conceptual dependency, including scripts, plans, goals, MOPs, and TOPs. The point of all these representations is to facilitate inferencing. Conceptual dependencies captured many inferences that can be made about physical actions. Since not all events are thought of in terms of physical features, other representations seemed to be necessary. Scripts captured knowledge about the most likely inferences to make in stereotypical situations. Plan and goal representations provided inferential capabilities at this level of understanding. TOPs combined goal/plan knowledge with contextual features, to provide more specific predictions useful in planning or plan understanding.

The inferential capabilities provided by these representations has led to a general approach to processing which emphasizes the use of knowledge to guide processing as much as possible. This has been true in many types of processing, such as language understanding, learning, planning, and reasoning. In this section, we will discuss some of the processing theories which have been developed around these representations. In particular, we will focus on two areas: language understanding and reasoning.

5.1. Language Understanding

Conceptual dependency was originally developed as a representation theory for language understanding. The processing theory which accompanied it was highly knowledge-based. One of the central points of this theory was that *expectations* could be generated from representations, which would then play a key role in guiding the processing of subsequent input.

This general approach to language understanding was radically different from most other work of its time, which focused largely on syntactic processing (e.g., [14,15]). Because of the difference in focus, several issues arose. First, in the expectation-based approach, a key component of the language understander is a conceptual knowledge base. In addition, the system must have linguistic or syntactic knowledge. How should these two types of knowledge be integrated, if at all? Should syntactic knowledge and conceptual knowledge be completely separate, or completely integrated, or somewhere in between? And in processing, should syntactic and semantic analysis proceed in tandem, or should the process be modularized?

Another issue also arose from the different goal of a conceptual analyzer. Since the final product of the parser is not syntactic in nature, what should the role of syntax be in conceptual analysis? Is it necessary to build a syntactic representation of an input text during parsing, if

this representation is simply going to be thrown away afterward, leaving the conceptual representation? Or can a conceptual analyzer get away with less syntactic analysis, since this analysis is not the final goal of the parser?

5.1.1. Request-Based Parsing

One approach that has been taken to conceptual analysis has involved the use of *requests*, or test-action pairs, to encode parsing knowledge. Requests were first used in Riesbeck's analyzer [16]. The requests in this parser were mainly stored in the lexicon.

A request could be in one of two states: active or inactive. A request was activated when the parser encountered a word whose dictionary entry contained that request. Once active, a request stayed active until it *fired*, or was executed; or until it was explicitly deactivated by another request. A request fired if it was in the active state and the conditions of active memory satisfied the test portion of the request's test-action pair.

Requests were largely responsible for building conceptual representations. Thus, common actions performed by requests were building an instantiation of a particular concept, or filling a slot in an already-built concept. For example, the dictionary entry of verb "ate" contained a request to build an instantiation of the concept INGEST (the Conceptual Dependency [1] primitive underlying eating and drinking); and a request to look for a noun group after the verb which referred to a food item, which, if found, was placed in the OBJECT slot of the INGEST.

Some requests in Riesbeck's parser were not stored in the lexicon. For example, at the beginning of a sentence a request was activated which looked for a noun group. When one was found, it was placed in a variable called #SUBJ. Later, when a request from a verb took the noun group from #SUBJ and placed it in the appropriate slot (usually the ACTOR slot) of the verb's representation. However, the number of requests like this was quite small, and thus most of the requests in the system were lexically-based.

The request-based method of parsing has been used in many other parsers since Riesbeck's parser. In the Conceptual Analyzer (CA) [17], requests were used in much the same way as in Riesbeck's analyzer, but with the elimination of many non-lexically-based requests. Thus, instead of a request activated at the beginning of a sentence which looked for a noun group, CA had requests in the dictionary entries of verbs, looking back in the sentence for a noun group which could function as the subject. For instance, the verb "ate" had a request looking for a noun group to its left, which was an ANIMATE. If such a noun group was found, it was placed in the ACTOR slot of the INGEST. Other parsers using request-based knowledge include the Integrated Partial Parser (IPP) [18], the Word Expert Parser (WEP) [19], and BORIS [13].[3]

