A Cognitive Architecture for Learning in Reactive Environments

Pat Langley

Irvine Computational Intelligence Project
Department of Information & Computer Science
University of California, Irvine, CA 92717

Technical Report 86-21

December 1, 1986

The ideas in this paper have resulted mainly from the author's discussions with other members of the UCI World Modelers Group (Rick Granger, Dennis Kibler, Dan Easterlin, David Benjamin, Howard Henry) and the CMU World Modelers Group (Jaime Carbonell, Greg Hood, Keith Barnett, Klaus Gross, Hans Tallis). In particular, Dan Easterlin made significant contributions to the section on object concepts.

This research was supported by Contract MDA 903-85-C-0324 from the Army Research Institute and by a gift from Hughes Aircraft Company. Approved for public release; distribution unlimited. Reproduction in whole or part is permitted for any purpose of the United States Government.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Technical Report No. 1	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* A Cognitive Architecture for Learning in Reactive Environments		5. TYPE OF REPORT & PERIOD COVERED Annual Report 7/85–6/86
		6. PERFORMING ORG. REPORT NUMBER UCI–ICS Technical Report 86–21
7. AUTHOR*(s)* Pat Langley		8. CONTRACT OR GRANT NUMBER*(s)* MDA 903-85-C-0324
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Information & Computer Science University of California, Irvine, CA 92717		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Army Research Institute 5001 Eisenhower Avenue Alexandria, Virginia 22333		12. REPORT DATE December 1, 1986 13. NUMBER OF PAGES 28
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

cognitive architectures
reactive environments
integrated models of learning
goal-based learning
concept formation
procedural learning
cognitive maps
heuristics learning

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

OVER

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

Goals of the Research

Machine learning is concerned with the mechanisms by which intelligent systems improve their performance over time. Early research in artificial intelligence emphasized issues involving learning, and in recent years machine learning has reemerged as one of the central research areas of AI. There are two clear reasons for the growing interest in learning, one practical and one theoretical. On the practical side, a better understanding of learning methods would let us automate the acquisition of the domain-specific knowledge bases for new expert systems, and thus greatly speed the development of applied AI programs. On the theoretical side, expert systems are unattractive because they lack the *generality* that science requires of its theories and explanations. On this dimension, the study of learning may reveal general principles that apply across many different domains.

Although the field of machine learning has made significant strides over the past decade, we feel that most of this work has been limited along important dimensions. In this paper, we outline a research plan that responds to these limitations, and that we believe will extend our understanding of learning in important directions. Before proceeding, we should briefly review how our approach differs from earlier work in the area.

- *Learning in a reactive environment.* The vast majority of machine learning research has focused on bounded, symbolic domains such as occur in puzzles and mathematics. In contrast, we are examining learning in a complex, reactive environment that cannot be completely predicted.
- *An integrated model of learning.* Previous machine learning research has tended to focus on a single facet of learning. Instead, we are studying the interaction between methods for concept formation, procedural learning, and motor learning, and we plan to develop an integrated cognitive architecture and a common representational scheme that supports all these forms of learning.
- *A model of continuous learning.* Existing work in machine learning has studied short-term learning, involving runs of a few CPU hours at most. We plan to model learning over much longer periods of time, and to address problems of memory organization (such as indexing and retrieval) that are introduced by such continuous learning.
- *Goal-based learning mechanisms.* The majority of machine learning research has ignored the role of the learner's goals. In contrast, goals will have a central place in our cognitive architecture, directing the learner's search for useful concepts and procedures.
- *Modeling human learning.* Few machine learning researchers have attempted to model the mechanisms of human learning. We plan to use knowledge of human behavior to constrain our architecture, and we hope the resulting models will lead to empirical predictions about human learning.

Taken together, we believe that these research biases will lead us down quite different paths than those taken in previous efforts, and the following pages partially confirm this belief. Moreover, we feel that the resulting framework will lead us to a qualitatively better understanding of learning, and thus to a better understanding of intelligence in general.

Now that we have reviewed the motivations underlying our approach, let us turn to the research through which we hope to accomplish our goals. We begin with a description of the simulated world we have developed as our task domain. With that as background, we move on to the aspects of learning that we intend to explore in this environment. After outlining our initial designs for a cognitive architecture, we address four aspects of learning that we have chosen as foci – object concepts, stored procedures, cognitive maps, and problem solving heuristics. Our research is still in its preliminary stages, and we have tested only a few of our theoretical ideas by constructing running systems. Undoubtedly, our ideas will change as we gain more experience with the details of the task domain, so this document should be read more as a research proposal than a final report.

A Simulated Environment for Machine Learning Research

If one is concerned with complexity and reactivity, then the real world is an ideal task domain. Unfortunately, the field is still far from understanding low-level perceptual and motor behavior in sufficient detail to seriously consider building robots that learn. However, a simulated environment that approximates the real world would enable one to sidestep the issues of low-level vision and motor control, and to focus immediately on higher level issues of learning. There is some precedent for this approach to studying intelligent behavior. Winograd (1972) employed a fairly complex blocks world in his studies of natural language understanding and problem solving, while Becker (1970) has described another simulated world. Unfortunately, Winograd's simulation was completely predictable and non-reactive, while Becker's environment was too simple to support interesting tasks.

The Simulated Environment

We have implemented a simulated environment that is both complex and reactive, and which we believe will provide a suitable domain in which to study different forms of learning and their interaction. The simulation is implemented in the C programming language and runs in Unix 4.2 on Vaxen and similar machines.

The environment supports three-dimensional solid objects. Primitive objects consist of spheres, cylinders, circles, and polygons, but these can be combined to form complex objects such as tables and chairs. Complex objects are defined hierarchically, with primitive objects as terminal nodes in each object tree. Each object has a number of features, including its physical dimensions, orientation in space, texture, and color. In addition, sounds and odors exist for brief periods after they are generated.

One major aspect of the real world is that it changes over time, and the simulated environment also has this characteristic. Time proceeds in discrete steps, with objects obeying the laws of Newtonian physics. The positions of objects at each time increment are updated as a function of their previous positions, velocities, and the forces applied to them. The world supports three basic forces – gravity, torque, and friction – and these determine the motions of objects in the world. Collisions can cause the direction of motion to change, leading to phenomena such as bouncing balls. With respect to forces and collisions, complex objects are rigid figures that act as units.

Although we have made efforts to simulate the real world as closely as possible, we have been forced to omit many details. Thus, only objects with regular shapes are supported, and liquids and gases are notably absent. However, we believe that the environment contains sufficient variety to provide real challenges to our models of learning. We have described the environment in more detail elsewhere (Langley, Nicholas, Klahr, & Hood, 1981; Carbonell & Hood, 1985).

Simulated Agents

The environment also supports simulated robots which are composed of primitive object types. We have experimented with a number of basic designs, all relatively simple. For instance, one organism has a cylindrical body with one eye and no arms.[1] The existing simple robots are sufficient to let us explore initial navigation and recognition tasks, but we plan to implement more complex versions in the near future.

The body of the simulated robot is controlled by an intelligent agent. The agent has control over a few primitive actions, such as applying a linear force in any direction (allowing rectilinear motion), applying rotational force to the left or right (allowing turning), and turning the eye to the left or right (allowing changing views). These basic effectors can be applied with varying amounts of force, and they are the building blocks from which more complex behaviors are constructed. Future organisms will have effectors controlling each of their joints, allowing even greater variety in motor behavior.

The agent also has sensing abilities that let it determine the shapes and distances of objects in its field of view. Rather than model the complex process of transforming retinal images into three-dimensional representations, we assume that the agent directly perceives complete shape descriptions. Thus, the sensory interface provides the agent with the shapes, dimensions, orientations, and distances of objects. Research by Marr (1982) and others suggests that bottom-up visual processes lead to such shape descriptions, making this a reasonable simplification for our purposes.

However, the agent does *not* have direct access to information about all objects. The agent is only provided with descriptions for objects located in its cone of vision. Moreover, one object may occlude another, causing the latter to be blocked from view. Finally, information is made 'fuzzier' with distance, making descriptions less certain when an object is far from the robot. In other words, the agent's knowledge of the world is incomplete, and this provides a natural motivation for movement and information gathering.

Although vision is the primary sense, it is not the only one. The agent can also hear sounds and note smells occurring in the environment, it can feel the texture of objects that it touches, and it has direct perception of the position of its body parts. This multi-modal flow of information provides the opportunity for noting correlations between senses and using these relations in learning.

---

[1] We are currently extending the simulation to support jointed arms. Until we have this capability, the forms of motor behavior described in later sections will be impossible.

