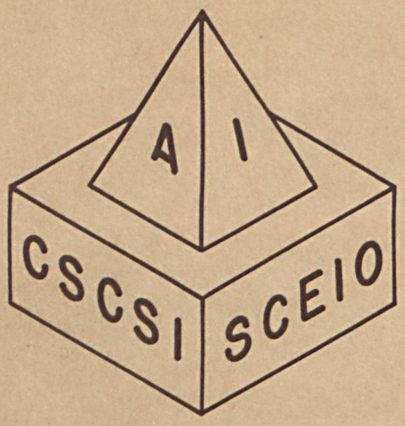
# PROCEEDINGS




## CONFERENCE 1980

VICTORIA, B.C.

14, 15, 16 MAY 1980

3. Goal Subsumption - Goal subsumption gives rise to dramatic situations when a subsumption state is terminated. For example, if John is happily married to Mary, and then Mary leaves him, all the goals subsumed by their relationship may now be problematic - John may become lonely, and miss his social interactions with Mary, for instance. Closely related to problems based on goal subsumption are those caused by the elimination of normal physical states. For example, becoming very depressed or losing a bodily function can give rise to the inability to fulfill recurring goals, and can therefore generate some interesting problems.

The resolution of goal subsumption termination involves establishing a new subsumption state to re-subsume the recurring goals.

## 4.0 GOAL RELATIONSHIP POINTS

Goal subsumption termination is a problem point component because previously subsumed goals become problematic. Goal conflict and goal competition endanger the fulfillment of some of a character's goals, and therefore generate dramatic impact. On the solution side, we have goal conflict resolution, goal abandonment, antiplanning, re-subsuming subsumption states, and spontaneous conflict and competition removal.

However the dramatic nature of goal relationships is not independent of how these relationships are presented in a text. For example, consider the following misuse of a potentially poignant goal relationship:

(7) John lost his job. Then he found another one.

This is not a particularly dramatic situation. It contains an instance of goal subsumption termination (John losing his job) and a solution to the problem this creates (John getting a new job). Nevertheless, (7) hardly qualifies as an interesting story.

The problem with (7) is that it contains the cause of the problem, the termination of a subsumption state, but no description of the problem itself. Contrast (7) with the Xenon story given at the beginning of this paper. John also lost his job in that story, but the situation contains considerably more dramatic impact. In the Xenon story we are given a description of John's problem state. He could no longer afford all the things he had become used to. Since the problem is spelled out in this story, its dramatic effect is more fully realized.

Thus the mere appearance of a problematic goal relationship does not guarantee its poignancy. The problem must appear in a form that spells out its implications. I call these forms point prototypes. A point prototype is a kind of distillation of the dramatic element of which the goal relationship is a part. The Xenon story above will serve to illustrate such a prototype.

The problem for John in the Xenon story is caused by a goal subsumption state terminating. To make this poignant, the story uses the problem point prototype in Figure 2 to fill out the circumstances of the problem.

Figure 2
Goal Subsumption Termination Prototype

1. Subsumption state
2. Cause of termination event
3. Problem state description
    1. Unfilled precondition
    2. Problematic goals
    3. New goal (optional)
    4. Emotional reactions (optional)

That is, to use subsumption state termination as a problem point, first state the subsumption state, followed by the cause of termination event. Then describe the problem state itself by listing the goals that are no longer subsumed; the goal of re-establishing a subsumption state may be stated also, along with any emotional reactions to the termination.

In the Xenon story, this prototype is instantiated as is shown in Figure 3.

Figure 3
Instantiated GST Prototype

1. Subsumption state - John has job.
2. Cause of termination event - Boss fires John.
3. Problem state description
    1. Unfilled precondition - John doesn't have enough money.
    2. Problematic goals - Maintaining car and house.
    3. New goal - John wants to resubsume these goals.
    4. Emotional reactions - not explicitly stated.

This problem is resolved in the story through a very common solution point called Fortuitous Circumstances. Spontaneous goal conflict resolution and external goal competition removal are also instances of this solution point component, which is shown in Figure 4.

Figure 4
Fortuitous Circumstance Solution Prototype

1. Undesired state
2. Fortuitous event
    1. Incidental action
    2. Fortuitous outcome
    3. New state
3. State consequence description

This solution prototype is instantiated in the Xenon story as Figure 5 shows.

Figure 5
Instantiated FC Solution Prototype

1. Undesired state - John doesn't have enough money.
2. Fortuitous event
   1. Incidental action - John saves rich man.
   2. Fortuitous outcome - Rich man gives John money.
   3. New state - John is rich.
3. State consequence description - John is happy and gets lots of possessions.

4.0.1 Some More Solution Point Components

Solution point components and their associated prototypes have not yet been analyzed in as much detail as the problem components have been. However, in addition to the fortuitous circumstances solution given above, several other solution point components seem to be common.

One such solution is called "More Desperate Measures". In this point, a problem is attacked by some plan that is normally not considered because of its high risk. Because of this risk, More Desperate Measures solutions tend to generate goal conflicts their user, thus creating another problem point component for the story. For example, in the Xenon story, after John loses his job, he might decide to rob a bank to get some money. Robbery entails a number of risks, so the use of this plan would create a goal conflict for John between his desire to have money and to preserve his well-being. This point would then be developed further in the story.

Overcoming a Limitation is another solution point seen with some frequency. This case can occur when a problem is based in part on a character's inability or lack of courage. Here the character attempts to overcome his personal limitation or see the error of his ways in order to resolve a problematic situation. For example, a typical fairy tale type plot might involve a character who is a subject of ridicule by his piers because he is a coward, and then overcomes his cowardess in some heroic deed.

5.0 CURRENT STATE OF PAM

As was mentioned previously, the naive explanation algorithm fails to find proper explanations for events in stories involving goal relationships. However, a more sophisticated version of PAM has been implemented that possesses knowledge about the goal relationships described above. PAM can use this knowledge to infer explanations for events in many complex goal relationship situations.

The following simple examples illustrate some of the situations involving goal relationships that PAM can understand:

**********

Goal Subsumption:

Input text:

JOHN AND MARY WERE MARRIED. THEN ONE DAY, JOHN WAS KILLED IN A CAR ACCIDENT. MARY HAD TO GET A JOB.

Input: WHY DID MARY NEED EMPLOYMENT?
Output: JOHN DIED AND SO SHE NEEDED A SOURCE OF MONEY.

**********

Pam infers that John's death terminates a subsumption state for Mary, and that she may seek to replace it. PAM uses this inference to infer that the explanation behind Mary's goal of getting a job.

**********

Goal Conflict:

Input texts:

WILMA WANTED TO HAVE AN ABORTION. WILMA WAS CATHOLIC. WILMA CONVERTED FROM CATHOLICISM TO EPISCOPALIANISM.

WILMA WANTED TO HAVE AN ABORTION. WILMA WAS CATHOLIC. WILMA WENT TO A ADOPTION AGENCY.

FRED WANTED TO TAKE HIS GUN HUNTING. FRED WANTED WILMA TO HAVE A GUN AT HOME. FRED ONLY HAD ONE GUN. FRED BOUGHT ANOTHER GUN.

**********

In the first two stories, PAM detects a conflict between Wilma's goal of having an abortion and her inferred goal of not having an abortion because she is Catholic. In the first story, PAM infers that Wilma resolved the conflict by changing the circumstance that gives rise to one of her goals, and fulfilled the other (i. e., she decided to have the abortion). In the next case, PAM infers that Wilma abandoned her goal of having an abortion because it menat less to her than violating her religious beliefs.

The third story is a goal conflict based on a resource shortage. Here PAM infers that Fred bought another gun so he could take one with him and leave one at home.

**********

Goal Competition:

Input text:

JOHN WANTED TO WIN THE STOCKCAR RACE. BILL ALSO WANTED TO WIN THE STOCKCAR RACE. BEFORE THE RACE, JOHN CUT BILL'S IGNITION WIRE.

Input: WHY DID JOHN BREAK AN IGNITION WIRE?
Output: BECAUSE HE WAS TRYING TO PREVENT BILL FROM RACING.

**********

This story contains an instance of a goal competition situation involving anti-planning. PAM explains John's action as part of a plan to undermine Bill's efforts by undoing a precondition for Bill's plan.

PAM also have been given some knowledge about poignancy. In particular, PAM knows about goal subsumption termination problem components, and fortuitous circumstance solution points. With this knowledge, pam can now understand the following version of the Xenon story:

**********

JOHN GRADUATED COLLEGE. JOHN LOOKED FOR A JOB. THE XENON CORPORATION GAVE JOHN A JOB. JOHN WAS WELL LIKED BY THE XENON CORPORATION. JOHN WAS PROMOTED TO AN IMPORTANT POSITION BY THE XENON CORPORATION.
JOHN GOT INTO AN ARGUMENT WITH JOHN'S BOSS. JOHN'S BOSS GAVE JOHN'S JOB TO JOHN'S ASSISTANT. JOHN COULDN'T FIND A JOB. JOHN COULDN'T MAKE A PAYMENT ON HIS CAR AND HAD TO GIVE UP HIS CAR. JOHN ALSO COULDN'T MAKE A PAYMENT ON HIS HOUSE, AND HAD TO SELL HIS HOUSE, AND MOVE TO A SMALL APARTMENT.
JOHN SAW A HIT AND RUN ACCIDENT. THE MAN WAS HURT. JOHN DIALED 911 THE MAN'S LIFE WAS SAVED. THE MAN WAS EXTREMELY WEALTHY, AND REWARDED JOHN WITH A MILLION DOLLARS. JOHN WAS OVERJOYED. JOHN BOUGHT A HUGE MANSION AND AN EXPENSIVE CAR, AND LIVED HAPPLY EVER AFTER.

**********

In addition to the many inference that are made to understand this story, PAM also recognizes that John's losing his job is an instance of a Goal Subsumption Termination problem, and that the hit and run victim rewarding John is an instance of a Fortuitous Circumstance solution to this problem. This representation could then be used by a summarization program to produce a summary that included only the events of John losing his job, the problems this caused, John's saving the rich man, and the rich man rewarding him (A summarization component that actually performs this taks in presently under construction. Although it has not yet been completed, it does not appear to be problematic, since all the information it needs is present in the structures PAM already produces).

## 6.0 SUMMARY

Stories constitute a subset of coherent natural language texts. For texts to be stories, they must be poignant in addition to being coherent. This point structure of a story serves to organize the representation of a story in memory so that more important episodes are more likely to be remembered than trivial events. Points also serve to generate expectations about what will happen next in a story, since a story reader is looking for the point of a story as the text is being read.

An important class of story points deals with human dramatic situations, and these most often contain a set of interacting goals that create difficulties for a character. A taxonomy of these goal relationships and the situations they give rise to is useful for detecting a point of a story, as well as for establishing its coherence as a text. When a goal relationship situation occurs as a problem point component, it will occur as part of a point prototype. These prototypes specify those aspects of the situations that should be mentioned in order to produce a dramatic effect.

The notion of a story point competes with the idea of story grammars as a way to characterize story texts. The story grammar approach attempts to define a story as a text having a certain form, while the story point idea defines a story as a text having a certain content. The form of a story is viewed here as being a function of the content of the story, not a reasonably independent object. Understanding stories, then, is not so much a question of understanding the structure of a text, but of understanding the point of what the text is about.

## References


1] Black, J. B. and Wilensky, R. (1979). An evaluation of story grammars. Cognitive Science, vol. 3, no. 3.

2] Charniak, E. (1972). Towards a model of children's story comprehension. AI TR-266, MIT.

3] Cullingford, R. E. (1978). Script Application: Computer Understanding of newspaper stories. Yale University Research Report #116.

4] DeJong, G. F. (1979). Skimming stories in real time: An experiment in integrated understanding. Yale University Research Report #158.

5] Kintsch, W., and Van Dijk, T. A. Recalling and summarizing stories. Language, 40, 98-116.

6] Mandler, J. M. and Johnson, N. S. Remembrance of things parsed: Story structure and recall. Cognitive Psychology, 9, 111-151.

7] Minsky, M. (1974). A framework for representing knowledge. MIT. AI Memo No. 306.

8] Rumelhart, D. E. (1975). Notes on a schema for stories. In D.G. Bobrow and A. Collins (eds.) Representation and Understanding: Studies in Cognitive Science. Academic Press, New York.

9] Rumelhart, D. E. (1976). Understanding and Summarizing brief stories. Center for Human Information Processing Technical Report No. 58. University of California, San Diego.

10] Schank, R. C. and Abelson, R. P. (1977). Scripts, Plans, Goals, and Understanding. Lawrence Erlbaum Press, Hillsdale, N.J.

11] Schank, R. C. and Wilensky. R. (1978). A Goal Directed Production System for Story Understanding. In D. A. Waterman and F. Hayes-Roth (Eds.), Pattern-directed Inference Systems. Academic Press, New York.

12] Schank, R. C. and Yale A. I. Project (1975). SAM -- A story understander. Yale University Research Report #43.

13] Stein, N. L. and Glenn, C. G. (1977). An analysis of story comprehension in elementary school children. In R. Freedle (Ed.) Multidisciplinary perspectively in discourse comprehension. Lawrence Erlbaum Associates, Hilldale, New Jersey.

14] Thorndyke, P. (1977). Cognitive Structures in Comprehension and Memory of Narrative Discourse. Cognitive Psychology, 9:88-110.

15] Wilensky, R. (1978a). Why John married Mary: Understandind Stories Involving Recurring Goals. Cognitive Science, vol. 2 no. 3.

16] Wilensky, R. (1978b). Understanding goal-based stories. Yale University Research Report #140.

# Speech Acts and the Recognition of Shared Plans

Philip R. Cohen
Center for the Study of Reading
University of Illinois &
Bolt Beranek and Newman, Inc.
Cambridge, MA

Hector J. Levesque
Dept. of Computer Science
University of Toronto
Toronto, Ontario

## Introduction

The purpose of this paper is to simplify Perrault, Allen, and Cohen's [1,2,9,10,19,20] plan-based theory of speech acts by revealing an important redundancy -- illocutionary acts. We show that illocutionary act definitions can be derived from more basic statements describing the recognition of shared plans -- plans based on the shared beliefs of the planner and some intended recognizer. Eliminating the redundancy is important for competence models of speech acts [10,19], but maintaining and exploiting it may be useful for computational and linguistic models [1,11,32] especially for those dealing with the "short-circuiting" of certain implicatures [4,18,32]. Our primary interest here is in competence models.

A plan-based theory of speech acts specifies that plan recognition is the basis for inferring the illocutionary force(s) of an utterance. The goal of such a theory is to construct a plan generation and recognition formalism that treats communicative and non-communicative acts uniformly. Such a theory should therefore state the communicative nature of an illocutionary act as part of that act's definition. A reasoning system would then not have to employ special knowledge about communicative acts; it would simply attempt to achieve its goals.