5.1.2. Integration and Request-Based Parsing

One of the main tenets behind request-based conceptual analysis has been that the traditional separation of text analysis into morphological, syntactic, semantic, and pragmatic phases should be eliminated. This is for two reasons: first, given that the goal of conceptual parsing is to build a meaning representation, and not a syntactic analysis, there is no *a priori* reason for a separate syntactic analysis phase to exist. Second, since semantic and pragmatic information can sometimes help to eliminate syntactic, or even morphological, ambiguities, semantics should be brought into the parsing process as quickly as possible. An example given in [20] involved the following sentence:

> Hunting dogs can be dangerous.

Out of context, this sentence is syntactically (and semantically) ambiguous. However, in the following contexts, the ambiguity is eliminated:

> Do you want to try shooting those dogs that have been pillaging the village?—No, hunting dogs can be dangerous.

[3] The test-action pairs in WEP and BORIS were called *demons*, rather than requests.

Let's take some dogs with us when we go to hunt moose.—No, hunting dogs can be dangerous.

In the context of shooting dogs, in the first example, the words "hunting dogs" make more sense as a gerund and its object, since the semantic interpretation of this syntactic sense fits into the semantic context. However, in the second sentence, since the context is hunting moose, and since we know that dogs are often used to assist in the hunting of other animals, the better interpretation of "hunting dogs" is that "hunting" is an adjective, modifying "dogs."

The desire to do away with the separation of syntax and semantics has been articulated further in [21] in terms of the *Integrated Processing Hypothesis*. Schank and Birnbaum suggest that the integration of syntax and semantics in a parser can be measured in terms of three aspects of the parser: its *control structure*, or the processes which occur during parsing; its *representational structures*, or the representations which it builds during the parsing process; and its *knowledge base*, or the set of rules which the parser draws on and which drive the parse of a text. A parser with an integrated control structure would have no separate syntactic or semantic processes or phases; one with integrated representational structures would not build any separate syntactic structure during the parsing process, but instead only semantic representations of its text; and a parser with an integrated knowledge base would have no rules which contained only syntactic knowledge.

The Integrated Processing Hypothesis states that syntax and semantics should be completely integrated in the control structure and representational structures, and that much of the knowledge base should also be integrated, although some separate syntax will exist in the knowledge base. Thus, it advocates that no separate phases of parsing should exist, that no purely syntactic information should be contained in the representations built during the parsing process, and that only some rules will exist in the parser's knowledge base which are purely syntactic.

In light of the goal of bringing semantic and conceptual information to bear as early as possible in the parsing process, at least some of the motivation behind the Integrated Processing Hypothesis is clear. Since semantics can aid even the earliest stages of the parsing process, it is important that a parser's control structure be highly integrated. One way to ensure that the control structure is integrated is to integrate the semantic and syntactic knowledge in a parser as much as possible. Thus, its knowledge base should also be highly integrated. Finally, with regard to representational structure, the goal of conceptual parsers is to build a conceptual representation of the meaning of a text, not to produce a syntactic parse tree for the text. Therefore, syntactic representations should not be built during parsing unless it is clear that they aid in the process of building the conceptual representation. Given these motivations, then, the Integrated Processing Hypothesis takes almost as strong a stand on the integration of syntax and semantics as can be taken.

The result of these principles has been the use of lexically-based requests to encode syntactic knowledge. Lexically-based requests meet, more or less, with the criteria of the Integrated Processing Hypothesis. Certainly syntax and semantics are completely integrated in terms of process. Requests carry on in parallel any syntactic processing with the building of conceptual structures. Requests are also as integrated as possible with respect to the knowledge base, since they usually contain both syntactic and semantic knowledge. For example, the requests discussed earlier for the word "ate" contained the conceptual knowledge that "ate" refers to the concept INGEST, and that the ACTOR of INGEST should be an ANIMATE and the OBJECT of INGEST should be a food item. These same requests also contained the syntactic information that the ACTOR of "ate" occurs before the verb in an active sentence, and the OBJECT after the verb. In representational structures, also, the integration is high using requests, since no syntactic representations are explicitly built by the requests.

