CASAP! A TESTBED FOR PROGRAM FLEXIBILITY

Robert M. Balzer

USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90291

Abstract

CASAP attempts to create a more flexible knowledge-based system for performing actions. It Is based on the combination of a procedural representation of actions, and an Information retrieval subsystem used to dynamically obtain all Information required to perform an action. It Is argued that such a combination eases the specification of new actions and increases their ability to obtain, from a wide variety of possible environments, the Information they require.

A simple example demonstrates this flexibility.

Introduction

The effort to build Intelligent programs has received a great deal of interest for a number of years. The early attempts were of the theorem proving type characterized by the Logic Theorist 11J, which attempted to Implement the rules of propos-itional calculus. Using these rules the system attempted to move from a stated set of axioms to a theorem to be proved by a generalized search technique. This system was the genes is for the later General Problem Solver (GPS) [2]. GPS applied a series of user specified operators to move from the initial state to the goal state as determined by rules of applicability of those operators and as directed by a difference table to eliminate differences between the current situation and the goal state.

Both systems *are* essentially table driven and represent an attempt to build a single general purpose problem solver capable of accepting valid operations In a wide variety of domains and using them to move from initial to goal states. The problem with such attempts has been that the general mechanisms contained within them for deciding In what order to apply the operators to make progress towards the goal have not been adequate for the problems under Investigatlon. Also, specific knowledge about how to proceed In a particular domain has been difficult to state In a general form for such systems.

This research is supported by the Advanced Research Projects Agency under Contract No. DAHC15 72 C 0308, ARPA Order No. 2223/1, Program Code No. 3D30 and 3P10.

These difficulties with general problem solvers led to a second approach centered around the incorporation of specific knowledge about how best to operate in a particular domain. Since this knowledge was not well codified, these systems represented It in the most general way known! I.e., in terms of programs. Experience with systems of this type led to the development of PLANNER [3] as a way of systematizing some common aspects of this approach. These common aspects Include provt sions for backtrack!ng, provisions for the dynamic update and maintenance of a data base, and pattern directed invocation (i.e., routines are invoked not by name, but by the results they promise to deliver. The promise is pattern matched against the current subgoal and Is invoked if the match is satisfied) - the procedural analog of the difference table In GPS.

These systems have traded increased operational power for loss of awareness. Because the knowledge is represented procedurally, the system is less capable of using It deductively or In determining what the consequence of particular actions may be.

CASAP represents a combination of these two approaches! the procedural representation of knowledge together with the general mechanism for assembling the Information required for actions or inferences.

There are two main problems with the procedural approach. The first Is the concurrent transfer of both control and data between routines. Typically, when a routine Is invoked, the data that it is to process Is also passed to It as part of the Invocation. Thus, the caller needs to know what data is requl*red* by the called routine. Such an organization is much too rigid. All that the caller should know about the called program is the result that It promises to deliver. Since routines are to be Invoked on the basis of the result that they promise to deliver, if for one reason or another they are inapplicable In a given situation, then they will inform the caller of this and alternative action can be taken. It shou]d therefore be the responsibility of the called program to obtain the Information it requires.

Towards this end, some systems keep the data In a common global data base which Is directly accessible by all routines. Hence, any routlne can get at whatever data it requires. This, too, Is an unacceptable solution; for it requires too wide a

distribution of the knowledge of how and where to find Informatlon, and unnecessarlly complicates each of the routines In such a system,

We propose instead. In CASAP, to place a single Informatton retrieval routine between any routine requiring data and the global data base Itself. Thus, all each routine requires Is the knowledge that It needs a certain piece of information which It then requests from the Information retrieval package. Knowledge about the data base Is thus centralized In a single routine.

The second major problem with the procedural approach is the polntwlse applicability of procedures. Either one procedure or another is active, but not both. This greatly limits the ability for two or more procedures to perform actions in a coordinated way. The closest thing we have to a solution for this problem are the demons which exist in various systems. These demons watch for certain events. in either the control or data spaces, and when such events occur, Invoke an associated routine which then gets control. This allows for more dispersed and low level Interactions between various routines toward a coordinated goal, but does not. In fact, lead to coordinated actions.