## Communication and the recognition of shared plans

Communication is intimately tied to plan-recognition. Grice [14] showed that "simple" recognition of intention(1) as might be performed by an unseen observer (cf. [24,31]) is insufficient as a basis for defining communicative acts. Instead he argued that speakers must plan for hearers to recognize their plans, and hearers must recognize the plans they were intended to recognize. Unifying Grice's analysis with Austin's [3], Searle [27,28] proposed that a speaker who is performing a speech act, such as a request, must intend to produce the effect of that action (to get the hearer to want to perform the requested act) by means of getting the hearer to recognize the speaker's intention to produce it. It was on this basis that Perrault and Allen [1] developed a scheme for recognizing indirect speech acts.

This research was supported primarily by the Advanced Research Projects Agency of the Department of Defense, monitored by the Office of Naval Research under Contract N00014-77-C-0378, and also, in part, by the National Research Council of Canada.

(1) For this paper, "intention" and "plan" should be considered synonymous

However, Schiffer [26] has argued that, to avoid counterexamples based on deception, the Gricean program (and its amendments [30,27]) produce an infinite regress of intending that one recognize an intention. To avoid this difficulty, he claims that the recognition of intention must be mutually believed. In other words, in order to communicate, speakers must make their plans shared or public knowledge.

In view of this problem, Perrault and Allen have suggested that their model be reformulated in terms of mutual beliefs. Since we are proposing such revisions, we shall discuss the essentials of their scheme.

## Allen and Perrault's Model

Building on prior attempts to link speech acts to plans [5,6,7,9,10,24], Allen and Perrault proposed two levels of speech act operators: surface and illocutionary. Illocutionary operators, e.g., REQUEST, are defined by stating propositions as preconditions, bodies, and effects with the understanding that:

a) preconditions are necessary for the successful "execution" of the act (or procedure) described by the operator;

b) effects are conditions that become true after the execution, and

c) the body is "a set of partially ordered goal states that must be achieved in the course of executing the procedure." [19, p.23].

Searle's recognition of intention" condition on speech acts is incorporated by defining an illocutionary operator's body to be "hearer believes speaker wants E" (abbreviated HBSW(E)), where E is the operator's effect. So, given the above understanding of operators, the illocutionary acts' operator's body needs to be achieved in the course of executing the operator. The standard ways of achieving them are by surface operators.

The classification of an utterance as a surface operator depends on the utterance's mood -- declaratives become S-INFORMs, imperatives become S-REQUESTs, and questions become S-REQUESTs to INFORM. Surface operators are considered to be primitive -- they represent what agents actually perform -- and consequently have no bodies. Their effects are defined to match the corresponding illocutionary operators' bodies -- i.e., HBSW(E). Thus, the standard way of achieving the body of a REQUEST is via an S-REQUEST. However, different combinations of surface act and propositional content can ultimately yield the same effect.

Mediating between the effects of surface operators and the bodies of the illocutionary ones is a set of plan-recognition inferences. Generally speaking, the inferences take the form: "the agent wanted P to be true because that would enable him to do A (precondition/action inference), which would result in E (Action/effect), which is a means of achieving B (body/action). The agent is then regarded as having wanted [to do] A, E, and [to do] B.

The inference process begins by observing an act and then assuming it was intentional -- the agent wanted to do it. The application of the action/effect inference speech act operator thus results in a proposition of the form: HBSW(HBSW(E)). Perrault and Allen supply rules for inferring new propositions E' such that HBSW(HBSW(E')). Each such E', inferred by these intended plan recognition rules, is regarded as having been communicated, in the Gricean sense.

Illocutionary act identification occurs when the body/action inference applies to the embedded HBSW(E) proposition, yielding, for instance, HBSW( REQUEST(S,H,B)). If the body/action inference occurs immediately after the expansion of a surface act, then a literal interpretation has been found. If there are intervening inferences, an indirect interpretation has been inferred.

Their uncovering of the inferences needed to arrive at indirect interpretations is the key accomplishment. But once those inferences are known, formal (and perhaps computational) models need not recognize illocutionary operators in order to communicatively infer their effects. Since, for their model, illocutionary force is being discovered by a hearer motivated to recognize the speaker's plan in order to facilitate it, the effects are all that is needed. We suggest, then, that the body/action inference collapses two distinct kinds of inference processing -- means/end reasoning and summarizing. The latter has not been shown to be essential to helpful plan recognition.

To demonstrate this point, Perrault and Allen's model will be elaborated upon in two ways:

1) To relate an illocutionary operator's body and effect, a plan will be stated that produces the effect once the body is achieved.

2) To capture the communicative nature of illocutionary acts more accurately, the steps of those plans should be mutually believed.

Surface speech act operators will be redefined to produce mutual beliefs about the speaker's goals, much as Perrault and Allen suggested (cf. Clark and Marshall's [8] analysis discussion of situations producing mutual beliefs). Once these steps are taken, the body/action inference will be unnecessary and illocutionary operators will reduce to redundant theorems.

## The Formalism

This section formalizes actions and plans, in conjunction with a planner's beliefs and desires. The style of formalization owes much to the literature on axiomatic specification of programming languages, and to Moore[17]. We do not intend to give the impression that a complete language with proof and model theories is lurking somewhere offstage. Although we will describe the formalism in terms of axioms, rules of inference and possible world states, it should be understood that these are intended to be more suggestive than definitive and that the formalism itself remains a topic of on-going research.

The formalism is a language with expressions of various types formed from primitive elements through rules of composition. Among the types of expressions we will discuss here are logical expressions (or wffs), action terms and terms denoting agents. For the remainder of this section, we will use "p", "q" and "r" as meta-variables ranging over wffs, "a" and "b" ranging over action terms and "x" and "y" ranging over agent terms. In addition, we will use "|-" as a predicate over wffs holding when the wff is a theorem.

By an action, we mean something an agent can do to change the state of the world. For example, the action term

(GIVE x o)

denotes the giving of the object denoted by "o" to the person denoted by "x". Notice that the action term does not mention the agent involved. This allows actions to be combined into more complex ones without having to fuss over the resulting agent. Examples of complex acts are

```
(IF p a b)  a conditional action
(SEQ a1 ... ak)  a sequence of
    actions
(WHILE p a)   an iterated action
```

Among the actions required for communication, we assume

```
(S-INFORM x p) saying to x that p
  is true
(S-REQUEST x p) asking x to make p
  be true
```

Among the wffs, we assume the usual connectives

```
(IMPLY p q)     (NOT p)
(AND p1 ... pk)     p = q
(FORALL v p)
```

In addition, there are wffs pertaining to communication

```
(ATTEND x y)   true iff x is
  attending to y
(BEL x p)   true iff p follows from
  what x believes  [15,16]
(WANT x p)  true iff p follows from
```

what x wants

Note that just because a BEL or WANT is true does not mean that the agent involved actively believes or wants the proposition in question. All that can be said is that in every world state that is consistent with what the agent believes, the second argument to BEL will be true. One particular kind of wff peculiar to actions is

```
(RESULT x a p)
```

which is true iff "p" is true in the world state resulting from the execution of "a" by "x" (or "a" does not terminate).

The behaviour of the expressions is governed by the axioms and rules of inference of the formalism. For example, action terms are specified using RESULT as in

```
|- (IMPLY (OWN x o) (RESULT x
                         (GIVE y o)
                            (OWN y o)))
```

The composite actions can be treated much like the axiomatic specification of programming language constructs. The IF rule, for example is

```
|- (IMPLY (AND (KNOWIF x p)
               (IMPLY p (RESULT x a q))
                    (IMPLY (NOT p)
                    (RESULT x b q)))
          (RESULT x (IF p a b) q))
```

where

```
|- (KNOWIF x p) = (OR (KNOW x p)
                       (KNOW x
                          (NOT p)))
```

and

```
|- (KNOW x p) = (AND p (BEL x p))
```

Similarly, the rule of consequence becomes

```
If |- p   then  |- (RESULT x a p)
```

Note that this must be a rule of inference in that the corresponding axiom (as an implication) cannot be a theorem. A related notion to this rule is the wff

```
(CAUSE x p q)
```

governed by the axiom

```
|- (CAUSE x p q) = (FORALL a/ACTION
          (IMPLY (RESULT x a p)
                    (RESULT x a q)))
```

so that a CAUSE is true iff anything that "x" does to bring about "p" also results in "q" being true. In other words, making "p" true makes "q" true.

Of crucial importance to the definition of speech acts, is the concept of mutual belief (or MB) governed by

```
If |- p then |- (MB x y p)
```

and

```
|- (MB x y p) = (BEL x (AND p
                          (MB y x p)))
```

The axiom states that mutual belief is equivalent to an infinite conjunction of beliefs in that, allowing that

```
|- (BEL x (AND p q)) = (AND (BEL x p)
                          (BEL x q))
```

then the following are implied by a mutual belief:

```
(BEL x p)
(BEL x (BEL y p))
(BEL x (BEL y (BEL x p)))   ...(1)
```

Given the notion of mutual belief, we can now state the two rules governing two primitive communication acts, S-INFORM and S-REQUEST:

```
|- (IMPLY (MB x y (ATTEND y x))
          (RESULT x (S-INFORM y p)
             (MB y x (WANT x (BEL y
                              (BEL x p]

|- (IMPLY (MB x y (ATTEND y x))
          (RESULT x (S-REQUEST y p)
             (MB y x (WANT x (BEL y
                              (WANT x p]
```

When discussing the behaviour of goal-directed agents, a useful concept is that of an agent being able to bring about some state of affairs that he wants:

```
|- (CAN x p) = (EXISTS a/ACTION
                   (KNOW x
                      (RESULT x a p)))
```

Note that this is an example of "quantifying in" in that it is not sufficient for "x" simply to know the existence of an action that results in "p", he must also know what action it is. On the other hand,

```
(BEL y (CAN x p))
```

could be true without "y" knowing how "x" will achieve "p" since, in this case, the quantifier is within the belief context. Given this characterization, we now turn our attention to plans, which, loosely speaking, are simply proofs that, given some set of beliefs, an agent is able to achieve some goal. More formally, the definition is

---

(1) Schiffer's [26] definition of mutual belief also includes an infinite conjunction starting from (BEL y p). Since we shall only be concerned about one person's point of view, we only deal with beliefs about mutual beliefs, which reduce to the above.

A plan for agent "x" to achieve some goal "q" is an action term "a" and two sequences of wffs "p0", "p1", ... "pk" and "q0", "q1", ... "qk" where "qk" is "q" and satisfying

```
1.  |- (BEL x (IMPLY p0
                   (RESULT x a q0)))
2.  |- (BEL x (IMPLY pi
                  (CAUSE x qi-1 qi)))
       i=1,2,...k
```

In other words, given a state where "x" believes the "pi", he will believe that if he does "a" then "q0" will hold and moreover, that anything he does to make "qi-1" true will also make "qi" true. Consequently, a plan is a special kind of proof that

```
|- (BEL x (IMPLY (AND p0 ... pk)
                  RESULT x a q)))
```

and therefore, assuming that

```
|- (IMPLY (BEL x p) (BEL x
                       (BEL x P)))
```

and

```
|- (IMPLY (BEL x (IMPLY p q))
          (IMPLY (BEL x p)
                 (BEL x q)))
```

a plan is a proof that

```
|- (IMPLY (BEL x (AND p0 ... pk))
                  (BEL x (CAN x q)))
```

Notice that the assumptions "pi" may be simplified in a plan in that if we have that

```
|- (BEL x (IMPLY p (RESULT x b
                    (AND p0 p1 ... pk))))
```

then we have a reduced plan for "x" to achieve "q" since

```
|- (BEL x (IMPLY p (RESULT x
                     (SEQ b a) q)))
```

This process can, of course, be iterated on the new assumptions. (Since action "b" achieves <u>all</u> the prerequisites, the "non-linearity" problem [21] remains.)

Among the corollaries to a plan are

```
|- (BEL x (IMPLY (AND p0 ... pi)
                  (RESULT x a qi))) i=1,...k
```

and

```
|- (BEL x (IMPLY (AND pi ... pj)
                  (CAUSE x qi-1 qj)))
          i=1,...k  j=i,...k
```

There are two main points to be made about these corollaries. First of all, since they are theorems, the implications can be taken to be believed by the agent "x" in every state. In this sense, these wffs express general methods believed to achieve certain effects provided the assumptions are satisfied. The second point is that these corollaries are in precisely the form that is required in a plan and therefore can be used as justification for a step in a future plan in much the same way a lemma becomes a single step in the proof of a theorem.

We therefore propose a notation for describing many steps of a plan as a single summarizing operator (akin to MACROPs in STRIPS [11]). An operator consists of a name, a list of free variables, a distinguished free variable called the agent of the operator, an effect which is a wff, a optional body which is either an action or a wff and finally, an optional prerequisite which is a wff. The understanding here is that operators are associated with agents and for an agent "x" to have an operator "u", then there are three cases depending on the body of "u":

1. If the body of "u" is a wff, then

```
|- (BEL x (IMPLY prerequisite
             (CAUSE agent body effect)))
```

2. If the body of "u" is an action term, then

```
|- (BEL x (IMPLY prerequisite
     (RESULT agent body effect)))
```

3. If "u" has no body, then it is simply an action and

```
|- (BEL x (IMPLY prerequisite
              (RESULT agent u effect)))
```

An example of this last kind of operator is the action GIVE, described above, which becomes the operator

```
[GIVE y o] agent: x
    effect: (HAVE y o)
    prereq: (HAVE x o)
```

One thing worth noting about operators is that normally the wffs used above

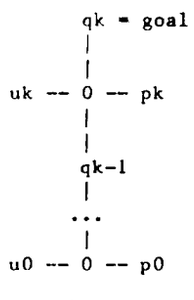
```
|-(BEL x (IMPLY prerequisite ...))
```

will follow from the more general wff

```
|-(IMPLY prerequisite ...)
```

as in the case of the GIVE example. However, this need not be the case and different agents could have different operators (even with the same name). Saying that an agent has an operator is no more than a convenient way of saying that the agent always believes an implication of a certain kind.

Before considering some examples of operators and their use in plans, we introduce the notation for describing plans.




where the "pi" and the "qi" are as before and the "ui" are the operators justifying the transition given "pi" from "qi-1" to "qi". In the simplest case, "pi" will be the prerequisite of "ui", with "qi-1" and "qi" the body and effect respectively. More generally, we need only require that

```
|- (BEL x (IMPLY pi prerequisite))
|- (BEL x (IMPLY qi-1 body))
|- (BEL x (IMPLY effect qi))
```

to satisfy the definition of a plan.

Operator Definitions

Given the above understanding of operator definitions, we present those operator schemas needed for our derivation of REQUEST. The first argument in the parameter list for a schema will be the agent.

[CAUSE-TO-WANT x y p]

```
effect: (WANT y p)
body:   (BEL y (WANT x p))
prereq: (AND  ~(WANT y ~p)
          (HELPFUL y x))
```

Provided y doesn't want NOT(p), and y thinks she is feeling helpfully disposed towards x, then getting y to believe that x wants p will get y to want p. Though this may be one way to influence someone's goals, more generally, one would like to state "y is given a reason for wanting p".