5.1.3. Semantics-First Parsing

Another system which utilized a more explicit grammar, yet was highly influenced by semantic information was the MOPTRANS system [22,23]. MOPTRANS translated short (1–3 sentences) newspaper stories about terrorism and crime. Source language included English, Spanish, Ger-

man, and Chinese; target languages were English and German. In this system, although knowledge was modularized into relatively separate syntactic and semantic components, the application of knowledge was integrated at processing time.

As MOPTRANS parsed a piece of text, the semantic and syntactic representations that it built were kept in its active memory. During parsing, new constituents were added to active memory as each new word was read. As new constituents were added, semantics was asked if anything in active memory "fit together" well; that is, if there were any semantic attachments that could be made between the elements in active memory. If so, MOPTRANS' syntactic rules were consulted to see if any of these semantic attachments were syntactically legal. If so, the appropriate syntactic rule was run, making syntactic and semantic attachments, and possibly removing some constituents from active memory. In other words, semantics proposed various attachments, and syntax acted as a filter, choosing which of these attachments made sense according to the syntax of the input.

To make this more clear, consider how the following simple sentence was processed by MOPTRANS:

John gave Mary a book.

MOPTRANS' dictionary definitions contained information about what semantic representation the parser should build when it encountered a particular word. Thus, "John" caused the representation PERSON to appear in the parser's active memory. At the same time, since "John" is a proper noun, the syntactic class NP was also activated.

When the word "gave" was processed, MOPTRANS' definition of this word caused the CD representation ATRANS to be placed in active memory. At this point, then, active memory contained the following:

```
"John"                "gave"
NP                    V
(PERSON NAME JOHN)    (ATRANS)
```

MOPTRANS considered the two semantic representations in active memory, PERSON and ATRANS. The semantic analyzer tried to combine these representations in whatever way it could. It concluded that the PERSON could be either the ACTOR or the RECIPIENT of the ATRANS, since the constraints on these roles are that they must be ANIMATE. It also concluded that the PERSON could be the OBJECT of the ATRANS (that is, the thing whose control or possession is being transferred). However, since this role was expected to be a PHYSICAL-OBJECT, a more general category than ANIMATE, the match was not as good as with the ACTOR or RECIPIENT roles.[4]

This is the point at which the MOPTRANS parser utilized its syntactic rules. Semantics determined that two possible attachments were prefered. Now the parser examined its syntactic rules to see if any of them could yield either of these attachments. Indeed, the parser's Subject Rule would assign the PERSON to be the ACTOR of the ATRANS. The Subject Rule is shown in Figure 12. This rule applied when an NP was followed by a V, and when the NP could fill the ACTOR slot of the semantic representation of the V. The NP would be marked as the SUBJECT of the V, and the V would be marked as having a subject (S). As dictated by the RESULT of the rule, the S would remain in active memory, but the NP would be removed, since its role as subject would prevent many subsequent attachments to it, such as PP attachments. In addition to these syntactic assignments, the semantic representation of the NP "John" would be placed in the ACTOR slot of the ATRANS representing the verb.

After the execution of this rule, then, active memory contained the following:

```
"gave"
S
(ATRANS ACTOR (PERSON NAME JOHN))
```

[4]For details about how the semantic analyzer reached these conclusions, see [22].

```
Subject Rule

Syntactic pattern:          NP, V (active)
Additional restrictions:    NP is not already attached syntactically
Syntactic assignment:       NP is SUBJECT of V, V is indicative (S)
Semantic action:            NP is ACTOR of V (or another slot, if specified
                               by V)
Result:                     S


Dative Movement Rule

Syntactic pattern:          S, NP
Additional restrictions:    S allows dative movement
Syntactic assignment:       NP is (syntactic) INDIRECT OBJECT of S
Semantic action:            NP is (semantic) RECIPIENT of S (or another
                               slot, if specified by S)
Result:                     S, NP


Direct Object Rule

Syntactic pattern:          S, NP
Syntactic assignment:       NP is (syntactic) DIRECT OBJECT of S
Semantic action:            NP is (semantic) OBJECT of S (or another
                               slot, if specified by S)
Result:                     S, NP
```

Figure 12. Some grammar rules used in MOPTRANS.