The Graphics User Interface

In addition to the simulated world and the sensory-effector interface, we have also developed a graphics-based user interface. This is also written in C and runs on Perq-II graphics workstations. The interface allows the user to view the simulated environment from any point in three-dimensional space, including the viewpoint of the learning system. It includes facilities for easily creating new objects, and we plan to add capabilities for directing a tutor organism by remote control; this will be necessary for the work on learning by imitation described below.

The graphics capability will prove essential for debugging extensions to the simulated environment. However, it will also be necessary for understanding the behavior of our learning systems. Since we are developing programs that will learn over long periods of time, we must deal with knowledge structures too large for traditional debugging methods. In many cases, we hope to be able to infer the system's goals and rules from its behavior, and for this we must be able to observe this behavior in relation to the environment. We also plan to implement graphics tools for displaying knowledge structures as they are retrieved during preformance and modified during the learning process. Thus, we feel that powerful graphics workstations will be essential to the success of the project.

Design for a Cognitive Architecture

In recent years, attempts to develop computational models of human learning and performance have led to the notion of a *cognitive architecture.* Theoretically, a cognitive architecture represents the *invariants* of the human information processing system. These characteristics are 'hard wired' into the system and thus affect behavior in a wide range of situations. However, a cognitive architecture does not attempt to specify how these invariants are implemented at the hardware (neuron) level; rather, it describes them at a *functional* level. Such an architecture contains significant theoretical assumptions about representation, performance, and learning, and thus constrains cognitive models considerably more than general formalisms such as Lisp or Prolog. We will see examples of such assumptions as we proceed.

Classes of Cognitive Architectures

A number of cognitive architectures have been proposed over the last five years, but most fall within two major classes – production system formalisms and schema-based approaches. One well-known example of the former is the ACT architecture, which Anderson (1982) has used to model a variety of psychological phenomena surrounding the acquisition of skilled behavior. Another more recent example is Laird, Rosenbloom, and Newell's (1984) SOAR, which combines the production system approach with the problem space hypothesis – that all cognitive behavior involves search through some problem space. Perhaps the best known schema-based framework is Schank's (1982) MOPS architecture, in which knowledge is organized into interconnected structures that describe scenes and events.

Within this classification, our architecture must be grouped with the schema-based approaches, since it contains interconnected 'scripts' or procedures containing sequentially ordered information about 'scenes'. However, it differs from earlier schema-based frameworks in the nature of these scripts; while earlier approaches assumed a very abstract, propositional representation of states and events, our architecture posits a much lower-level sensori-motor representation. We will see examples of this representation in later sections.

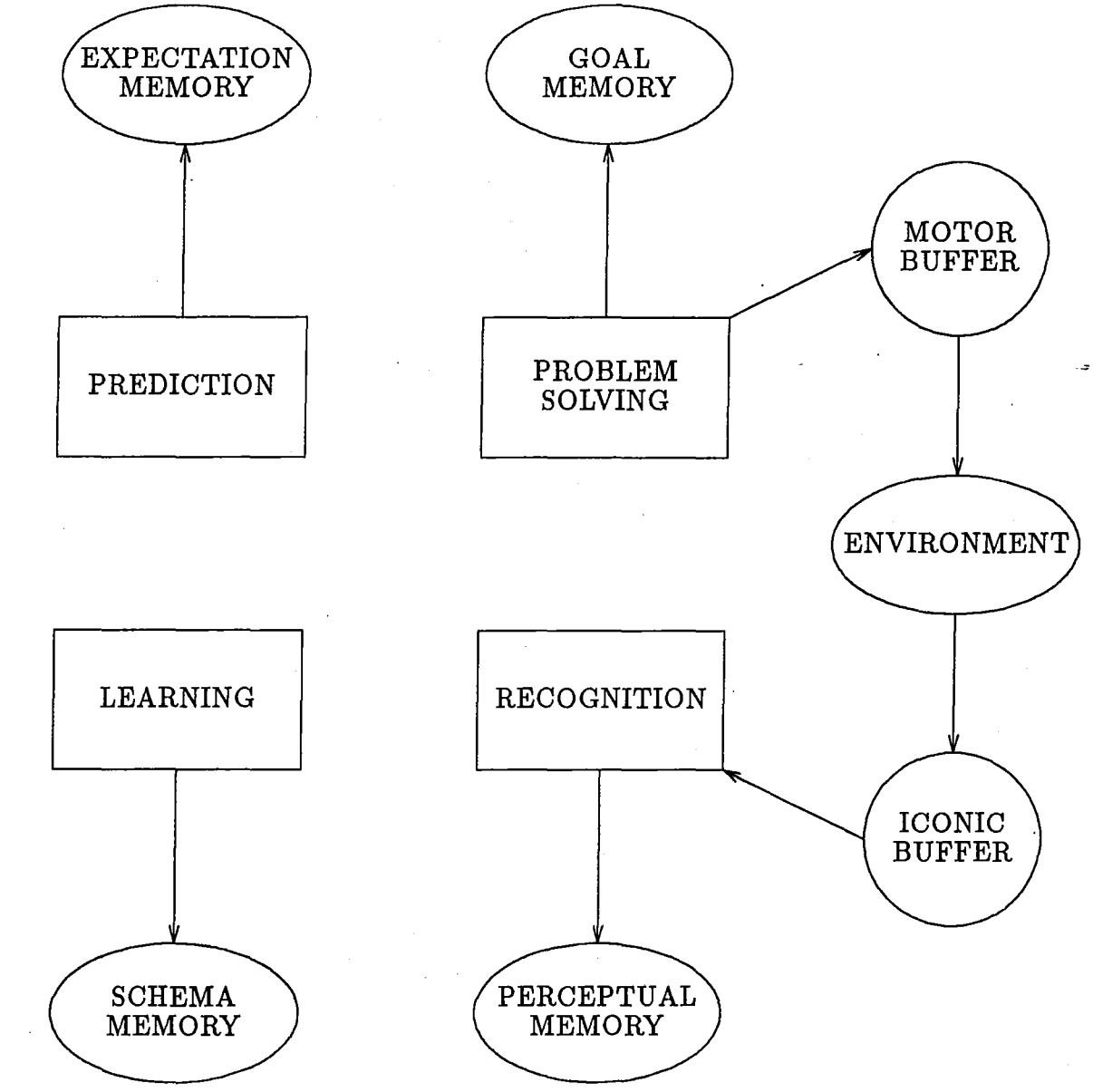

Figure 1. Memories and processes of the cognitive architecture.

Types of Memories

Figure 1 presents an overview of the architecture in terms of memories and mechanisms. The cognitive architecture includes four basic memories:

- *Schema memory.* This contains general knowledge about the world, including object concepts (such as *table* and *ball*) and stored procedures (such as *throw* and *bounce*). These schemas are 'fuzzy' in that they can match against items in the other memories to greater or lesser degrees.
- *Perceptual memory.* This contains descriptions of what the agent has experienced in the world, as well as specific beliefs that it has inferred from these observations. This includes instances of object concepts and procedures (e.g., the dog chased the ball).
- *Goal memory.* This contains descriptions of the agent's goals, including desired states (transform goals) and desired actions (apply-operator goals).
- *Expectation memory.* This contains the agent's predictions about what will occur in the environment, including the locations of various objects and the occurrence of procedures and events.

There are two interesting aspects to these memories. First, both objects and procedures are represented at a sensori-motor level, including details about shape, size, location, and orientation of objects. Second, the contents of the four memories have nearly identical forms. In other words, it is possible for the same description to occur as a schema, a perception, a goal, and an expectation; however, this description will be interpreted quite differently depending on the memory in which it occurs.

Each of these memories contain both 'object-like' elements and 'procedure-like' elements. Thus, one can believe that a chair is nearby, and that another agent just sat in that chair. Similarly, one can desire to see food, as well as wanting to eat that food. In addition, most memories have both a short-term (active) partition and a long-term partition (which stores past experiences). The exception is schema memory, which contains only long term structures.

In addition to the four basic memories, the architecture also includes a motor buffer and iconic memory. The former holds motor commands for very brief periods of time, and these commands cause the effector interface to alter the simulated environment. The iconic memory plays an analogous role for perception, holding low-level perceptions for brief periods of time until attention mechanisms transport some of these descriptions into perceptual memory.

Types of Processes

Each main memory has an associated process that is responsible for modifying the contents of that memory. These processes may *access* information from any of the four memories, but they may *write* only to their associated store. These processes include:

- *Recognition.* This process is responsible for recognizing instances of schemas (object concepts or stored procedures), and adding this information to perceptual memory. This in-

volves retrieving candidate concepts (possibly using goals) and invoking a partial matching mechanism to determine the degree of match between concepts and sense data. For example, one may recognize an object as an instance of a chair.