The SHARED-RECOG operator describes shared recognition of the agent's goals:

[SHARED-RECOG x y p q]

```
effect: (MB y x (WANT x q))
body:   (MB y x (WANT x p))
prereq: (MB y x (CAUSE x p q))
```

Of course, not every action produces mutual beliefs about someone's goals. Usually, the two parties must be mutually aware of the other's presence. However, once it is shared knowledge that x wants p, if its mutually believed that anything x does to make p true makes q true, then it will be mutually believed that x wants q. Clearly, we are exploiting the "follows from what the agent wants" interpretation of WANT here -- an agent wants all the inevitable results of his wants. Since this interpretation is currently forced on us by our formal tools, and since we want to formalize shared plans, we WANT this interpretation.

The next operator provides for private recognition of the agent's goals. It is similar to Perrault and Allen's [19] Plan-Deduce operator.

[PRIVATE-RECOG x y p q]

```
effect: (BEL y (WANT x q))
body:   (BEL y (WANT x p))
prereq: (BEL y (BEL x (CAUSE x p q)))
```

PRIVATE-RECOG should appear in plans when SHARED-RECOG is inappropriate, for instance when the conditions implying CAUSE statements are not mutually believed. Lack of shared knowledge can arise because of third parties (e.g., someone tells you what I want), because of the modality of communication (e.g., telephone conversations), or because one of the parties is an unseen observer.

The operator ACHIEVE models getting someone else to make p true.

[ACHIEVE x y p]

```
effect: p
body:   (WANT y p)
prereq: (CAN y p)
```

All that is required is that y know of some action resulting in p (x does not have to know which action that is). Then, simply by getting y to WANT p will CAUSE p to hold. Of course this idealization ignores the possibility of y's being unable or unwilling to actually perform the action. Future versions of CAN, using Moore's [17] RES modal operator, may ensure that y can also perform the action.

To allow for another way of influencing someone's goals, we define:

[FORCE-TO-WANT x y p]

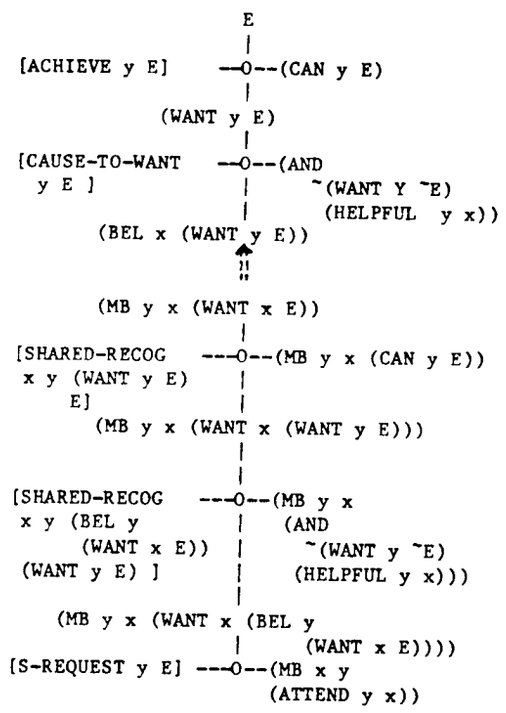
```
effect: (WANT  y p)
body:   (BEL y (WANT x p))
prereq: (BEL y (HAS-AUTHORITY-OVER x y))
```

The semantics of HAS-AUTHORITY-OVER (interpreted as x has authority over y) could be stated by filling out an organizational chart, or determining the status relationships between the parties.

Finally, the last operator we shall need is S-REQUEST, as defined earlier, to produce mutual beliefs about the speaker's goals. The prerequisite is that it be mutually believed between x and y that y is attending to x. (Note the order of x and y -- x must actually believe y is attending.) A crucial but as yet unanalyzed condition on classifying an utterance as an S-REQUEST to some particular hearer H is that it be mutually believed between the speaker, S, and H, that H is the intended addressee. This condition is not always satisfied, since some computer systems are conceptualized as "overhearing" (e.g., Genesereth's [13] ADVISOR).

A Plan

The following is x's plan to achieve E:




Given the individual operators and the interpretation of operators as theorems, the plan itself should be relatively self-explanatory. The prerequisites of the SHARED-RECOG operators shown imply those necessary for each individual step. For instance, since all theorems are mutually believed:

```
|-(BEL x (MB x y [IMPLY (CAN y E)
                        (CAUSE x (WANT y E)
                                 E],
```

therefore

```
|- (BEL x (IMPLY [MB y x (CAN y E)]
                 [MB y x (CAUSE x
                                (WANT y E) E)]))
```

the precondition of

(MB y x (CAN y E)) is shown. We have made one such implication explicit in the diagram -- the one marking the transition from shared to private beliefs.

Summarizing the plan

Various portions of the plan can now be summarized. First of all, consider the summary operator REQUEST:

[REQUEST x y E]

```
effect: (MB y x (WANT x E))
body:   (MB y x (WANT x (BEL y
                          (WANT x E))))
prereq: (AND (MB y x (CAN y E)
             (MB y x  ~(WANT y ~E))
             (MB y x  (HELPFUL y x))
```

If the prerequisite holds, any action making the body true achieves the effect. The propositions in the plan not summarized by this operator are achieved by virtue of y's private beliefs. The decision to include illocutionary or perlocutionary effects as part of some operator cannot be made solely on formal grounds. Also, notice that the third argument in the REQUEST schema is a proposition and not an action. While it would be desirable to derive a REQUEST to use an action, the formalism forbids its use since WANT takes a proposition as its argument.

We can also define other operators from this same plan. For instance,

[COMPLY y x E]

```
effect: E
body:   (MB y x (WANT x E))
prereq: (AND  ~(WANT y ~E)
              (HELPFUL y x))
              (CAN y E))
```

Clearly, we could have made the effect (WANT y E). COMPLY subsumes the remainder of the above plan, and progresses from shared beliefs to private ones (which cause y to achieve E). However, it is unclear which proposition should be chosen as the body. Should the body be a mutual belief (therefore involving a previous communication act) or need it only be a private belief? Finally, if E is a KNOWIF or KNOWREF proposition [1,2,10,19], then a more specific operator, ANSWER, can be defined.

Multiple summaries can occur because of some indirect uses of surface speech acts -- as in with an S-INFORM of x's WANT that leads to the same effect as an S-REQUEST [1,2,10,19]. Not only could the early part of the plan be summarized as an INFORM, and the later stages as a REQUEST, but a perhaps computationally useful operator would be one subsuming both the INFORM and REQUEST; call it a WANT-REQUEST. This formalizes the technique used in Woods et al's [32] system to "short-circuit" various chains of reasoning involving indirect speech acts.

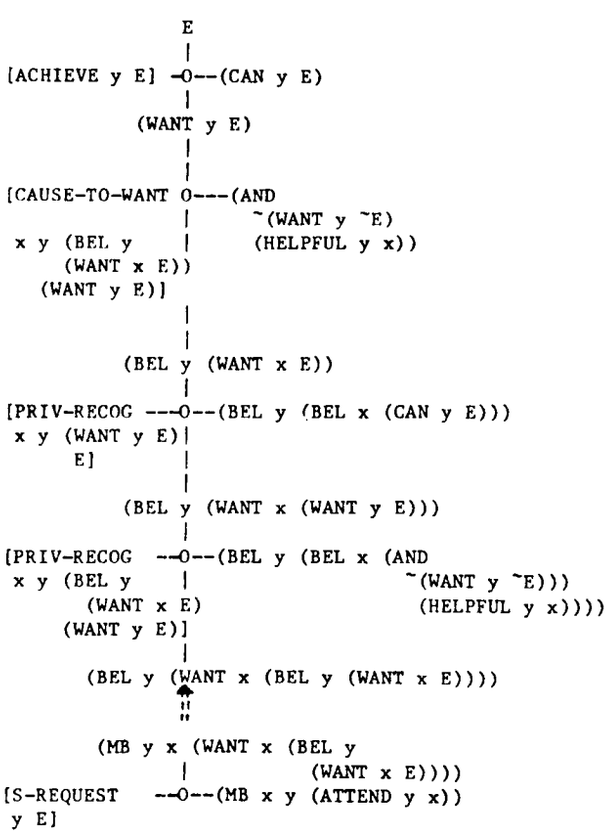
Substituting FORCE-TO-WANT for CAUSE-TO-WANT into the above plan allows us to create a summary termed COMMAND as follows:

[COMMAND x y E]

```
effect: (MB y x (WANT x E))
body:   (MB y x (WANT x (BEL y (WANT x E)]
prereq: (MB y x (BEL y
                   (HAS-AUTHORITY-OVER x y]
```

COMMAND differs from REQUEST in its being insensitive to the hearer's helpful (or non-helpful) disposition and to her prior desires.

Finally, we can create a plan in which the effect takes hold in a non-communicative manner:




Again, the implication marks the shift from mutual beliefs to private ones. By Schiffer's [26] definition, any effect obtained on the basis of private beliefs was not communicated. Thus, on philosophical grounds, one would not classify a summary of this plan as describing an illocutionary act.

## Possible Uses of Illocutionary Operators

The formalism indicates that certain illocutionary operators are redundant--they can be derived from other independently motivated operators. However, the redundancy is only relevant to achieving the illocutionary operator's effects. For the reasons stated below, the redundancy may be useful.

Illocutionary operators might be used to represent the meaning of illocutionary verbs. Consider verbs that report on social interaction. Corresponding operators can be defined to span multiple agents' achievements (e.g., COMPLY and ANSWER). Summary operators can perhaps be used for verbs requiring "uptake" [3]. Thus, a plan summarizable as a bet could contain portions summarizable as offerings and acceptances. The major questions for this approach would be when and to what end would those summaries expanded in the course of processing an utterance. Obviously the linguistic questions related to performatives are also relevant but as yet remain unanswered.

From a computational perspective, summary operators are useful in limiting a planner's search, as demonstrated by the use of MACROPs in STRIPS [12]. Summary operators allow for "short-circuiting" the interpretation of certain indirect speech acts ([18,2,4,31,32]). Further reduction in search could follow ABSTRIPS [22] in assigning priorities to the summary operator's prerequisites. Speech act plans could first be sought using high priority preconditions and later pruned by lower ones. Given suitable priority and threshold schemes, indirect achievement of a communicative goal may be as efficient as direct achievement.

Finally, the issues of dynamically acquiring summary operators, as in STRIPS, become relevant. Though a system may summarize a shared plan, there may be no corresponding illocutionary verb in its lexicon to describe that plan. This problem then presents an interesting challenge to a model of language use -- how could a system plan communicative acts to establish a jointly agreed upon vocabulary?

In summary, our model proposes a foundation for defining a class of illocutionary verbs. However, as the next section shows, there are formal and descriptive limitations to be overcome. Furthermore, other tests need to be applied to support the model.

## Limitations

Our scheme has only been applied to a narrow range of phenomena. First of all, we have only shown the redundancy for two illocutionary verbs ("requests" and "command") though a similar analysis has been done on "inform." Since these verbs are prototypical of Searle's [29] "directive" and "representative" classes, our hope is that this style of formalism can be extended to other members of those classes. Such an analysis is currently limited by our understanding of concepts such as benefit (for suggestions) and danger (for warnings).

We have not yet attempted to handle the class of indirect speech acts addressed by Perrault and Allen. Our efforts are currently hampered by the KNOWIF(P) --> P(or ~P) recognition inference stating that if you believe an agent wants to know whether or not P is true, then it is plausible to believe that agent wants P (or, wants ~P). The inference arises because a planner must determine whether or not an action's preconditions hold. In order to formalize the inference, an axiomatization of the behavior of a planner or a plan-recognizer is needed. Such a formalism would also have to capture stopping conditions for shared and private plan-recognition [1,2,13,25,32], and perhaps rating schemes for choosing the best plan [1,2,32].

Regarding the formalism, a major difficulty is the lack of an adequate axiomatization and semantics for BEL and WANT. For instance, the distinction between actively desiring, and "putting up with" (as the lesser of two evils)

needs to be drawn formally. BEL, given Hintikka's [15,16] treatment is the better understood concept.

A bothersome quirk of the formalism is that actions cannot appear as objects of want, and hence do not appear in the REQUEST summary operator. We are therefore searching for a propositional way to state that an action was done.

## Conclusions

The primary reason for pursuing this formalism is that is allows one to express naturally the communicative nature of illocutionary operators in terms of shared plans. It leads us to conclude that summarizing an utterance as the performance of an illocutionary act is not necessary to helpfully motivated plan recognition. The illocutionary operators that we have studied are redundant for achieving their effects, since the shared plans provide all the power, and their components are independently motivated. However, though we have suggested such operators are unnecessary, we cannot formally prove the point without further research, especially on the logic of WANT. The formalism has led to a foundation for "short-circuiting" certain implicatures, as recommended by Morgan[18], Perrault and Allen[19], and as attempted in Woods et al's [32] natural language system. Finally, it reveals the arbitrary nature of operator definition. Some choices can be decided using the adequacy test of third-party speech acts proposed by Cohen and Perrault [10]. Other decisions must await empirical evidence.

## References


1. Allen, J. A plan-based approach to speech act recognition (Doctoral dissertation, University of Toronto, 1979). Technical Report No. 131/79, Dept. of Computer Science, University of Toronto, January, 1979.
2. Allen, J.F., & Perrault, C.R. Analyzing intention in dialogue, forthcoming.
3. Austin, J.L. How to do things with words. J.O. Urmson (Ed.), Oxford University Press, 1962.
4. Brown, G.P. Indirect Speech Acts in Task-Oriented Dialogue: A Computational Approach, unpublished ms, MIT, 1979.
5. Bruce, B. Belief systems and language understanding (BBN Report No. 2973). January, 1975(a).
6. Bruce, B., Generation as a Social Action, Proceedings of the Conference on Theoretical Issues in Natural Language Processing, Cambridge, MA, 1975(b)
7. Bruce, B., & Schmidt, C.F. Episode understanding and belief guided parsing. Presented at the Association for Computational Linguistics Meeting at Amherst, Massachusetts (July 26-27, 1974).
8. Clark, H.H., & Marshall, C. Definite reference and mutual Knowledge. In A.K. Joshi, I.A. Sag, & B.L. Webber (Eds.), Proceedings of the Workshop on Computational Aspects of Linguistic Structure and Discourse Setting. New York: Cambridge University Press, in press.
9. Cohen, P.R. On knowing what to say: Planning speech acts (Doctoral dissertation, University of Toronto, 1978). Technical Report No. 118, Department of Computer Science, University of Toronto, January 1978.
10. Cohen, P.R. and Perrault, C.R., Elements of a plan based theory of speech acts, Cognitive Science, 1979, 3, 177-212.
11. Fikes, R., & Nilsson, N.J. STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, 1971, 2, 189-208.
12. Fikes, R., Hart, P., & Nilsson, N.J. Learning and executing generalized robot plans, Artificial Intelligence, 1972, 3, 251-288.
13. Genesereth, M.R., Automated consultation for complex computer systems, (Doctoral dissertation), Dept. of Computer Science, Division of Applied Sciences, Harvard University, September, 1978.
14. Hintikka, J. Knowledge and belief. Ithaca: Cornell University Press, 1962.
15. Hintikka, J. Semantics for propositional attitudes. In J.W. Davis et al. (Eds.), Philosophical Logic. Dordrecht-Holland: D. Reidel Publishing Co., 1969.
16. Grice, H.P. Meaning. The Philosophical Review, 1957, 66, 377-388.
17. Moore, R.C. Reasoning about knowledge and action (Doctoral dissertation, Massachusetts Institute of Technology, 1979). Artificial Intelligence Laboratory, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, February, 1979.
18. Morgan, J.L. Two types of convention in indirect speech acts. In P. Cole (ed.), Syntax and Semantics, Volume 9: Pragmatics, New York: Academic Press, 1978.
19. Perrault, C.R., & Allen, J.F. A plan-based analysis of indirect speech acts.
20. Perrault, C. R., Allen, J. F., & Cohen, P. R., Speech acts as a basis for understanding dialogue coherence, in Proceedings of the second conference on theoretical issues in natural language processing, Champaign-Urbana, Illinois, 1978.
21. Perrault, C.R., & Cohen, P.R. Inaccurate Reference, In A.K. Joshi, I.A. Sag, & B.L. Webber (Eds), Proceedings of the Workshop on Computational Aspects of Linguistic Structure and Discourse Setting. New York: Cambridge University Press, in press.
21. Sacerdoti, E.D. A structure for plans and behavior (Doctoral dissertation, 1975). Technical Note 109, Artificial Intelligence Center, Stanford Research Institute, Menlo Park, California, August 1975.
22. Sacerdoti, E.D. Planning in a Hierarchy of Abstraction Spaces. Proceedings of the Third International Joint Conference on Artificial Intelligence, Stanford, Calif., 1973.