The rest of the sentence was parsed in a similar fashion. After the word "Mary" was read in, active memory contained two nodes: the node above representing "gave," and a node for "Mary" which was an NP whose semantic representation was (PERSON NAME MARY). To determine how these nodes should be attached, semantics was asked for its preference. It determined that the RECIPIENT slot of the ATRANS was the best attachment.[5] Syntax was consulted to see if any syntactic rules could make this attachment. This time, the Dative Movement Rule in Figure 12 was found. When applied, this rule assigned "Mary" as the indirect object of "gave," and placed the PERSON concept representing "Mary" into the RECIPIENT slot of the ATRANS, leaving active memory containing the following:

```
"gave"                                    "Mary"
S                                         NP
(ATRANS ACTOR (PERSON NAME JOHN)          (PERSON NAME MARY)
        RECIP (PERSON NAME MARY))
```

The final NP in the sentence, "the book," was attached to "gave" in a similar way. Semantics was asked to determine the best attachment of "book," which was represented as a PHYSICAL-OBJECT, to other concepts in active memory, which at this point contained the nodes for "gave" and "Mary." Semantics determined that the best attachment was to the OBJECT role of the ATRANS, since a book is a type of PHYS-OBJ, which is what ATRANS expects to fill its OBJECT slot. The syntactic rule which could perform this attachment was the Direct Object Rule, also shown in Figure 12. This rule was applied, yielding the final semantic representation:

[5] Just as earlier, "Mary" could either be the ACTOR or RECIPIENT of the ATRANS, but "John" has already been assigned as the ACTOR.

Directed prediction

1. Predict a concept.
2. Retrieve associated patterns which will recognize that concept.
3. Predict patterns' "leading edges" (concepts or lexical items).

Opportunistic recognition

1. Activate a concept.
2. Find all referenced predictions.
3. Refine to more specific predictions.
4. If the predictions' pattern is complete then activate the predicted concept; else predict the pattern's next element.

Figure 13. Components of DMAP.

```
(ATRANS ACTOR  (PERSON NAME JOHN)
        OBJECT (PHYS-OBJ TYPE BOOK)
        RECIP  (PERSON NAME MARY))
```

Note that although the Direct Object Rule could have applied syntactically (and even semantically) when "Mary" was found after the verb, it was never even considered. This is because the semantic analyzer prefered to place "Mary" in the RECIPIENT slot of the ATRANS rather than the OBJECT slot. Since a syntactic rule was found which accomodated this attachment, namely the Dative Movement Rule, the parser never tried to apply the Direct Object Rule.

5.1.4. Direct Memory Access Parsing

One of the common themes of many of the processing structures in the conceptual dependency family is that these structures also serve as memory structures. However, many of the language understanding programs have not utilized this fact. Martin has integrated processing and memory further in the Direct Memory Access Parser (DMAP) [24]. In DMAP, there is no distinction between representations built during parsing and episodic memory. Representations simply consist of activated memory nodes. This enables the system to make inferences based on all information in the memory.

DMAP allows any concept in memory, at any level of generality, to post expectations for other concepts or lexical items. The posting and verification of expectations is broken down into two components: *directed prediction* and *opportunistic recognition* [24, p. 476], shown in Figure 13.

Patterns occur at all levels in memory, and can refer to lexical items or phrases, or schemas such as scripts or MOPs. Thus, inferences and lexical expectations are handled with the same mechanism. Due to this complete integration, changes in DMAP's memory result in automatic changes in pattern-driven expectations.

5.2. Planning and Reasoning

Many of the conceptual structures discussed in this paper suggest a paradigm for planning, problem-solving, and other types of reasoning tasks. The main message is that memory contains many different types of structures, which can make very specific predictions that can be useful in these tasks. Thus, the general approach taken using these structures is that, rather than viewing these tasks as a search problem, much of planning and reasoning is a memory retrieval task.