- *Prediction.* This process is responsible for generating expectations from beliefs about the current state of the environment and previously generated predictions. For example, one may predict that a chair has a fourth leg, even though this component is occluded by other objects.
- *Problem solving.* This process is responsible for generating subgoals in response to existing goals and beliefs. This involves a combined process of forward-chaining and backward-chaining known as means-ends analysis. Goals describing primitive actions are deposited into the motor buffer rather than goal memory.
- *Learning.* This process is responsible for modifying the contents of schema memory. This can occur as the result of accurate or violated expectations, successful or failed goals, and correct or incorrect classifications. Learning involves both the creation of new schemas and the incremental modification of existing structures.

The contents of schema memory play a central role in each of these processes. The object concepts and procedures stored in this memory are essential in recognizing objects and events, in making predictions about events to come, and in carrying out actions and creating subgoals. Other memories may modify these processes (e.g., goals may slant the recognition mechanism), but long-term schemas are the major determiners of behavior.

Clearly, each of these processes is quite complex, and we have only attempted to summarize them here. In the following pages we describe them in more detail, but we have not organized our treatment along the lines used above. Instead, we have divided the architecture according to alternative forms of knowledge. The first of these involves object concepts (such as tables and chairs) and their recognition. The second concerns the recognition and application of stored procedures (such as throwing a ball). The third involves knowledge of spatial relations in terms of 'cognitive maps'. Finally, we examine the use of goals and heuristics in the problem solving process. In each case, we consider the representation of knowledge, the mechanisms that operate on these representations, and the processes by which these structures are learned.

The Formation of Object Concepts

The vast majority of AI research on concept learning has assumed that concepts can be represented in terms of necessary and sufficient conditions (Mitchell, 1977; Hayes-Roth & McDermott, 1978; Michalski, 1980). Although this view holds for mathematical concepts such as 'triangle', it seldom works in the real world. For instance, some birds (like robins) are more prototypical than others (like penguins), and some chairs are better than others (some may have broken legs). Rosch and Mervis (1975) have discussed this issue at length, and have noted that humans acquire certain categories earlier than they master others. One aspect of our model focuses on the acquisition of such object concepts. However, before we can address the issue of how such 'family resemblance' concepts are learned, we must find some way to represent them and to use them during recognition.

Representing Object Concepts

We hypothesize that object concepts are represented as abstract *prototypes*, in which some features are labeled as more criterial than others. We envision structural descriptions similar to Binford's (1971) generalized cylinders, with actual numeric values replaced by means for each parameter. For instance, one might represent a prototypical chair as consisting of four legs, a seat, and a back arranged in particular spatial relations to each other.[2] In addition, each component would have associated numeric means, such as orientation, length, and diameter (normalized for the overall size of the object).

In addition to associating a mean value with each numeric feature, we also associate a *variance* based on the degree to which instances of the concept have varied along that dimension. A high variance implies that a wide range of values are acceptable, while low variances imply that only small variations from the mean can be tolerated. We posit that the criteriality of a numeric feature is equivalent to the *inverse* of its variance. This gives low criteriality to features with widely varying values, and high criteriality to features with nearly constant values. For instance, the legs of a chair are nearly always half the length of the entire chair's height. Thus, this feature would have a low variance and be highly criterial, giving it an important role in judgements of prototypicality.

Clearly, some concepts (such as *food*) must be defined in functional rather than structural terms. Many of our everyday concepts have structural components, and we have chosen to focus on these in our initial work, but the architecture also provides a natural explanation of functional definitions. As we will see later, procedural concepts describe common event sequences, and each procedure may refer to object concepts that occur as arguments to that procedure. Now assume that stored procedures are indexed by the objects to which they can be applied. One may then interpret the set of procedures that can be applied to any given object concept as the *functional definition* of that object concept. Although we will not focus on such modes in the present paper, the basic architecture shows promise for function-based reasoning about the usefulness of given concepts.

Recognizing Object Concepts

Now that we have considered the representation of object concepts, let us turn to their recognition. We have seen that traditional AI approaches to concept learning assume necessary and sufficient conditions, and it is natural that most have employed complete matching methods for concept recognition. However, in rejecting the 'all-or-none' assumption, we are inevitably led to replace this with some form of partial matching mechanism. Hayes-Roth (1978) has argued that partial matching is computationally expensive, and the best known algorithm is exponential in the general case. Therefore, we must take advantage of constraints to make the task manageable.

Recall that we are assuming different weights on the various conditions composing the

[2] Not all descriptions need be so visually oriented, and even primarily visual concepts may have other associated features. However, since vision plays such a major role in human concept recognition, we will focus on visual features in our examples.

concept's 'definition'. To a certain extent, we can constrain the partial matching process by attempting to match more criterial conditions (those with higher weights) first, and leaving less criterial features and relations until later. We envision a beam search through the space of partial matches, which is computationally much more attractive than an exhaustive version. Like all heuristic search approaches, the method is not guaranteed to find the optimal solution (in this case the best partial match), but it will nearly always find a satisfactory one with considerably less effort.

However, recall that an experienced agent must choose between hundreds (or even thousands) of competing concepts, and it is unlikely that the above method will suffice. But suppose we assume that concepts are created because their instances have been instrumental in achieving the learner's goals (more on this below). In this framework, it is natural to organize concepts around the goals they help satisfy.[3] If we index concepts by the goals with which they are associated, then the agent can use its currently active goals as probes to retrieve potentially relevant concepts. As a result, the partial match is constrained to those concepts likely to aid in achieving the current goal, presumably a few instead of thousands.

Since it is central to the recognition process, we should say a little more about the partial matching mechanism. Given the description of some object and the characterization of some concept, the matcher returns a mapping between the two structures, along with the degree to which the match was successful. If the match was high, then the agent can infer that the object will prove useful for satisfying the goal under which its concept was indexed. If the match is only fair, then it may still want to use the object, provided no better objects are found in the immediate vicinity. For instance, one may sit on a table (thus treating it as a chair) when no chairs are available.

There remain questions about the manner in which the matches for different feature are combined into an overall prediction for an object. We assume that the degrees of match for all features are combined linearly, with each feature weighted by its criteriality; the resulting sum represents the degree to which the object matches the concept. Missing structural components (e.g., an occluded leg of a chair) have scores of zero assigned to their features. Since one can seldom observe all components of object, a visual object concept will almost never achieve a perfect match; however, a given object will certainly match some concepts better than others.

The Clustering Process

Now that we have considered the representation and recognition of object concepts, let us examine how they might be acquired. An inspection of the concept formation task reveals three issues that must be addressed:

1. *Clustering.* One must decide which objects to group together into an abstract class, such as table or chair.
2. *Characterization.* One must generate some intensional description of the objects in the

[3] More precisely, object concepts should be indexed by the procedures in which they occur, with procedures themselves being indexed by the goals they satisfy.

class, so that future instances can be recognized.

3. *Indexing/Storage.* One must store the characterization in such a way that it can be retrieved when needed, as during the recognition process.

The vast majority of AI research on concept learning has occurred within the paradigm of learning from examples (Winston, 1975; Hayes-Roth & McDermott, 1978; Mitchell, 1977; Michalski, 1980). In this approach, the clustering problem is sidestepped, since a tutor provides positive and negative instances of the concept to be learned. In addition, there is no significant storage problem, since only one or a few concepts must be acquired. In other words, the task of learning from examples can be viewed as 'distilled' characterization.[4] Although much has been learned from this approach, we believe that learning in a reactive environment requires one to deal with all three issues, and our model has responses to each.

Since children learn many concepts before they come to understand language, it is clear that they do not learn from examples in any traditional sense. However, by rejecting this approach to concept learning, we must find some other solution to the problem of clustering objects. We have already argued that goals are important to the recognition and retrieval of object concepts, but we believe that they are equally important to the clustering process. More precisely, we feel that objects are grouped together on the basis of their ability to satisfy (more or less) common goals.

In describing their means-ends analysis theory of problem solving, Newell, Shaw, and Simon (1960) distinguish between various types of goals. Our concern here is with *apply-operator goals*, in which one desires to apply an operator to some object or state. Such apply goals that can be used to direct the clustering process. For instance, suppose the agent is tired, and decides to apply the operator *sit-down*.[5] This operator requires some object upon which to sit, and the agent will scan its immediate environment for a likely candidate. The important point is that by applying its operator to candidate objects, the agent will discover that some objects produce better results than others. These will be good instances of 'sittable' objects, while others (such as chairs with uneven legs, or with a tack on their seat) will be poor instances. In any case, only objects to which the operator has been successfully applied are passed on to the characterization process.