23. Schank, R., & Abelson, R. Scripts, plans, goals, and understanding. Hillsdale, N.J.: Lawrence Erlbaum Associates, 1977.
24. Schmidt, C.F., Understanding human action, Proceedings of the conference on theoretical issues in natural language processing, Cambridge, MA, 1975.
25. Schmidt, C.F., Sridharan, N.S., & Goodson, J.L., The plan recognition problem: An intersection of artificial intelligence and psychology, Artificial Intelligence 10, 1979.
26. Schiffer, S. Meaning. Oxford: Oxford University Press, 1972.
27. Searle, J.R. Speech acts: An Essay in the philosophy of language. Cambridge: Cambridge University Press, 1969.
28. Searle, J.R. Indirect speech acts. In P. Cole & J.L. Morgan (Eds.), Syntax and semantics, (Vol. 3), Speech acts. New York: Academic Press, 1975.
29. Searle, J. R. A Taxonomy of Illocutionary Acts, in K. Gunderson (ed.), Language, Mind, and Knowledge, University of Minnesota Press, 1976.
30. Strawson, P.F. Intention and convention in speech acts. In The Philosophical Review, 1964, 5, 73.
31. Wilensky, R., Understanding goal-based stories, (Doctoral dissertation), Research report # 140, Dept. of Computer Science, Yale University, September, 1978.
32. Woods, W.A., Bobrow, R., Brachman, R., Cohen, P., Klovstad, J., Sidner, C., & Webber, B., Natural language understanding, Annual Report, Bolt Beranek and Newman, Inc., 1980

# UNDERSTANDING ARGUMENTS

Robin Cohen
Department of Computer Science
University of Toronto
Toronto, Ontario M5S 1A7

Abstract

This paper outlines a preliminary design for a system to understand one-sided arguments. These are a particular kind of conversation, where the speaker has one main objective: to convince the hearer of a particular point of view. Arguments are thus characterized by having an overall point, defended by some logical chain of reasoning. We develop methods to analyze arguments, considering them as intentional behaviour. For this first design, we concentrate on developing methods to recognize the logical form of the argument, by examining the relations between sentences.

## 1. The problem area

We are studying a particular kind of conversation - the one-sided argument. This is a speech with a main objective of convincing the hearer of a particular point of view. Arguments differ from other texts in that: (i) there is an overall point (untrue of stories) (ii) the point is an opinion which is to be defended (untrue of news reports) (iii) the individual sentences serve to support the point (untrue of informing rather than convincing arguments) (iv) there is an overall logical form: a method of reasoning, holding the argument together (untrue of non task oriented conversations).

Our main objective is to analyze arguments, producing a representation which reveals (i) the point and overall opinion (ii) the chain of reasoning supporting the point. The restricted form of arguments is used to develop a classification for each sentence as either claim, evidence for some claim, or a statement of control (i.e. a sentence about the structure of the argument - e.g. "We now present our conclusion"). To classify sentences and record the relationships between them, frames are defined for each of the basic logical rules of inference. The main operation of our analysis is thus a matching onto frames, which hold our representation and facilitate further processing.

The underlying philosophy of this system is that arguments may be considered as intentional behaviour. One motivation for this pragmatic approach is that there are some clear distinctions between shared and private knowledge in arguments. Speaker (S) and hearer (H) share some knowledge: both know that the main purpose is to convince; both are aware of standard techniques to convince (e.g. using analogies, contrast, examples, etc.). On the other hand, both the statements of S and the connections between them may be unknown to H. So H's task is both to recognize the logical forms being used and to believe that they are appropriate.

Another important reason for considering intentions is to facilitate the understanding process. H's comprehension process often involves deciphering and interpreting unstated assumptions. H may be able to determine unstated opinions of S or overall argument structure by

examining, for example, the choice of words (e.g. "however", "only"). But H also knows that S must facilitate H's understanding, in order to succeed in convincing H of his main point. H can thus postulate rules of coherence to interpret S's intentions and aid in analyzing the argument.

There seem to be two main levels to H's processing: determining what S believes, and deciding whether or not he, himself, believes it. The second task involves judging the credibility of arguments, and will not be addressed in this paper (See Section 3: Future Work).

This problem area, as defined, is different from other language understanding projects.

DeJong's FRUMP <DeJong 79> analyzes newspaper stories. This kind of text is similar to arguments in that (i) there may be statements of evidence and sources quoted (ii) it is important to believe the story. However, FRUMP does not concern itself with the underlying opinion on the overall topic, or with credibility. In contrast to FRUMP, we must distinguish the evidence in the argument and determine how the evidence supports the main opinion. Further, there is a basic representational difference between arguments and stories. DeJong himself addresses the issue in <DeJong 79>, indicating that his program can't handle editorials because these present arguments in a novel form, and scripts can't be written ahead to include these new ramifications.

Carbonell's POLITICS <Carbonell 78> analyzes opinionated text. But his system is given the underlying opinion (in the form of an ideology, represented as a set of goals). In our case, H assumes that the argument will conform to one ideology, but he must determine that ideology by examining the form of the argument. Furthermore, the main purpose of our analysis is distinct from Carbonell's: we are concerned with the overall form - why sentences are put together in a particular order. Carbonell concentrates more on analyzing individual events.

Allen <Allen 79> analyzes conversation as intentional behaviour. But again the goal of his system is distinct from ours. He is interested in recognizing speech acts; we know that the main purpose is to convince, but must determine how the form of the argument succeeds in convincing. Some of Allen's methods of plan deduction to uncover intentions may be useful to us.

In sum, our problem area presents us with a new language understanding task. We are concerned with determining form and uncovering intentions to perform analysis.

## 2. The Analysis Process

### 2.1 Overview

This section describes the basic procedure the hearer (H) follows to determine the logical form of an argument, leaving aside the issue of credibility.

The basic step in the analysis process is for H to take a sentence of the argument and to determine whether it is a new claim or evidence for some previously stated claim. In this way, H can uncover the intended function of each sentence. The basic unit of analysis is actually a proposition - the propositional content extracted from a sentence. (A simplifying assumption for our system right now is that the propositional content is made available).

To help H in classifying a proposition, there is a standard set of frames, representing rules of inference. In addition to frames representing correct rules like modus ponens and modus tollens, there are some representing bad logic, which is often used in arguments (either intentionally or in an attempt to justify bad evidence). Consider the following set of frames:

SET OF FRAMES:

(Abbreviations: MAJOR - major premise, MINOR - minor premise, CONC - conclusion)

| (correct) | MAJOR | MINOR | CONC |
|---|---|---|---|
| MODUS PONENS | A-->B | A | B |
| MODUS TOLLENS | A-->B | ~B | ~A |
| MODUS TOLLENDO PONENS | Aor~B | B | A |
| MODUS PONENDO TOLLENS | Aor B | B | ~A |
| (incorrect) | | | |
| ASSERTING CONSEQUENT | A-->B | B | A |
| DENYING ANTECEDENT | A-->B | ~A | ~B |

This selection of frames is motivated by <Sadock 77>, which indicates those correct and incorrect logical rules that occur most often in conversation. Our analysis of examples so far seems to function well with this restricted set.

For each of these frames representing rules of inference, it is often the case that they are not completely spelled out in the argument. Any one of the major premise, minor premise or conclusion may be omitted, and H must still be able to recognize the logical form intended, by filling in the missing detail. (This kind of argument is referred to as "modus brevis" in <Sadock 77>). H is aware of these variations in frames.

CLASSIFICATION OF FRAMES(e.g:Modus Ponens)

| | | |
|---|---|---|
| normal | A-->B,A | /B |
| normal MAJOR | A-->B | /B |
| normal MINOR | A | /B |
| MAJOR | A-->B | (assume rest) |
| MINOR | A | (assume rest) |
| CONC (hard) | B | (assume rest) |

How can H make use of these frames to represent the logical form of an argument? Consider MAJOR premise, MINOR premise, and CONCLUSION to be slots of a frame, with the constraint that the premise slots must lead to the conclusion. H is motivated in filling frames in order to classify propositions: we say that A is evidence for B iff they both fill slots in a frame such that A is a premise for B. H tries to instantiate a frame by filling its slots with propositions of the argument, possibly inferring premises that are not "spelled out", and thus choosing one of the "missing" versions of frames. The result is an indication of the logical relations between propositions in an argument.

## 2.2 Details

The overview illustrates the basic frame matching technique used to classify propositions. This section examines the analysis process in detail. In particular, a proposition may be classified in many different ways. We develop a scheme which formulates hypotheses for each proposition as to how it can fit with the rest of the argument, and then rates these hypotheses to determine the most likely interpretation. The rating scheme is based partly on fitting into our logic frames, and partly on other heuristics - e.g. based on the actual choice of words. In addition, this section describes the processing of the entire argument in more detail: how the classification of one proposition affects

another, how to isolate sub-arguments, and what kind of representation to build for the overall argument.

Rating Hypotheses

Consider the following classification scheme for a single proposition:

HYPOTHESES FOR CLASSIFICATION OF PROP(i)

(i) new claim
(ii) evidence for some future claim
(iii) evidence for PROP(i-1)
evidence for PROP(i-2)
...
evidence for PROP(1)

To illustrate that more than one hypothesis is probable for a given proposition, and that rating is thus necessary, consider the following example:

EX1: 1)There is too much crime in the city
2)We need more police

This example gives insight into the possible functions of a proposition, and the need to rate hypotheses. Consider 1) in isolation: it can be either a claim (and we'd expect evidence about the amount of crime) or evidence for some claim. Upon seeing 2), a connection is found between 1) and 2) (e.g. "more police --> less crime"), so 1) is interpreted as functioning as evidence for 2).

Determining whether a proposition is evidence for another is done by trying to fill slots in a frame, as described in Section 2.1. Since there are many frames in our system, each of the hypotheses in (iii) really represents a variety of options - e.g. evidence for PROP(i-1) by modus ponens, evidence for PROP(i-1) by modus tollens, etc. Since propositions are processed one at a time from the start, the only options that can be directly measured are those using propositions that have already been processed - hence the distinction between (ii) and (iii) above.

TRYING FRAMES

The first step is to determine the ratings for hypotheses in (iii) by actually trying to fit frames. To ensure that correct logic frames are given preference over bad logic frames, consider a frame system where the bad logic frames are connected to their correct logic counterparts using SIMILARITY links (as described by <Minsky 75>). In the spirit of <Tsotsos 80>, similarity links trigger alternatives when an exception is raised in trying to fill a slot in a frame. For example:

EX2: 1)Whenever the stock market crashes Carter refuses to appear on TV
2)Carter has refused to appear on TV
3)So the stock market must have crashed

With 1) and 2), modus ponens fails - we have A-->B, then B. So we try "asserting consequent", and with A asserted in 3), find that the bad logic frame succeeds. So bad logic frames are only tried when correct logic ones fail.

Each hypothesis in (iii) thus gets expanded into a list of options: one for each of the correct logic frames. Then, each option is tried. If frame constraints can't be satisfied, the option is given a very low rating. As H tries to instantiate frames, he must be aware that he is often interpreting beliefs of the speaker. So, for instance, modus ponens is usually recognized as: (S believes (A-->B)), (S believes A) thus (S believes B). (And not as "(A-->B) is believed to be true by H"...etc.). This introduces an interesting sub-topic of how H distinguishes beliefs, wants, and goals of S to aid analysis (see Section 3: Future Work).

Even when H succeeds in instantiating a frame, the rating for that frame may be lowered if it was "difficult" to fill in

the necessary "chains of reasoning". Consider the following:

EX3a: 1) There is a national railroad system in the US today
2) Railroads serve more than local needs

EX3b: 1) Railroads deliver goods across state lines
2) Railroads operate on a national scale

To determine 2) as evidence for 1) in 3a requires a rather long chain of reasoning (something like "exists national system --> operates on national scale --> carries goods between localities --> serves more than local needs"). In 3b, the chain is brief ("across state lines --> national"). H may wish to lower the ratings for longer, less certain chains.

In addition to actually measuring the options in (iii), we also consider some heuristics to affect ratings, which include an assessment for (i) and (ii).

LINGUISTIC CLUES

Sometimes H is aided in the classification of propositions by the actual choice of words. For example, H can recognize (and S knows that he will recognize) the organizational function of certain words and phrases, and can thus detect structure. For example, a classification like <Hobbs 78> is reasonable:

| CATEGORY | EXAMPLE |
|---|---|
| SUMMARY | in conclusion |
| PARALLEL | in the first place |
| EXAMPLE | for example |
| DETAIL | in particular |
| CONTRAST | on the other hand |
| CAUSAL | therefore |

The different functions can be interpreted in terms of claims and evidence for H to expect.

In addition, the choice of connective between propositions should provide H with an insight into S's intentions. Certainly, compound sentences suggest a common topic for their clauses - but the presuppositions attached to the words can indicate the function of each clause in the overall form. We are working on a precise description of connectives, and of particular constructions like analogy, with a view to developing a description for overall logical form that further specifies the evidence / claim distinction. It is also interesting to examine the motives of S in choosing a particular construction - an issue which relates more to "credibility".

SYSTEM RULES

Furthermore, there are heuristics called SYSTEM RULES to indicate preferred classifications. These are rules motivated by the intentional nature of arguments. H expects S to conform to certain coherence rules, because he knows S must be clear in order to convince H of his point of view. H thus has some defaults about how propositions relate. Consider the following:

1) a proposition which is a statistic is likely to be evidence

2) a proposition which quotes a particular authority is likely to be evidence

(based on the idea that claims are considered to be disputable, while statistics and quoting authorities are less disputable material)

3)a change in topic often signals a new claim

4) repetition of topic suggests some connection (propositions may support one or another, or both support a common third)

(judging continuity of topic)

5) rules of distance: prefer connections between propositions located closer

together in the argument

* when a proposition rates high as a new claim, strenghten the ratings of previous propositions which let them relate to each other (since it should be hard to find evidence located after this new train of thought)

something like the rule of distance applies for frame fitting:

* if the "chain of inference" needed to instantiate a frame is long, decrease the rating for that hypothesis

(other sub-rules based on distance may develop)

EX4: 1) Peter is a good musician
2) He has produced 50 songs this year

2) can be either:
*new claim
*evidence for future claim
*evidence for previous claim

Not only does 2) --> 1) fit into a frame (with chain "prolific --> good") but because of the statistic in 2), all evidence options are strengthened.

