A COMPUTER-ASSISTED FORMULATION OF AN ABSTRACT MODEL FOR SOME ASPECTS OF NEOCORTICAL LEARNING

by

Stephen Fraser Zeigler

Computer Sciences Technical Report #341

November 1978

A COMPUTER-ASSISTED FORMULATION OF AN ABSTRACT MODEL FOR SOME ASPECTS OF NEOCORTICAL LEARNING

BY

STEPHEN FRASER ZEIGLER

A thesis submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

UNIVERSITY OF WISCONSIN - MADISON

1978

© Copyright by
Stephen Fraser Zeigler
1978

All Rights Reserved

A COMPUTER-ASSISTED FORMULATION OF AN ABSTRACT MODEL FOR SOME ASPECTS OF NEOCORTICAL LEARNING

Stephen Fraser Zeigler

Under the supervision of Professor Larry E. Travis

ABSTRACT

This dissertation presents an abstract model for some aspects of neocortical operation. Contributions in three areas have been attempted: Knowledge representation, learning mechanisms, and neocortical modeling. Recognizer/Predictors (RPs), data structures resembling small, uniform frames, but applied in parallel, are proposed to represent knowledge. Pattern induction is proposed as a mechanism for learning and correcting RPs, by detecting and abstracting regularities from episodic memories of RP usage. RPs and pattern induction are described in the context of MUL, a computer program modeling infant-like development in a simple environment.

The abstract concepts of RPs and pattern induction are used to suggest a new model for neurons of the neocortex. In this model, neocortical neurons begin in an unfixed state in which they have no meaningful output. Each neuron records episodes, storing the activity the neuron receives

at its input synapses from fixed neurons. Each neuron performs pattern induction upon its collection of episodes until it discovers some suitable regularity; it then undergoes fixation, by modifying its input synapses to recognize (become activated by) future instances of that regularity. Fixed neurons correspond to RPs.

Among other mechanisms introduced are default activation and inhibition. Default activation of an RP (or fixed neuron) involves activation in the absence of an instance of the regularity recognized by that RP (or fixed neuron). Such default activation is shown to be useful in initiating motor responses, approximation, generalization, and anticipation of future experience. Inhibition is a mechanism to correct improper defaulting behavior; inhibition is implemented by inhibitory RPs (or inhibitory neurons), and the learning of inhibitions is also accomplished by a pattern induction process.

Among the advantages claimed for the proposed neocortical model:

- A definite, unchanging meaning is assigned to each neuron.
- Neurons may discover regularities in the activities of other neocortical neurons, to recognize progressively more complex concepts.
- Neural activation patterns may be meaningfully recorded as episodic memories.

- Pattern induction provides a plausible mechanism to assign meanings to neurons, even in the absence of direct feedback or obvious examples, provided by an external "teacher", and in spite of noise or malfunction.
- Pattern induction, and the resulting RPs (or fixed neurons), are useful independently of the meaning of the inputs to the RP or neuron.
- The proposed neocortical model is compatible with biochemical and neurophysiological evidence of neocortical function.

Acknowledgements

I deeply appreciate the efforts, encouragement, and guidance of Professor Larry Travis. His invitation to participate in the TELOS design and development project precipitated the first successes of this research, as well as providing valuable experience and financial support.

Marty Honda and Rich LeBlanc know that working with them and Larry on TELOS was the highlight of my university career.

I would like to thank the other members of my committee, especially: Charlie Fischer, whose humor provided perspective as his teaching provided knowledge; and Raphael Finkel, whose insights were as valuable as his rewrites.

Thanks also to my friends in Madison, and to Thom, Linda and Bill for the roost and boost.

I wish, finally, to thank my family, especially my parents. They gave me curiousity, persistence, and an appreciation of knowledge, as well as their love.

I dedicate this dissertation to Marti, my friend, my co-worker, my support, and my inspiration. You have given me a past filled with joys, yet with each dawn fulfill a promise of a still better future.

TABLE OF CONTENTS

Chapter 1: Introduction

"We, like most AI researchers, wish to build a learning machine. We believe that the most promising approach is to model natural developmental stages (as exhibited by children), and that a learning machine will use processes similar to those employed by children."

- R. Shank [Shank 77]

This dissertation presents a model for aspects of intellectual development in the human infant, specifically for some operations of the neocortex in learning to perceive and respond to the environment. The model evolved in response to explorations of the properties of computer simulations. Speculation about the brain has influenced the model, but no attempt was made directly to imitate brain parts during program development. Once the model stabilized and reached its final form, in order to evaluate its plausibility and credibility, attempts were made to correlate it with neuroscientific hypotheses concerning neocortical structure and operation at the level of individual neurons.

The research is intended as a contribution in three related areas, knowledge representation, learning, and

neural modeling. Later sections of this chapter review previous work by other researchers in each of these areas. Chapter 2 describes the evolution of the computer model's knowledge representation. Chapter 3 defines the learning mechanism used to build knowledge for the model. Chapter 4 presents the operation of the model as it applies learned knowledge to recognize and anticipate its sensory inputs, describing results from program executions. Chapter 5 compares aspects of the program's structures and behaviors with those of neurons in the cortex, touching upon the roles of subcortical elements. Chapter 6 concludes with a review of the model's salient features and suggests some areas for future work.

1.1 The world of the neonate

The neonate faces a difficult task, at least difficult for observers to understand. It receives massive amounts of input having diverse meanings and varying importance. It must control a body by manipulating hundreds of outputs affecting both its relations with the external environment and its internal body states. No external teacher is easily or generally available at this stage: the infant cannot isolate a teacher as an object, nor is there a teacher always present. Elders, when available, do little more than

move the infant about, feed it, and clean it. It must begin to make sense of all those inputs and outputs on its own.

The infant, though without a teacher, is not without help. Evolution has provided, within the infant's body, useful machinery. Many outputs are controlled by specific "hard-wired" (that is, non-learned) subsystems. Many inputs are preprocessed by genetically preprogrammed processors that may cause specific motor responses in certain simple but crucial situations (e.g., reflexes). An important question is "How much of the infant's activity is genetically preprogrammed and how much has to be learned?"

This research assumes that a fairly large repertoire is preprogrammed, but that learning may occur in many ways and at many places in the brain. A recent popular book, "The Dragons of Eden", gives a readable description of assumptions about brain organization adopted for this research [Sagan 78]: the many parts of the brain are categorized into three subsystems: 1) the reticular system, composed of the reticular formation and many special-purpose brain elements (e.g., the collicullus superior); 2) the limbic system, composed of the limbic cortex and several other elements including the thalamus and hippocampus; and 3) the neocortex, a thin, crumpled, comparatively uniform sheet of about 10 billion neurons, accepting inputs from and producing outputs to the lower layers of the brain.

These three subsystems are presented above in order of decreasing evolutionary age, of increasing architectural uniformity, and - it is assumed - of decreasing influence from genetics on specific function. Learning is thus assumed to play a correspondingly increasing role in the development of function in these subsystems. The "lower" limbic and reticular systems play the dominant role in the responses of the infant, as it must depend on preprogrammed activity until it can learn. The neocortex is thought to assume partial control of various body parts as it learns their operation and effects. However, the lower systems are not to be considered "replaced" by the cortex. They remain active, carrying out the actions initiated by the cortex and continuing their preprocessing roles for sensory input. They can even seize or assume control from the cortex during panic situations or during familiar tasks like walking.

The cortex is taken to be the site of most learning in the brain. Too little is known about the cortex for its operation to be deduced directly. Instead, the research began with the following unsupported and vague hypothesis for how learning <u>might</u> occur:

> In the infant, new knowledge is obtained by detecting and abstracting <u>regularities</u> in memories of past experiences.

This hypothesis differs only in form from speculations of Aristotle and others on what has since been called

"intuitive induction" [Cohen and Nagel 34]. Regardless of form, the hypothesis raises at least the following questions:

1) How is the knowledge to be represented?

2) How are past experiences to be remembered?

3) What are regularities in experience?

4) How can regularities be detected and used to construct new knowledge?

1.2 Background: knowledge representation

The field of artificial intelligence has seen the development of a variety of structures for the representation of knowledge. This research has been strongly influenced by the representation structures of L. Uhr and J. Becker.

Uhr is concerned primarily with visual perception, attacking the problem at about the same level as does this research [Uhr 74, Uhr 75, Uhr 77]. That is, his SEER model receives input on a retina; that input is preprocessed by "hard-wired" elements before being delivered to the learned perception mechanisms. Uhr suggests that knowledge can be encoded by structures he calls transforms.

"A transform specifies:

A) A set of If-conditions to be satisfied,

B) a Threshold test for success or failure,

C) a set of consequences, the Implieds,

D) a Region within which the transform is to be applied, and

E) a set of Expectations." [Uhr 74, p.18-19]

Uhr presents a structure he calls the "recognition cone" to control transform application. The cone consists of a sequence of rectangular buffer arrays, diminishing in size to a single-celled apex. Each transform is bound to a particular buffer (perhaps to a subregion of that buffer), examining the contents of that buffer (or region) and looking for information that satisfies its If-conditions. Whenever the If-conditions are satisfied sufficiently to pass the Threshold test (probabilistic weights are used), then the transform succeeds and the Implieds are effected. Implieds could specify values (names) to place in cells in the next buffer toward the apex of the cone, or suggest another transform to apply, or direct that some motor output be performed.

The recognition cone structure has found application in this research, as will be seen in later chapters. While transforms are not employed, several of their properties are considered to be important for knowledge representations:

1) hierarchical construction - each transform might use results of other transforms.

2) probabilistic applicability - experiences are not exact, but may be incomplete, noisy, or otherwise incorrect.
3) parallel applicability - at least in part (layer by layer, for example).
4) directed application - the ability to suggest other transforms to apply.

The principal reason that transforms were not used directly was their complexity. Understanding of the learning mechanisms we were investigating required their application to simpler structures.

The work of J. Becker on intermediate level cognition contains a proposal for a simpler knowledge representation, and some suggestions as to how it might be learned. In Becker's words:

> "The heart of JCM [as Becker called his model] is a particular formal structure for representing information. Such a structure, called a "schema", is essentially a sequence of observations in which one observation has gained predominant emphasis. To illustrate this very sketchily, let us suppose that a schema has the form:
>
> [Sensation 1 -> Action 1 ->
>
> Action 2 => Sensation 2]

> The element 'Sensation 2' is emphasized, as is shown by the big arrow, '=>', preceding it. This schema may be interpreted as saying 'Sensation 1, followed by Action 1, followed by Action 2, leads to Sensation 2'."
>
> [Becker 73, p.396]

Sensations and actions are each encoded as kernels, composed of predicate and arguments. For example, the information that Becker's model senses the color red in a retina cell number 01532 would be expressed by the appearance of this kernel:

<color 01532 red>

Input to Becker's JCM is a stream of kernels. The goal of the model is to encourage the appearance of certain "pleasurable" kernels by introducing appropriate action kernels. Schemata are used in a goal-directed manner: the model searches for schemata having pleasurable kernels emphasized; the unemphasized portions of these schemata are then compared against the currently received kernels; if needed action kernels are not present, they may be introduced; if needed sensory kernels are not present, then the entire schemata utilization algorithm can be recursively applied with the missing kernels marked as "pleasurable". The schema that entails the fewest (or easiest) actions to obtain a pleasurable kernel is applied, in the sense that the action(s) it suggests are actually performed.

Becker's schemata have several appealing features. Their structure is simple enough to be an object of comprehensible learning mechanisms. They are meant to operate at about the same sensory/motor level as was planned for this research. They have an elaborate weighting system (not illustrated in the example) that allows their probabilistic application, making them suitable for inexactly-observed (real) environments. Schemata also have deficiencies. Their goal-directed, serial application scheme is not generally useful since the vast majority of sensory kernels are neither good nor bad. Schemata are not hierarchically formed, although they may be used cooperatively. Finally, schemata have no facility for suggesting other schemata to be applied.

The representation structures developed for this research have their roots in Becker's schemata. More will be said about them in the next section (concerning Becker's proposals for learning) and in Chapter 2 (where our knowledge representation structure is detailed).

Productions, as developed in [Newell and Simon 73, Waterman 75] and reviewed in [Davis and King 77], share many properties with Becker's schemata. The general form of a production is:

<antecedent item(s)> => <consequent action(s)>

Productions (as used by Newell and Simon) are applied to a "short-term memory" (STM) containing a limited, ordered set of objects. A production is applicable if the items in its antecedent match objects in the STM. When a production is applied, the action(s) specified in its consequent part are performed.

There are many variants of productions, and many different ways to determine which production(s) to apply. In "pure" production systems (i.e., Newell and Simon's), the antecedent items and the contents of STM are atomic symbols. All productions are ordered in a list. During the application cycle, the productions are examined in order until a production is found applicable - that is, its antecedent symbols appear in proper order in STM. This production is then applied. Variations on this scheme allow more complex objects in STM (e.g., assertions, or Becker's kernels) and correspondingly more complex descriptions in the antecedent part. The application system also has variants, including goal-directed application mechanisms similar to Becker's.

Productions have, or can be altered to have, properties (hierarchical construction, probabilistic and parallel applicabilities, directed application, and simplicity) desirable in a knowledge representation. When kept close to their "pure" form, they are simple. Their application can be made probabilistic by the addition of weights like those

on Becker's schemata. They could be applied in parallel, as is suggested in [Uhr 78]. They could be hierarchical in construction, by putting special objects in the consequent part in order to "key" higher-level productions. Several variants of productions suggest further productions to apply, either with such special objects or by direct specification of productions in some consequent function. However, were all the above alterations made, productions might lose the very important property of simplicity.

Another knowledge representation that has had an important influence on this research is the frame proposed by Minsky [Minsky 75, Kuipers 75]. As an information structure, the frame is a record-like structure whose fields are called slots. When a frame is used, its slots are filled with appropriate values or objects, possibly including other frames. Associated with each slot are, at the least, restrictions on the kind of objects or values that can be put in the slot. Other slot information might include indications for steps to be taken when the slot cannot be filled, might provide a default value for the slot or suggest ways to get a value, or might suggest other frames that might be more applicable.