Reflecting this general approach, the *case-based reasoning* paradigm has developed. We can contrast case-based reasoning with more traditional planning systems. In traditional planning, a goal is pursued by constructing sequences of operators (a plan) which will satisfy it (e.g., [25–29]). Operators have *preconditions* which must be satisfied before they can be applied. Thus, if an operator is selected whose preconditions are not true, then *subgoals* are generated. If several goals are to be pursued at once, then plans are found to satisfy each goal individually, and then interactions between the plans are checked, to make sure that the side-effects from one plan does not destroy the preconditions necessary for other plans.

This approach to planning is very search-intensive. Operators must be selected, from a set of possibly many operators which could apply, then preconditions must be planned for by selecting more operators. If a precondition cannot be satisfied, then the planner must backtrack, selecting another possible operator to satisfy the original goal.

In contrast to this approach, case-based planners search their memory for the closest plan previously constructed, based on a set of possible indices including the goal or goals to be satisfied, and other features of the context which may be relevant. This is very similar in spirit to the notion of TOPs, which are also organized according to a goal configuration as well as specific features of the context. Once a previous plan is selected, it may have to be modified to meet the specifics of the current planning situation. This involves applying plan transformation rules which dictate, according to the differences in the former and current situations, how the plan must change.

The case-based approach was used in the CHEF program [30], which planned Szechuan recipes given a set of constraints, such as ingredients to be used, style, and desired flavor. CHEF found previous recipes in its memory, indexed according to these features, as well as the interactions between features. Then the previous recipe was modified, if it differed from the constraints of the current case. Modification of recipes could involve further goal interaction, which either resulted in further modifications, and/or possible abandonment of one or more goals.

An example of CHEF in operation was its planning of a stir-fry dish with beef and broccoli. In this case, CHEF's goal was to build a recipe satisfying three constraints: the goal to have a stir-fried dish, the goal to include beef, and the goal to include broccoli. Initially, CHEF found a previously constructed plan (recipe) in its memory for beef with green beans, another stir-fry dish. The initial modification was to simply substitute the broccoli for the green beans. A set of *object critics* was responsible for making this substitution, and for identifying differences in the recipe required by the change, such as chopping the broccoli, etc.

The simple substitution created a goal interaction problem, however: the brocolli would become soggy if cooked in the same way as the green beans. Given this failure, CHEF had to construct an explanation of why this happened. After analysis of the failure, CHEF further modified the new plan by suggesting that the broccoli should be added later in the cooking process than the green beans.

The important thing to note in this example is the marked difference in the plan construction process: rather than reasoning from first principles, CHEF created plans by recalling previous plans, then modifying them accordingly. Sometimes this would lead to planning failures, which would then have to be explained, leading to further plan modifications.

Several other case-based systems have been written to perform other planning and reasoning tasks. Simpson's PERSUADER [31] played the role of a mediator to resolve adversarial conflicts. JUDGE [32] reasoned about criminal trials in order to decided upon appropriate sentences. More recent systems have investigated the combination of case-based and rule-based reasoning. These include MEDIATOR [33], which investigated Simpson's domain further using a hybrid approach; and CABARET [34] which operated in the domain of legal reasoning.

Similar work has also been done in an explanation-generating program called ABE (Adaptation-Based Explanation) [35]. ABE tried to explain events in terms of *explanation patterns* [36], or pre-packaged explanations. These patterns were explanations of previously encountered explanation failures. The same basic paradigm was followed: an explanation pattern was retrieved from memory, according to the features of the events being explained. In the case that no exact match was found, the retrieved explanation was modified according to a set of rules so as to fit the current situation. The result was a system which produced explanations without having to generate causal chains from scratch for each explanation.

6. CONCLUSION

The representational structures that have descended from conceptual dependencies are many and varied. They range from low-level descriptions of physical actions, such as CD's themselves, to compilations of sequences of these descriptions such as scripts, abstractions of these sequences

such as MOPs, and goal-oriented representations such as planboxes, named plans, goal categories, and TOPs.