We are assuming that means-ends analysis is one of the main mechanisms underlying problem solving. Of course, it is clear that some behavior (even some search behavior) does not involve means-ends reasoning, and it is likely that the clustering problem is not always solved in this manner. However, we believe that the majority of human concepts are goal-based, and that they result from goal-based problem solving. It is interesting to note that although the majority of machine learning research emphasizes the importance of search,

[4] The term 'generalization' is often used in place of 'characterization'. However, we prefer the latter term, since the former has the second meaning of starting with some specific hypothesis and making it more general over time.

[5] Obviously, the action *sit-down* is not primitive in any sense; it is a high-level procedure that must also be acquired from experience. We discuss the issue of procedural learning in a later section.

little work has focused on the role of goals in learning.[6]

Characterization and Indexing

Let us now turn to the mechanism of characterization, by which the learner goes from instances of some concept to a description of that concept. We are assuming that the clustering process has determined which instance or object should be incorporated into which concept description, and that it also provides the degree to which that object satisfies the relevant goal. The task of characterization is to modify the existing description to better predict the 'goodness' of the current object. The environment can seldom be completely predicted, so we do not require that the concept perfectly summarize the instances it covers.

We also assume that instances are processed incrementally, since the agent generally interacts with one object at a time (or a few at most). Thus, each instance leads to only minor modifications in the concept description. However, before a concept description can be altered, it must first be created, and issues arise about the nature of such initial descriptions. Since early descriptions are based on a single instance, one might make each feature very specific by having a criteriality of one. As additional instances are observed, variation will likely be noted, and constraints on feature values will become looser. Alternatively, one might begin with low criterialities and increase them if additional evidence warrants this step. The first approach is similar to generalization learning (modified to deal with prototypes), while the second is similar to discrimination learning. A third alternative would be to employ the number of instances in determining the degree of match, improving the fit when fewer objects have been observed. This would lead to a form of 'recognition by analogy' at the outset, and more abstract forms of recognition after additional experience had been gained.

Once a stable description has been formed, the feature values of new instances are used to modify the means and variances associated with each numeric feature. By retaining the number of instances that have been observed so far, one can easily compute the new values for each feature. This may lead to gradual changes in the concept description over time. For example, if the learner began to see chairs with longer legs, his coefficients for the 'length of leg' features would slowly be revised. Thus, this method can respond to changing environments, unlike most traditional approaches to concept learning.

However, if the agent encounters an object with feature values that fall far outside previous experience, this is an occasion to generate a disjunctive version of the current concept. For instance, if one sees a chair in which the legs are substantially longer than expected (such as a baby's high-chair), then it is natural to distinguish this from other chairs that more closely match one's expectations. Such variants are stored near to the initial concept, but are characterized independently of the original version. Note that this implies the order of presentation is relevant to learning. If gradual changes in feature values are observed, a single concept will be learned; however, if instances with extreme values are alternated, disjunctive concepts will be acquired instead.

---

[6] Recent papers by Ohlsson (1983), Anderson (1983), and Laird, Rosenbloom, and Newell (1984) describe exceptions to this trend.

In addition to the processes of clustering and characterization, concept formation also requires one to store the concept description in some manner that will let the agent efficiently recognize future instances of the concept. We have already discussed the retrieval process, and the role that goals play in indexing concept descriptions. The storage process is simply the means by which new concepts are indexed under the goals they satisfy. Complications may arise when the same concept proves relevant to multiple goals, but there is nothing to prevent the learner from generating multiple indexes for a concept. Basically, we believe that the storage process is relatively straightforward, at least compared to the complementary mechanisms for clustering and characterization.

In summary, our model of concept formation relies heavily on goals for both the clustering of instances and the retrieval on candidate concepts. Since the problem solving process relies on procedural schemas to generate such goals, let us now turn to the acquisition of procedural concepts.

The Acquisition of Procedural Concepts

In addition to object concepts for describing its world, an intelligent agent must have some form of procedures for accomplishing its goals. These may be generated from known components on the fly, or they may be stored in plans that can be accessed when needed. The former involves problem solving and search, and we discuss these in a later section. The latter relies on stored procedures, and in this section we consider the representation, use, and acquisition of such procedures.

We plan to explore procedural learning in two contexts – learning to manipulate objects and learning navigation skills. Both of these involve higher level planning components as well as lower level physical skills. Below we argue that higher level components can be learned by imitation, while the lower level ones will require trial-and-error practice, though guided by the acquired plans. Both tasks should also provoke goal-based concept formation, since different classes of objects can be manipulated in different ways (some can be lifted, while others cannot), and different types of objects have different implications for navigation (some can be pushed aside, while others must be avoided).

Representing Procedures

In our framework, procedures have much in common with object concepts. We represent a procedure as a sequence of state descriptions, in which each state is described with the same sensori-motor level language as used for object concepts. But despite this similarity, procedures differ from object concepts in two important respects – they are sequential in nature, and they involve the application of operators. As an example, let us consider the throw procedure, which might consist of six ordered states:

1. The agent is near an object but is not holding it;
2. The agent is enclosing the object in his hand, in front of his body;
3. The agent is enclosing the object in his hand, above and behind his head;

4. The agent is enclosing the object in his hand, in front of his body at eye level;
5. The agent is holding his hand in front of his body at eye level, not enclosing the object.
6. The agent's arm is hanging to his side, and the object is in the air in front of the agent.

Of course, the actual descriptions would be stated using a scheme similar to Binford's generalized cylinder notation, but the English paraphrases convey the basic idea.[7] The similarity to object concepts should be clear. In fact, we believe that each state description also contains mean values and variances for numeric features (e.g., giving the position of the hand behind the head), which translate into expected default values and the allowed variance for each feature.

However, unlike simple object concepts, the state descriptions that make up procedures are ordered sequentially and are connected by operators.[8] The interpretation of these operators differs somewhat from the standard sense of 'operators' in heuristic search problems. Rather than applying an operator once, each operator is applied iteratively until some halt state is reached. For instance, in the *throw* procedure, the *lift-arm* operator would be applied repeatedly until one reached the desired state in which the arm was above and behind the head. This 'repeat-until' notion is very similar to Miller, Galanter, and Pribram's (1960) Test-Operate-Test-Exit (TOTE) units, and seems particularly well suited for representing motor procedures. Moreover, there is evidence (Keele & Summers, 1976) that human motor activity is organized along a similar lines.

These operators may take arguments, such as the speed and force with which they should be applied. The arguments may be passed on from the calling procedure to its subroutines, since in order to throw objects with varying force, one must evoke the *lift-arm* and *move-arm-forward* operators with varying force. In addition, state descriptions may be connected by more than one operator, letting actions occur in parallel. Thus, in throwing a ball, one also moves the opposite arm forward (for balance) while moving the grasping arm behind the head. However, such parallel motions typically begin and end at the same time, employing the same test to determine when to halt. (In fact, Klapp and Greim (1979) present evidence that humans are not able to carry out two movements of different duration.) Finally, we should note that procedures can be organized hierarchically, with the operators of one procedure expanding to a lower level procedure. Thus, the procedure *making a double play* would contain the *throw* procedure as one of its components (actually occurring twice in this case).

---

[7] Again, we do not mean to limit our descriptions to visual features, especially in representing motor skills. Other forms of information (such as proprioceptive feedback) would also be included, but we have focused on the visual aspects here for the sake of clarity.

[8] Actually, many everyday object concepts can exist in multiple states. Doors can be open or closed, lights can be on or off, and many of these even vary continuously. We can represent such objects in terms of the procedures for causing their state changes. Thus, one might have a procedure for opening a door. In some cases (such as the weather), the state changes cannot be controlled by an agent, and the operators must be omitted. One can still recognize such procedural concepts, but can never enact them. This representation of state changes is similar to De Kleer and Brown's (1983) representation for qualitative mental models.

Recognizing Procedural Concepts

Like object concepts, stored procedures can be used for recognition, in this case for the recognition of familiar events. In many cases, one is attempting to understand another agent's goals and strategies based on his observed behavior. This is the *plan recognition* problem studied by Schmidt and Sridharan (1978), Schank and Abelson (1977), and others. The task confronting our model differs from these in that our system must interface with an external world, and must deal with actual scenes rather than the abstract representations output by natural language systems.

We will start by considering the process of matching a given procedure against a sequence of observations, and then address the issue of retrieving plausible procedures. Recall that procedures are composed of two different structures – state descriptions and the operators connecting them. In understanding another's actions, one can never observe the operators themselves; one can only infer them based on their effects. For instance, suppose the agent sees another agent grasp an object, lift his arm behind his head, move the arm forward rapidly, and then release the object. Actually, the agent would not see this at all. Rather, he would see a series of 'snapshots' in which the other agent's position (and the object's) had changed. One must *infer* that certain low-level operators were applied to achieve these state changes, and one must divide this sequence into episodes (grasping the object, lifting the arm) by matching against procedures in memory.