Frames are of greater complexity than Becker's schemata, especially since their semantic interpretation in the AI literature generally concerns abstractions of entire

situations or complex events. When considered strictly as data structures whose size can be arbitrarily small, frames contribute a counterpoint for schema-like representations. They also contribute the idea that no slot need be "emphasized" (in Becker's words), and that all slots can be filled either with actual or default values. These ideas will be discussed at greater length in Chapter 2.

Transforms, schemata, productions and frames are not an exhaustive list of knowledge representations that have influenced this work. Others will be mentioned in the next section, which discusses the learning of knowledge representations. Also, this research included participation in the design and implementation of TELOS, a programming language based on PASCAL and intended for research in artificial intelligence [LeBlanc 77, Travis 77]. The strengths of this language significantly influenced the approach to the research by allowing a very flexible evolutionary approach to knowledge representation development. TELOS and its contributions to this research are discussed further in Appendix 1.

1.3 Background: learning

The learning investigated in this research is a matter of detecting and abstracting regularities in memories of

past experience. The approach is to be distinguished from other current investigations of learning models in several ways:

1) It is at the level of individual inputs to and outputs from the neocortex. Cortical inputs may have straightforward interpretations (for example, red is seen at retina position 01532), complex interpretations (for example, "there is a moving edge crossing a particular spot of the retina"), generated by visual preprocessing elements, or unknown interpretations (most inputs to the neocortex have unknown functions). The learning model is therefore designed to be independent of the meanings of its inputs and outputs. (For purposes of exposition, discussions in this dissertation will involve easily-interpreted sensory and motor inputs.)

2) The number of observables is expected to be <u>very</u> large. Observables include not only the original cortical inputs and motor outputs, but also internally generated observables corresponding to features detected or predicted among the other observables. Further, the observables arrive in parallel.

3) No "teacher" is present (externally). Experience is <u>not</u> broken into neat pieces called examples. Nor is

> essential input information especially segregated from unimportant input information. Finally, there is no direct feedback as to what is good or bad, except perhaps in some few situations where pleasure or pain is involved.

Learning in the absence of examples and feedback has not been a popular topic in recent AI work. A perusal of the "knowledge acquisition" section of the proceedings of the fifth international conference on AI [IJCAI 77] confirms Feigenbaum's observation that research on learning has moved toward automation of knowledge acquisition and away from unassisted general learning [Feigenbaum 78]. For example, research on skill acquisition for Meta-Dendral [Buchanan 72], for annotated production systems [Goldstein and Gumson 77], and for interactive transfer of expertise [Davis 77] are too domain-specialized and too linguistically oriented (and thus at far too high a level) to have much significance for our problem of developing mechanisms and structures applicable to general perceptual learning.

Other work, while not directly applicable, is sufficiently general that concepts have been influential. Winston's research introduces a method for learning predicate-net descriptions of block structures, by providing examples, non-examples, and "near-misses" [Winston 75]. While his work is too concerned with examples to be directly

applicable here, it does introduce the useful idea of comparing two complex descriptions to assess similarities and differences. Hayes-Roth generalizes these ideas to Interference Matching, a technique that finds a maximal common description given two or more descriptions of examples [Hayes-Roth 77, 78]. Interference matching is general enough to be applied with any knowledge representation, but is computationally intractable and requires heuristic solution. Since interference matching tends to discard irrelevant knowledge, the technique could be applied in situations where examples are less neatly delineated. However, given current computing economics, the expense required for such matching is intolerable.

Michalski attacks a related problem of generalizing rules from incomplete descriptions [Michalski 77]. Generalizing rules as such is, in itself, a knowledge acquisition problem not faced at the level of learning in this research. What is of interest here are Michalski's mechanisms for generalizing rules: dropping predicates from a description; introducing variables for constants; listing simple exceptions (by indicating that an expression is true except for values in a particular set); generalizing to ranges for linearly-ordered values, or to first-common-ancestor for tree-ordered values; introducing new functions to describe the generalizations. While these

generalization techniques were eventually discarded in this research, they did inspire fruitful experimentation.

Becker, mentioned in the previous section for his knowledge representation, proposes some methods for learning his schemata. He suggests that totally new schemata can be introduced when an input kernel is not matched by the emphasized part of any schema. That kernel becomes the emphasized part of the new schema. The new schema is additionally composed of kernels that precede the now emphasized kernel in STM. In addition, Becker speculates that unneeded parts of old schemata can be deleted by altering the many weights associated with the schemata. Finally, Becker suggests that when a schema is unreliable - that is, its emphasized part does not always appear when the actions it includes are taken - the unreliable schema can be discarded or modified by differentiation. Differentiation is a process that adds requirements for new kernels to the unemphasized parts of old schemata, in order to further specify the conditions producing the emphasized part.

Learning in accordance with Becker's suggestions can be performed in the absence of an external teacher. However, it raises several problems. Because of the large number of parallel inputs expected, there is no easy way to select kernels for the unemphasized parts of new kernels. Learning by weight modification often produces oscillating weight

values - not much information is preserved in a weight. The lack of hierarchical representation in schemata will make complete schemata very cumbersome - meaningful situations cannot often be described in three or four sensory/motor kernels. Further, if schemata can be made hierarchical, then modification of a schema would be a serious affair, in effect altering the meanings of any schema that references it.

The idea that learning can be accomplished by detecting regularities in memories of past experiences is being explored by Langley [Langley 78a, 78b], concurrently with this research. His system, called BACON.1, is currently being programmed and no results are yet available. BACON.1 appears to be a descendent of Lenat's AM, itself an interesting approach to learning by discovery [Lenat 76].

Langley's BACON.1 operates in simple environments similar in complexity to those used for tests in this research. However, the concepts involved are high-level ones. For example, his model is being programmed for planetary motions with the aim of discovering Kepler's laws. Toward that end, he specifies several heuristics to recognize constancies and trends among incoming data. These heuristics specify rather precise actions. For example, one states that if the value of an attribute of object x is consistently less than the value of that attribute for some

object y, then introduce a new attribute calculated as the ratio of the values for the two objects.

BACON.1, while applying its learning to higher level problems than are approached in this research, is encouragingly similar in intent to that of this research. The results of the two projects may well fruitfully contrast and/or interact.

1.4 Background: neuroscience and psychology

As was indicated in the introduction, neuroscientific and psychological concepts and theories played only an indirect role in the development of this research. The concepts and theories were consulted after the fact (i.e., after our model had stabilized) to see whether they added any credibility to it and whether they suggested possibly interesting interpretations. Specific sources used in this consultation are cited at the appropriate point of reference within the dissertation. The remainder of this section is intended only as a brief overview of some of the important literature covering an immense field of research, starting from psychologists examining the whole person and working down to chemists examining molecules.

For insight into the intellectual development of the child, the work of Piaget remains classic [Piaget 54, Piaget

and Inhelder 69]. A more modern treatment is given in [Hamilton and Vernon 76], a collection that also abstracts many representative experiments from experimental psychology. For computer models of development in the child, see [Shank 77]. All of these works characterize the infant's development in terms of stages, illustrating the stages with observations on emerging capabilities. Section 5.6 in this dissertation presents such a characterization.

For comments on current theories concerning the architecture of the neocortex, [Brazier and Petsche 78] is useful. Of particular interest in this collection is [Creutzfeld 78], which supports the idea that the neocortex is a single uniform organ differentiated only by the nature of the inputs it receives. Theoretical models of the cortex are often based on the work of Hebb, who introduced the concept of the cell assembly [Hebb 49]. Inhibitory effects were later introduced into cell assemblies [Milner 57]. A group of researchers including B. Zeigler and S. Kaplan have had some success in statistical and computer-simulation modeling of such cell assemblies [Whitehead 77]. In the tradition of Hebb, these workers suggest that each assemblage of neurons represents and recognizes some entity. There have also been interesting attempts to model systems of individual neurons [Witte 73]. The whole field of neuronal modeling is given a good review in [MacGregor 77].

There has been a great deal of work on biochemical analyses of neuron functions. Recent work is covered well in [Davison 77], which discusses biochemical correlates of brain structure and function. Modeling done at this level is largely a matter of mathematical characterization of properties of synaptic transmission and impulse propagation [Scott 77].

1.5 An overview of MUL

The model developed in the next chapters is called MUL, an acronym for Method Underlying Learning. Since the method investigated involves "mulling" things over to find regularities, 'MUL' is doubly appropriate.

The following chapter, Chapter 2, gives a historical evolution of MUL's knowledge representation structure. In its final form, called the RP for Recognizer/Predictor, the structure is even simpler in form than schemata or productions. It resembles a mini-frame having a small number of slots (called components), usually no more than two or three. Each component references another RP. RPs are used as recognizers in much the same way as Uhr's transforms: an RP is "activated" as a recognizer when all the RPs referenced by its components are activated at appropriate times. Under certain conditions, an RP may cause the default activation of some RPs referenced by its

components, a process similar to defaulting for frames. Improper defaulting may be prevented by a mechanism called inhibition. A unique inhibitor RP is associated with each RP; an RP may not be activated by default if its associated inhibitor RP is already active.

The RP, with inhibitions, appears to have a number of advantages over other knowledge representations. For example, the desired properties listed in Section 1.2 - hierarchical structure, parallel application, probabilistic application, and directed application - will be shown to be properties of RPs. Other advantages are discussed later. In addition, RPs are simple and thus are good candidates for the object of general learning. Chapter 3 describes the pattern induction learning method implementing the detection of regularities in memories.

Finally, RPs and inhibitions can be related to cortical neuron structure, as is done in Chapter 5. That chapter suggests correspondence with evidence from neuroscience and from experimental psychology.

Chapter 2: Knowledge representation

The structures representing knowledge in MUL must meet three requirements. They must be learnable in the sense that a relatively small program can fabricate such structures in a reasonable period of time. They must be usable in the sense that the program can apply them to recognize and anticipate its experiences. They must be powerful in the sense that the program must recognize and differentiate a variety of possibly complex relationships.

Like the rest of MUL, the knowledge representation medium has evolved during this research. To give an understanding of the final form of the representation, the stages of that evolution will be presented in an historical fashion.

2.1 Becker's schemata

The proposals of J.D. Becker [Becker 73] inspired MUL's initial knowledge representation. His schemata, discussed in Chapter 1, have the general form:

[condition(s) -> action(s) => result(s)]

When the antecedent condition(s) are met and the action(s) are performed, then the result(s) may be expected.

Sensory information is provided to Becker's model in the form of kernels, predicate-argument structures described in Section 1.2. To provide a source for concrete examples, the next section presents a variant of Becker's shelf environment.

2.2 The shelf environment

The shelf environment is a smooth shelf nine units long populated by tall smooth white blocks and short rough black blocks. All blocks are one unit wide. Short ones are one unit high, and tall ones are two units high. Blocks are positioned so that they do not overlap unit boundaries. Our organism sits before the shelf and may move left or right in one unit jumps. Its eye sees the 3x3 unit section of shelf immediately before it, including a spot of red where its hand lies. The hand may touch the top, middle or bottom units of the shelf, but must remain directly in front of our organism. Ten kernels of sensory information are provided to our organism at the beginning of each time unit: nine retina kernels (e.g., <color bottom middle red>) and one touch kernel for the texture of the object immediately behind the hand (e.g., <texture smooth>). For each time unit, the organism may select a motor output from among: <move left>, <move right>, <raise arm>, <lower arm> or <rest>. The output will be executed if possible.

2.3 Becker's schemata illustrated

The following is a schema for the shelf environment as it might appear in Becker's proposed long-term memory.

<color(10) middle(0) left(7) none(10)> (10) -> (10)
<move left>(9)
=> <color(10) middle(1) middle(8) none(5)> (10)
charge: 0 confidence: 9 cost: 2

It consists of a set of kernel descriptions (partially) ordered by little and big arrows. The numbers in parentheses are weights between -10 and 10 associated with the immediately preceding schema feature. A high positive weight means that the associated feature must be present if the schema is to be applied. A high negative weight means that the associated feature must not be present. A weight near zero indicates that the associated feature is not important. Components of a kernel description that have near-zero weights will be replaced by variables for expository purposes only - no change is meant in Becker's representation. Such variables are prefixed with a "?", following the MICROPLANNER convention. The weight on each kernel description indicates whether a kernel must (or must not) be found to match that kernel description. The weight (10) on the first arrow ("->") means that the <move left> must be generated following the observation of a kernel matching <color ?x left none> or the remainder of the schema

cannot be expected. The variable "?x" in our revised notation means that the matching kernel may be for any of top, middle or bottom retina positions because the weight on middle is zero.

The above schema therefore has the following interpretation: if a <color top left none>, a <color middle left none>, or a <color bottom left none> kernel is observed, and MUL causes a <move left> kernel, then a kernel matching <color ?y middle ?c> should be observed. From Becker's original schemata, c has a strong but not absolute requirement to bind to "none". Becker proposes that the weights result from learning. A generalization operation, for instance, might continue to weaken the requirement that ?y be bound to "middle" in Becker's schemata. (The process for establishing and adjusting these weights is only weakly defined in Becker's proposals.)

The charge of zero indicates that the desirability of obtaining a kernel matching <color ?y middle ?c> is neutral. The cost of two indicates that, once this schema is suggested by the occurrence of a kernel matching <color ?x left none>, two units of energy must be expended to create a <move left> kernel. Finally, the confidence of nine indicates that 90% of the times this schema has been activated, a kernel matching <color ?y middle ?c> has been seen.

2.4 The recognizer/predictor (RP)

Becker's schemata lack descriptive power because they are not hierarchical in construction. A schema is formed of descriptions for kernels, and kernels represent - for Becker - only primitive sensory inputs or motor outputs. In order to specify the conditions of its applicability, each schema must include a long list of descriptions for sensory and motor kernels that should or should not be present. Even in the simple shelf environment, quite complex schema are necessary to describe, for example, occasions when the color black is to be expected at the lower left; since only conjunctions of kernel descriptions are allowed in schemata, a separate schema is needed to represent each such occasion.

Becker's mechanism for schema application is also inappropriate for our present modeling. Becker requires that sensory information arrive and schema be applied in series - parallel arrival and application are desired for MUL.