Given the wide range of structures, it might seem that they do not share much in common. However, the motivation behind all of them is the same: to facilitate inferences. As people, we can make inferences at all sorts of levels of generality; the range of proposed structures reflects this.

As these theories have developed, another common theme has been that processing structures and memory structures are the same. That is, not only are these structures useful in processing tasks such as language understanding, planning, and reasoning, but they are also used to store and retrieve episodes. As a result, much of the processing done using these structures is memory storage and retrieval. For example, in planning, rather than constructing a plan from first principles every time a goal arises, the case-based approach suggests that much of what is going on is finding the appropriate previously-built plan in memory, and then adapting it to the current situation.

One might feel that the notion of building structures around inferential capability is a painfully obvious one. However, if we examine related work in language understanding, problem solving, reasoning, and other tasks, we find that this does not seem to be the case. For example, in natural language processing, semantic theories that existed before conceptual dependencies (and even afterward) did not address this issue. Representations such as Katz and Fodor's [37] and Fillmore's case grammar [38] made some attempt to categorize dependencies between constituents in sentences (although many of these dependencies were still highly syntactic in nature), but lexical items (or word senses) were used as the fundamental pieces out of which the representations were built. There was no attempt to capture the similarities in meaning across different lexical items; hence these representations would not have easily facilitated inferential capabilities. Even now, many natural language semantic representations such as SNePS [39] and HPSG [40] do not attempt to capture these meaning similarities in the representational predicates used.

The same can be said of many planning representations. Although operators are usually written so as to be general and facilitate the planning process (a form of inference), in constructive planners no attempt is made to represent knowledge about what sequences of operators are best for particular situations. This information has to be re-derived over and over again. This is a fault of the representation: operators are too general, and do not capture this level of inferential capability.

Thus, perhaps the most important common thread that connects the CD family of research is the assumption that intelligence is knowledge-intensive, rather than processing-intensive. If the representations used to perform a task are rich enough, then processing strategies traditionally associated with AI programs, such as search, become much less important. The result is simpler processing theories which are highly dependent on a very rich, and highly organized, knowledge base.

REFERENCES

1. R. Schank, Conceptual dependency: A theory of natural language understanding, *Cognitive Psychology* 3, 552–631 (1972).
2. R. Schank, *Conceptual Information Processing*, North-Holland, Amsterdam (1975).
3. R. Schank and R. Abelson, *Scripts, Plans, Goals, and Understanding*, Lawrence Erlbaum Associates, Hillsdale NJ (1977).
4. C. Rieger, Conceptual memory and inference, In [2].
5. M. Minsky, A framework for representing knowledge, In *The Psychology of Computer Vision*, (Edited by P. Winston), McGraw-Hill, New York (1975).
6. D. Rumelhart, Notes on a schema for stories, In *Representation and Understanding: Studies in Cognitive Science*, (Edited by D. Bobrow and A. Collins), Academic Press, New York (1975).
7. D. Bobrow and D. Norman, Some principles of memory schemata, In *Representation and Understanding: Studies in Cognitive Science*, (Edited by D. Bobrow and A. Collins), Academic Press, New York (1975).
8. G. Bower, J. Black and T. Turner, Scripts in memory for text, *Cognitive Psychology* 11, 177–220 (1979).