Suppose the system is considering the *throw* concept, and attempting to match an observed sequence of events against this procedure. It begins with the description of the start state. This is probably the most abstract of the state descriptions in the *throw* concept, since it involves the most variation. This means that many scenes would match well against it. However, the next state description in the *throw* procedure is more constrained – the agent must be holding an object in front of its body. Of course, one will observe many scenes between these which are not explicitly represented in the *throw* concept.

However, an important relation must hold between these states and the two descriptions (the ungrasped and the grasped object) – each successive state must take the agent *closer* to the grasped state. We will call this the *progress* constraint. In computational terms, every successive scene must have a better partial match to the grasped state description. If this relation holds, then the *throw* procedure usefully describes the agent's behavior; otherwise, some other concept may be more appropriate. Once the agent has actually grasped the object (and a nearly perfect match is found), the next episode is initiated. Now each successive state must be closer to the succeeding description, in which the agent's arm is cocked behind his head. This process continues until the final description of the *throw* concept has been observed, in which the agent's arm is relaxed, and the object is flying through the air some distance from the thrower.

Like object concepts, procedural concepts can be matched to a greater or lesser degree. Since each state description consists primarily of object concepts in particular configurations, we can use the same partial matching mechanism as we use for recognizing object concepts. Some descriptions may have little variance and are thus highly constrained, while others allow many specific scenes to match. Tentatively, we have defined the total degree of match

for a procedural concept as the average degree of match for its component descriptions. Thus, the *throw* procedure would average over the degree to which each of its five descriptions were matched by some observed scene, assuming that each of these was matched sufficiently well (in sequence) to retain the *throw* hypothesis. Thus, some instances of *throw* will be more prototypical than others, just as some chairs better fit the standard notion of chair.

Procedural structures share another issue with object concepts, in that they must be indexed in some manner that narrows the search for relevant concepts, enabling the partial matching process to be computationally tractable. The progress constraint will help significantly, since many competing procedures will be eliminated from the competition early on. However, two other types of information can also greatly constrain the search for useful matches. First, each procedural concept contains descriptions of the objects involved in that procedure. These may be more or less constrained, but suppose we assume that procedures are indexed by the objects to which they can be applied. Given such connections, whenever one notices an object changing state, one simply retrieves all procedures in which that class of objects occur. Few (or possibly none) of these will explain the observed state change, but the set of resulting hypotheses will be much smaller than the set of known procedures, and one can reasonably apply the above partial matching method to this reduced set.

The second method takes advantage of the inferred goals of the observed agent. Suppose that procedures are indexed not only by the objects involved in them, but by the final effects of the procedure (such as an object flying through the air). If we know the goal of the agent (e.g., to rapidly transfer an object from his possession to another's possession), then these indices can be used to retrieve potentially useful procedures (in the transfer case, the *throw* procedure). Moreover, since procedures are organized hierarchically, the invocation of a high-level procedure leads one to consider its component procedures as well. This expectation-driven approach to procedure recognition will only apply when one can infer an agent's goals, but it should be very useful in such cases.

Applying Procedures

Unlike static object concepts, procedural concepts can be 'run' to produce behavior and thus affect the external world. Let us consider how our model will produce such behavior, given the representation of procedures we have described. We will not examine how the agent decides to evoke a particular procedure here; such decisions fall within the realm of problem solving, which we discuss in a later section.

Suppose a procedure like *throw* is called with a few arguments specifying parameters such as speed, force, and direction. Before the procedure can be applied, it must first be retrieved from memory and expanded into its components. These components consist of (1) intermediate state descriptions and (2) the operators connecting these descriptions. Naturally, if the current state of the agent and/or the relevant objects do not closely match the initial state description, then the procedure cannot be applied. In this case, the agent must retrieve or construct (through problem solving) some procedure for transforming the current state into the required starting state. Once this has been achieved, the procedure can proceed.

Research on human motor behavior (Keele & Summers, 1976; Summers, 1981) suggests two modes of operation, which have been labeled *closed loop* and *open loop* processing. In the former, the agent continually checks against predicted feedback (both visual and proprioceptive) to ensure that he is carrying out the action correctly. Closed loop behavior typically occurs when the details of a procedure are still being acquired, and performance is gradually improving. In open loop behavior, the agent ignores predicted feedback and carries out the procedure much more quickly than in closed loop mode. Feedback can still be used for error recovery (e.g., if someone grabs the agent's arm while throwing), but it is generally bypassed. Open loop behavior typically occurs after a procedure has been overlearned and automatized.

Although we are not attempting to model human motor behavior in detail, we believe that the closed loop/open loop distinction is useful for any system that operates in a reactive environment. Moreover, these two forms of processing arise naturally from the representation we have proposed for procedures. In closed loop mode, the agent starts from the initial state description and applies the specified operators. After each application, the resulting state is compared to the succeeding state in the procedure's definition. If the observed state of the world matches the predicted state sufficiently well, then the agent moves on to the next operator and repeats the process. For instance, in throwing one stops lifting one's arm at a certain position behind the head, and then moves the arm forward from that position.

However, for many real-world procedures the same operator must be applied repeatedly. This occurs if the observed state matches the predicted state *better* than the previous state – in other words, if some *progress* has been made toward the expected state. If this occurs, the operator is repeated, a new state is generated, and the test is again repeated. This 'repeat-until' process continues until the expected state is achieved, at which point the agent moves on to the next operator.[9] In our *throw* example, the *lift-arm* operator is applied repeatedly until the desired/expected position behind the head is reached. However, if progress stops at some point before reaching the expected state, some problem has arisen and the agent must abandon the procedure. In throwing an object, another person may grab the agent's arm, thus halting the process before reaching the cocked position.

Thus, in closed loop behavior the agent is continually comparing the current state of the world to the next desired/expected state, and continues applying the same operator(s) until an acceptable match is found. In contrast, during open loop behavior the agent bypasses the match process entirely (or uses it only occasionally as a backup). However, this forces one to use some other method for specifying the number of times each operator should be applied.

In our framework, each instance of an operator (within a given procedure) has associated parameters that specify the length of time it should be applied, as well as the degree of

[9] The 'repeat-until' metaphor is only partially correct. Actually, the degree to which an operator is applied is proportional to the difference between the current state and the goal state. In other words, an operator is initially applied with large increments, but these decrease in size as the goal is approached. In using this approach, we are borrowing from extensive earlier work within the framework of servomechanisms (Arbib, 1972).

application. Like the numeric features of object concepts, these parameters contains both a mean and a variance. When a procedure is first acquired, both values are unspecified, but as the agent runs the procedure in closed loop mode, he begins to acquire data about the duration and force for each operator application, averaging the observed values. At any point in the learning process, the agent can run the procedure in open loop mode if absolutely required, ignoring feedback and relying on the estimated values. However, these estimates will be poor early in the learning process, and open loop behavior will usually lead to poor results. As more closed loop experience is gained, knowledge is transferred from the state descriptions (the tests) to the estimated number of operator applications (the generators). Eventually, the agent can run the procedure in open loop mode and perform as well as it would in closed loop mode.

Although the closed loop/open loop distinction is based on data from low-level human motor behavior, we believe that higher level analogs will also prove useful. Basically, we think the distinction will hold for any procedure that requires interaction with the world, and that involves the repeated application of subprocedures. This includes low-level motor phenomena such as throwing a ball, but it also includes high-level phenomena such as navigating around a number of obstacles to a goal object. Of course, this is ultimately an empirical question, but we plan to test the usefulness of both concepts in our research on procedural learning.

Learning by Imitation

Much of human learning occurs through the imitation of others, and we believe this method will prove essential in any environment in which complex procedures can occur. Although one can conceive of an agent discovering the *throw* procedure on its own, it is much clearer how one might learn the concept through watching another agent throw an object. At first glance, learning by imitation appears straightforward: one simply stores away the sequence of actions carried out by the tutor, and retrieves them when required. However, this overlooks some important facts. First, the learner observes only the successive states of the tutor; these are an important component of procedures, but before the agent can run the procedure himself, he must infer the operators the tutor employed. If the new procedure is composed of known procedures, then the learner must recognize the component skills before constructing the higher level skill.

Second, there may be differences between the tutor and the learner, such as their sizes, which hand each prefers to use, and so forth. The learner must take such differences into account through some form of analogy. In this process, the learner must determine the similarities between himself and the tutor, and then use these similarities as a guide to learning. At the same time, he must determine the differences and use his basic problem solving abilities to fill in the gaps.