These problems forced the alteration of both kernel and schema. To facilitate hierarchical knowledge representation, kernels were given a general role: instead of representing only primitive sensory/motor information, kernels were changed to represent an instance of any situation or feature of a situation that can be represented

by an RP. An RP (Recognizer/Predictor) was introduced as the new knowledge representation unit. This early version of the RP retained the structure of the schema except that, because activity proceeds in parallel, Becker's arrows (as indicators of temporal sequencing) were no longer useful. As an example, the schema discussed in Section 2.3 could be partially represented by this RP:

#102: (<color ?x left none> <move left>) =>
<color ?y middle ?c>

Sensory and motor kernels are still present, of course. They are to be regarded as activations of preprogrammed RPs.

Learned RPs like #102 are used in a way resembling the use of productions in production systems, except that more than one RP may be used at a time. RP #102 could be used when a <move left> kernel occurs in parallel with at least one of <color top left none>, <color middle left none>, or <color bottom left none>. In general, however, kernels will have the form:

<RP_identifier {actual parameters, if any}>

(Notation: items within '{' and '}' are optional.)

For example, when RP #102 is activated, a kernel will be created having #102 as its RP_identifier and having as its three actual parameters the values that were bound to the three variables of #102, ?x, ?y, and ?c.

Such RPs are activated when kernels are received that match all the kernel descriptions (components) of the RP.

For example, RP #102 has three components. Suppose a <move left> kernel, a <color top left none> kernel, and a <color bottom middle black> kernel are concurrently received. Then RP #102 could be activated, producing a new kernel, <#102 top bottom black>. This kernel commemorates a particular instance when the situation described by #102 was actually observed.

If such RPs are activated only when all their components have been matched, then they act only to recognize situations. This version of the RP could also be used in predicting occurrences by interpreting its right-most kernel description to be "emphasized" in Becker's sense. If all the other kernel descriptions can be matched with actual kernels, then MUL can anticipate a kernel matching the emphasized kernel description.

Such RPs are not yet effective structures for knowledge representation. They have inherited from schemata a tendency to become meaninglessly overgeneralized. For example, #102 anticipates the arrival of a kernel matching the description <color ?y middle ?c>, an anticipation that will inevitably be met; the anticipation is therefore useless. (The next section discusses possible remedies for overgeneralization.) Also, parallel kernel arrival mitigates the usefulness of temporal relationships among kernels, but at least gross inter-kernel timing must be expressible. (This timing problem is discussed in Section

2.7.) To their credit, however, such RPs can build on the descriptive power of other RPs to describe complex situations. For example, such an RP might now contain a kernel description for RP #102's activation kernel.

Though such RPs may be built to represent arbitrarily complex events, each individual RP may be of bounded (even a small fixed) size. For example, it is conceivable that (<#1000...> <move left>) => <#1902...> might represent the appearance of a black block at the left of the visual field after a move left from a particular spot on the shelf: RP #1000 describes the particular spot and RP #1902 describes the appearing black block.

2.5 Interparameter relations

Unfortunately, this early RP design inherited from Becker's schemata an inability to preserve relationships among the values bound to parameters when the RPs are generalized. For example, the following RP has 4 parameters (?x,?c1,?c2, and ?t):

103: (<color bottom middle white> <arm ?x>) =>
(<color middle middle ?c1> <color top middle ?c2>
<texture ?t>)

If #103 is to be used to predict incoming kernels, then considerably more information is needed. As it stands, an application of #103 predicts that a kernel matching <texture

?t> will be observed. But this condition is always true, since some (perhaps null) texture is always felt. It is of greater interest to predict a value for ?t. This prediction is possible; when x = 'raised', then t = 'none', otherwise t = 'smooth'. Similarly, both c1 and c2 can be calculated given knowledge of the value of x.

Interparameter relations are intended to save this information. Our example, RP #103, might have been represented in this way given such relations:

(<color bottom middle white> <arm ?x>) =>

(<color middle middle f(!x)> <color top middle g(!x)> <texture h(!x)>)

where

(x=raised) => (f(x)=white, g(x)=red, h(x)=none)

(x=centered) => (f(x)=red, g(x)=none, h(x)=smooth)

RP #103 might then have an RP activity kernel of <#103 !x> since all other parameters are dependent on x.

Of course, RP #103 is a particularly simple example for such interparameter relations. One could easily imagine RPs for which parameter values might be functions of several parameters, some or all of which appear only in the "result" or right-hand side of the RP. Still worse, the interparameter relationship may not be single-valued (i.e., may not be a function). In these cases, the expected kernels can not be uniquely described - any of a class of kernels may fulfill the expectation.

Although a mechanism was formulated for the representation of arbitrary relationships among parameters, it was abandoned because the learning procedure for the relationships was very similar to the procedure used to learn new RPs themselves. The information contained in the relationships can not actually be lost except through overzealous generalization. For instance, in our example RP #103, the information could be preserved by maintaining two distinct RPs:

103.1 (<color bottom middle white> <arm raised>) =>
(<texture none>...)

103.2 (<color bottom middle white> <arm centered>) =>
(<texture smooth>...)

The overzealous generalization expected in Becker's proposed system was not a problem in this research. The learning procedures employed preserved the distinct RPs #103.1 and #103.2. Alternative mechanisms for generalizing are presented in Section 2.11, involving RPs that behave as disjunctions of kernel descriptions.

2.6 Defaults, and the removal of "=>"

As the role of the RP shifted toward recognition and away from Becker's goal-pursuit, the concept of prediction gave way to the broader concept of defaults. From the above production-like form, having a distinguished right-hand

side, RPs were altered to a frame-like form, with any component being defaultable. This change in RP structure was accomplished by a corresponding change in the "goal" of RP usage, from an attempt to produce certain kernels to an attempt to anticipate all kernels. Both changes were motivated by the change in the meaning and behavior of kernels, as is explained below.

Kernels now represent instances of arbitrarily complex situations because they represent activations of arbitrarily complex RPs; kernels may now appear or fail to appear for correspondingly complex reasons:

1) A kernel may not have appeared yet, because the situation it represents may not have occurred or been recognized yet. In this case, it may be useful to create a matching kernel on the anticipation that such a kernel will be observed. Further expectations can then develop on the basis of this expectation.
2) A kernel has not appeared, because the situation it represents has not and will not occur.
3) A kernel has not been obtained because of interference from other kernels. Interference is often observed in the computer model, where time and space limitations dictate heuristic implementation. Processing to determine which RPs can be activated at any given time may be quite expensive, so some

deserving RPs may not be activated, or their activation forgotten. Time or space resource limitations in the computer model might cause such lapses.

4) A kernel may not have appeared because it in turn awaits the appearance of another kernel. Determining if a kernel can appear may therefore involve determining if other kernels can appear, potentially an exponentially expensive task.

5) A kernel may fail to appear because of noise or sensory malfunction.

Although Becker's schemata and learning mechanisms are adequate for prediction (#1 above), they can be expected to behave very poorly for the other cases. Whenever a schema matches in all but one kernel, Becker's generalization process tries to remove that kernel from the schema. Removal may be an acceptable strategy when only simple sensory kernels are present, but compounded RPs introduce the cases described by 3 and 4 above. Case 5 would cause instability in the schema feature weights.

These weaknesses make it productive for RPs to behave as though any unmatched component were a consequent. When suitable matches are found for some of the components, the RP creates a "pressure" towards finding matches for the remaining components. Such pressure causes a search for an existing (but overlooked) kernel. If the search fails, a

sufficiently great pressure causes the construction of an appropriate kernel, called a default kernel. Default kernels may be later supported, either because an appropriately matching actual kernel is eventually experienced, or because kernels matching the default's components are later encountered or defaulted.

Forming a default kernel is a rather drastic step. Until support for the default is found, the default represents an imaginary fact. Defaults should therefore be constructed only when the probability of finding support is high. To estimate that probability, RPs maintain a statistic for each component recording the historical success of defaulting that component, as measured by whether the default was eventually supported. This statistic appears as a factor between 0.0 and 1.0 in parentheses after each component. A value near 1.0 indicates that defaults of the associated component have been very successful. A value near 0.0 indicates that no defaulting should be done for that component.

2.7 Representation of time in RPs

Becker's proposed program requires serial input of kernels and assumes that the order of arrival of the kernels is important. In MUL, on the other hand, kernels arrive and are processed in parallel; the relative timing between

appearance of kernels is not very predictable. In addition, because MUL may choose to terminate processing on a kernel due to resource limitations, some RPs activated by each kernel may not be applied immediately. (They may be applied later as a result of searches triggered by defaulting.) Finally, kernels may appear out of normal recognition sequence because of defaulting.

For all these reasons, inter-kernel temporal sequencing in MUL carries less information than for Becker's schemata. However, at least gross temporal sequence information must still be represented. In MUL, components of an RP may be required to match kernels observed in preceding time units. A time unit begins with the input of all sensory kernels that represent the current state of MUL's environment, and it continues long enough for MUL to perform some (but likely not all possible) steps of perception and learning from this new information.

For example, the following RP was formed in this version of MUL:

#143: (<color middle middle red> (0,0) previous
<color middle middle red>(.8))

This RP recognizes the concept that the hand is seen in the middle position of its vision field at two consecutive time units. The fractions in parentheses on each component measure the past success MUL has had in defaulting each component should it not be observed. The attribute

"previous" may be attached to any component of an RP with the effect that a kernel matching that component must be present, not in the current, but in the immediately previous time unit.

Episodic memory, discussed in more detail in Section 3.2, maintains a record of each kernel including the time unit in which it occurred or was expected to occur. In this way, MUL may receive the actual sensory inputs from time unit t, but enter expected kernels for t+1, t+2, and so on into the future, or for t-1, t-2, and so on into the past. Expectations about the future are evaluated for correctness when the corresponding time unit is actually experienced. Expectations about the past must be verified by existing kernels. [In what follows, the time of occurrence or appearance associated with a kernel indicates the time unit associated with that kernel in episodic memory. This time bears no direct relation to the "current time", which names the time unit actually being experienced, and is important only to the environment simulator program.]

A typical activation of RP #143 would begin with the appearance of a <color middle middle red> kernel at some time t. The kernel may appear because of sensory stimulation (MUL is actually experiencing time unit t) or because the activation of some other RP caused the appearance of this kernel by default (MUL is experiencing t-1, or t-2, or so forth). Whatever the cause of its

appearance, this kernel matches either component of #143, if #143 is examined for possible activity. If the kernel is bound as a match for the second component, then #143 will be activated at t if <color middle middle red> has appeared for the time unit t-1. Because the default coefficient of the first component is zero, no attempt is made to default <color middle middle red> at t-1. When the new kernel is bound as a match for the first component of #143, the second component suggests there is an 80% probability that a similar kernel will appear in the following time unit t+1. If no such kernel is already expected at t+1 (it may have appeared as the result of defaulting by other RPs), then RP #143 would cause the second kernel to appear as a default. When time t+1 arrives, sensory input of <color middle middle red> would validate this default; otherwise the default is in error. Of course, unless some mechanism inhibits the use of RP #143, MUL could reapply RP #143 to the defaulted kernel to default a similar kernel at t+2, then at t+3, and so forth. The importance factor maintained for each kernel is one mechanism that limits defaulting purely on the basis of other defaults.

2.8 The importance factor

MUL must continually make decisions concerning allocation of its time and space resources. These decisions

are based on a measure of each kernel's potential importance to MUL's activity. One contributor to importance is the confidence MUL has in a given kernel. Since default kernels can be introduced in the absence of direct support, they are not as confidently accepted (and thus have diminished importance). In turn, RP activity kernels activated by imagined kernels have even less importance.

Unreliable kernels aside, MUL must still face resource allocation problems. For example, a MUL process might build thousands of RPs to distinguish various nuances in the orientation of grass blades in a lawn. Even if all their recognitions and defaults are absolutely trustworthy, a single RP capable of anticipating a potential disaster might deserve greater attention.

Section 4.3 discusses a "limbic" mechanism for evaluating the potential benefit or danger of events. Such a mechanism is fundamental not only in resource allocation, but also in general motivation and choice (see Section 4.9). In MUL experiments with the shelf environment, the performance (behaviors) of the programs never reached sufficient complexity for questions to have to be decided on motivational grounds. Another environment, discussed in Section 4.9, did require motivational direction for MUL's activities.

2.9 Examples of perception by RPs

MUL's perceptions of and responses to its environment are indicated by its RP activity. The RP activation process is discussed in detail in section 4.1. We pause now to follow examples of that process.

Suppose at time t MUL sees a black block toward the left of its visual field. Information that the color black is present in that area is transmitted by activation of appropriate sensory RPs, some of which might be:

RP# 7: componentless, sensing "color lower left black"

RP#11: componentless, sensing "color middle left black"

RP#14: componentless, sensing "color upper left none"

[Note: "componentless" RPs have no components (that is, no kernel descriptions) and therefore are never activated by recognition. Each componentless RP has a specific duty or task in MUL. For example, sensory RPs like #7, #11, and #14 above are connected to sensory devices and are activated by conditions external to MUL. Motor output RPs like #4: componentless, requesting "lower arm", are connected (indirectly) to motor effectors for MUL; activating a motor RP - possible either by default or by "reflex" - results in an attempt to perform the associated motor activity. Inhibitory RPs, to be introduced in Section 2.12, are componentless RPs that act to prevent the default activations of certain other RPs. In general,

componentless RPs are given meaning both by the duty assigned them and by what other RPs can activate them by default.]

These sensory RPs may be activated asynchronously or, as in the MUL versions examined, synchronously with each step of the environment. (Synchronous environment "stepping" enhances the understandability of MUL activity.)

At time t, RPs #7, #11, and #14 will be activated by the presence of the black block. The activation of these RPs may in turn cause activations of other RPs. Suppose MUL has learned an RP of the form:

RP#105: (<color lower left black>(0.0)
<color middle left black>(0.0))

When RPs #7 and #11 are both active concurrently, all conditions are met for the activation of #105. Given a sequential machine, detecting that an RP, in this case #105, can be activated is too expensive an operation to perform for each of hundreds of RPs. For the purpose of simulation, only important activity will trigger computation. Suppose the activation of #7 at time t is sufficiently important to trigger computation. Then MUL finds RPs that have a component matched by an activation of #7. RP #105, by virtue of its first component, is such an RP. The other component of #105 is then inspected to determine if #105 can become active. Since the activation of #11 matches this

second component, #105 is activated: an RP activity kernel for #105 is created in time t.