9. E. Charniak, A framed PAINTING: The representation of a common sense knowledge fragment, *Cognitive Science* 1, 355–394 (1977).
10. R. Schank, *Dynamic Memory*, Cambridge University Press, Cambridge U.K. (1982).
11. R. Wilensky, Understanding goal-based stories, Ph.D. Thesis, Department of Computer Science, Yale University, Research Report #140 (1978).
12. C. Seifert, G. McKoon, R. Abelson and R. Ratcliff, Memory connections between thematically similar episodes, *Journal of Experimental Psychology: Human Learning and Memory* **12**, 220–231 (1986).
13. M. Dyer, *In-depth Understanding: A Computer Model of Integrated Processing for Narrative Comprehension*, MIT Press, Cambridge, MA (1983).
14. W. Woods, R. Kaplan and B. Nash-Webber, The lunar sciences natural language information system: Final report, Technical Report #2378, Bolt Beranek and Newman, Inc., Cambridge, MA (1972).
15. T. Winograd, *Understanding Natural Language*, Academic Press, New York (1972).
16. C. Riesbeck, Conceptual analysis, In [2].
17. L. Birnbaum and M. Selfridge, Conceptual analysis of natural language, In *Inside Computer Understanding*, (Edited by R. Schank and C. Riesbeck), Lawrence Erlbaum Associates, Hillsdale, NJ (1981).
18. M. Lebowitz, Generalization and memory in an integrated understanding system, Ph.D. Thesis, Department of Computer Science, Yale University, Research Report #186 (1980).
19. S. Small, Word expert parsing: A theory of distributed word-based natural language understanding, Ph.D. Thesis, Department of Computer Science, University of Maryland (1980).
20. C. Riesbeck and R. Schank, Comprehension by computer: Expectation-based analysis of sentences in context Research Report #78, Department of Computer Science, Yale University (1976).
21. R. Schank and L. Birnbaum, Memory, meaning, and syntax, Research report #189, Department of Computer Science, Yale University (1980).
22. S. Lytinen, The representation of knowledge in a multi-lingual, integrated parser, Ph.D. Thesis, Department of Computer Science, Yale University, Research Report #340 (1984).
23. S. Lytinen, Dynamically combining syntax and semantics in natural language processing, In *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA, 574–578 (1986).
24. C. Martin, Pragmatic interpretation and ambiguity, In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society, Ann Arbor, MI, August*, 474–481 (1989).
25. A. Newell and H. Simon, *Human Problem Solving*, Prentice-Hall, New York (1972).
26. R. Fikes and N. Nilsson, STRIPS: A new approach to the application of theorem proving to problem solving, *Artificial Intelligence* **2**, 189–208 (1971).
27. E. Sacerdote, A structure for plans and behavior, Research Report #109, SRI Artificial Intelligence Center, Palo Alto, CA (1975).
28. G. Sussman, *A Computer Model of Skill Acquisition*, American Elsevier, Artificial Intelligence Series, New York (1975).
29. J. Laird, A. Newell and P. Rosenbloom, Soar: An architecture for general intelligence, *Artificial Intelligence* **33**, 1–64 (1987).
30. K. Hammond, *Case-Based Planning: Viewing Planning as a Memory Task*, Academic Press, Cambridge, MA (1989).
31. R. Simpson, A computer model of case-based reasoning in problem-solving: An investigation in the domain of dispute mediation, Ph.D. Thesis, School of Information and Computer Science, Georgia Institute of Technology (1985).
32. W. Bain, Case-based reasoning: A computer model of subjective assessment, Ph.D. Thesis, Department of Computer Science, Yale University (1986).
33. K. Sycara, Resolving adversarial conflicts: An approach integrating case-based and analytic methods, Ph.D. Thesis, School of Information and Computer Science, Georgia Institute of Technology (1987).
34. E. Rissland and D. Skalak, Combining case-based and rule-based reasoning: A heuristic approach, In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Detroit MI, August*, 524–530 (1989).
35. A. Kass, Adaption-based explanation: Extending script/frame theory to handle novel input, In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, Detroit MI, August*, 141–147 (1989).
36. R. Schank, *Explanation Patterns: Understanding Mechanically and Creatively*, Lawrence Erlbaum Associates, Hillsdale, NJ (1986).
37. J. Katz and J. Fodor, The structure of a semantic theory, *Language* **39**, 170–210 (1963).
38. C. Fillmore, The case for case, In *Universals in Linguistic Theory*, (Edited by E. Bach and R. Harms), pp. 1–88, Holt, Rinehart, and Winston, New York (1968).
39. S. Shapiro, The SNePS semantic network processing system, In *Associated Networks: Representation and Use of Knowledge by Computers*, (Edited by N. Findler), Academic Press, New York (1979).
40. C. Pollard and I. Sag., *Information-Based Syntax and Semantics*, Center for the Study of Language and Information, Menlo Park, CA (1987).