Updating ratings

So far we have described how H can do a thorough analysis of a single proposition. We now consider how the ratings for one proposition can affect others. Recall that our control structure is such that propositions are processed one at a time. Now, once a proposition is processed, the ratings of previous propositions are re-evaluated. Consider the following interactions:

(i) when PROP(i) is processed, go back and evaluate the "evidence for future claim" option for previous propositions, using PROP(i)

EX5: 1)Motorcycle gangs caused 50% of the deaths in our small town
2)These gangs are dangerous

None of the hypotheses for 1) can be directly measured since it is the first sentence. When 2) is processd, the hypothesis "2) as evidence for 1)" is measured and rates low (hard to establish). Then 1) is re-processed, and "1) as evidence for 2)" is measured. This works with modus ponens and missing premise "caused lots of deaths --> dangerous".

(ii) when PROP(i) rates high as evidence for PROP(j), increase the claim rating for PROP(j)

(iii) if a new proposition is created - i.e. filled in as missing detail in a frame - then this proposition is added to our system. Then, if a future proposition rates high as being related to one of these derived propositions, we increase the rating for the hypothesis which "created" it

Overall form

We now begin to have a feel for how rating propositions can propagate through an argument. We must as well try to develop a representation for the overall argument. What our classification of propositions has done so far is to indicate logical connections between sets of propositions. This, in fact, isolates sub-arguments, each with its own claim and set of supporting evidence. To complete the analysis, H must first of all determine the boundaries, where one sub-argument ends and another begins. Consider a methodology in the spirit of <Tsotsos 80>, strengthening hypotheses that indicate good "continuation" into a separate unit. Since a proposition can participate in more than one frame, the most likely option must be chosen. Some ideas on how to measure boundaries include using (i) change of topic (ii) ratings for "new claim" option (iii) combined ratings

for hypotheses indicating a unit: claim plus some evidence.

EX6: 1)Rogers is a talented songwriter
2)He has won 6 Junos
3)His son Peter has been active on Broadway for the past 15 years
4)Peter has won 4 Tony awards
5)Both Rogers and son are very talented

Here 2) rates high as evidence for 1), and 1) as a claim. Then 3) rates high as a new claim (new topic), so we strengthen ratings for 2) and 1) to consider them a separate unit.

Once sub-arguments have been isolated, using the hypotheses for propositions with the highest ratings, the overall argument structure can be analyzed. For now, this process is done at the end of processing. We try to relate all the individual claims from the sub-arguments into some overall form. Our final output is thus an indication of the most likely structure for the argument, in the form of relations between sub-arguments and forms of sub-arguments, represented as instantiated rules of inference.

Continuing with EX6:
4) and 3) share a common topic. 4) as evidence for 3) is possible, but with a weak inference. 3) and 4) both as evidence for a common claim rates high. Then, with 5) we reach the end and have to wrap up. 5) splits into 5a)Rogers is talented and 5b)Son is talented. We find 3) and 4) are evidence for 5b), and 1) is evidence for 5a). The most likely overall form is thus: two sub-args, one with 5a) as claim, with 1) as evidence (and 2) as evidence for it) and one with 5b) as claim and 3) and 4) as evidence.

Our next step will be to develop a more sophisticated control structure. Ideally, we will be analyzing overall form in parallel with our detection of sub-arguments.

## 3. Future Work

Section 2 presented a preliminary design for the analysis of arguments. In fact, most of the ideas for the design developed from an examination of many examples of arguments ( including a good selection from <Holmes and Gallagher 17>). The set of rules developed so far seem to function well, but are certainly not presented as an optimal solution.

We have already alluded to several areas that need more development, including: (1) more precise formulation of linguistic clues (2) more precise measure of frame fitting (3) more sophisticated overall control structure and (4) thorough description of the rating mechanism, including some actual figures to show comparative worth of different rules.

The major area for future work, however, is to develop mechanisms for H to determine and interpret the intentions of S. Issues of concern include: (i) recognizing and making use of control sentences - sentences about the structure of the argument (We have skipped discussion of this kind of sentence in this paper) (ii) distinguishing the wants, beliefs, and goals of S to aid in analysis (as alluded to before, H often recognizes relations between propositions not as logical truths but as beliefs of S. In addition, H may recognize wants, in particular with claims suggesting a "course of events". For example, when S says "There should be less war", H must recognize this as "S wants (less war)", and when S then says "Less war would mean more prosperity", H must recognize this as a belief of S, and through some logical reasoning conclude "S wants (more prosperity)".) (iii) determining the

motive behind S's choice of argument form and content - examining issues like order of presentation, choice of evidence, and deliberate deception. Furthermore, this investigation of intentionality should lead to some comments on credibility - the factors that H has available to influence his belief in the argument.

## 4. Conclusion

This paper presents some ideas for the design of a system to understand arguments. We give insights into how analysis can proceed: the rules available and the form of representation we want to build. We have only begun to look at the intentional aspect of arguments: how H can guide his analysis by his expectations about S. But we begin to see how this particular natural language task has restrictive characteristics which enable us to formulate specific methods of analysis: not only is there an overall form, but the speaker is forced to limit his choice of overall form so that the hearer can recognize all the points, and be convinced.

## Acknowledgements

I am grateful to Ray Perrault for his supervision of this research, and to Alex Borgida for his comments on this paper.

## Bibliography

<Allen 79> Allen, J.; "A Plan Based Approach to Speech Act Recognition"; U. of Toronto Dept. Comp. Sc. Tech. Rept. No. 131

<Carbonell 78> Carbonell, J.; "POLITICS: Automated Ideological Reasoning"; Cognitive Science 2,1

<DeJong 79> DeJong, G.; "Prediction and Substantiation: Two Processes that Comprise Understanding"; IJCAI 79

<Hobbs 78> Hobbs, J.; "Why is Discourse Coherent?"; SRI International Tech. Note No. 176

<Holmes and Gallagher 17> Holmes, H.W. and Gallagher, O.; Composition and Rhetoric

<Minsky 75> Minsky, M.; "A Framework for Representing Knowledge"; in The Psychology of Computer Vision, P. Winston, ed.

<Sadock 77> Sadock, J.; "Modus Brevis: The Truncated Argument"; in Papers from the 13th Regional Meeting, Chicago Linguistic Society

<Tsotsos 80> Tsotsos, J.; "A Framework for Visual Motion Understanding"; PhD Thesis, Dept. Comp. Sc., U. of Toronto

# Example Generation

Edwina L. Rissland

Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

Abstract

This paper addresses the problem of generating examples that meet specified properties which are used to direct and constrain the generation process, which we call CONSTRAINED EXAMPLE GENERATION. We begin by presenting a few examples of CEG taken from protocols. Based upon such examples, we present a model of the CEG process. We describe the architecture of a system that generates examples from specifications and present examples of problems that it has solved.

## 1. INTRODUCTION

The ability to generate examples that have specified properties is important in many intellectual areas, such as mathematics, linguistics and computer science [Collins 1979]. It is important from the standpoints of learning and teaching as well as performing research. For instance, examples are needed for inductive reasoning, sharpening of conjectures, and concept formation and refinement [Polya 1968, 1973; Lakatos 1963; Winston 1975; Lenat 1976; Soloway 1978]. Having a rich stock of examples is intimately related to understanding [Rissland 1978a, b]. Thus, examples lie at the heart of efforts to learn and reason in a subject.

When an example is called for, one can search through one's storehouse of known examples for an example that matches the properties of the desired example. If a satisfactory match is found, then the problem has been solved through retrieval.

However, when a match is not found, how does one proceed? In many cases, one modifies an existing example that is judged to be close to the desired example, or to have the potential for being modified to meet the constraints.

In some cases, generation through modification fails. Experienced researchers, teachers and learners do not give up however. Rather they switch to another mode of example generation which involves building up an example from very elementary consituents through careful attention to the desiderata and "unpacking" of the concepts involved. This phase of CEG is usually more difficult than either retrieval or modification.

This paper presents a model of CEG that incorporates three phases: RETRIEVAL, MODIFICATION, and CONSTRUCTION. This model is based upon analyses of protocols of example generation tasks taken from mathematics and computer science [Rissland 1979, Woolf and Soloway 1980].

## 2. PROTOCOLS OF CEG

In this section, we describe some protocols for CEG tasks taken from the domain of elementary function theory in mathematics (which deals with concepts such as continuity) and from elementary LISP programming (especially with regard to concepts concerning list structure).

### 2.1 Examples of Retrieval

The type of questions that most people answered through retrieval is:

Give an example of a function that is continuous
but not differentiable (at a point).

Give an example of a list
with three elements.

Most people handled these problems by offering their favorite standard "reference" examples [Rissland 1978a, b]: for the first problem, the absolute value function (at the origin) and for the second, a list like "(A B C)". Responses were usually immediate indicating that the retrieval was very readily made.

2.2 Examples of Modification

An example of a problem solved through modification of a known example is:

Give an example of a list
with three elements
where the depth of the first atom is 3.

Subjects frequently modified an example, such as "(A B C)" by adding two more parentheses around the first element, to produce the list

( ((A)) B C)

Other subjects truncated a longer list such as the list of digits or added to a shorter list such as (0 1), as well as adding parentheses. The example chosen for modification depends on the context of the problem (e.g., the sequence of recently solved problems) and the subject's data base of examples and its epistemology (e.g., his favorite references).

An example of a mathematics problem which every subject solved by modification is the following:

Give an example of a non-negative,
continuous function
defined on the entire real line
with the value 1000 at 1, and
with area under its curve less than 1/1000.

Most protocols for this question began with the subject selecting a function (usually, a familiar reference example function) and then modifying it to bring in into agreement with the specifications of the problem.

There were several clusters of responses according to the initial function selected and the stream of the modifications pursued. A typical protocol went as follows:

"Start with the function for a "normal distribution". Move it to the right so that it is centered over x=1. Now make it "skinny" by squeezing in the sides and stretching the top so that it hits the point (1, 1000)."

"I can make the area as small as I please by squeezing in the sides and feathering off the sides. But to demonstrate that the area is indeed less than 1/1000, I'll have to do an integration, which is going to be a bother."

"Hmmm. My candidate function is smoother than it need be: the problem asked only for continuity and not differentiability. So let me relax my example to be a "hat" function because I know how to find the areas of triangles. That is, make my function be a function with apex at (1, 1000) and with steeply sloping sides down to the x-axis a little bit on either side of of x=1, and 0 outside to the right and left. (This is OK, because you only asked for non-negative.) Again by squeezing, I can make the area under the function (i.e., the triangle's area) be as small as I please, and I'm done."

Comments

Notice the important use of such operations as "squeezing", "stretching" and "feathering", which are usually not included in the mathematical kit-bag since they lack formality, and descriptors such as "hat" and "apex". All subjects made heavy use of curve sketches and diagrams, and some used their hands to "kinesthetically" describe their functions. Thus the representations and techniques used are very rich.

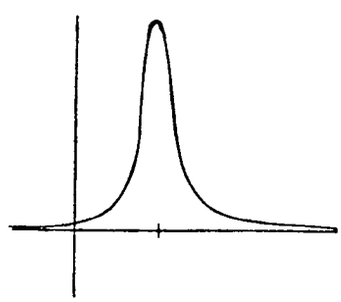


FIG 1A

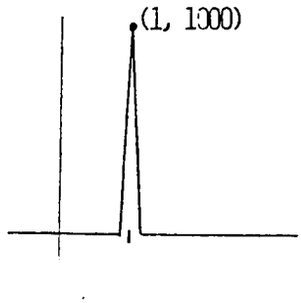



FIG 1B

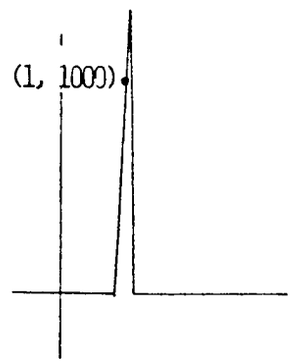



FIG 1C

Another thing observed in all the protocols (of which there were about two dozen for this problem) is that subjects make implicit assumptions about the symmetry of the function (i.e., about the line x=1) and its maximum (i.e., occuring at x=1 and being equal to 1000). There are no specifications about either of these properties in the problem statement; however, they are mathematically simplifying and cognitively natural.

These are the sort of tacit assumptions that Lakatos talks about [Lakatos 1963]; teasing them out is important to study both mathematics and cognition.

Example functions for protocols are shown in Figures 1a and 1b; another mathematically permissible example is shown in 1c.

## 2.3 An example of Construction

In this subsection, we present a protocol of example generation in which the example is built largely "from scratch" by working with the concepts involved in the specifications of the desiderata, instantiating them, and combining exemplars to produce a new example. The problem is:

> Give an example of a list of lists
> each of which has two elements
> the first of which is a literal atom.

A typical protocol began with the subject sketching out the overall structure of the desired list as:

```
( (A1 L1)
  (A2 L2)
  (A3 L3)
  ...
        )
```

where in each sublist, Ai stands for a literal atom, and Xi the second element.

The subject next focussed his attention on instantiating the Xi's. Since he wanted to emphasize the fact that the elements of the sublists could themselves be lists -- "there's a lot of embeddedness possible here" -- he made each of the Xi's a list of atoms (a "LAT").

The subject began to write each Xi as (Ai1 Ai2 ... Ain) and then remarked that this level of generality was more than the problem called for. In particular, nothing was said about keeping the Xi's different: "So, why not make them all the same, like (00 01)".

The candidate example now looks like:

```
( (A1 (00 01))
  (A2 (00 01))
  (A3 ...

                )
```

The subject next decided to pin down the length of the "big" list by making it be "not too short, like 2, and not too long either; why not 7". He tended to the Ai's by noting that A1, A2, A3, ... A7 are perfectly fine literal atoms.

The list thus offered is:

```
( (A1 (00 01))
  (A2 (00 01))
  (A3 ...
  ...
  (A7 (00 01)) )
```

Even though the subject was satisfied with this answer, he noted that it really didn't have to be so complex or long; the following list would do:

```
( (A1 1) (A2 2) (A3 3) )
```

He said he made his list have a length longer than 2 because he didn't want it to be confused with the length of the sublists (i.e., 2). However, he said that a list of length two would be acceptable, but a list of length one would not since "after all the problem called for a list of lists".

"The list:

```
((A B) (A B))
```

would also do just fine. In fact, the possibilities are endless."

### Comments

There are several observations to be made on this protocol. First, the subject had a general model of a list and procedures to instantiate it (e.g., generate literal atoms and lists) and he had procedures to modify lists and properties of lists. Second, the subject made several implicit assumptions on the example to be generated, such as (1) its length, (2) the non-repeatedness of some elements, (3) its complexity (e.g., depth), and (4) uniformity (e.g., of list-structure).