Sometimes RPs are found to be only partially activated (have at least one unmatched component). For instance,

RP#106: (<color lower left black> previous (0.0)
<color lower left black> (0.9))

In the case of RP #106, the activation of #7 at t matches the first component, suggesting that #106 might be activated in the next time period t+1. Presuming that no activation of #7 at t+1 has already been defaulted, MUL will anticipate that #7 will become activated at t+1 on the strength of the 0.9 statistic for success in defaulting #106's second component. MUL is still experiencing time t; the anticipated kernel enters episodic memory for time t+1, and will be supported at that time if RP #7 is indeed activated by sensory input when t+1 is experienced.

In addition, the activation of #7 at t matches the second component of #106 if #106 is activated at t. MUL will then search to see if #106's first component can be matched at t-1 (i.e., was #7 active at t-1?). If a black object was present in the left of MUL's visual field at t-1, then #7 will have been activated and #106 can be activated at t. If not, the first component will not be defaulted. The low statistic (0.0) indicates that defaults of this first component have not been successful.

Some RPs may cause defaults of higher level RPs, which causes effort toward verifying the default. In the case of RP #150 below, the activation of #7 matches component one.

RP#150: (<color lower left black> (0.0)
<RP#120> (0.9))

Component two of #150 anticipates the activation of some RP#120. For instance:

RP#120: (<color lower right white> (1.0)
<color middle right white> (0.6))

RP #150 indicates that MUL has reason to believe that if black is seen in the lower left, then there is a good chance of seeing a tall white block to the right. Since RP activity for the right side of the retina may not have been examined yet, #150 can cause future examination to look for a white block by defaulting RP #120. Later processing may verify this default.

This example is meant only as an introduction to the fundamentals of MUL perception. Details of the process are presented in Chapter 4.

2.10 RPs in relation to frames and productions

At this stage in its evolution, the RP shared some properties with the frame formalism of Minsky [Kuipers 75]. RP structure was more uniform than that of frames: such RPs have only a small number of uniform slots (usually 3 or

less), each with an index of the suitability (measured by historical success) of filling the slot by default. The uniform structure facilitates learning. In this version of the RP, slots are filled by any kernel that 1) occurs in the proper time period; 2) represents activation of the proper RP; and 3) contains any required values.

Such an RP is unlikely to possess the same "unified" interpretation as a frame. The RP will not, in general, categorize an object or characterize a scene. Rather, it is lower level: it represents a piece of an object (in a particular context), or a part of a scenario. Such RPs are uniformly small compared to frames. However, the interaction among RPs can result in a "mass" of RPs performing the more complex functions of the higher-level Minsky-type frames.

RPs lack the features of frames that perform various error-recovery functions. Frames can suggest alternative frames when serious anomalies prevent the continued use of the current frame. In masses of RPs, this direction can be performed by any RP: suggesting an alternate RP is as simple as defaulting an RP activity kernel for that RP. Previous computations (recognition and default by RPs) remain usable in support of the new RP, analogously to the computations which establish "terminals" that may be applied to alternative frames in frame theory.

The adaptive property of frames, their ability to adjust to minor changes in supporting information, was not yet a demonstrable property of this version of RPs. RPs had to be made more flexible, able to recognize generalized concepts. This problem is the subject of the next section.

The major difference between perception by frame and perception by RP is that many RPs are active, while only one frame is active at any time. MUL's perception of its experiences is therefore difficult for a human observer to decipher. (Performance evaluation is discussed in Section 4.4.)

The RP has advantages over frames for the study of learning mechanisms because its small size and uniform structure require less implicitly coded high-level knowledge in the programs.

The RP and its use share some features with productions [Davis and King 77]. Like pure productions, RPs have a simplicity of form conducive to machine learning. RPs have no separate consequent part; components may sometimes be defaulted, however. This default process, especially since it results in kernel creation, is analogous to the introduction of items into the processing stream during production triggering. In addition, defaulting a motor-output kernel requests the performance of that motor

activity, in much the same way that evaluation of a production's consequent may precipitate motor activity.

Some differences between RPs and typical production systems are important, however:

- RPs are meant to operate in parallel: many RPs may exist in partially matched states at any given time.
- The order in which kernel patterns are matched is unimportant, except where defaulting may result.
- Typical production systems apply productions to kernels present in a small "working memory". While MUL employs a working memory (called the short term memory or STM), kernels are not required to be in that memory when causing RP activity. The only requirement is that the kernels exist in the proper relative time units of the very large episodic memory.
- RPs are not required to have any defaultable components, nor are they restricted from having several. In this way, RPs can behave as though they have several, or no, consequents.
- RP activation causes creation of an appropriate RP-activity kernel.
- RP use does not involve backward chaining, since no specific goal is present.

RPs may be activated in both a bottom-up (recognition) mode and a top-down mode (default). It may be argued that

RPs are a special variety of production, just as they may be considered small and primitive frames. The RP structure appears to occupy middle ground, combining the structure of frames with the diminutive size and simplicity of productions.

2.11 Generalization in RPs

The RP structure discussed in preceding sections was used in the first MUL versions [Zeigler 78]. However, it was not the final structure explored. One source of difficulty was the use of incompletely-specified components (for example, <texture ?t>) in defaulting. Defaulting an incompletely-specified component can be effected either by creating an incompletely-specified kernel corresponding to the component or by creating a "typical" kernel matching the component. Alternatively, the default of inexactly-described components could be forbidden. Both of the latter possibilities were investigated experimentally - the incompletely-specified kernel possibility is equivalent to defaulting a completely-specified component referencing an appropriately generalized RP. (For example, the default of <texture ?t> can be achieved by defaulting RP #155 if #155 has the single kernel description <texture ?t>.) Forming a "typical" kernel has the benefit of establishing a connection with episodic memory, since the construction of a

typical kernel is accomplished by searching episodic memory for a good match (see Section 4.2). Problems arise, however, in detecting whether the default was later supported - the typical kernel may not be easily associated with the actual kernel appearing later. Not only does this lack of associability make the evaluation of default success difficult and expensive, it also produces many similar new kernels that disrupt the perception process.

A second problem arose in designing a language for inexact description. Becker's inexact descriptions could by degrees require a particular value, forbid a particular value, or accept any value. This language is not very expressive, but it is easy to implement. A variation on Becker's descriptions is to specify a set of permissible values rather than a single value. Value sets were used in early MUL versions. While more expressive than the all-or-one patterns of Becker, they still could not represent requirements concerning permissible combinations of parameter values. Certainly, sets are not the only language for expressing interparameter relationships and parameter restrictions; Michalski discusses a number of such mechanisms together with methods for introducing them during an induction learning task [Michalski 77]. However, his mechanisms all rely on orderings on possible parameter values. Orderings are difficult to preprogram for parameters of sensory RPs (how is "red" order-related to

"black"?), and can not be anticipated for learned RPs. Such order-based generalization techniques were therefore not employed in MUL.

A third problem with inexactly-described components is that they are computationally more awkward than ungeneralized RPs, as examining a single referenced RP is easier than discovering and examining each of a set of RPs matching some description. While this problem is one of implementation on a serial computer, its effect is to squander resources that could better be directed toward learning, the real object of this research.

The need for inexact component descriptions is eliminated by introducing a new and different kind of mechanism to represent generalization. RPs are already generalizations in two ways:

- Each RP deals with only a small number of the many kernels present. The RP has generalized that the other kernels are irrelevant for its purposes.
- RPs may be applied in the absence of particular kernels, by defaulting them. Defaulting allows the recognition process to be "inexact", in that kernels may appear either by recognition or by default, and even when appearing by recognition may be supported by other defaults.

When MUL is presented with an entirely new scene, it is capable of partly interpreting the scene by using RPs which

generalize in the above ways. These ways are not sufficient, however. The RP, as it stands, can only represent conjunctions. Defaulting, which allows kernels to appear for reasons other than recognition, is performed only on the expectation that the default will be supported - that its components will be matched, or (if it is sensory) that it will correspond to actual input, or (if it is motor) that its associated action will be performed.

What is needed is a mechanism that can represent disjunction, to be activated if any of its components is matched without requiring anything of its unmatched components. An RP structure was designed to accomodate disjunction by becoming active if any of its components was active, but this design was not used: instead, the effect of disjunction is achieved by introducing another variety of componentless RP, called disjunctive RPs. Unlike the already present sensory/motor RPs, these new RPs are not connected to any external faculty. Therefore they can be activated only by default, and require no later support.

To illustrate the correspondence between componentless disjunctive RPs and disjunctions, suppose that a disjunction of some RPs #1 and #2 is of interest. Introduce three new RPs:

#3: componentless

#4: <#1>(0.0) <#3>(1.0)

#5: <#2>(0.0) <#3>(1.0)

Given these RPs, when #1 is activated, #4 will be activated, causing the default activation of #3. Similarly, when #2 is activated, #5 will be activated, causing a default activation of #3. The result: #3 is activated if either #1 or #2 is activated and no further support is required. The behavior of #3 corresponds to the behavior expected for a disjunction of #1 and #2.

Disjunction could have been represented by the new RP mentioned above that requires the activation of only one component to become active itself. Componentless RPs are used instead for several reasons:

- Componentless RPs are already present in MUL, so no additional mechanism is required.
- Componentless RPs may be activated by any number of other RPs. For example, adding some RP#6 to the disjunction represented by #3 requires only the addition of:

  #7: <#6>(0.0) <#3>(1.0)

  Thus, componentless RPs can correspond to disjunctions with a variable number of disjuncts.
- Each element of the disjunction (#2 above, for example) may not precisely describe an element of the disjunction. By using a default mechanism to activate a disjunctive RP, the problem of refining the descriptions of disjuncts reduces to the more general problem of controlling defaults, a problem solved in the next section.

The meaning of a disjunctive RP is thus far determined only by what other RPs can default it. However, the general meaning of RPs cannot be allowed to change because that would invalidate episodic memory and alter the meaning of any RPs that reference the altered RP. Disjunctive RPs, then, must be assigned some particular meaning if they are to be useful in MUL. The next section introduces the inhibitory RP as one type of disjunctive RP with definite meaning.

When RP components are required to reference a particular RP, rather than inexactly describe some class of RPs, the form of the RP is as follows: each RP is a (possibly empty if componentless) list of several other RPs (with timing information). The format for RP display in the remainder of the dissertation will be:

RP #n: (<#component 1, factor 1 {,previous} >
<#component 2, factor 2>)

Where

#n: The identifying number of this RP.

#component 1: The identifying number of the RP referenced as the first component.

factor 1: A fraction between 0.0 and 1.0. If factor 1 exceeds .75, then component 1 can be defaulted, with confidence proportional to the value of factor 1. Factors below .75

are shown as 0.0 to emphasize that they prevent defaulting.

previous: An optional attribute whose presence indicates that component 1 is to be active in the time unit preceding the activation of component 2, and any resultant activation of RP #n.

#component 2: The identifying number of the second component's referenced RP.

factor 2: The same as factor 1, but for the second component.

Componentless RPs are described as such, and a brief explanation of their meaning is provided. For example:

RP #n: componentless, sensing "color middle middle red"

RP use is straightforward: when all RPs on the component list are active at appropriate relative times, then the containing RP becomes activated; default activation of components may occur when a sufficient number of the other component RPs are active, provided that the history of such a default is favorable. (Further control of defaulting is discussed in the next section.) The kernels representing activations of the components of a kernel are called the <u>support</u> for that kernel. Kernels that originate by recognition always have such supporting kernels for each component, and are therefore "completely supported". Kernels that originate by default may lack support for

several, even all, components. Ideally, all kernels are eventually completely supported. "Indirect support" for a kernel refers to the support (including indirect support) of the of the support for that kernel.

2.12 Inhibition

The RP structure described in the preceding section was studied in the second major MUL version (discussed in Chapter 4). The process of defaulting was found to be too inexact, however. The only restraint on defaulting had been a simple measure of historical success. For example, consider the defaulting activity of RP #143 from Section 2.7. In its now ungeneralized form, RP #143 looks this way:

RP #143:(<#15,.2,previous> <#15,.8>)

where RP #15 is a sensory RP representing <color middle middle red>.

When RP #15 is activated at times t and t+1, RP #143 can be activated for time t+1. If MUL is currently experiencing time t, no sensory information has yet arrived for t+1, so #15 can not already be active at t+1 by sensory activation. (#15 might be active at t+1 due to defaulting by other RPs.) In any case, #143 will insure the activation of #15 at t+1 with an importance factor about .8 that of #15's activation at t. This reduction in importance reflects the statistical history of such defaulting: about 4 out of 5 times that #15

is activated in some time unit, #15 will also be activated in the following time period. This RP has a straightforward interpretation: if MUL's hand is in front of its eye, about 4 out of 5 times the hand will still be there at the next time unit.

Such a default, though often correct, is not as useful as one would like. The more confidence MUL can place in a default, the greater the usefulness of that default. Unfortunately, even the problem of deciding whether or not a default is "correct" is not well defined. The only method yet mentioned has involved verifying whether or not the default receives suppprt - but defaults may not receive support for several reasons without being incorrect: they may not have been sufficiently important to command resources for support detection; they may involve speculation about past events that have been partially erased from episodic memory; they may be unsupported because of noise or sensory malfunction. Worse, incorrect defaults may receive support from other incorrect defaults.

For the purposes of this section, we assume that at least some incorrect defaults of a given RP can be detected as incorrect. Further, we assume the existence of some mechanism that discovers RPs whose activation forewarns that defaulting the given RP will be incorrect. That is, RPs that are often activated before the given RP is incorrectly defaulted, but which are seldom activated when the given RP

is correctly defaulted. Mechanisms to perform these tasks are described in Chapter 3.

With these assumptions, suppose that RP #15 above is being incorrectly defaulted. The assumed mechanism is able to suggest that activation of RP #126 below is often observed in the time unit before improper defaults of #15.

RP #126: (<#5,0.0> <#15,0.0>)

where RP #5 is the motor RP meaning 'lower arm'.

If MUL can be warned that defaulting #15 will be incorrect when #126 is active in the previous time unit, then many improper defaults of #15 will be avoided.