A third complication involves goals. Humans carry out very few actions without having some goal in mind, and if the learner hopes to acquire useful procedures by imitation, he must determine the goals of the tutor in carrying out an action. Unfortunately, the reasons for the tutor's actions are seldom made explicit; in many cases his procedures are so well compiled that he cannot recall any but the highest level goals. In these cases, the learner

must infer the goal tree of the tutor, and incorporate these goals into the procedures he acquires.[10] Inferring another's goals and subgoals is a challenging task, and may involve recalling similar situations and the goals that were useful in these contexts.

Taken together, these difficulties suggest that the most one can hope to learn by imitation is some abstract plan, in which the state descriptions are approximately correct at best. The agent must then attempt to fill in the details based on his own experience. This is the role of practice, to which we now turn.

Improving Skills With Practice

Skill improvement is complementary to learning by imitation. On the one hand, it cannot occur unless an initial procedure or plan exists, however fragmentary, and we have seen that such plans can be learned by imitation. On the other hand, skill improvement fills in the details of procedures that learning by imitation cannot provide. This view is consistent with current psychological theory concerning motor learning (Summers, 1981). Since we know that humans improve their motor skills gradually, Anderson's (1983) model of knowledge compilation immediately suggests itself as a candidate explanation of this process. However, although Anderson's compilation process accounts for the speedup observed in motor learning, it does not explain the more *efficient* motions that occur with practice. Although something like knowledge compilation may be involved, some additional learning mechanism is also required. Similar comments hold for Laird, Rosenbloom, and Newell's (1984) theory of learning by chunking.

Our model of skill improvement relies on the distinction between closed loop and open loop behavior, and on the assumption that practice during closed loop behavior must precede effective open loop actions. We believe that two related learning mechanisms are in operation. First, the agent runs the new procedure in closed loop mode, observing the extent to which the resulting action leads to his desired goals. In our *throw* example, the agent would (after constructing an initial plan by imitation) throw an object at a desired location. If the results are poor, the agent 'loosens' the match process responsible for comparing observed to predicted states, leading to slight variations in the procedure each time it is evoked. Better results draw the means in the expected state descriptions toward the observed values, while poorer results push the means away from the observed values. In the case of throwing, the initial state descriptions are gradually transformed from ones suitable for the tutor into ones suitable for the learner. Thus, the agent carries out a hill-climbing search through the space of possible state descriptions, using the initial plan as the starting point.

While this process is occurring, a second form of learning is also taking place. Each time a procedure is run, the agent revises the open loop parameters for each operator in the procedure (specifying the length of time and the degree to which it should be applied). However, since the means for each operator reflect the knowledge implicit in the expected state descriptions, these values will not stabilize until the state descriptions have themselves

[10] This makes the task of learning by imitation appear very similar to the language acquisition task, since the language learner must infer an adult's goal tree from a sequence of actions – the words that he utters (Langley, 1983).

been optimized. Meanwhile, the variance of each operator parameter will be high, since the number of times they are applied will vary as the agent experiments with different instantiations of the procedure. Ultimately, the closed loop behavior will stabilize, and the open loop behavior will eventually come to reflect this. However, since the agent never 'knows' when this stabilization occurs, open loop learning must proceed even before this situation is reached. Note that we assume simple averages for the open loop parameters, rather than ones which are weighted by the degree of goal satisfaction. This is because goal information has already been incorporated into the numeric features of the state descriptions used in closed loop mode.

As an example of the basic process, note that one can learn by imitation to go around an obstacle, but one learns by trial and error how far to skirt the obstacle, and one slowly improves with practice (inevitably gaining some bruises along the way). We have already discussed how variants on the basic plan are tried during closed loop learning. This approach is similar to the *perturbation* method used by Kibler and Porter (1983) on more symbolic tasks, but the details of our method differ due to the sensori-motor level representation we employ. Now let us turn to another form of knowledge that is necessary for the robust application of procedures – cognitive maps.

The Construction of Cognitive Maps

Before an intelligent agent can interact with its environment successfully, it must have some model of that environment. Such models have sometimes been called *cognitive maps*, and if we want our system to behave robustly in reactive environments, then we must include this form of knowledge in our cognitive architecture. Cognitive maps serve a number of functions, such as aiding navigation and easing the acquisition of desired objects. Below we consider two quite different forms of spatial knowledge, and relate them to other components that we have already discussed.

Local Cognitive Maps

One common usage of the term 'cognitive map' refers to spatial knowledge of one's immediate surroundings. We plan to represent such knowledge in schemas that state the positions and orientations of local objects in terms of a three-dimensional coordinate system. For instance, such a schema would specify the relative locations of the walls, windows, and pictures in a room. Like object concepts, these cognitive maps summarize experience over long periods of time, and will be 'fuzzy' in cases where the component objects do not have constant locations. For example, the walls and windows in a room never move, while the location of furniture in may change over time. The locations of some objects would thus be more 'criterial' to the cognitive map than others, depending on the degree of variation.

In fact, we feel that such local spatial knowledge is represented in the same manner as object concepts and that it is acquired using the same learning mechanisms. The main difference lies in the degree of constancy that occurs in object concepts and local maps. Most object concepts vary in their relative location to other objects over time, and thus these aspects of their structural descriptions become unimportant. In contrast, the objects

composing a local map remain in the same spatial relationship over time, so locational information remains a significant part of the description. The result is a 'meta-object' that describes both the shapes of stationary objects and their relative positions. In other words, we hope to account for this form of cognitive map without introducing additional knowledge structures or learning mechanisms. This seems an important claim, and one that we should subject to empirical tests.

Local cognitive maps can be used in the same ways as object concepts – for recognizing familiar scenes, for predicting the location of objects, and so forth. However, since they generally describe larger regularities than object concepts, one can also use them for local navigation, planning manipulative acts, and the like. Note that cognitive maps do not explicitly store all the possible relations between component objects (as would be required in a propositional scheme); rather, the numeric information about position and orientation permits the computation of any spatial relation that is needed, when it is requested.

We have argued that local maps are acquired through the same process as that used to form object concepts. However, since the visual field cannot take in the whole surroundings at once, the learner must look in different directions to construct a model of his local surroundings. Moreover, he may acquire knowledge of object positions by different senses (such as sight and touch), which are then combined into a coherent whole. As with object concepts, an initial local map will be quite specific, since it is based on information over a short period of time. But as more experience is gained with this area, variations will be observed and the map will become more general. In some cases, component objects will move freely or be missing entirely, and these objects will eventually be dropped from the description. In other cases, objects may be in two (or a few) stable positions (e.g., a window may be open or closed), and this may lead the learner to split the map into two specializations of the original map. In cases of slight variation, the map will come to contain means and variances similar to those used in object concepts.

Route Knowledge

A second usage of 'cognitive map' refers to the route knowledge used in larger scale path planning and navigation. While local maps are similar to object concepts, these global maps are analogous to stored procedures. Thus, global maps store route information in terms of the operators required to traverse those routes, and they refer to local maps in their state descriptions; the latter can be used as landmarks to check one's progress towards the goal state. Just as procedures are stored hierarchically, so are global maps stored at multiple levels of aggregation, with lower level routes being treated as operators by higher levels.

Route knowledge provides a means for navigating beyond the immediate field of vision. In planning a path to some goal object, one retrieves those maps that connect the goal's position to the current position, and uses the resulting information to constrain the search for a useful route. Kuipers (1983) has proposed computational models of human route planning, while Crowley (1984) has described methods for robotic navigation; both approaches share the notion of using route knowledge to plan useful paths. One can also use such maps to generate expectations which can then be compared against sensory input. This process is

essential in carrying out one's plans, since the location of objects may have changed since the last time they were observed, and one must be able to modify plans dynamically when this occurs.

Just as we intend to model local maps as object concepts, we also plan to model route knowledge as stored procedures, using the same representational scheme and the same learning mechanisms.[11] Although local maps can be formed simply by observing one's immediate surroundings, global maps require the exploration of adjacent local environments. As the agent stores sequences of actions that lead to a successfully thrown ball, so he stores the sequences of steps that successfully lead between two locations. The same processes of learning by imitation and incremental refinement that apply to procedures also apply to route knowledge. And just as stored procedures may refer to object concepts in their state descriptions, so may route knowledge refer to local cognitive maps.