If viewed In this manner, CASAP does not address this Issue at all. However, from a different standpoint, what we desire Is the abl]lty for one routine to Influence the behavior of another routine, that is, to set up constraints or suggestions which *are* dynamically used to modify the behavior of the Invoked routines as directed.

CASAP, in a limited way, performs such modifications by utilizing a context which is established and maintained by the Information retriever, and used whenever Information is required. Thus, the Information obtained at any point, in response to a quest!on, is dependent upon the context that has been previously set up. Through this mechanism an Invoking routine can establish the context in which the request for Information from invoked routines will be answered and thus changes the perceived state of the world for that routine.

This mechanism Is, however, quite limited from two standpoints. First, It only deals wlth Informatlon that the invoked routine requires In a particular situation. Information which Is not needed by the Invoked routine has no way of Influencing the behavior of such a routine. Secondly, It Is an information-based context and not an action-based one. Hence, It only plays a part when I nformation is required and not when actions are being performed. With these restrictions In mind, however, ft is a step In the direct lon of a system which utilizes context for the Interpret/on of actions and Information dynamically required.

It is our belief that moving from essentially macro based languages to languages that are essentially context dependent for the interpretation of both actions and data is the next major needed advance In programming languages.

System Organization

CASAP represents our attempt to test these Ideas In an operational way and has demonstrated, In a very limited way, the feasibility. If not the practicality, of such an approach.

Organizationally, CASAP was predicated on the principle that there would be no subroutines In CASAP programs (the system Itself was Implemented by standard technique); that there would be no hard, well-defined Interfaces between the specification of act!ons to be accomplished; and that Information would flow among the processing components as required by the particular example, rather than as preplanned. Thus, both the decision of what actions to Initiate, and how and where the Information for those actions Is obtained, Is dynamically determined while the system is In operation.

The system logically consists of three parts: an Interpreter responsible for carrying out Inltlated actions; an information retrieval part responsible for obtaining and putting Into usable form the Information required by Initiated actions and/or the user; and finall, a modification component responsible for altering an action to fit it Into a dynamic context. Currently the modification component Is null and the system has been constructed so as not to Invoke this function.

As part of this experiment we wanted to investigate the build-up and use of commonly used concepts In a natural way. Hence, we have decided to use English as the way of specifying the actions, the concepts, and the Information Input to the system. However, to avold concentration on parsing of natural language, all input to the system was pre-parsed by the author, although It still contained the original English words.

Each input by the user was examined to see whether it was a concept or fact to be added to the data base, or the initiation of an action. In the former case the concept or Information was merely stored In the data base In the input form. Commands, however, caused the system to determine which action should be invoked. This Is done by pattern directed Invocation centered around the verbs. That Is, the system first determines what actions it knows about In the data base that correspond, either directly or by Inference with the specified command word (verb). A second level of applicability check Is made to see If the action Is appropriate for *the* current situation.

If any such actions are appropriate, the system selects one and applies Its definition. This process is repeated until one of the primitives of the system Is applied. These prlmltlves cons!st of the set manipulating routines of Insert, remove, create, and destroy which enable the system to add, delete or modify attributes and/or their values to objects In such sets. As the specifications

of the operations are being applied, contexts are being built up, which enable the Information retriever to locate the relevant data for the operations to be performed.

System Operation

Let's consider some examples from the protocol in the appendix. All the examples In this paper will be drawn from the domain of card games, and particularly from the game of Hearts. The system knows nothing about either cards, card games, or the game of Hearts Initially but merely the idea of ordered sets and the manipulations previously Indicated as be! ng primitive.