The mechanism that warns of improper defaulting is called Inhibition. With any RP, there may be associated a single, unique, componentless inhibitory RP. For example, when it is first decided that #15 is to be inhibited, the following componentless RP is created:

RP #402: Componentless, inhibiting #15

Whenever MUL is about to default an RP, it first determines if the inhibitory RP associated with the RP to be defaulted is active in time unit t. If so, no defaulting is done. If not, the default activation proceeds as usual.

MUL can prevent the improper defaulting of #15 by activating #15's inhibitor, #402, at appropriate times. RP #500 activates #402 whenever #126 is activated, warning MUL not to default #15.

RP #500: (<#126,previous,0.0> <#402,1.0>)

The following is a reiteration of the RPs accumulated to govern the default of #15:

15: componentless, sensing 'color middle middle red'

#143: (<#15,previous 0.2> <#15,0.8>)

5: componentless, requesting 'lower arm'

#126: (<#5,0.0> <#15,0.0>)

#402: componentless, inhibiting #15

#500: (<#126,previous,0.0> <#402,1.0>)

In operation, these RPs have the general meaning:

> If the hand is centered at time t, anticipate that the hand will also be centered at time t+1 unless the arm is to be lowered.

These six RPs are not a complete RP mass for #15. Other RPs than #143 could default #15; other RPs than #500 could default #402 to inhibit #15.

The concept of inhibition and its implementation via componentless RPs has several convenient properties.

- Inhibitions control defaulting without altering the content or meaning of existing RPs. It is important that RPs have fixed meanings. First, the learning mechanism depends on an episodic recording of past experience. That recording is made by remembering which RPs were active at what times. If the meaning of an RP were allowed to change, the episodic memory for that RP would be invalidated. Second, RPs are hierarchical. If an RP may

change in meaning, then all RPs that reference that RP directly or indirectly will also have altered meanings.

- Inhibitory RPs represent, semantically, the negation of the RP they inhibit. For example, the activation of #402 can be interpreted as meaning "the color of (middle,middle) is NOT red". Inhibitory RPs are a form of disjunctive RP. Their disjuncts are all RPs that default them, each disjunct representing a situation when the inhibited RP should not be activated.
- Inhibitory RPs are themselves activated by default; thus their behavior may in turn be controlled by other inhibitory RPs. For example, #402 can itself be inhibited if it is being defaulted at inappropriate times.
- Inhibitions provide the foundations for the detection of incorrect defaults. Inhibitions prevent activation by default only. They do not prevent activation from sensory input or by recognition. What, then, is the significance of the concurrent activation of both an RP and its inhibitor? The occurrence of this situation, called a conflict, is evidence that an improper default has occurred. For example, suppose #402 is activated for time unit t, but when t is experienced, #15 is activated by sensory input. Then #402 has been improperly defaulted. More will be said about conflicts later in the dissertation.

Chapter 3: Learning by pattern induction

Learning in MUL is accomplished by introducing new RPs. The method for learning is pattern induction: a study of the history of past RP activity (and/or RP structure) to discover a pattern (regularity) in that activity. New RPs are then constructed to describe a typical instance of that pattern to enable recognition and anticipation of future instances of the pattern.

3.1 An Example

The following is a scenario of the progress of learning as it occurs in later MUL versions. Later sections of this chapter explain the inner workings of pattern induction, here mentioned only for its outputs. To simplify the presentation, only the following five sensory/motor RPs are mentioned. The actual MUL interface with the shelf environment has 44 sensory/motor RPs.

RP #1: componentless, sensing "color top middle red"

RP #2: componentless, sensing "color top middle none"

RP #3: componentless, sensing "color middle middle red"

RP #4: componentless, sensing "color bottom middle red"

RP #5: componentless, requesting "raise arm"

The history of activity for these RPs might reveal a number of patterns. For instance, the color at the top middle retina position will always be none (unless the hand is raised) because no blocks are 3 units high. Pattern induction might therefore produce the RP:

#10: (<#2,0.90,previous> <#2,0.95>)

The interpretation of RP #10 is that if RP #2 is active at time t, there is a 95% probability that #2 will be active at t+1. Conversely, if #2 is active at t, there is a 90% probability that #2 is active at t-1. This RP's activity will be refined when further pattern induction discovers that when #2 is incorrectly defaulted at t+1 by RP #10, the <raise arm> RP #5 is present. In the most advanced MUL version, this new knowledge is represented by the introduction of one or two new RPs (if #2 already has an inhibitor, then #11 is not needed - the old inhibitor is used):

#11: componentless, inhibiting #2

#12: (<#5,0.0,previous> <#11,1.0>)
<if "raise arm" seen at t, then default "NOT color top middle none" at t+1; however, the activation of #11 at t does not suggest the default of #5 at t-1>

RP #2 cannot be defaulted at t+1 after any RP defaults #11 at t+1. The componentless RP #11 represents the situation "the color of (top, middle) is <u>not</u> none".

This RP mass will be further refined when later experience exhibits cases where #2 is improperly inhibited by #11. In particular, if the hand is lowered (that is, rests in bottom, middle) and the <raise arm> RP #5 is activated, the resulting activation of #12 activates #11. But #2 is activated by external sensory information because <raise arm> moves the hand 1 unit to the (middle, middle) position.

This situation is an example of a conflict: the activation of #2 says that the (top,middle) retina spot sees "none", but the activation of #11 says that the color seen in the (top,middle) is NOT "none". Pattern induction might remedy this conflict by inhibiting #11:

#13: componentless, inhibiting #11

#14: (<#4,0.0,previous> <#13,1.0>)

The resulting RP mass, composed of RPs 10,11,12,13 and 14, can be verbalized as follows:

> "Predict that the color of (middle, middle) will remain none except when <raise arm> is activated and the arm is not lowered."

This RP mass is not yet completely accurate in anticipating the activity of #2 - certainly <lower arm> can have effects comparable to <raise arm>. The important thing is that the RP mass can be further refined if required by experience.

In general, then, the task of pattern induction is to discover patterns of RP activity exemplified by "RP x (or some combination of RPs) regularly occurs at some interesting times". Later sections of this chapter describe mechanisms for performing this task, including a MUL-specific interpretation for "interesting".

3.2 Episodic memory: the history of MUL experience

A history of MUL experiences is the foundation of pattern induction. This history, called episodic memory, is composed of one record for each RP activation in each time unit. Each record, called a kernel, identifies the time unit containing it, the RP it represents an activation of, the kernels for activations of its components (if any), and (if it originated by default) the kernel for the activation of the RP that defaulted it. Since both the sensory/motor experiences of MUL and MUL's processing of real or imagined experiences are represented by RP activations, the episodic memory is a history of both mental and physical experience.

Kernels are erased (forgotten) only when no space remains in the machine. In current MUL versions, the oldest kernels are forgotten first (FIFO). More sophisticated forgetting schemes would assess the relative importance of kernels to future pattern induction, perhaps themselves using inductive mechanisms.

3.3 An overview of pattern induction

The task of pattern induction is to detect and describe regularities in memories of past experiences - that is, implement learning as hypothesized in Section 1.1. Given that knowledge is to be represented by RPs, the task can be rephrased: detect regularities in the RP activation history of episodic memory, and describe those regularities by introducing new RPs. Still undefined, however, is the nature of a regularity and the mechanism for detecting one.

Becker's suggestions for introducing new schemata provide a model for one sort of regularity, called a predictive regularity. When an RP is activated unexpectedly, MUL may be motivated to find grounds for anticipating that activation in the future. (An RP activation is unexpected if it is not a default: default activations anticipate experience.) Becker's approach - building a new schema from the kernels that preceded the unexpected kernel - is inappropriate in MUL: RP activations occur in parallel, complicating the concept of "preceded"; RP activations are multitudinous, making such selection of components for new RPs impossibly arbitrary. MUL can, however, make use of its episodic memory. Instead of attempting to build new RPs on the basis of just one unexpected activation, MUL waits until several activations of that RP have been unanticipated. New RP(s) are built to

reflect the similarities in events leading up to the unexpected activations. Further, episodic memory may be examined for occasions when the (unanticipated) RP was not activated, to detect differences between the events leading up to activations and the events that did not lead up to activations. The process of assessing similarities and differences in sets of examples and non-examples is similar to the experimental methods of agreement and difference as described by John Stuart Mill [Cohen and Nagel 34]. The process is also comparable to that investigated by Winston [Winston 75] and later by Hayes-Roth [Hayes-Roth 78], except that the choice of incident, and of examples and non-examples of that incident, is left to MUL. The mechanism implementing the discovery of predictive regularities, called predictive pattern induction, is detailed in Section 3.7.

Predictive regularities are neither easily described nor easily discovered among RP activations for the shelf environment. An RP is created primarily to recognize a feature of the flow of RP activations. If (as yet) unobserved RP activity can be predicted, the default mechanism can represent it and inhibitions can make those predictions reliable. But the central problem was one of introducing RPs to recognize features of the flow of RP activations, the task of <u>non-predictive</u> pattern induction. The implementation used in this research depends on a

selection process to suggest an "interesting" RP activation. This process is not so straightforward as selecting an unanticipated activation, although unexpectedness may be one factor in the selection (see Section 3.4). Once an activation is selected, episodic memory is searched to find other activations of the same RP. Subsequent investigation attempts to discover similarities among the events (RP activations) accompanying activations of the selected RP, similarities that can be abstracted as new RPs. Occasionally, the new RPs can default (and thus anticipate) future activations of the selected RP. Defaulting roles are assigned to the components of the new RPs by later analysis of the new RPs' activity when applied to episodic memory. Most components are not found to be defaultable; most RPs act only to recognize events.

Both predictive and non-predictive pattern induction are used in MUL. Outlines of algorithms for both varieties are presented later in this chapter. Non-predictive pattern induction is used to introduce new RPs to define and recognize new features of an environment. Predictive pattern induction is unsuitable for learning arbitrary new RPs because few features involve predictive relationships. Predictive pattern induction, on the other hand, is used to introduce inhibitions that prevent improper defaulting. Non-predictive pattern induction is unsuitable for introducing inhibitions because the new RPs must anticipate

a particular event, namely an impending conflict, and prevent it by activating an inhibitory RP.

3.4 Instance selection

Both predictive and non-predictive pattern induction begin with the selection of an instance, an anomalous occurrence that, if more were known about it, would have been anticipated or avoided. In predictive pattern induction, the object of attention is a kernel that caused a conflict and therefore should have been inhibited. The mechanism for choosing such a kernel is described in Section 4.5. Briefly, the kernel must have appeared by default, not have been completely supported, and have caused a conflict. In the case of non-predictive pattern induction, the object of attention is a kernel selected as "important" to MUL and was unanticipated. Various measures of importance are examined in the remainder of this section.

One measure of importance is the "importance factor" associated with each kernel for answering resource allocation questions (Section 4.3). This factor is primarily a measure of the confidence MUL has in a kernel. As such, it is of some use in deciding what kernel to learn about: kernels that have high importance factors represent real experiences; those having low importance factors are supported only by defaults and so in some sense represent

imaginary experiences. Pattern induction is best confined to dealing with real experiences (see Section 4.6).

Importance factors are also a function of an "intrinsic" importance of the kernel to MUL. Kernels are given higher importance factors if they represent unanticipated sensory experiences. Intrinsic importance may also be assessed by specialized mechanisms. For example, when pain and pleasure sensors are added (Section 4.9), kernels for their activation are considered very important.

Importance factors alone are not sufficient to select a particular kernel as a subject for non-predictive pattern induction if only because many kernels may have equally high importance factors. What is needed is some measure of whether more should be learned about a kernel. Many MUL versions selected kernels that precipitated the fewest activations of RPs, on the assumption that little must be known about the kernel if it caused little RP activity. Choosing kernels that precipitate the greatest RP activity can also be rationalized: if the kernel precipitates many RP activations, then it must be a good kernel to anticipate (its arrival is telling MUL a lot).

3.5 Cause-based relations between kernels

Non-predictive pattern induction implements a search for similarities among RP activations at several particular

time units. The search is too expensive to be performed over more than a small fraction of the RPs activated during a single time unit, as 200 RPs are likely to be activated each time unit even for simple environments like the shelf. Therefore, a small number (about eight) RPs are selected for examination, forming a set called RELATED_RPS. Ideally, activations of the selected RPs are the most likely to be related (as cause-effect or by common cause) to the appearance of the "interesting" kernel, K, selected for pattern induction. In practice, the RPs must be selected by the heuristic methods described below.

In early versions of MUL, RELATED_RPS was filled by random selection on the rationale that activations of several RPs are likely to be causally related to the appearance of K, and random choice would be likely to net one of them. The choice was restricted to RPs activated in the same time unit as, or one unit before the appearance of K. Random choice results in a large number of essentially useless RPs, exemplified by:

(<color middle middle none> <color top left none>)

Performance markedly improved with the introduction of additional relatedness requirements to control membership in RELATED_RPS. These requirements were preprogrammed and at first applicable only to sensory/motor RPs. For example, when non-predictive pattern induction is applied to activations of the sensory RP <color middle right black>,

preprogrammed restrictions confine RELATED_RPS to RPs for neighboring retina positions (e.g., <color lower right black>) or to the RP sensing the color seen in the (middle right) position in the immediately preceding time unit.

Spatial and temporal contiguities limit RELATED_RPS during low-level sensory learning, but must be generalized to apply to learned RPs. MUL later employed an area/level construction to define relatedness among RPs, similar to Uhr's recognition cone in both effect and justification [Uhr 77]. Each sensory/motor RP is assigned to some area of level 1 of a recognition-cone-like formalism. In general, RELATED_RPS may contain only other RPs at the same level and area as the RP whose activation is recorded by K, and that are activated either during the same or the immediately preceding time unit as K. Any new RP formed is assigned to the next higher level in an area given by a preprogrammed transition function. In all current versions, the transition function is a simple fixed mapping from area to area, but could conceivably be more complex (see Section 6.3). The area/level mechanism is a crude way to organize RPs into categories in which each RP deals with similar information at a similar conceptual level. Level 1, for example, contains an area receiving inputs from sensory RPs from the bottom three retina cells, another area receiving inputs from the middle three retina cells, and so forth. At

higher levels, the number of areas diminishes until, at levels 4 and higher, only one area is present.

3.6 Non-predictive pattern induction

This section presents an outline of the algorithm used for non-predictive pattern induction.