Problem Solving and Heuristics Learning

We have already described our scheme for representing procedures in long-term memory, for using those procedures in recognition and the generation of behavior, and for acquiring this procedural knowledge. However, the very nature of complex reactive environments leads inevitably to some situations that cannot be covered by existing procedures. If an agent is to respond adaptively to such an environment, it must be able to generate new plans and procedures dynamically, and to implement these plans to achieve its goals. The framework of means-ends analysis (Newell, Shaw, & Simon, 1960) provides a general approach that takes advantage of goals to direct the search for useful plans, and we have incorporated this method into our model of performance and learning. Since the resulting plans will be specific versions of the abstract procedures we described earlier, we will focus instead on the representation of goals that lead to these plans. After this, we describe the version of means-ends analysis that we plan to employ, and we outline some methods for learning heuristics to improve this planning process as the result of experience.

Representing Goals

Newell, Shaw, and Simon's (1960) theory of means-ends analysis distinguishes between three types of goals – *transform* goals, *apply-operator* goals, and *reduce-difference* goals. In many ways, the third goal type is redundant, so our reasoning component uses only the first two goal types. Since most of the agent's goals will relate to desired physical states (either in the world or internal to the agent), let us consider how we represent both forms of goals.

Transform goals involve changing the current state of the world into some desired state. For instance, if the agent sees a particular object in the distance and would like to be closer to that object (say close enough to pick it up), then it would create a goal to transform the current situation into one in which it was the specified distance from the object in

[11] Note that this does *not* imply that route planning depends on the particular operators stored with the procedure. If a given operator or subprocedure cannot be applied, one can always resort to problem solving to transform one state description into another.

question. This goal would be abstract in the sense that it would ignore many aspects of the environment (such as the positions of other objects). However, it would be quite specific regarding the agent and the desired object, representing each in terms of the generalized cylinder scheme we described earlier. Of course, the generalized cylinder notation cannot be used to represent internal states, such as hunger or exhaustion, so we must incorporate other notations as well.

Apply-operator goals refer to applying some operator to a specific object, such as lifting one's arm or throwing a ball. In our framework, such operators may be either primitive actions (applying a force to part of the agent's body) or complex procedures (constructed from lower level operators). In both cases, one requires a transparent representation of the operator to employ means-ends analysis, since the method employs both the preconditions on operators and their expected results.[12] We have already described our plans for representing the initial and final states in complex operators in terms of the generalized cylinder notation used for object concepts. The first state of a complex operator (like *throw*) can be viewed as its preconditions, while the final state can be considered as its expected results. Apply-operator goals must also take arguments which further detail how the operator should be applied (e.g., the direction in which to throw an object, how hard to throw it, and so forth).

Problem Solving through Means-Ends Analysis

The method of means-ends analysis centers around the notion of *differences* between states. It can be applied to a problem only if such differences can be computed, making it less general than simpler heuristic search schemes like best-first search. Fortunately, one can compute differences for many of the problems that arise in the real-world, such as the navigation and manipulation tasks that we plan to study. In addition, operators must be indexed either by the differences they reduce, or by their preconditions and expected results, so that they can be retrieved selectively. We tentatively plan to take the latter approach, and to let the resulting differences be represented implicitly.

Let us consider a simple example of reasoning by means-ends analysis as it might be used in our simulated environment. Suppose the agent sees an object in the distance, and creates the high-level goal of changing its current location to one close to the object. It would then look for operators in long term memory that lead to changes in location, retrieving an operator like *slide* (our simulated robot has no legs, but can move by applying force in any direction). The resulting apply-operator goal would specify the operator to be applied, along with various arguments, such as the direction in which the agent should move, in this case the direction in which the object lies. The agent would then apply the *slide* operator repetitively until it reached the desired location.

However, suppose that another object blocks the agent's path, preventing it from applying the *slide* operator (actually, preventing it from having the desired effect). In this case, the agent must create a subgoal to transform the current state into one which satisfies the preconditions on the *slide* operator. In other words, it must find some location which will

[12] This leads to another learning task that we discuss below – the acquisition of transparent operator models from before-after state descriptions.

allow it to move in a straight line to the desired object. Once such a location (to either side of the blocking object) has been identified, the agent searches memory for an operator which will let it achieve this subgoal. Again, the *slide* operator would suggest itself, and an apply-operator subgoal would be proposed. If this had the desired effect (i.e., if there are no further blocking objects), then the agent will achieve the new state, and refocus its attention on the original transform goal.

However, if we assume the agent can only move forward (rather than sideways), then it cannot apply the *slide* operator directly, since it is no longer facing the object. Again it must find some operator that will transform its current state (facing in one direction) into another state (facing in the desired direction). This time the turn operator suggests itself, and since nothing keeps this operator from applying, the agent would turn, and then apply the *slide* operator iteratively until it reached the original goal state near the object. Note that the problem solver does not completely plan out its solution path before actually taking action, but that it is able to recover when complexities arise along the way. This distinguishes means-ends analysis from other planning methods, such as those implemented in STRIPS (Fikes & Nilsson, 1971) and NOAH (Sacerdoti, 1974).

We are not suggesting anything especially new here, and our discussion has been largely a review of an approach to problem solving that was developed 25 years ago. However, we feel that this method has been largely abandoned by the AI community, and that more research remains to be done within the means-ends analysis framework. In particular, we must extend the approach to deal with representations of physical objects and their locations. This contrasts sharply with the purely symbolic approach to robot planning taken in STRIPS, and we believe that it will lead to insights about the representation of plans and methods for generating them.

Learning Operators and Search Heuristics

As we have noted, means-ends analysis relies on transparent representations for the operators used in problem solving. We believe that one can acquire such representations by observing the state of the world before an operator is applied, and comparing it to the state of the world after application. Porter and Kibler (1984) have described an approach to this problem for symbolic domains (solving simultaneous equations and symbolic integration), and Vere (1977) has explored the similar problem of inferring 'relational productions' from before-after pairs. We hope to extend these methods to learning operators for our simulated environment. In particular, we think that many instances of an operator's application must be observed before a general description of its effect can be learned, and that this process holds much in common with the process responsible for learning object concepts.

The second learning process related to problem solving involves acquiring the *heuristic* conditions on various operators. In the heuristic search framework, one begins with a set of *legal* conditions on each operator, but these are not sufficient to eliminate search. For this, one must determine additional conditions that specify the situations under which each operator *should* be applied. Researchers in machine learning have devoted considerable attention to this problem over the past five years (Mitchell, Utgoff, & Banerji, 1983; Langley, 1983; Kibler

& Porter, 1983), and substantial progress has been made.

However, the existing work on heuristics learning has two limitations. First, nearly all approaches have been implemented within the heuristic search framework, focusing on simple strategies such as breadth-first and best-first search. As a result, most of the research in this area has ignored the key role of goals in learning search strategies. Second, while the notion of 'legal' conditions makes sense for abstract problems such as the Tower of Hanoi and symbolic integration, it has much less relevance for problems such as manipulation and navigation. In the latter domains, the question is not so much whether one *can* apply an operator (such as applying some force), but whether this operator will have the desired effect.

Our response to the first drawback is to learn search heuristics within the means-ends analysis framework. One of the standard approaches in the work on heuristics learning is to assign credit and blame on the basis of complete solution paths to some problem (Sleeman, Langley, & Mitchell, 1982). Naturally, this approach works only for problems with relatively small search spaces. However, the means-ends analysis framework provides a 'solution path' every time a goal is achieved, since one knows that the sequence of operators applied while the goal was active led to that goal's satisfaction. In addition to letting one deal with large problem spaces, this method of credit assignment also enables learning during the search process, so that one can take advantage of acquired heuristics before the original (top-level) goal is achieved.

We should note that Newell, Shaw, and Simon's (1960) approach to means-ends analysis assumed an ordering on the differences between states, and this information greatly constrained the search required to solve problems. Although Eavarone (1969) has shown how one can automatically determine the ordering on such differences, we do not believe this approach generalizes to reactive environments. As a result, we believe that our implementation of means-ends will initially require more search than the traditional version. However, we also believe that methods for learning search heuristics from experience can be used to acquire knowledge equivalent to these difference orderings, and that these methods will prove useful within our simulated environment.

Our response to the second issue is to modify the preconditions of an operator whenever it fails to have the expected effect. We assume that the transparent representations for most operators are learned under idealized conditions. For example, instances of the macro-operator for moving an arm forward would not include cases in which an obstacle blocked the arm's motion. As a result, the initial operator descriptions will be overly general, leading the agent to apply them in cases where they will not succeed. When this happens, the learning system will invoke a discrimination (specialization) process, leading to more a specific version of the operator with preconditions that will keep it from being applied to similar cases in the future. Moreover, these additional preconditions will lead to changes in the agent's problem solving process. In the future, if the agent creates a goal to apply the *move-arm-forward* operator and then finds that the new precondition is violated (e.g., an obstacle is blocking the arm), it will create a subgoal to transform the current state of the world into one in which this precondition is met.