## 3. A CEG MODEL

From analyses of protocols such as presented in Section 2, we developed the following general model of the CEG process. Presented with a task of generating an example that meets specified constraints, one:

> (1) SEARCHES for and (possibly) RETRIEVES examples satisfying the constraints. This is done by searching through the knowledge base and judging examples for their match (or partial match) to the desiderata;
>
> (2) MODIFIES an existing example judged to be close or having the potential for fulfilling the desiderata;
>
> (3) CONSTRUCTS an example from elementary knowledge, such as definitions, principles and more elementary examples from the knowledge base.

Thus, there is a spectrum of responses to a CEG task ranging from having a ready answer as in (1) to having no especially close fitting candidate as in (3). In general, Task N depends on and follows Task N-1.

This information processing model of CEG is useful not only in describing human protocols, but also precisely specifying a computational model.

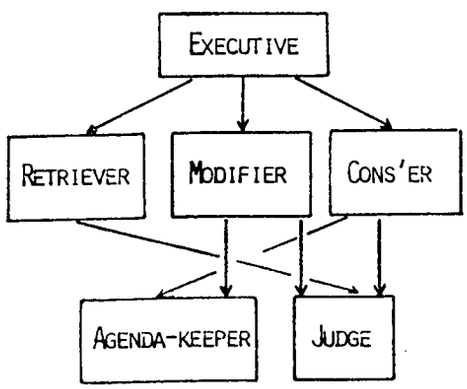



FIG 2

## 4. ARCHITECTURE OF A CEG SYSTEM

From the model of the last section, we have developed a system that solves CEG problems in the LISP domain. It has also been used to hand-simulate CEG problems in the mathematics of linear and piece-wise linear functions.

We have implemented this CEG model in the LISP domain. Written in LISP, it currently runs interpretively on a VAX 11/780 running under VMS. Examples of problems and solutions are given in Section 6.

The knowledge in our CEG system resides in two major sources: the knowledge base upon which the system runs, and the knowledge embedded in the processes operating on that base. The knowledge consists of general epistemological knowledge (e.g., the structure and types of examples) and domain-specific knowledge (e.g., particular example modification techniques).

The system consists of several components -- roughly one for each of the three phases of the model -- which handle different aspects of CEG. The flow of control between the components is directed by an EXECUTIVE procedure. Figure 2 shows the general architecture of our system.

The components use a common knowledge base which consists of two parts: (1) a "permanent" knowledge base of "Representation-spaces" [Rissland 1978]; and (2) "temporary" knowledge generated during the solution of a CEG problem.

There are four representation spaces, each of which is a set of items, represented as frame-like data structures, and organized according to predecessor-successor relationships. Examples-space, which is by far the most heavily used in our current system, consists of known examples organized according to the relation of constructional derivation reflecting which examples are constructed from which others. The other spaces and their relations are: Concepts-space: definitional dependency; Results-space: logical dependency; and Procedures-space: procedural dependency.

Before the system is given any CEG problems to work on, we create an initial set of representation spaces. The initial state of the Examples-space for the set of problems described in this paper is shown in Figure 3. The spaces are modified -- mostly through the

addition of examples to Examples-space -- as the system works through CEG problems.

The temporary knowledge held by the system during a CEG problem run includes a list of the constraints of the problem, an agenda of candidate examples, and various bookkeeping parameters such as "boxscores", "constraint satisfaction counts" and "recency counts".

## 5. CEG SYSTEM COMPONENTS

(1) The EXECUTIVE is responsible for initializing the system for a CEG problem, directing the flow of control among the components, and cleaning up afterwards. It accepts a CEG problem in prescribed format from the user and sets up the problem specifications in the temporary knowledge base.

The problem desiderata are kept on the CONSTRAINT-LIST, which has as many entries as there are constraints. Each constraint is recorded as a pair of properties DESIRED-PROPERTY and DESIRED-VALUE. For instance, the specification of the three constraint problem of "a list, of length 3, where the depth of the first atom is 1" is recorded by the following properties (PLIST's) for the constraints:

```
CONSTRAINT-1    DESIRED-PROP: (TYPE X)
                DESIRED-VALUE: LIST

CONSTRAINT-2    DESIRED-PROP: (LENGTH X)
                DESIRED-VALUE: 3

CONSTRAINT-3    DESIRED-PROP: (DEPTH
                    (FIRST-ATOM X) X)
                DESIRED-VALUE: 1

                Problem 1
```

The EXECUTIVE dictates the behavior of the system as a whole by specifying the orderings used by the other processes, such as the order of retrieval of candidate examples used by the RETRIEVER and the order of application for modification techniques used by the MODIFIER.

(2) The RETRIEVER searches the knowledge base for examples on request from the EXECUTIVE. It searches through Examples-space by examining examples in an order specified in terms of attributes such as "epistemological class" [Rissland 1978], position in the Examples-graph, and recency of creation.

In the problems described in Section 6, the "retrieval order" used was:

```
reference examples before
counter-examples before
start-up examples before
examples without epistemological
        class attribute
```

and in the case of ties

```
predecessors before
successors.
```

This retrieval order biases the system to examine ubiquitous and earlier-contructed examples before others. The order of CANDIDATES retrieved from the initial Examples-space of Figure 3 is thus:

(A B C)

(0 1 2 3 4 5 6 7 8 9)

(0 1)

( )

(A)

(A B C D E)

With each new example selected, the RETRIEVER calls the JUDGE to evaluate the example to ascertain how well it satisfies the desiderata.

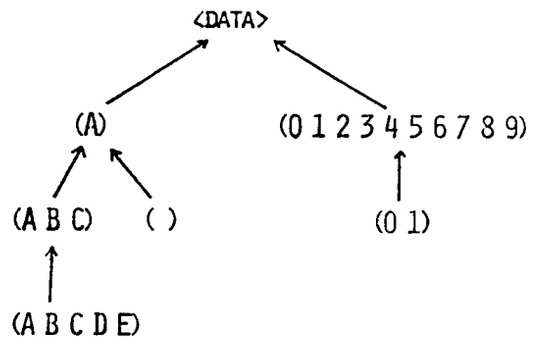



FIG 3

(3) The JUDGE evaluates a CANDIDATE example by cycling through all of the DESIRED-PROPERTY/DESIRED-VALUE pairs on the CONSTRAINT-LIST, comparing them with the actual properties of the CANDIDATE, and recording the results of the comparison. Thus, the JUDGE's basic cycle is evaluation, comparison and record.

The JUDGE records the results of the comparison by FILLING-IN the BOX-SCORE and the CONSTRAINT-SATISFACTION-COUNT ("CSC") slots in the representation frame of the CANDIDATE. The CSC is simply the number of desiderata met by the CANDIDATE.

The BOX-SCORE is a list of 2-tuples, one for each constraint, of the form (ACTUAL-VALUE, T or NIL). The ACTUAL-VALUE is the CANDIDATE's value for the DESIRED-PROPERTY; T is entered if the ACTUAL-VALUE equals the DESIRED-VALUE, and NIL if not.

The BOX-SCORE for the example "(A B C)" in Problem 1 would be:

( (LAT T) (3 T) (1 T) )

The CSC for this example would be 3, that is, all the constraints are met; the success of this example would be recorded as a T in its "SF" (SUCCESS/FAILURE) slot. With the above retrieval order on the Examples-space of Figure 3, Problem 1 would be solved with the first example retrieved.

If the example "(A)" were judged, its BOX-SCORE would be:

( (LAT T) (1 NIL) (1 T) )

The CSC for this example would be 2.

(4) The MODIFIER is invoked by the EXECUTIVE when the RETRIEVER has been unable to find an example meeting the constraints from its search through Examples-space.

The MODIFIER calls the AGENDA-KEEPER to set up an agenda of examples to be modified. The MODIFIER then works down the AGENDA trying to modify each entry in turn until success is achieved or the agenda exhausted.

To modify an example, the MODIFIER examines its BOX-SCORE for the constraints that were unsatisfied. It calculates the DIFFERENCE between the DESIRED-VALUE and the ACTUAL-VALUE for each DESIRED-PROPERTY not satisfied. Using the DESIRED-PROPERTY and the DIFFERENCE as an index to a difference-reducing table, the MODIFIER's DIFFERENCE-REDUCER finds and then applies modification techniques to the example.

For instance, for the example "(A)" with a CSC of 2 for Problem 1, the property not met is that of having a length equal to 3. The DIFFERENCE between the DESIRED- and ACTUAL-VALUE is +2. The difference-reducing technique MAKE-LONGER is found by looking for modification techniques affecting the LENGTH attribute of a list and reducing the DIFFERENCE, i.e., by making it longer by 2. (If the difference were -2, as would be the case for the example "(A B C D E)", the appropriate technique would be MAKE-SHORTER).

When there is more than one unsatisfied constraint, the MODIFIER orders its modification attempts according to the order specified by the EXECUTIVE. For the sample problems of this paper, the modification order is to apply techniques that affect:

TYPE before
LENGTH before
DEPTH before
GROUPING

The modified example is then re-judged and a new BOX-SCORE and CSC calculated. If the CSC is improved, the MODIFIER prints a message to the user of "success" or "failed" and asks whether it should continue modifying this example by going through another difference analysis, difference reduction, judgement cycle. If the CSC goes down, the MODIFIER abandons its attempt to bring the example into line, goes on to the next example on the AGENDA, and does not re-queue the example. Thus the MODIFIER engages in a form of hill-climbing.

The modified example must be re-judged for two reasons: (1) some techniques are heuristic and do not guarantee successful modification; and (2) there can be interaction between the constraints, that is, a successful modification for one constraint may undo satisfaction of another.

For instance, the system can make a NESTED-LIST from the LAT "(A B C)" by GROUPing "A" and "B", i.e., "( (A B) C)". However, before the modification technique was applied the LENGTH was 3, but now, after modification, it is 2. Satisfaction of the NESTED-LIST constraint has undone the LENGTH 3 constraint.

In the next version of our system, we shall re-judge an example after each modification, and also protect some contraints.

(5) The AGENDA-KEEPER is called by the MODIFIER and CONS'ER to set up the AGENDA of examples to be modified or instantiated.

When called by the MODIFIER, the AGENDA-KEEPER compiles an agenda of items to be modified based upon the CSC's calculated and recorded during the retrieval phase: the examples are ranked in order of their CSC's. Thus, the CSC is used as a measurement of the closeness of the example to meeting the constraints. In the case of a tie, the retrieval ordering is used.

(6) The CONS'ER is called by the EXECUTIVE when the MODIFIER is unsuccessful in its attempts to produce a solution or a model needs to be instantiated. The CONS'ER uses the procedural formulations of concepts stored in Concepts-space.

## 6. SAMPLE PROBLEMS

[NOTE: Text in this section is actual computer output generated by our CEG system; however explanatory text has been added (indicated by a "$") and some output modified to improve readability.]

### Problem 2

$Problem 2 asks for a list of length 3 whose first atom has a depth of 3:

```
(x1 (desired-value list desired-prop
 (typep candidate)))
(x2 (desired-value 3 desired-prop (length
 candidate)))
(x3 (desired-value 3 desired-prop (depth
 (first-atom candidate) candidate)))
```

$The retrieval phase is entered with the Examples-space of Figure 2. The retrieval order of candidates is:

<abc>
<**digits>
<**bits>
<empty>
<a>
<abcde>

$The RETRIEVER reports on each candidate tried, by printing out its BOXSCORE, CSC and SF:

```
   candidate name = <abc>
 candidate-value = (a b c)
     csc = 2   sf = nil
(entry-x1 (lat t))
(entry-x2 (3 t))
(entry-x3 (1 nil))
"failed"

   candidate name = <**digits>
 candidate-value = (0 1 2 3 4 5 6 7 8 9)
     csc = 1   sf = nil
(entry-x1 (lat t))
(entry-x2 (10 nil))
(entry-x3 (1 nil))
"failed"

   candidate name = <**bits>
 candidate-value = (0 1)
     csc = 1   sf = nil
(entry-x1 (lat t))
(entry-x2 (2 nil))
(entry-x3 (1 nil))
"failed"

   candidate name = <empty>
 candidate-value = nil
     csc = 0   sf = nil
(entry-x1 (atom nil))
(entry-x2 (0 nil))
(entry-x3 (0 nil))
"failed"

   candidate name = <a>
 candidate-value = (a)
     csc = 1   sf = nil
(entry-x1 (lat t))
(entry-x2 (1 nil))
(entry-x3 (1 nil))
"failed"

   candidate name = <abcde>
 candidate-value = (a b c d e)
     csc = 1   sf = nil
(entry-x1 (lat t))
(entry-x2 (5 nil))
(entry-x3 (1 nil))
"failed"
```

$The problem desiderata are not met by any example in the data base, and thus the modification phase is entered.

$The AGENDA of candidates for modification is (the CSC is given after the candidate's name):

(<abc> 2)
(<**bits> 1)
(<a> 1)
(<**digits> 1)
(<abcde> 1)
(<empty> 0)

```
$The MODIFIER goes to work on the first
candidate, (A B C):

-----------------------------------------
constraint = ((typep candidate) list)
actual score = (entry-x1 (lat t))

modify-candidate   ok

-----------------------------------------
constraint = ((length candidate) 3)
actual score = (entry-x2 (3 t))

modify-candidate   ok

-----------------------------------------
constraint = ((depth (first-atom
 candidate) candidate) 2)
actual score = (entry-x3 (1 nil))

  "find-diff"   (increase-depth-by 2)
  "apply-diff"
      reducer =  make-deeper-x
 new-candidate =  (((a)) b c)

modify-candidate   modified

-----------------------------------------

$The candidate's depth attribute has been
modified by the modification routine
MAKE-DEEPER-X to produce a new example,
which is then judged and added to the
Examples-space:

   candidate value =  (((a)) b c)
     csc =  3   sf =  t
(entry-x1 (nlist t))
(entry-x2 (3 t))
(entry-x3 (3 t))
("created new frame for example "
 mar11-009 (((a)) b c))
"success!!"
```

Problem 3

```
$The CONSTRAINT-LIST for the next problem
is:

(x1 (desired-value list
     desired-prop (typep candidate)))
(x2 (desired-value 5
     desired-prop (length candidate)))
(x3 (desired-value 2
     desired-prop (depth
         (first-atom candidate)
 candidate)))
(x4 (desired-value 3
     desired-prop (depth
         (first-atom (cdr candidate))
         candidate)))
```

```
$The order of candidates retrieved and
judged is:

         <abc>
         <**digits>
         <**bits>
         <empty>
         <a>
         <abcde>
         mar11-009

$Since no example meets the constraints,
the modification phase is entered with
the following AGENDA:

         (<abcde> 2)
         (mar11-009 2)
         (<**bits> 1)
         (<a> 1)
         (<**digits> 1)
         (<abc> 1)
         (<empty> 0)

$The MODIFIER sets to work on the first
candidate (A B C D E):

-----------------------------------------
constraint = ((typep candidate) list)
actual score = (entry-x1 (lat t))

modify-candidate   ok

-----------------------------------------
constraint = ((length candidate) 5)
actual score = (entry-x2 (5 t))

modify-candidate   ok

-----------------------------------------
constraint = ((depth (first-atom
 candidate) candidate) 2)
actual score = (entry-x3 (1 nil))

  "find-diff"   (increase-depth-by 1)
  "apply-diff"
      reducer =  make-deeper-x
 new-candidate =  ((a) b c d e)

modify-candidate   modified

-----------------------------------------
constraint = ((depth (first-atom (cdr
 candidate)) candidate) 3)
actual score = (entry-x4 (1 nil))

  "find-diff"   (increase-depth-by 2)
  "apply-diff"
      reducer =  make-deeper-x
 new-candidate =  ((a) ((b)) c d e)

modify-candidate   modified

-----------------------------------------
   candidate value =  ((a) ((b)) c d e)
     csc =  4   sf =  t
(entry-x1 (nlist t))
(entry-x2 (5 t))
(entry-x3 (2 t))
(entry-x4 (3 t))
```

$The modification is successful and the new example is added to the Examples-space.