1) [select an interesting kernel K]
   By the methods discussed in Section 3.4, a kernel, K, is selected as an instance requiring further learning. To review, K will generally be unanticipated and have a high importance factor. Further selection is based on lack of use as an RP component (in early versions) or on RP activity precipitated (in later versions). As a notational convenience, the function ACTIVATED_RP (K) designates the RP for which K is an activation, for any kernel K.
2) [find history of ACTIVATED_RP(K)]
   Construct a sample, PAST_OCCURRENCES, of other past time units in which ACTIVATED_RP (K) is also active.
3) [build RELATED_RPS]
   Construct a sample, RELATED_RPS, of RPs whose activations might be useful in anticipating K or might be of importance when seen to co-occur with K. As explained in Section 3.5, RELATED_RPS is generally a

random sample of RPs from the same level and area as ACTIVATED_RP (K), and active either in the same time unit as, or one time unit previous to K.

4) [abstract a history of the activations of RP(s) in RELATED_RPS in the time units of PAST_OCCURRENCES]

For each RP, R, in RELATED_RPS Do

For each time unit, T, in PAST_OCCURRENCES Do

TABLE [R,T] := 1 if R is activated at T

(T-1 if R entered RELATED_RPS because it was active in the time unit preceding that of K).

otherwise TABLE [R,T] := 0

5) [construct new RPs]

A new RP is created, with a first component requiring an activation of the RP, R, whose TABLE row is most frequently one. The second component of the new RP requires an activation of ACTIVATED_RP(K). Thus, the new RP has the form:

#n: (<R,P1{,previous}> <ACTIVATED_RP(K),P2>)

The attribute "previous" is included only if R entered RELATED_RPS because it was active in the time unit preceding K. "P1" and "P2" are statistical measures of the propriety of activating RP #n's components by default. To set initial values of P1 and P2, find a set, S1, of the most recent occurrences of R and a set, S2, of the most recent occurrences of ACTIVATED_RP (K).

PAST_OCCURRENCES might do for S2. P1 reflects the degree to which S1 is contained in S2, and P2 reflects the containment of S2 in S1.

Early MUL versions added a third component to the new RP if some RP, R2, had TABLE entries identical to those for R. The correspondence among three RP activations (for R, R2, and ACTIVATED_RP(K)) was considered less likely to be pure coincidence than the correspondence among just two (for R and ACTIVATED_RP(K)). Later work, particularly with hand-coded RPs, suggests that two-component RPs are adequate, and decrease redundancy.

In addition, several RPs might be produced for a single pattern induction attempt. Each RP whose TABLE entries contain more than some minimum number of 1's would be named as a first component of an RP like #n above. Multiple RP creations are advisable because of the expense of constructing TABLE.

3.7 Predictive pattern induction

Predictive pattern induction can be used to introduce RPs [Zeigler 78], but is generally ineffective because predictive relationships are not often observed among RP activations. In later MUL evolution, predictive pattern induction was given up as a mechanism for general RP

introduction though it re-emerged as a mechanism for learning inhibitions. The following algorithm prevents future conflicts by introducing inhibitive RP masses.

1) [selection of an RP X to inhibit]

An RP, X, is found (by methods described in Section 3.4) to have been improperly defaulted. MUL applies predictive pattern induction to define the conditions preceding the improper default so that an RP mass can be constructed to recognize those conditions and activate an inhibitory RP to prevent future improper defaults of X.

2) [formation of "not X"]

If X does not already have an inhibitor RP associated with it, create a componentless RP, N, as the inhibitor of X. N is called the "not X" RP of X, because its activation conceptually represents situations when X should not be activated. In some versions, X is made equivalent to "not N" by making X in turn the inhibitor of N, but for RPs, not (not X) is not equivalent to X.

3) [examine the history of X's activation]

Find several past activations of X that caused conflicts and should have been inhibited; record their times of appearance in an array, IMPROPER. Also find several activations of X that did <u>not</u> cause conflicts; record their times of activation in an array, PROPER.

4) [introduce an RP, Z, to activate N when X is to be inhibited]

Find or construct an RP, Y, that was often activated in (or just prior to) time units in IMPROPER, but that rarely was activated in (or just prior to) time units in PROPER. That is, Y is activated about the time that X should be inhibited. Introduce the RP:

#Z: (<Y,0.0{,previous}> <N,1.0>)

The "previous" attribute is included if activations of Y are found for time units just prior to those in IMPROPER. Now, Z can forestall the improper default of X by activating N (that is, "not X") under the conditions recognized by Y. Generally, possible Z's will be generated until all improper defaults of X are inhibited, or until some resource is exhausted.

3.8 Why pattern induction?

The lack of examples (explicitly identified as such) from which to learn is an influential constraint on the development of pattern induction. Pattern induction as we have studied it must not only select examples from which to learn, it must also decide the nature and content of those examples. That is, a set of similar events must be selected as a "training" set from a long history of mental and physical experience, some of which might be only

"imaginary". From the experiences preceding these similar events (again, hundreds or thousands of RP activations) must be extracted a useful (hopefully predictive) relationship between other events and events in the "training set" of similar events. The above process might be repeated for several "interesting" kernels in each step.

Limitations on resources available influence algorithm design. It would be helpful to construct more complete information about event similarity or spend more time devising descriptions of similarities in the circumstances of similar events. The interference-matching technique used in Hayes-Roth's SPROUTER program [Hayes-Roth 78] is an example of how greater resources could be well spent. However, to justify the increased expense, assurance is required that the end product is useful. This assurance could be at least partially based on a sophisticated (and expensive) selection of subject events. Further, a large amount of irrelevant information (at least to a particular subject event) is present for each event; some mechanisms (expensive, of course) must screen out such irrelevant coincidences. Even the Hayes-Roth technique does not often detect regularities in the events leading up to the occurrences of the subject events because the distinguishing features of those events have not yet been recognized - the appropriate RPs have yet to be learned.

3.9 The benefits of pattern induction

Pattern induction is expensive in the forms investigated for this research. It is important to understand the benefits expected from such pattern induction in return for the cost.

First, such pattern induction is inherently stable. It will not create drastic alterations in knowledge in an effort to recognize and anticipate new inputs. New RPs must usually prove usable in several past experiences before they can be introduced. Once introduced, RPs are never altered and so they always recognize the same general event. (Inhibitions change only the prediction behaviors of RPs. Also, RPs are not forgotten in current versions of MUL, though forgetting may eventually be required in future MULs to reclaim storage resources.)

Second, pattern induction as we have investigated it is a self-correcting learning process. RPs cannot be incorrect as recognizers, though they may be redundant or irrelevant to future experience. While RPs can be used to default kernels at improper times, these defaults (if truly harmful) will produce conflicts that later stimulate the learning of corrective inhibitions. Inhibitions themselves can be inhibited, to refine their behavior. It would be an interesting experiment to excise some RP(s) from MUL to observe MUL's recovery. Some sensory RPs might also be

removed, although performance must deteriorate with the loss of information.

Third, pattern induction, as we have investigated it, is universal. It may be applied to activation histories of both learned and preprogrammed RPs without alteration. Also, it may be applied to any environment with which an "RP activation" interface can be established. This versatility applies only to pattern induction and RP application: such tasks as importance assessment (Section 4.3), motivation (Section 4.9), performance of basic motor and sensory activities, and environment simulating require environment-specific code.

Finally, pattern induction, as we have investigated it, acts to remove the need for "close" matching during RP application. Close matching has not disappeared, but has moved to the relatively less sensitive and more stable area of pattern induction. RPs in later MUL versions always match exactly or not at all. "Close" matching is still allowed, but is incorporated in the default process. Since defaulting is controlled by inhibitions, MUL can learn and adjust "close" matching by learning new inhibitions.

Chapter 4: MUL

Preceding chapters introduced the recognizer/predictor (RP) as the repository for MUL's knowledge, and presented pattern induction as a class of mechanisms for learning new RPs. This chapter begins by developing the machinery for using RPs. Detailed consideration is given several problem areas for MUL design, notably the calculation of kernel importance, the evaluation of MUL behavior, and issues of motivation in MUL. The chapter concludes with discussions of various MUL experiments.

4.1 MUL's mental cycle

MUL's use of RPs in the perception of its environment, its learning of new RPs, and the activity of the environment itself are carried out in the fixed sequence described below. The rigidity of the cycle is a result of both the need to represent such tasks using a single-processor sequential machine and the desire to obtain comprehensible and reproducible results. MUL's activities could be performed by fairly independent concurrently-executing processes. Section 6.3 suggests a formalization of RP use that is particularly well suited to parallel processing.

The following algorithm describes MUL's activity cycle. Numbers in brackets refer to explanatory notes following the algorithm itself.

0. if resources for given environment step are exhausted then [update environment]
   a) select [1] and perform motor activity, if any (activate [2] motor RPs if a "flail" action [3]).
   b) update environment with respect to any changes from causes external to MUL.
   c) using the updated environment, activate the proper sensory RPs.
1. select a kernel, k, from a buffer called STM (ordered by importance value).
2. for each RP, r, having a component referencing ACTIVATED_RP (k) (up to some MAX number, ordered by RP importance):
   a) calculate the time unit, rp_date, in which RP r may be activated by k: given that kernel k occurs in time unit t, if the component of r referencing k has the attribute previous, then let rp_date be t+1, otherwise let rp_date be t.
   b) if r is not already active at rp_date, then
      b.1) see if kernels can be found (in episodic memory) that record activation(s) of the RP(s) referenced by the other component(s)

of r at proper times with respect to rp_date.

b.2) if a component RP is not already activated at the proper time, is defaultable [4], and is not inhibited at that time, then activate the component RP as a default.

b.3) if all components of r are active, then activate r for time unit rp_date.

3. if perception resources are not yet exhausted, then go to step 1.

4. while resources remain, do pattern induction.

Notes:

[1] Selecting a motor RP

1) if a motor RP has been activated for the current date,

2) and the action indicated by the motor RP is possible (a lowered arm cannot respond to <lower arm>, for example),

3) then alter the environment to reflect the outcome of the action.

- if MUL has activated more than one motor RP, choose the activation with the highest importance factor.
- if MUL has not activated any motor RP, then MUL's motor apparatus is considered to be uncontrolled; a

"flail" action is chosen at random and performed. Generally, the flail action is to "REST", doing nothing.

[2] Activating an RP: if the activation has sufficient importance, then

1) introduce a kernel recording the activation into episodic memory.
2) place a reference to the kernel into the STM buffer. If STM is full, then eliminate STM's least important member (by importance factor) or ignore the new kernel if it is least important.

[3] Initially, MUL knows nothing about its motor capabilities. To expose those capabilities, MUL is equipped to flail: If MUL is not generating motor outputs (thereby demonstrating control of those outputs) then some motor output will be randomly selected (usually "rest").

[4] A component of an RP is considered defaultable (in the MUL versions examined) if all other components have been activated and if the statistics for that component indicate success with at least 75% of past defaults.

4.2 Supporting defaults: the request for instantiation (RFI)

Default activations are initially unsupported. MUL may support a default by finding timely activations (called instantiations) for each component (if any) of the defaulted RP. As explained below, supporting a default involves investigation of whether particular RPs (those referenced as components of the defaulted RP) are active or can in turn be defaulted. In this way, RPs are able to suggest other RPs to apply.

Given an unsupported default of RP x, these steps are taken to support it:

1) If x has no components, do nothing (if x is a sensory RP that is incorrectly defaulted, then it will be in conflict with the actual sensory data received later).
2) Examine each component of x. If the component RP is active at the proper time, then record the kernel as supporting the activation of x.
3) If any component(s) of x are not already active at the required times, and the default of x has a sufficiently large importance factor, then appropriate kernels may in turn be defaulted for those components. The RP responsible for the default activation of x is also responsible for

kernels defaulted to support the activation of x. The activation of that responsible RP indicates that x is to be active, and thus indicates that x's components are to be active, as are x's component's components, and so forth. In practice, defaulting cannot be continued to components of defaulted components because defaulted kernels are given importance factors too small to enable further defaulting.

Later MUL versions employed a delayed instantiation implemented by the Request For Instantiation (or RFI) mechanism. Instead of immediately attempting to support a default RP activation, MUL creates an RFI referencing the default. When MUL has nothing better to do, it selects an RFI and tries to find or default support for the referenced default. As with facts in STM, the importance of the unsupported default is used to decide when and if an RFI is processed. The processing of RFIs is interleaved with STM processing.

4.3 Importance assessment

The importance factor first discussed in Section 2.8 is used ubiquitously to control the course of MUL's activities, as in the timing of RFI and STM processing.

The initial importance of kernels is a function of their origin. Kernels arise in four basic ways:

1. Kernels introduced by sensory mechanism have their importance initialized to a constant value (0.9). For the MUL model as it currently exists, all sensory information is equally important when it is first received. (More complex environments almost certainly require more complex calculations to establish this initial importance. In animals it is likely that several specialized subsystems exist in lower brain level, for example, the limbic system, apparently screening out unimportant information and calling attention to and emphasizing important information.)

2. Kernels introduced by recognition have their importance factor initialized to that of their least important supporting kernel. An alternative formula used in early MUL versions was to average the importance of the supporting kernels. Averaging was unacceptable because improperly defaulted kernels (having low importance factors) could often be used with good kernels to produce a moderately important but completely improper kernel. The minimum-component formula was much more conservative: once a kernel was considered

bad, it could never support kernels of greater importance than its own.

3. Kernels introduced by direct default are given an initial importance based on the activations of the RPs that defaulted them, called their defaulters. That initial importance is the product of the importance factor of the defaulter, a fractional importance diminisher (0.9), and the probability that the default will be supported.

4. Kernels introduced by default in support of an RFI have initial importance factor calculated as the importance factor of the RFI (a function of the importance of the default referenced by the RFI) diminished by a constant fraction (about 0.7). The great diminishing insures that only kernels which are relatively certain to appear are defaulted in this indirect way.

Importance factors may not remain constant. Three situations may change the importance of kernels:

1) Whenever a kernel is determined to be a cause for later conflict (discussed in the next section), then its importance is diminished by a constant fraction (0.75 in later MULs).