Testing the Theory of Learning

Naturally, we would like to test our architectural design and the specific models of learning that we implement within this framework. In fact, this was one of the original reasons for developing our simulated world – to provide a challenging testbed for integrated models of the learning process. This simulated world provides two natural classes of problems on which to test our models – navigation tasks and manipulation tasks.

In many ways, navigation problems appear easier since they do not require fine motor control or balance. Since our simulated robots have no legs, they move about the world by applying linear force (and thus sliding along the floor) and rotational force (and thus turning). In this way, the agent can carry out a visual search for some goal object, avoid obstacles lying in its desired path, and engage in general exploratory behaviors. At first glance these would seem to involve high-level planning rather than fine-grained motor behavior, and certainly one can use problem solving methods to generate useful paths. However, note that the most efficient paths require near-collisions with objects in the environment, and that humans probably acquire such navigational skills gradually by trial and error.

Manipulation tasks require the agent to interface its body parts with objects in the world, and this certainly requires fine control. However, we know that humans initially carry out manipulation procedures in an awkward manner, and our model should begin in this fashion as well. Thus, it should have difficulty in picking up food to eat and may even have trouble placing the food in its mouth. Actions that involve trajectories (like throwing or sliding an object) are even more difficult to acquire, since the agent loses contact with the object before the event is complete. But humans are able to master the fine control of even such complex procedures, and we would like our models to have the same ability.

Naturally, our implementation should also be able to recognize instances of useful object concepts and procedures, and it should be able to make use of both local cognitive maps and route knowledge. However, each of these behaviors should arise in the context of navigation and manipulation tasks, and these should emerge from the agent's innate goals and drives. The problem solving process will lead naturally to subgoals and thus to actual behavior, but we should briefly describe the top-level goals on which these subgoals will be based.

We assume that the agent has a few innate drives such as removing hunger, avoiding pain, and satisfying curiosity. These are implemented in terms of top-level goals that are added to memory under various conditions. For instance, if the agent's energy level drops below a certain amount, a goal to raise this amount would be automatically added to memory. This would lead the agent to retrieve procedures that it knows lead to reduced 'hunger', or to dynamically generate a plan (through problem solving) to satisfy this transform goal. Similarly, when the agent sees an unfamiliar object, a goal to approach the object would be automatically created, leading the problem solver to retrieve navigation procedures or to create a plan to approach the anomaly. The observation of a new action sequence would also attract the learner's attention, leading him to imitate the procedure and apply it to various objects.

Given the complexity of the environment and the likely complexity of our models, objectively evaluating our results will be a difficult process. However, for the interim we will be satisfied to develop initial agents that can interact with their surroundings, acquire object concepts, stored procedures, and cognitive maps, and use these structures to achieve their simple goals. We would also like our models of these processes to roughly correspond to our knowledge of human learning and performance, and in the preceding pages we have outlined some phenomena that we hope to explain. Clearly, we have set ourselves a difficult task, but we believe the nature of that task has already led to insights and approaches that might otherwise have been bypassed.

References

Anderson, J. R. and Kline, P. J. A learning system and its psychological implications. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*, 1979, 16–21.

Anderson, J. R. A theory of language acquisition based on general learning principles. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, 1981, 165–170.

Anderson, J. R. *The Architecture of Cognition.* Cambridge, Mass.: Harvard University Press, 1983.

Becker, J. D. An information-processing model of intermediate-level cognition. AIM 119, Department of Computer Science, Stanford University, May, 1970.

Binford, T. O. Visual perception by computer. Paper presented at the IEEE Conference on Systems and Control, December, 1971, Miami.

Carbonell, J., & Hood, G. The world modelers project: Objectives and simulator architecture. In *Proceedings of the Third International Machine Learning Workshop*, 1985, 14–16.

Crowley, J. L. Dynamic world modeling for an intelligent mobile robot. In *Proceedings of the IEEE Seventh International Conference on Pattern Recognition*, 1984, 207–210.

De Kleer, J. and Brown, J. S. Assumptions and ambiguities in mechanistic mental models. In D. Gentner and A. L. Stevens (Eds.), *Mental Models.* Hillsdale, N.J.: Lawrence Erlbaum, 1983.

Fikes, R. E. and Nilsson, N. J. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 1971, *2.*

Granger, R. Identification of components of episodic learning. *Cognition and Brain Theory*, 1983, *6*, 5–38.

Hayes-Roth, F. The role of partial and best matches in knowledge systems. In D. A. Waterman and F. Hayes-Roth (Eds.), *Pattern-Directed Inference Systems.* New York: Academic Press, 1978.

Hayes-Roth, F. and McDermott, J. An interference matching technique for inducing abstractions. *Communications of the ACM*, 1978, *21*, 401–410.

Keele, S. W. and Summers, J. J. The structure of motor programs. In G. E. Stelmach (Ed.), *Motor Control*, New York: Academic Press, 1976.

Kibler, D. F. and Porter, B. W. Perturbation: A means for guiding generalization. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 1983, 415–418.

Klapp, S. T. and Greim, D. M. Programmed control of aimed movements revisited: The role of target visibility and symmetry. *Journal of Experimental Psychology: Human Perception and Performance*, 1979, *5*, 509–521.

Kuipers, B. Modeling human knowledge of routes: Partial knowledge and individual variation. In *Proceedings of the National Conference on Artificial Intelligence*, 1983, 216–219.

Laird, J. E., Rosenbloom, P. S., and Newell, A. Towards chunking as a general learning mechanism. In *Proceedings of the National Conference on Artificial Intelligence*, 1984, 188–192.

Langley, P., Nicholas, D., Klahr, D., and Hood, G. A simulated world for modeling learning and development. *Proceedings of the Third Conference of the Cognitive Science Society*, 1981, 274–276.

Langley, P. Language acquisition through error recovery. *Cognition and Brain Theory*, 1982, *5*, 211–255.

Langley, P. Learning effective search heuristics. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 1983, 419–421.

Marr, D. *Vision: A computational investigation into the human representation and processing of visual information.* San Francisco: W. H. Freeman, 1982.

Michalski, R. S. Pattern recognition as rule-guided inductive inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1980, *2*, 349–361.

Miller, G. A., Galanter, E., and Pribram, K. H. *Plans and the Structure of Behavior.* New York: Holt, Rinehart, and Winston, 1960.

Mitchell, T. M. Version spaces: A candidate elimination approach to rule learning. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 1977, 305–310.

Mitchell, T. M., Utgoff, P., and Banerji, R. B. Learning problem solving heuristics by experimentation. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach.* Palo Alto, CA: Tioga Press, 1983.

Mozer, M. C., & Gross, K. P. An architecture for experiential learning. In *Proceedings of the Third International Machine Learning Workshop*, 1985, 133–136.

Newell, A., Shaw, J. C., and Simon, H. A. Report on a general problem-solving program for a computer. In *Information Processing: Proceedings of the International Conference on Information Processing*, 1960, 256–264.

Ohlsson, S. A constrained mechanism for procedural learning. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, 1983, 426–428.

Porter, B. W. and Kibler, D. F. Learning operator transformations. In *Proceedings of the National Conference on Artificial Intelligence*, 1984, 278–282.

Rosch, E., and Mervis, C. B. Family resemblance studies in the internal structure of categories. *Cognitive Psychology*, 1975, *7*, 573–605.

Sacerdoti, E. D. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 1974, *5*, 115–135.

Schank, R. C. *Dynamic memory: A theory of reminding and learning in computers and people.* New York: Cambridge University Press, 1982.

Schank, R. C. and Abelson, R. P. *Scripts, Plans, Goals, and Understanding.* Hillsdale, N.J.: Lawrence Erlbaum, 1977.

Schmidt, C. F., Sridharan, N. S., and Goodson, J. L. The plan recognition problem: An intersection of psychology and artificial intelligence. *Artificial Intelligence*, 1978, 45–83.

Sleeman, D., Langley, P., and Mitchell, T. Learning from solution paths: An approach to the credit assignment problem. *AI Magazine*, Spring, 1982, 48–52.

Summers, J. J. Motor programs. In D. Holding (Ed.), *Human Skills.* New York: John Wiley & Sons, 1981.

Vere, S. A. Induction of relational productions in the presence of background information. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 1977, 349–355.

Winograd, T. *Understanding Natural Language.* New York: Academic Press, 1972.

Winston, P. H. Learning structural descriptions from examples. In P. H. Winston (Ed.), *The Psychology of Computer Vision.* New York: McGraw-Hill, 1975.