("created new frame for example " mar11-011 ((a) ((b)) c d e)) "success!!"

The Examples-space after the successful solution of Problems 2 and 3 is shown in Figure 4.

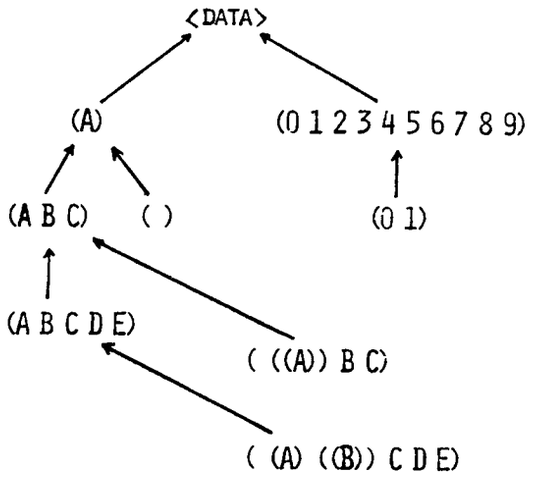



FIG 4

## 7. CONCLUSIONS

In this paper we have described a computer system that models Constrained Example Generation ("CEG") in domains from computer science and mathematics. We described how the CEG system generates examples of data in LISP.

We are currently using the system to explore issues such as

1. the effect of the initial contents of Example-space and the sequence of solved problems on the evolution of Examples-space;

2. the effect of alternative orderings on the retrieval and modification processes;

3. the effect of interacting constraints, e.g., impossible constraints.

We also plan to use our system to study machine learning by the incorporation of adaptive techniques, e.g., by keeping track of the performance of various orderings and techniques and choosing the ones that perform best. Such extensions of our system will enable it to "learn" from its own experience.

## References


Collins, A. and A. Stevens (1979) Goals and Strategies of Effective Teachers Bolt Beranek and Newman, Inc., Cambridge, Mass.

Friedman, D. (1974) The Little LISPer, Science Research Associates, Menlo Park, Calif.

Lakatos, I. (1963) Proofs and Refutations, British Journal for the Philosophy of Science, Vol. 19, May 1963. Also published by Cambridge University Press, London, 1976.

Lenat, D.B. (1976) An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search, Stanford Univ. Artificial Intelligence Memo 286.

Polya, G. (1968) Mathematics and Plausible Reasoning, Volumes I and II, Second Edition, Princeton Univ. Press, N.J.

Rissland (Michener), E. (1978a) Understanding Understanding Mathematics, Cognitive Science, Vol. 2, No. 4.

Rissland (Michener), E. (1978b) The Structure of Mathematical Knowledge, Technical Report No. 472, M.I.T Artificial Intelligence Lab, Cambridge.

Rissland, E. (1979) Protocols of Example Generation, internal report, M.I.T., Cambridge.

Soloway, E. (1978) "Learning = Interpretation + Generalization"; A Case Study in Knowledge-Directed Learning. Univ. of Massachusetts, COINS Technical Report 78-13, Amherst.

Winston, P. (1975) Learning Structural Descriptions from Examples, in The Psychology of Computer Vision, P. Winston (Ed.), McGraw-Hill, New York.

Woolf, B., and Soloway, E. (1980) Analysis of Student Protocals: Misconceptions in Understanding Programming in LISP, in preparation.


## Acknowledgments


Special thanks to my colleagues Elliot M. Soloway, for his invaluable assistance in bringing up the CEG system, and Oliver G. Selfridge, for many useful discussions and critiques.

# An Adaptive Sorting Program

Oliver G. Selfridge
Valerie I. Congdon
Stephanie R. Davis

Computer and Information Science
University of Massachusetts
Amherst, Massachusetts

ABSTRACT

This paper discusses the design, construction, and use of an adaptive sorting program, which selects and tunes the sorting algorithm according to its recent experience with the algorithms available to it. That is, the program adapts its behavior to try and minimize a cost function specified by the users.

The point of this exercise is to explore ways in which the computer program can carry some of the responsibility of optimizing its performance, instead of relying on a user to set rigid specifications. The purpose of choosing a good sorting algorithm is to minimize some kind of cost; the cost function used here is "virtual CPU time," computed by the program; we use that instead of measuring real CPU time because of the difficulties and unreliabilities of measuring it in a time-sharing environment.

Some of the adaptive programs discussed here perform better on some populations of lists than the standard workhorses found in many computer centers.

## 1.0 Introduction and Overview

There are a number of different algorithms that can be used to sort lists. Each has advantages and disadvantages that depend on the nature of the lists. This paper discusses an adaptive sorter, which selects the particular algorithm it uses according to the efficiency it has previously found. The sorter restricts itself to three algorithms: Straight Insertion, MergeSort, and QuickSort. It should be noted that each has operating parameters that have to be set before it can be considered well specified.

The task domain is the selection of algorithms to sort lists. For this paper, the lists were generated by a program. For some of the exercises, the characteristics of the lists changed slowly in time -- that is, there is not merely a single optimum sorting algorithm that is to be searched for, and, once found, maintained. Rather, the program must be capable of changing its selections as its environment changes.

In Artificial Intelligence (AI), there is another approach to this problem, namely the use of experts: find out from them the rules they follow, the diagnostics they use, and so on, and design those rules into a program. Such a technique has been very successful in some cases.(1) But sometimes the domains are too large to have rules of the necessary precision; or the rules seem to involve human judgment in a profound way, as if in artistic selection; or the best experts are just not very good; and so on. Sometimes the criteria for an "optimum" change -- in our example, the evaluation or cost function might be changed to include some measure of the cost of storage. That is, our underlying interest here is the use of adaptive techniques where there may be no experts, or where the problems may be too hard to understand or even to state. We use sorting as a domain, where there are experts, so that we can compare the efficiency of the adaptive techniques with that of the experts.

Our purpose here is to see to what extent we can give the program the responsibility of choosing the "optimum" algorithm, by providing it the experience of trying several algorithms and remembering how

(1) For example, see Feigenbaum and Lederberg (1974), with Dendral and Metadendral.

they performed (with respect to the cost function that defines the optimum).

Any program that can track a moving optimum must spend some extra effort deciding what the current optimum is, just as the experts spend effort on their diagnostic analyses. Our program is constantly checking the current best algorithm against its competitors; it does so less often when, whenever it does, the comparison is very one-sided, and more often when it isn't.

In general it is not possible to compare two algorithms with just two trials, because each has parameters that must be set for it to do best, so that the ideal settings have to be tracked. In the first set of experiments, however, we set the parameters by hand. A second set tries to optimize the parameter settings as well. A third set tries to find a good combination of useful diagnostics to help the program determine a good threshold function to help make a good selection of algorithm.

The general adaptive approach that the program exemplifies is discussed at length, considering especially its inherent limitations and the underlying assumptions about its application. For many domains of AI, we suggest that AI cannot afford the time spent to tap experts, and ought to try giving the program some responsibility in improving its behavior; that would be even truer if the experts were scarce or not very good. This program is, we hope, a beginning exploration of ways to do that.

At least one of the programs shown here seems to do better on the average than some of the workhorses used at computer installations; while we cannot be certain that it would be profitable to substitute them, it would certainly be worthwhile to check them with a broader range of sorting problems and algorithms, perhaps referring to human interaction, and to the expert work of, for example, Knuth (1974).

An example of the kind of problem that might be susceptible to the approach is the scheduler on a time-sharing system, where it is hard to decide what is the best way to satisfy user requirements, especially as they and the system change in unknown and unpredictable ways; another example is the control of a communication net with large swings in the volume and nature of the traffic, subject to changing priorities and channel capacities.

We use three sorting algorithms, Straight Insertion, Mergesort, and QuickSort, each of which is good for some applications, and which are described in section 2. The adaptive techniques are not particularly subtle, and are described in section 3. The structure and functions of the program as a whole are described in section 4, and section 5 presents the results of the several experiments. The final section discusses the results and draws conclusions from them.

## 2.0 Background: Sorting

Sorting is the task of arranging items in some desired order, like alphabetical. It was one of the first tasks assigned to automatic data processing machines, and was one of the first such problems to be thoroughly analyzed.

We chose sorting as a task domain for several reasons:

1. It is a well-known problem with several commonly used algorithms, whose relative costs vary widely with the sorting problems presented to them.

2. Sorting algorithms have been thoroughly analyzed, so that experts can be reasonably sure about the rules they follow; in that way the performance of the program can be properly evaluated.

3. We felt that it was likely that if the program worked as well as we hoped, it could lead to profitable improvements over some of the standard work horses now being used in computer installations.

The program deals with 3 sorting algorithms: Straight Insertion (I), MergeSort (M), and QuickSort (Q). These are described in the following subsections.

### 2.1 STRAIGHT INSERTION (I)

Insertion adds items one at a time to a previously ordered list. Since each insertion leaves the list ordered, a list n long takes merely (n-1) insertions. In the worst case, I makes i-1 comparisons to insert the ith item, so that the number of comparisons is O(n*n). Note that if the list is highly ordered to start with, then each insertion may be done with very few

comparisons. It is also usually a fast method for lists with few items, say, fewer than 15.(1) The program itself is short and easy to understand. Some other sorting methods may use it with short lists or sublists.

### 2.2 MERGESORT (M)

Merging is the technique of combining lists (in this case, two) by sequentially comparing the first elements of sublists, and moving them in the correct order into the merged list. M combines pairs of single elements into sublists, then the pairs themselves, and so on, each time dealing with sublists twice as large, until the process terminates.

M is an efficient sort, and is currently the system choice at the computer center at UMASS. It does, however, require a lot of storage. In running time, it takes O(n*log n). Its worst case is never much worse than that average.

### 2.3 QUICKSORT (Q)

Q is on the average the best sorting algorithm according to the experts.(2) Q is based on the notion that exchanges of items should be made over large distances in order to minimize the number of exchanges -- it is the diametric opposite of bubble sort, for example, which exchanges items out of order only with their neighbors. Q makes an arbitrary partition of the list into two parts, comparing items from both parts, and interchanging them when needed.

The running time for Q is on the average O(n*log n), but its disadvantage is that its worst case performance can be O(n*n), which is not true of M.

## 3.0 Background: Adaptation and Computer Learning

There is a long and rich history of attempts to make the computer (program) learn in the sense that children learn and grow. Nearly twenty years ago, there was a great deal of interest in 'self-organization,' by which the computer was to organize its data and restructure itself so as to perform better. Some of us remember the perceptron of the late Frank Rosenblatt(1) and its numerous companions. Recently such activity has waned, perhaps because of a more or less conscious decision by the AI researchers that it was not very productive. One of the questions that we raise here is whether the simple control mechanisms that we discuss can be applied to hard problems so as to make a beginning of an attack on the larger area of learning by computers.

The advantages of adaptation and learning, if any, are not had for nothing: it will always cost extra resources to make checks on the efficiency of the particular algorithms being used.

One method of improving a strategy is to try small changes in it, observing the changes in performance. If they are positive, continue to make such changes; if negative, undo them, and try other ones. This is generally known as hill-climbing, and it has a venerable history. Much of the power and difficulty of hill-climbing depends on the particular representation of the strategy, so that the changes are in some way related to the changes in performance. Indeed, AI researchers have long considered that the problem of finding good representation is one of the truly central ones in AI.(1)

## 4.0 The Adaptive Sorter

We deal with three different kinds of adaptive mechanisms. In the first, the program is given the cost of the algorithm when it tries it; its goal is to minimize the total cost of a long series of sorting problems. In order to make sure that the algorithm it is using is the best (that is, the cheapest), it must occasionally try the other algorithms, which costs it more resources. The underlying assumption behind this strategy is that the sorting problems in the series vary their characteristics only slowly, so that the best algorithm stays best for some long time.

---

(1) See Knuth (1974).
(2) Ibid, and Wirth (1976).

(1) For example, see Rosenblatt (1960). For the best discussion of perceptrons, see Minsky and Papert (1969).

(1) See Winston (1976) for a good general discussion.

The characteristics of the lists to be sorted that are relevant here are two: the length L, and the degree of randomness R. The latter is interesting -- Q, for example, takes nearly as long to sort a list that is already sorted as to sort one that is in random order. Straight insertion, on the other hand, takes but L - 1 comparisons to establish that a list is well-ordered.

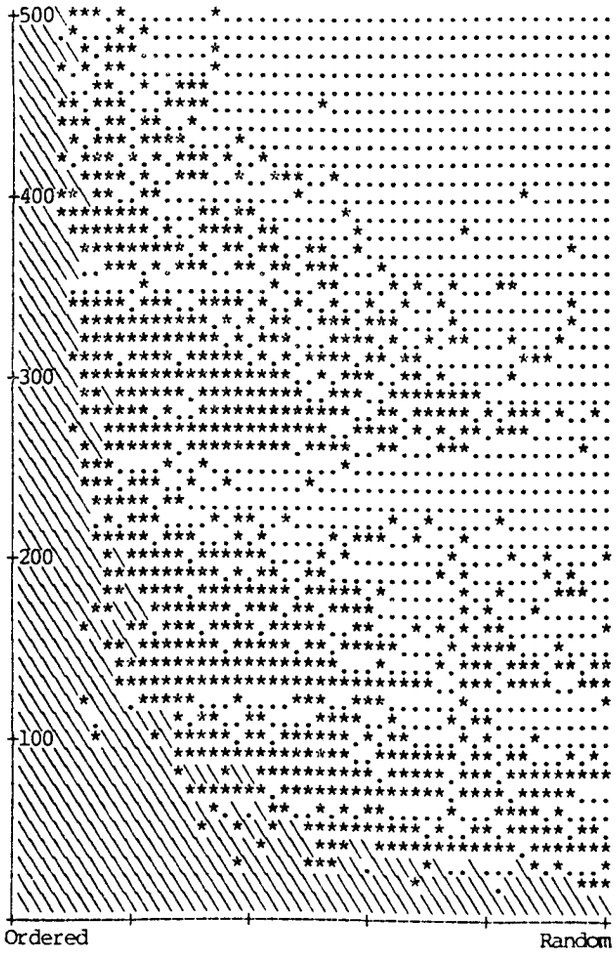



Figure 4.1 Optimum Method for Sorting
X-axis: Randomness R
Y-axis: Length L

\ : I . : M * : Q

Figure 4.1 is a length-randomness plot, with the length shown vertically, and randomness horizontally. The figure shows the regions where the three algorithms are superior to the others. The figure was constructed by generating 2500 lists using a random number generator. In general, small lengths or low randomness suggest I; M and Q are in fact fairly close over the remaining region, and the superiority of one over the other is usually but slight. It is possible to observe the extra time taken by M as the length rises above each power of 2. Clearly, if the program could detect the best method cheaply enough, perhaps it could make significant savings in resources.