We begin by telling the system that a hand Is a set of cards ordered by suit and card value and then tell It the ordering relationship both for suits and for the card values by explicitly naming the values *of* each of these attributes in order. Then we tell the system that all players have a hand, and ask It to create a player which we are going to call Player 1. The system creates such a player, notices that all players have a hand, and therefore, constructs with that created player, a set which is to be Its hand, which the system knows will be composed of cards. We similarly then create two other players named Player 2 and Player 3. Then we create five cards. Having now primed the system, we are ready to start investigating Its behavior In carrying out a series of actions. We begin with the command "Insert card] In the hand of Player 1". The "insert" primitive action Is applied and obtains both the object It Is to insert and the set In which It Is supposed to do the Insertion by asking questions of the Information retriever. In this case, the required Informatlon is immedlately avaliable; and no problems are encountered in this execution. However, when we ask the system to then Insert card 2 In the hand of Player 1, the system first finds out which card and in which set the object is supposed to be Inserted. Then, In the process of doing the Insertlon, it discovers that this set is ordered. it determines that the primary ordering relationship Is suit. Since there *already is* a card In that set. It must discover the relative ordering between those two cards on the basis of the suit attribute. Therefore, it attempts to find out the suit of the card It is now Inserting. An Internal question is formed for the informatlon retrlever, which looks through the current context and the data base, but can't discover the value of this attribute, and so,, asks the user. The user responds that Its a heart. The system then continues operation of the insert primitive, but sees that It must also find the suit of the already Inserted card so that a comparison may be made. Again a question is formed, and again the Information retriever asks the user. The user responds that the suit of card 1 Is a diamond. Because the two suits are different, the system can order them properly; and card 2 Is correctly Inserted In the hand of Player I.

Notice that we have invoked the same routine twice. In the first case, the only information that it needed was the primary objects that were manipulated — the object to be inserted and the object in which the Insertion was to be done. In the second case, those same pieces of Information were needed; and in addition, two other pieces of Information were required: the suits of the two cards that were being inserted, so that the ordering could be properly maintained.

Thus, the amount of Information required by this routine - or, in fact, any routine, - is context dependent. This Is one of the main reasons why we want such Information to be dynamically obtained through the information retriever package, rather than being passed down or discovered by Individual examination of *a* global data base.

To belabor the polnt even further, suppose we now Insert another card, and when the system asks us what the suit of that card is, we tell it that it is also a heart. Since there is already a card, which }s *a* heart, in the hand, It must obtain further Information to resolve the conflict. By examining the ordering relationship specified with hands, it finds that this relatlonshlp Is the card value. The insert routine must then discover the card value of both the new card being Inserted, and the old card with which there Is a partial conflict.

Skipping ahead In the protocoli after putting the players around in a circle by means of a relation called "to the left of", we reach an Interesting command "each player passes the highest heart". (For ease of explanation we will now deviate slightly from the protocol in the appendix.)

Once the system has obtained the relevant actions, of which there might be several, it sees which of them are applicable to the given situation on the basis of their applicability to the objects described. In the current example, the system had available a description of how to pass a card. To test the applicability of this action, an Internal question was generated as to whether the highest heart was a card. The Informatlon retriever examined the data base, and found that heart was the value of the suit attribute of the object card. Therefore, It assumed that highest heart was a descriptive term being used for card which Is the highest heart. Hence the description of how to pass a card was Invoked.

The interpreter retrieved the previously specified definition of "pass a card," which Is: "remove It from your hand and place It face down on the table In front of the player to your left" ("In front of" was taken to be a spec!flc location on the table for each player). The Interpreter then Invoked the remove operation by pattern directed Invocatlon. Thls operation is one *of* the primitives of the system consisting of all the set manipulation operations. To perform the remove operation, the objects which "remove" manipulates must be available. It obtains these through the Informatlon retrieval package, but first forms a question to discover from which set the objects are being removed. The Information retriever discovers

that !t is "your hand" and that there exist many hands In the data base. Thus "your" Is used to determine which one Is being specified. "Hand" is looked up In the data base and is found to be owned by "player." Thus "your" refers to a particular player. St nee it is not further spec!fled, the Information retriever assumes that the particular player is the one who is in context, that Ts, the one selected earlier by the interpreter as part of the Iteration of the "each player..." command. Thus, the information retriever returns to the Interpreter the specific set that has been referenced. Next, a questIon is formulated within the remove package to discover which object from that set Is to be removed. Again the question Is formulated and the Information *retri*ever, working through the context stack, finds that the object is the card being passed, and that the card being passed is the hi ghest heart. At this poi nt the informati on retriever computes which is the highest heart within the context of the previously selected set, I.e., the highest heart Is computed on the basis of the hand in question and not the highest heart in the deck.

As one can see, the success of the i nterpretation of this particular example is critically dependent upon the order In which the questions were asked; but the knowledge that they should be asked In this particular order, namely, finding the set first and then the object within the set Is part of the intelligence of the remove package, not part of the general intelligence of the system Itself.