2) Whenever a defaulted kernel is found to be in agreement with actual sensory input, its

importance is altered to a constant .8 value. Thus, unanticipated sensory inputs have a higher importance factor (.9) and are more likely to be selected for pattern induction. It is possible to diminish the importance of all correctly anticipated kernels, rather than just sensory kernels, but the cost of detecting such situations was considered to outweigh the benefit for the environment used.

3) Whenever an RFI is selected for processing, the importance of the kernel it names may be altered depending on whether and how support is obtained for it. If no supporting kernels must be defaulted, then the new importance value is the minimum importance of its supporting kernels. Otherwise, a loss of faith in the RFI kernel is effected by diminishing its importance by a constant factor (.75).

These formulas have so far been adequate even though there is no compelling rationale for the particular constant factors mentioned. MUL has so far proved relatively insensitive to any but major alterations in importance calculations. However, with more complex environments where only a small fraction of kernels could be examined, the treatment of importance values is likely to influence the

rate and perhaps even the long-term quality of MUL perception and learning.

One adjustment could significantly improve MUL's performance, though at a high cost. Suppose kernel A supports the activation of kernel B. If A undergoes a change in importance, B's importance might depend on A's and should therefore be recalculated. Changes in importance should also propagate down to kernels created by default, if more or less confidence can be placed in their defaulters. Propagation of importance changes has not yet been attempted.

4.4 Performance measurement

With all the hundreds of RP creations and thousands of activations, how is MUL's improvement in its ability to recognize and anticipate its environment to be detected? MUL's only observable activities (before the development of inhibition) were the construction of new RPs and RP use for recognition and default of kernels.

The earliest estimations of perception quality were subjective: a small number of the RPs were examined after each run, their quality assessed by guessing whether the RP would be useful to MUL or not. This evaluation was unacceptable for two reasons. First, due to their sheer numbers, not all RPs or their activities could be examined.

Second, it proved difficult to reliably assess RP usefulness.

The evaluation problem was partially solved with the adoption of a statistical approach. Counts of the number of occurrences of various events were automatically recorded for each environment step. Of interest were these events:

- number of defaults taken
- average importance of the defaults
- number of kernels defaulted in support of RFIs
- average importance of defaults in support of RFIs
- number of failures (Originally, a failure was an unverified default of a sensory RP. That is, sensory data received did not include an activation of that RP. Later, failures were defaults that caused conflicts and triggered attempts to introduce new inhibitions.)
- average importance of failures
- number of kernels generated
- number of kernels examined to see what RP activity they precipitate
- number of RFIs generated
- number of RFIs acted upon
- number of RPs generated
- number of sensory RP activations correctly defaulted (that is, defaulted without being identified as a

cause of a conflict, as discussed in the next section)

- time required for processing

By examining and comparing these statistics for different MUL versions, some opinions can be formed on their relative merits. One measure of MUL's "understanding" of an environment is its ability to anticipate future developments in that environment. Thus, a crude measure of performance was the number of correctly defaulted sensory RP activations.

The statistical approach was not without problems. Versions often differed extensively, prohibiting direct comparison of results by statistical means. A better measure for performance was needed, and was found in the "conflict".

4.5 The conflict

The development of mechanisms for inhibition opened a new dimension for performance measurement: the conflict. Historically, the idea of the conflict grew from the contradictions observed between defaulted and actual activations of sensory RPs. For instance, MUL might default <color middle middle red> in anticipation of events at time t, yet when t is experienced some incompatible RP is

activated instead. Before inhibition mechanisms were introduced, these contradictions were detected in an ad hoc manner. With inhibition the occurrences of <color middle middle red> were preprogrammed to inhibit the activation of any RP suggesting a different color than red for that retina position. When sensory data, for instance <color middle middle black>, arrived for t, a conflict was detected: the sensation-activated RP was inhibited. The conflict idea was extended to cover any instance when an inhibited RP is activated, whether by recognition or sensory input. (Defaulting cannot be done on inhibited RPs.)

The introduction of the conflict concept made performance measurement easier, by pointing out MUL's mistakes. Conflicts also formed the basis for creating new inhibitions: Conflicts show that some RP must have been incorrectly defaulted. But which RP? The appearance of both the inhibiting kernel and the inhibited kernel could be based on information obtained by default. The conflict could have been prevented by inhibiting any of the defaults supporting either of the conflicting kernels.

Instead of trying to select a culprit RP on the basis of one conflict, an algorithm was used to mark several possible culprit kernels. An improper default often causes several conflicts, and thus is repeatedly marked. The activated RP of the most heavily marked kernel can then be selected for inhibition.

The algorithm for marking culprit kernels hinges on two observations. First, the appearance of the culprit must support or indirectly support the appearance of one of the conflict kernels. Second, the culprit must be a defaulted kernel. A kernel, x, supports a kernel, y, if x is recognized by y (that is, ACTIVATED_RP (x) is named as a component of ACTIVATED_RP (y), and x occurs at the proper time with respect to y). A kernel x indirectly supports a kernel y if x supports some kernel z and z directly or indirectly supports y. Since MUL kernels preserve references to the kernels which directly support them, a recursive algorithm can generate all kernels supporting the kernels in conflict. Culprits are supporting kernels having insufficient support themselves (that is, introduced by default). In practice, kernels in direct support of a conflict are assumed to be more likely to be improper defaults than those at greater distances from the conflict. The distance between a kernel x supporting another kernel y is one if the x directly supports y. Otherwise the distance is 1 plus the distance between the kernel directly supported by x, and y. Since x may directly support several kernels, some of which may not support y at all, the support and distances are actually generated starting from y. MUL generates and marks culprits only if they are within a distance of four from the conflict.

4.6 Problems during MUL development

This section reviews some of the salient problems encountered during MUL's evolution. Substantial program alterations were often made (aided much by TELOS). Local program fixes were generally avoided in favor of appropriate alterations in underlying ideas.

The first MUL version used distinct structures for recognizing and predicting, although their net effect was similar to the RP structure described in Section 2.4 and [Zeigler 78]. Sets of allowed values were used to implement incomplete component description, for both recognizer and predictor structures. Interparameter relationships were not preserved. Inhibitions and RFIs were not yet part of the model. The pattern induction mechanism was essentially predictive, introducing recognizer structures only when no predictive structure was discovered.

A first problem with this MUL version was a predilection for schizophrenia. No mechanism distinguished what MUL imagined from what it truly experienced. MUL could fantasize (i.e., default) something and then support the fantasy with more fantasy, moving irretrievably into a dream world. A tag of "IMAGINARY" had to be added for meaningful experiments to proceed. In later versions, this tag was removed and the problem solved by assigning a decreased importance factor to unsupported defaults and then requiring

that any kernels used in pattern induction have at least a minimum importance factor.

Once MUL was persuaded to concentrate on the real world, it became obvious that not all RPs could be inspected during pattern induction. The RELATED_RPS set was introduced to restrict the induction. Initially, RELATED_RPS was constructed using specific relatedness guidelines like "RPs concerning a retina cell are related to RPs concerning neighboring retina cells". Only when the generalized level/area relatedness mechanism was introduced did the quality of pattern induction become acceptable (see Section 3.5).

Early MUL versions used an antecedent/consequent form for their predictor structure. This form was found inadequate to convey temporal sequence information for predicted kernels. For example, <color middle middle red> might, as a consequent, be predicted. Such a prediction is useless without information about precisely when that kernel is being predicted to appear in relation to other events. The RP structure attacked this problem by enabling kernels to be defaulted during particular time units.

Another serious problem in early MULs was prediction-reality discontinuity. Prediction-reality discontinuity often occurred when an inexactly-described consequent was predicted. The kernel created to instantiate such an inexactly-described consequent usually would not

correspond to any kernel actually experienced. A simple mechanism to circumvent this discontinuity was to require that all consequents predict unique kernels. Kernels then recorded recognizer activation. Recognizers could describe events incompletely by using incompletely-specified components; a unique predicted kernel could thus effect an incompletely-specified prediction. Normal recognizing and predicting actions would then verify the predictions if they were valid. Incompletely-specified component description was later abandoned in favor of the RP structure.

The incompletely-specified component knowledge representation form was abandoned in response to the overgeneralization problem discussed in Section 2.11. Overgeneralization frequently produced true but useless predictors exemplified by "when some color is seen in a retina position, anticipate that any color will later be seen in that position". Overgeneralization was also responsible for introducing incorrect predictors. Without inhibitions, predictors that are wrong even one percent of the time cause great confusion.

The next MUL versions were constructed around the concept of the RP, as described in Section 2.11. Inhibition mechanisms had not yet been introduced, but RFIs were introduced to establish better control of the "support default" task.

One of the first serious problems encountered in the RP versions was that of redundant RPs. ¿Pattern induction would often produce an RP identical to an existing RP. MUL has very limited space and time resources so that unlimited redundancy was considered too damaging to be allowed. This duplication was initially difficult to detect because RPs might have several components. Equivalence up to permutation in the order of components was complete operational equivalence for this model. This problem was solved by developing a canonical form for RPs. Canonical RP representation made it economical to detect and reject RPs containing the same components as already existing RPs as pattern induction produces them.

Redundancy is still present among RPs, however. For example, suppose three RPs #1, #2, and #3 are consistently coactivated. ¿Pattern induction might produce several new RPs:

#201: (<#1,0.0> <#2,0.0>)

#202: (<#2,0.0> <#3,0.0>)

#203: (<#1,0.0> <#3,0.0>)

#204: (<#1,0.0> <#2,0.0> <#3,0.0>)

Even worse, new RPs #201, #202, #203, and #204 would coactivate, together with #1, #2, and #3. MUL could be strangled with RPs like:

#205: (<#1,1.0> <#2,1.0> <#201,1.0>)

Fortunately, the level/area mechanism used to build RELATED_RPS for pattern induction eliminated many intolerable RPs like #205. But RPs like #206 still persisted.

#206: (<#204,0,0> <#202,1.0>)

RP #206 is not totally useless, however. Suppose an RP defaults #204. Activations can be defaulted for RPs #1, #2, and #3 if the RFI for #204 has a sufficiently large importance factor; if not, then #206 can activate #202 by default, to allow continued investigation based on the presence of #202. Thus, #206 is not strictly redundant.

In the above example, the default of #204 was sufficient to determine the status of RPs #1, #2, and #3 as activated. RP#206 served to assist only in rare cases. However, the activation of #206 would presumably be unanticipated and therefore would trigger pattern induction. Suppose some RP, #207, was learned that anticipated #206:

#207: (... <#206,1.0>)

Now, activations of #207 are unanticipated, triggering pattern induction to learn:

#208: (... <#207,1,0>)

Clearly there will <u>always</u> be unexpected activations, so that pattern induction must not be performed on activations solely because they are unexpected. A further requirement is that learning to anticipate the activation would be <u>helpful</u>. Anticipating an activation is helpful if that

activation directly or indirectly anticipates otherwise unanticipated sensory RP activations. This definition, however, was not explicitly encoded in MUL. Instead, the effect was approximated by lowering the importance factors of successfully anticipated sensory RP activations and raising the importance factors of unanticipated sensory RP activations. Later RP usage then produces kernels of greater importance for any RPs involving the unanticipated sensations.

Adjusting importance calculation reduced redundancy to tolerable levels. Still further reduction is possible if the number of components for RPs is kept at two or less. The trade-off is that a greater number of RPs must be activated to recognize any particular event. Future research may find limitation to or, at least, emphasis on two-component RPs desirable.

Redundant RPs can also be reduced by taking suitable precautions during the construction of RELATED_RPS. An RP, x, is redundant with respect to RP, y, if <u>all</u> situations in which x is activated will result in an activation of y. A method was developed to test for such redundancy. First, a prototypical situation activating y is created, by hypothesizing activations (and RFIs) for each component of y. The RFIs will result in activations (and RFIs) for the components of the components of y, and so forth until the default process dies from lack of support. If the normal

perception cycle is performed on this hypothesized set of activations, any RP which can be activated must in some sense be redundant with y and can therefore be excluded from RELATED_RPS. Unfortunately the cost of this hypothesizing technique is excessive. In order to be useful, RELATED_RPS must not contain RPs redundant with the pattern induction target y, and it must not contain an RP, z, if y is redundant with z. Practical resource limitations prevented this sort of exacting approach to redundancy elimination.

Another problem encountered in early RP-type MULs was a tendency toward "tunnel vision". MUL is composed of parallel processes at least at lower levels of the level/area RP structure. During MUL simulation as a sequential process, an important kernel can result in an avalanche of RP activations from only a few level/areas. These bursts of activity often dominated fixed-size structures like STM, pushing out kernels from other level/areas. To mitigate this effect, fixed-size structures were constructed with flexible partitions by area, guaranteeing that no area could be totally ignored. Section 6.3 suggests even more complete partitioning of MUL resources by level/area, an approach that would further decrease tunnel-vision effects.

The development of the inhibition mechanism led to yet other MUL experiments and still more problems. Salient among these problems was that of inhibition timing. Inhibitions were often activated after they were needed - that is, after the RP they inhibited had been defaulted. For example, the following RP mass predicts #1 if #2 is present and #3 has not already been activated.

#200: (<#2,0.0> <#1,0.9>)

#201: componentless, inhibiting #1

#202: (<#3,0.0> <#201,1.0>)

If #3's activation is examined for RP activity before #2's activation, then #202 will default #201 and thus correctly prevent use of #200 to default #1. If, however, #2's activation is examined before #3's, then #1 will be defaulted by #200 before #3 has an opportunity to inhibit #1 by defaulting #201. The result is a conflict, resulting (most likely) in an attempt to inhibit #1 by other means.

Later MUL versions combatted this problem by being careful when introducing new RPs for inhibitory purposes; new RPs must have components that are always activated prior to conflict-producing activations of the RP to be inhibited. This precaution was not foolproof because MUL shows extraordinary flexibility in the order of kernel occurrence. While it appears that no system can completely eliminate the problem of inhibition timing, it is hoped that the more rigid partitioning of resources suggested in Section 6.3 may

provide a more predictable time frame for kernel appearance in MUL, so that inhibition timing can be more precisely accessed and controlled.

4.7 Results with the shelf environment

Once the RP formalism and RFI were incorporated, MUL versions began showing respectable performance in the shelf environment. This section will discuss the performance of the first respectable version and the final version for the shelf environment.