In the second kind of adaptation, the program takes advantage of knowing, either by experiment or by our having told it, the contents of figure 4.1. The program's task is then to make good estimates of length and randomness. In our sorting, the length is provided as a given with the submission of the list; the question is then how to estimate the randomness cheaply enough to make it profitable.

A third experiment in adaptation selects a good simple combining function of L and R.

There is a kind of zeroth order adaptation, in which the best overall method is used consistently; given the population of lists that we presented to the machine, and presuming a uniform distribution of lists over the variables L and R, that method was in fact M. the first job of any adaptive scheme, obviously, must be to do better than M.

## 5.0 Experiments and Results

We ran a number of experiments with some interesting results. The first task was to generate the lists; how we did that is described in section 5.1 below. Each list was in fact evaluated separately with each of the three methods, and what was presented to the program was merely the characteristics of each list and the resources it would take according to the three methods. In that way, it was possible to compare different adaptive strategies without actually running the algorithms over and over again, which would have consumed considerable computer time.

### 5.1 Generation of Lists

The lists had two controllable attributes -- length L and randomness R. The value R set the fraction of the list that was constructed at random, the other elements being generated in order. For example, for a list 100 long, the ith element defaults

to just i itself, for R = 0, the perfectly ordered case. If R is 0.5, then in exactly half the elements, the value chosen is just 100 times a random number uniformly distributed between 0 and 1.

Note that lists can be generated with **negative** ordering, that is, backwards, but we did not use such lists in our experiments.

For each experiment we generated lists that form the basis of the data used in the adaptation. Each list was then sorted by all the three methods. The data is contained in an array whose columns were:

1. The length of the list
2. The randomness of the list
3. How long it took to sort with I
4. How long it took to sort with Q
5. How long it took to sort with M

For each test of the adaptive strategy, we computed the costs by merely referring to the array, instead of generating lists and sorting them. In this way the individual adaptive run could be tried with very little CPU time.

Inspection of figure 4.1, generated in this way, will reveal a certain noisiness in the data. That arises from the use of the random number generator in making the lists.

## 5.2 First Adaptive Scheme

In the first scheme, the program initially tries the three algorithms, and then continues by using only the best one. Best is defined by an estimate of the CPU time used by the algorithms. The program checks the validity of its choice by trying the other ones occasionally; if one of the other algorithms proves to be shorter, the program switches. The underlying assumption for this scheme is that the attributes of the lists do not change very fast.

In fact, the lists were selected from a population whose characteristics followed the trajectory shown in figure 5.1, which covered 1000 lists. Note, by comparing that figure with figure 4.1, that the trajectory runs through all three regions where the different algorithms were optimum.

The program worked by computing the cost of the preferred sorting algorithm. Some fraction of the time, it also tested by trying the other algorithms; that meant adding their costs to the costs already incurred. The cost of the tests on the non-preferred algorithms was minimized by ceasing the test whenever its cost exceeded that of the preferred one.

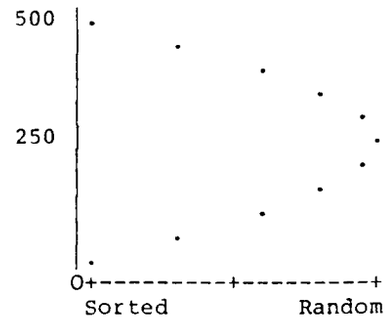



Fig. 5.1 Trajectory for a Population of Lists

The behavior of the program is about equal to the best single algorithm, M. In this case, then, adaptation does not provide any real profit. The extra expense of making checks causes the program not to outperform the best single algorithm in a significant way.

This kind of scheme is efficient only when the population characteristics change slowly, of course. In that sense, it is unrealistic to expect that it can provide a vast improvement in sorting efficiency in an operational environment. By analogy, however, we might hope that such a scheme applied to tasks harder to analyze, like certain kinds of scheduling in time-sharing systems, could lead to really worthwhile savings.

## 5.3 Second Adaptive Scheme

The second scheme used the data shown in figure 4.1. The program has but to know where on the figure the new list is, and it can choose the best sorting method. The length L is given; since the program is not given R, it has to estimate it. The adaptation here is how much resources to put into estimating R. If the whole list is examined, that represents a fairly massive expense; and if, say, but ten items are examined, then there is a fairly large

chance of making a bad estimate. The parameters of the procedure for estimating R are tuned so as to minimize the cost. The difficulty with such a scheme is the length of time needed to be sure of the results of tuning. It is clearly cheaper to use short samples, except that then the probability of picking the wrong selection by chance increases; that may lead to the selection of a much more expensive algorithm.

We decided that the parameter of estimation that ought to be optimized was the fractional size X of the sample that gave the estimate of R. That is, if X were 0.10 then a list of length 500 would be tested for R with a sample of length 500*0.10=50. The estimation does not pretend to be an accurate measure of randomness, which is in any case undefined except insofar as it is defined by the generation process itself.

If Y is the fraction of successive differences between successive elements in the list that are negative, then the program makes the randomness estimate R´ = 2*Y. The process is illustrated

| LIST | | | | | | | | | | Y | R |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | | | | | 0.0 | 0.0 |
| 0 | 6 | 2 | 3 | 4 | 5 | | | | | 0.2 | 0.4 |
| 6 | 10 | 1 | 5 | 4 | 9 | 2 | 3 | 8 | 7 | 0.5 | 1.0 |

and these agree obviously with the generation process in a gross way. The cost of the estimate is some approximately linear function of the number Ll=L*X of the items in the sample. There is no **a priori** reason why the optimal value of X should not be a function of L, and in truth it may be; furthermore, the optimum is not even well defined in any absolute sense, and must depend on the distribution of the population. This point is considered further in section 6.

Once the (R,L) was established for the given list, the method to be selected was found by examining the region around the point (R,L) in figure 4.1. Since that figure is drawn from noisy data, as discussed in section 5.1, we averaged the region around (R,L), using a 5X5 window, and the program chose the most frequent best algorithm in that window.

We selected a population of lists so as to exaggerate the effects and success of the scheme, by picking lists where the differences in performance are marked. The results for various values of X, that is, the fractional sample size for estimating S, are shown in the table:

| | COSTS of Sorting | Estimating | Total |
|---|---|---|---|
| X | | | |
| .05 | 2930 | 16 | 2946 |
| .10 | 2817 | 33 | 2850 |
| .15 | 2856 | 50 | 2906 |
| .20 | 2833 | 68 | 2900 |
| .30 | 2808 | 102 | 2910 |
| .50 | 2823 | 171 | 2994 |
| 1.00 | 2812 | 344 | 3156 |

The COSTS are in arbitrary units. This shows a shallow but definite minimum at X=0.10.

Using this value of X, then, let us compare this program with the single algorithms separately, and with the best possible selection (BP):

| Program | M | Q | I | BP |
|---|---|---|---|---|
| 2821 | 3457 | 3476 | 17991 | 2744 |

So it is clear that the program is not quite the best possible, but is still some 20% better than M or Q, even allowing for the extra resources used in making the estimate.

## 5.4 **Third Adaptive Scheme**

The third scheme illustrates the selection of a usable adaptive method from a set of possibilities. Each method is not precisely prespecified, but must be adaptively improved before it can be reasonably evaluated. This does not demonstrate the full construction of a possible complex processing scheme from a **tabula rasa**, but it does show how easy it is to test possible combinations of simple subprocesses to generate useful processes.

The supposition is that the program knows, from previous experiences that we do not specify, that the values R and L are significant parameters in the choice of the best sorting algorithm. What the program does not know is how best to combine those two parameters to get a reasonable function that will help to make the decision about the algorithm to be selected.

Again, we used the data that produced figure 4.1. The figure was divided in regions bounded by smooth curves, so as to smooth over the irregularities caused by the random number generation that made the figure. The boundary of the region where I is the best policy is like a hyperbola, for example, separating it from where Q is

best. The boundary between Q and M is somewhat more complicated, and shows clearly that M (if we didn't know) is a binary merge, losing a little efficiency relative to Q every time the length L rises above another power of 2. For the sake of simplicity, this scheme selects only between Q and I, ignoring M: that is because the boundary between Q and M is not easily describable.

The point of this scheme is not to have to store the relatively large amount of data in figure 4.1, but to approximate its contents with a simple formula. We suppose that the program does not initially have the concept or idea of hyperbolas, and cannot make algebraic inferences from pictures like the figure. What it can do is to try schemes and pick the best performing one. This conceptually simple plan is complicated by the fact that each scheme will be seen to have parameters of operation, just like the one in section 5.3 above. Since we need to compare the best examples of each scheme, that means that we must optimize each one before we compare them all and choose the best.

We decided to do that in parallel. The schemes we tried were all to use a function of L and R in combination with a threshold; if the function was greater than the threshold, select Q, and otherwise I. The functions were simple arithmetic combinations of L and R:

(L) (R) (L*R) (R/L)

There are obviously others; and those can be generalized in obvious ways. But perhaps they can be considered a fair sample. Remembering the observation two paragraphs above that the boundary between Q and I was approximately hyperbolic, the reader will suspect that the best function ought to be (L*R), since L*R=constant is a family of hyperbolae.

The program ran all of those schemes, represented by the different function forms, in parallel, keeping track of the costs; after adapting the thresholds to somewhere near optimum, the costs were compared and the best one chosen. The experiment used 2500 lists, the ones that were used to make figure 4.1. Using L as the function, the best threshold turned out to be 40; using R, the best threshold was .08 - .10. This is shown in the following table, using arbitrary units for the costs:

| LENGTH THRESHOLD | TOTAL COST |
|---|---|
| 10 | 6185 |
| 20 | 6182 |
| 30 | 6181 |
| 40 | 6179 |
| 50 | 6180 |
| 60 | 6187 |
| 70 | 6194 |

| RANDOMNESS THRESHOLD | TOTAL COST |
|---|---|
| .03 | 5995 |
| .04 | 5944 |
| .05 | 5944 |
| .06 | 5916 |
| .07 | 5916 |
| .08 | 5912 |
| .09 | 5912 |
| .10 | 5912 |
| .11 | 5933 |
| .12 | 5990 |

Using L*R, the best threshold was 30, and the cost was better than for either L or R by themselves.

| L*R THRESHOLD | TOTAL COST |
|---|---|
| 10 | 5957 |
| 20 | 5881 |
| 30 | 5877 |
| 40 | 5942 |
| 50 | 6066 |
| 60 | 6268 |

Using the other functions produced no usable threshold at all. The differences in performance are not enormous, it must be remarked, but they are all in the right direction.

## 6.0 **Discussion**

What we have tried to show here is that some conceptually very simple methods of adapting certain parameters that govern the selection of an algorithm in a computer program can produce profit for the system. Learning what is the best thing to do, and when to do it, always entails more work than merely doing what is the standard; sometimes, however, it more than pays for itself. If one is in a situation that is hard to model -- like a fickle and changing set of computer users -- the default procedure has to be to see experimentally what works best, and then to take advantage of what one finds out.

In our paradigm, sorting lists in an adaptive way, testing to see which is the best algorithm is expensive; usually more expensive than doing the task. It is as though we are slogging through mud of varying depth on submerged wooden tracks, but we do not know which is the track that is nearest the surface. To test the other tracks may require complete submersion, but it may result in finding a track that is only a couple of inches deep. So how often should one take the plunge?

Adaptation at another level is shown by the second scheme. Here we are provided with a good map of the terrain, showing the depth of the tracks. It's just that we do not know where we are unless we swim around in the mud getting bearings -- the more we swim the better the estimate.

The third scheme handles two adaptations at once. One of them is a simple tuning of a threshold, the other a simple discrete choice. The important aspect is that the second choice depends on having done the first adaptation well. It is an easy example of a hierarchy of adaptations. We use it not so much as to produce a useful program by itself, but to illustrate the kind of adaptations that must be used in the development of systems that may be hard to model or even understand.

It will be clear that a crucial role in this attack is played by the representation of the possibilities, the different functions in our case. As we have mentioned already, Winston (1976) lays much stress on that point. Behind it lie some other ideas. Before the representation can help, there must be the possibility of searching for help in the first case. L and R are merely two attributes of a situation where a program has a task to do. In some way the program seeks to use the information in R and S; and furthermore, it seeks to optimize the diagnostic functions of the observables R and S, by improving them. Typically in current system design, the possible realm of modifications is sharply restricted and tightly delineated, so as to make it less likely that bugs will arise. Systems, it is claimed, should always work in known ways with tried and true algorithms. What we are talking about here is a system that ought to be able to **notice**, for example, that small values of L should mean to use I; and from noticing, to make inferences about good rules of behavior. Good rules of behavior -- in our case, that L can be used as an indicator or diagnostic, and that so can R -- can be combined or modified to make better ones. That process of course does not stop in a single application, but continues, guided by the enlarging set of tasks that the system is faced with. Indeed, the rules of combination and modification themselves ought to be considered as modifiable in the same way, but perhaps that is more ambitious than we are dealing with here.

The efficient functioning of the adaptations described here does not depend on having an accurate model of what sorting is or how the individual algorithms work. Rather, we claim, the program tries to learn from various kinds of experience. This is far from saying that good models, mathematical or otherwise, should be avoided. If we can get good models, we should use them. But there are many instances when it is difficult to produce and deal with accurate models of the tasks ahead; in such cases, some of the adaptive techniques shown here, or ones like them, may be useful.

That may be true, for example, in the control of a complex communications net, or in setting or tuning the scheduling algorithm of a time-sharing system. Typically such models have to assume certain kinds of random distributions in order to be mathematically tractable; and all too often those assumptions are grossly incorrect.

## 7.0 BIBLIOGRAPHY


Feigenbaum, E., and Lederberg, J., "Heuristic Programming Project," in Earnest, L., "Recent Research in Artificial Intelligence, Heuristic Programming, and Network Protocols," Stanford AI Lab Memo AIM-252, July 1974.

Knuth, D., **The Art of Computer Programming; Volume 3: Sorting and Searching**, Addison-Wesley, Reading, Mass., 1973.

Minsky, M., and Papert, S., **Perceptrons; an Introduction to Computational Geometry**, M.I.T. Press, Cambridge, Mass, 1969.

Rosenblatt, F., "The Perceptron - a Theory of Statistical Separability in Cognitive Systems," Cornell Aero. Lab., Report VG 1196, G1 & G2, 1958.

Winston, P.H., **Artificial Intelligence**, Addison-Wesley, Reading, Mass., 1977.

Wirth, N., **Algorithms + Data Structures = Programs**, Prentice-Hall, Englewood Cliffs, N.J., 1976.