However, the remove routine is actually a little bit more general than indicated. Had the information retrieval package not been able to find a specific set for the remove operation, then the question would have been asked as to whether the object being specified was a named object or a computed object. If it was a named object, then the question *of* which set would have been re-asked In the context of those sets that contain the specific object In question. In the current example the object was a computed one; and hence, the strategy would have failed.

Conclusion

It is important to note at this point the flexibility that has been demonstrated by this admittedly simple example. The correct interpretation of the command was accomplished *by* the combination *of* the following:" Information In the data base which allowed us to determine that the highest heart was an instance of card; the context set up by a definition of one of the operatons used in carrying out that command, namely, the context established by the set specification in the definition; and the intelligence incorporated In the system at the lowest level In the order In which the remove package dynamically obtalned its parameters. The combination of these elements in an environment that dynamically obtalns Information as required has produced an instance of the flexibility that people seem to have in locating and utilizing Information as opposed to the rigidity displayed by most computer systems.

Appendix

```
Sys:  CASAP Initialized
User:  Hand is set(card, ordered-by suit and
       card-value)
Sys:  Input accepted
User:  Value-set(of suit, is clubs, and
       and diamonds, and hearts, and
       spades)
Sys:  Input accepted
User:  Value-set(of card-value, is ten, and
       Jack, and Queen, and King and Ace)
Sys:  Input accepted
User:  Player(which all, has hand)
Sys:  Input accepted
User:  Create object player named Player1
Sys:  Input accepted
User:  Create object player named Player2
Sys:  Input accepted
User:  Create object player named Player3
Sys:  Input accepted
User:  Create object card named Card1
Sys:  Input accepted
User:  Create object card named Card2
Sys:  Input accepted
User:  Create object card named Card3
User:  Input accepted
User:  Create object card named Card4
Sys:  Input accepted
User:  Create object card named Card5
sys:  Input accepted
User:  Insert(object card1, in hand of
       Player1)
Sys:  Input accepted
User:  Insert(object card2, in hand of
       Player1)
Sys:  What is the suit of Card2?
User:  Hearts
Sys:  What is the suit of Card1?
User:  Diamonds
Sys:  Input accepted
User:  Insert(object Card3, In hand of
       Player1)
Sys:  What is the suit of Card3?
User:  Spades
Sys:  Input accepted
User:  Insert(object card4,in hand of
       Player1)
Sys:  What is the suit of Card4?
User:  Hearts
Sys:  What is the card-value of Card4?
User:  Jack
Sys:  What is the card-value of Card2?
User:  Queen
Sys:  Input accepted
User:  Insert(object Card5, In hand of
       Player1)
Sys:  What is the suit of Card5?
User:  Hearts
Sys:  What is the card-value of Card5?
User:  King
Sys:  Input accepted
User:  Remove(object hearts which highest,from
       hand of Player1)
Sys:  What is the means of highest?
User:  Highest means card-value most large
Sys:  Input accepted
User:  To-the-left(of Player1,is Player2)
Sys:  Input accepted
User:  To-the-left(of Player2,is Player3)
Sys:  Input accepted
User:  To-the-left(of Player3,is Player1)
Sys:  Input accepted
```

Users Passtwho player which each,object card<sult hearts, which lowest))
Sys: What Is the means of pass?
User: Pass(obJect card:*1,means remove(object *1,from hand whjch your,then Insert (object *l,tn hand of player whlch-ls-to-the-left of player)))
SysJ What Is the means of lowest?
Users Lowest means card-value most small
Sys* Input accepted
User: stop

References

1. Allen Newell, J.C.Shaw, and Herbert A. Simon, Empirical Explorations with the Logic Theory Machine: A Case Study In Heuristics, Western Joint Computer Conference 1957, pp 218-239.

2. Allen Newell, J.C. Shaw, and Herbert A. Simon, Report on a General Problem-Solvlng Program. The RAND Corp., Report P-1584, 1959-

3. Carl Hewitt, PLANNER: A Language for Provlng Theorems in Robots, International Joint Conference on Artificial Intelligence, 1969, pp 295-301