Recall that the shelf environment provides ten sensory input kernels during each time unit, and expects one motor output kernel. Of the sensory kernels, two are always activated (<color top left none> and <color top right none>) and should therefore be easily predicted. The remaining eight sensory kernels are dependent on the position of the organism and the orientation of its hand. These kernels can be accurately anticipated only if the next movement is known and the territory revealed by the movement is known. The movements are decided by random flailing in all tests because the program has never been run long enough that it might gain control of its motor capabilities. Thus eight of the ten kernels are difficult to anticipate.

The following graphs show results on various performance measures for a first respectable MUL version

(called PRIMITIVE) and the final MUL version run on the shelf environment (called ADVANCED). PRIMITIVE used the RFI mechanism. ADVANCED incorporated inhibitions and many alterations designed to attack problems mentioned in the preceding section.

Graph I shows the number of anticipated sensory inputs. In the graphs, smoothed (by averaging) values are represented for ADVANCED by the solid line and for PRIMITIVE by the dashed line. ADVANCED does considerably better than PRIMITIVE, averaging 5.92 correct to PRIMITIVE's 2.64. Initially, neither version was able to predict anything, as only sensory/motor RPs were known. At time unit 5, enough experience had been gathered to support pattern induction to learn more RPs. Fluctuations in anticipation quality are caused mainly by MUL's movements. For example, at time unit 6 a <move right> put both versions into unknown sensory territory. That MUL is able to anticipate anything about unknown territory indicates that its RPs indeed generalize.

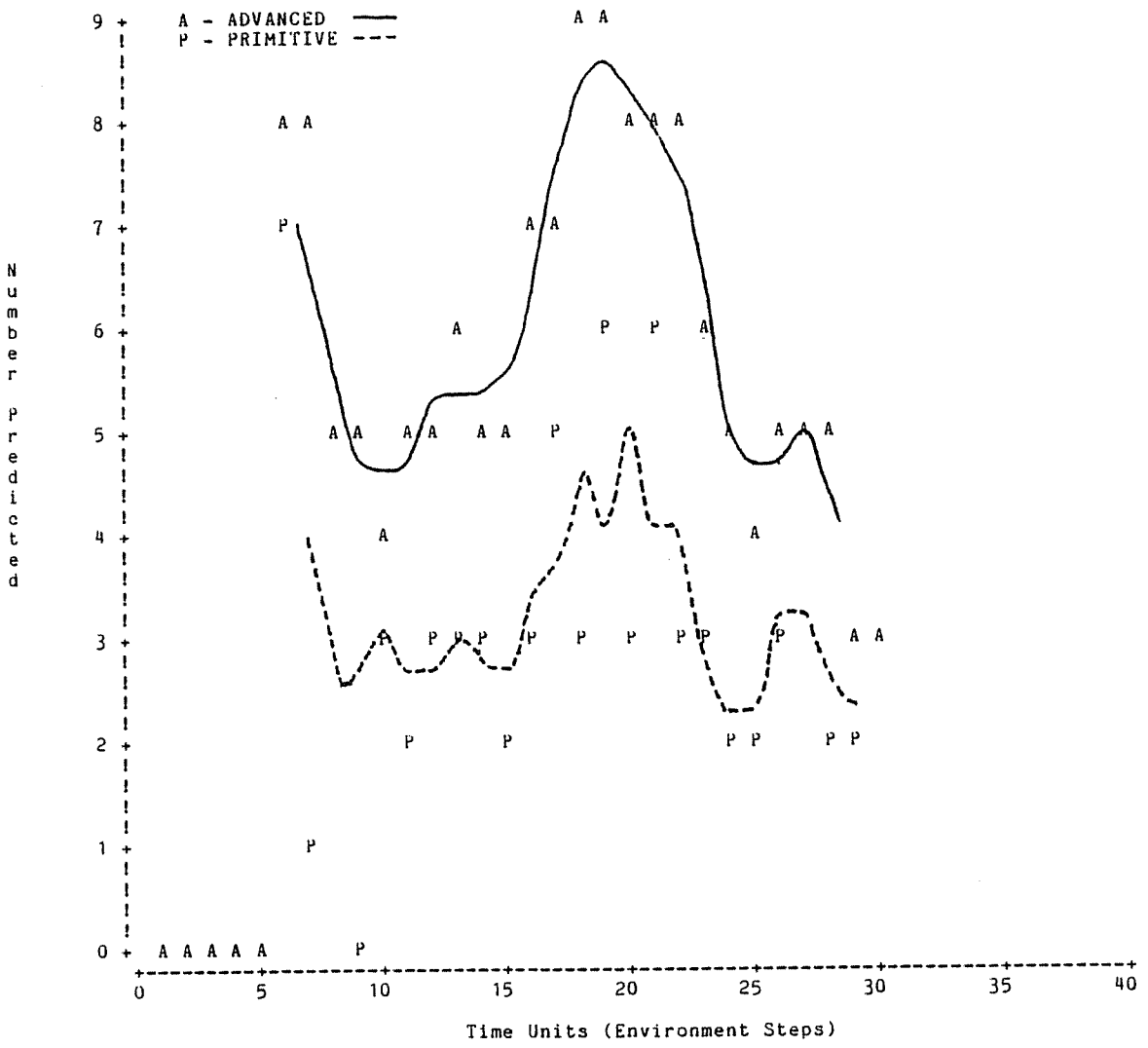
The accuracy of MUL's sensory anticipation is a good measure of its knowledge of its environment. However, it is an all-or-nothing measure. In order that a sensory event be anticipated, an activation of the appropriate sensory RP(s) must be defaulted. Often, however, the RPs that might default the sensory kernel are activated with importance factors too small to cause the default of a component. In such cases, the sensory event has still been anticipated (as

a part of a higher level event) but MUL has not considered it worthwhile to instantiate the anticipation by activating supporting sensory RPs.

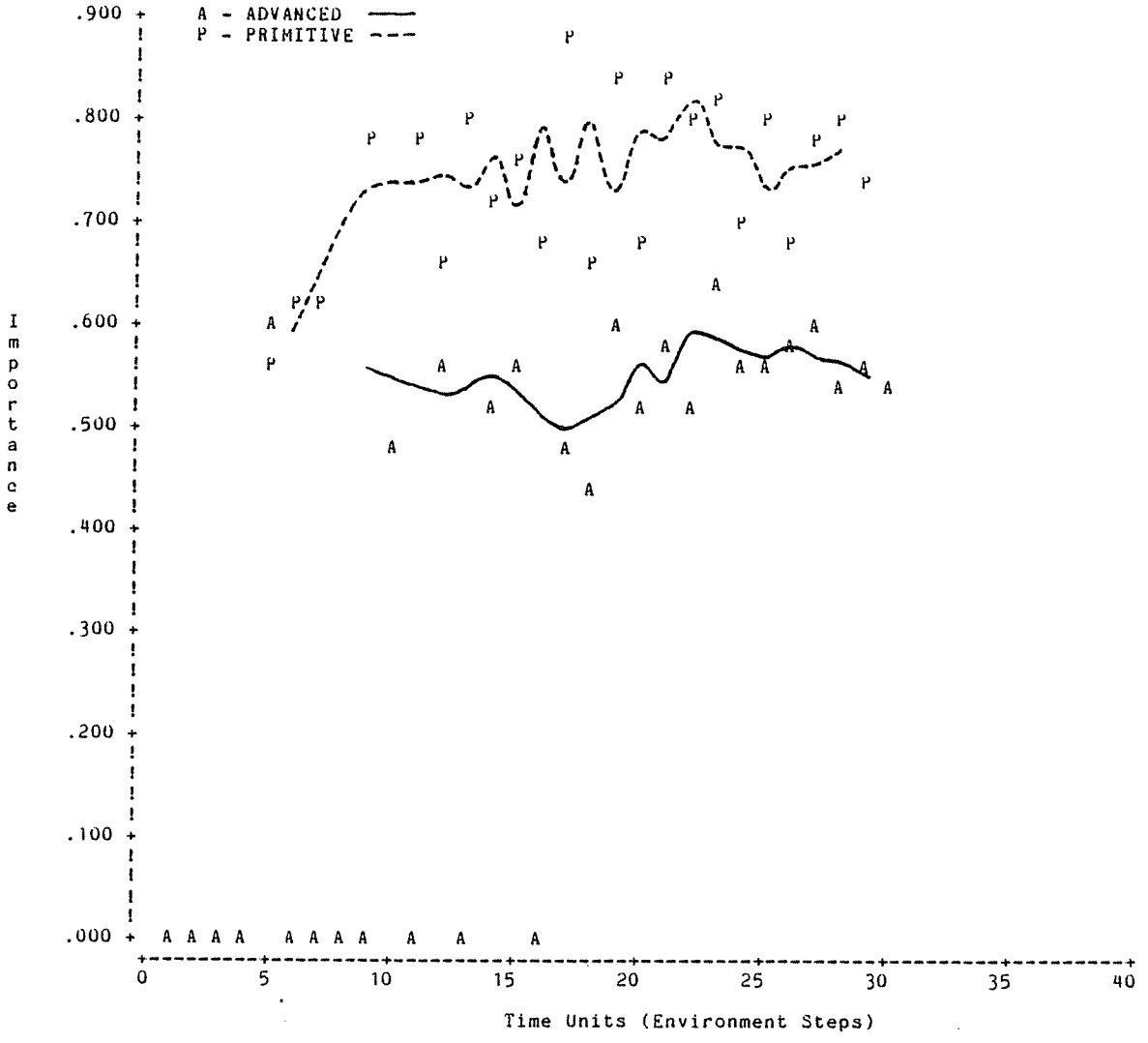
Graphs II and III present measures of MUL's anticipation quality that are sensitive to uninstantiated high-level defaults. Graph II shows average overall default importance. Graph III shows the average importance of defaults that caused conflicts. A comparison of these graphs indicates that importance values are far higher for proper defaults, as desired; MUL pays greater attention to proper defaults. These graphs may also explain why high-level defaults are often left uninstantiated: defaults have low importance factor values compared to the 0.9 importance values normally observed for activations originating from recognition. In ADVANCED the average default has insufficient importance to cause the default of supporting kernels.

Generally lower average default values of ADVANCED are a result of modifications in the importance calculation formulas. The modifications were introduced to steer pattern induction toward dependable activations and away from unsupported defaults. ADVANCED shows a decidedly lower average failure importance, however, more a result of better overall RP quality than importance calculation modifications.

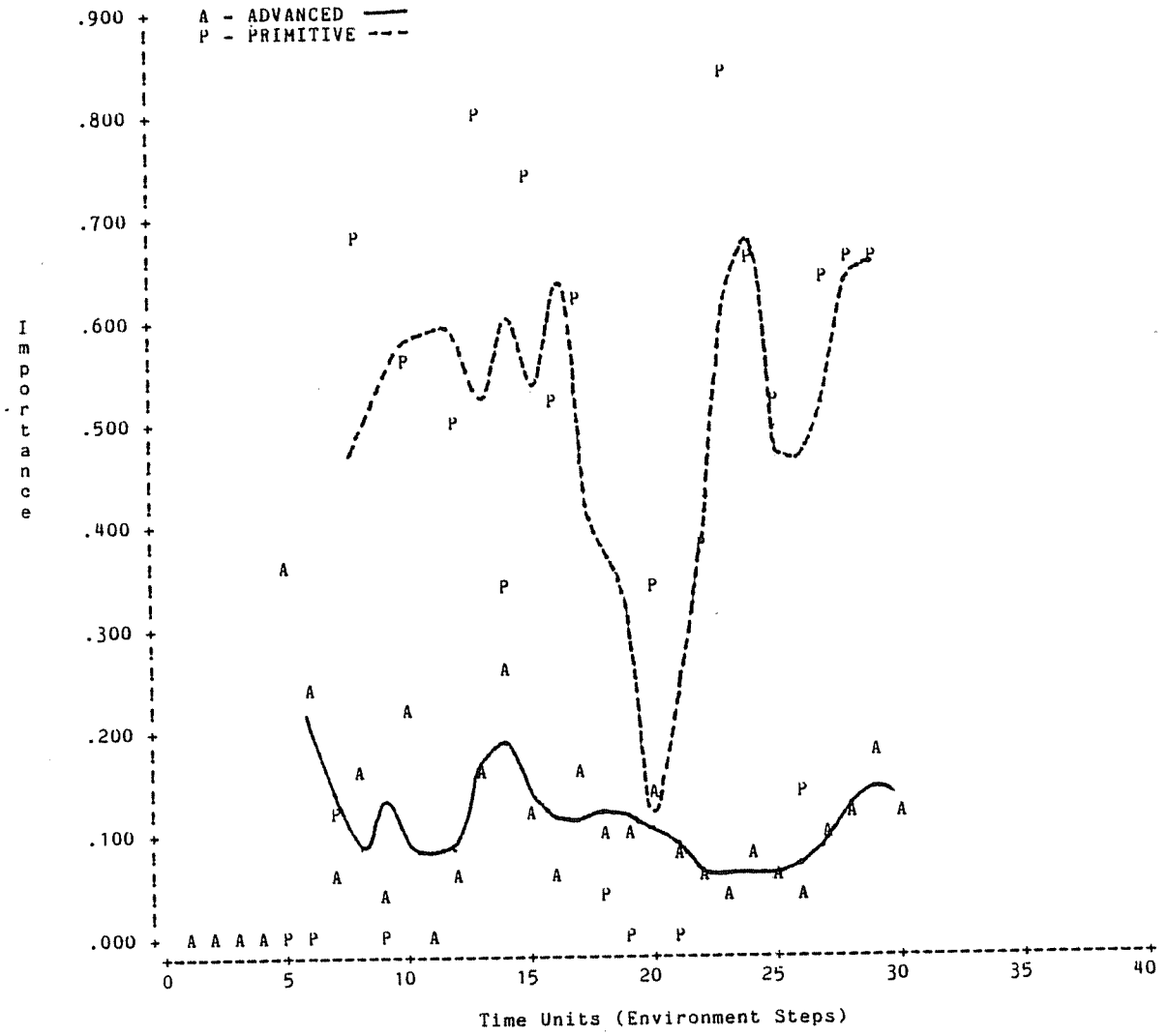
Graph I: Number Of Sensory Inputs Anticipated



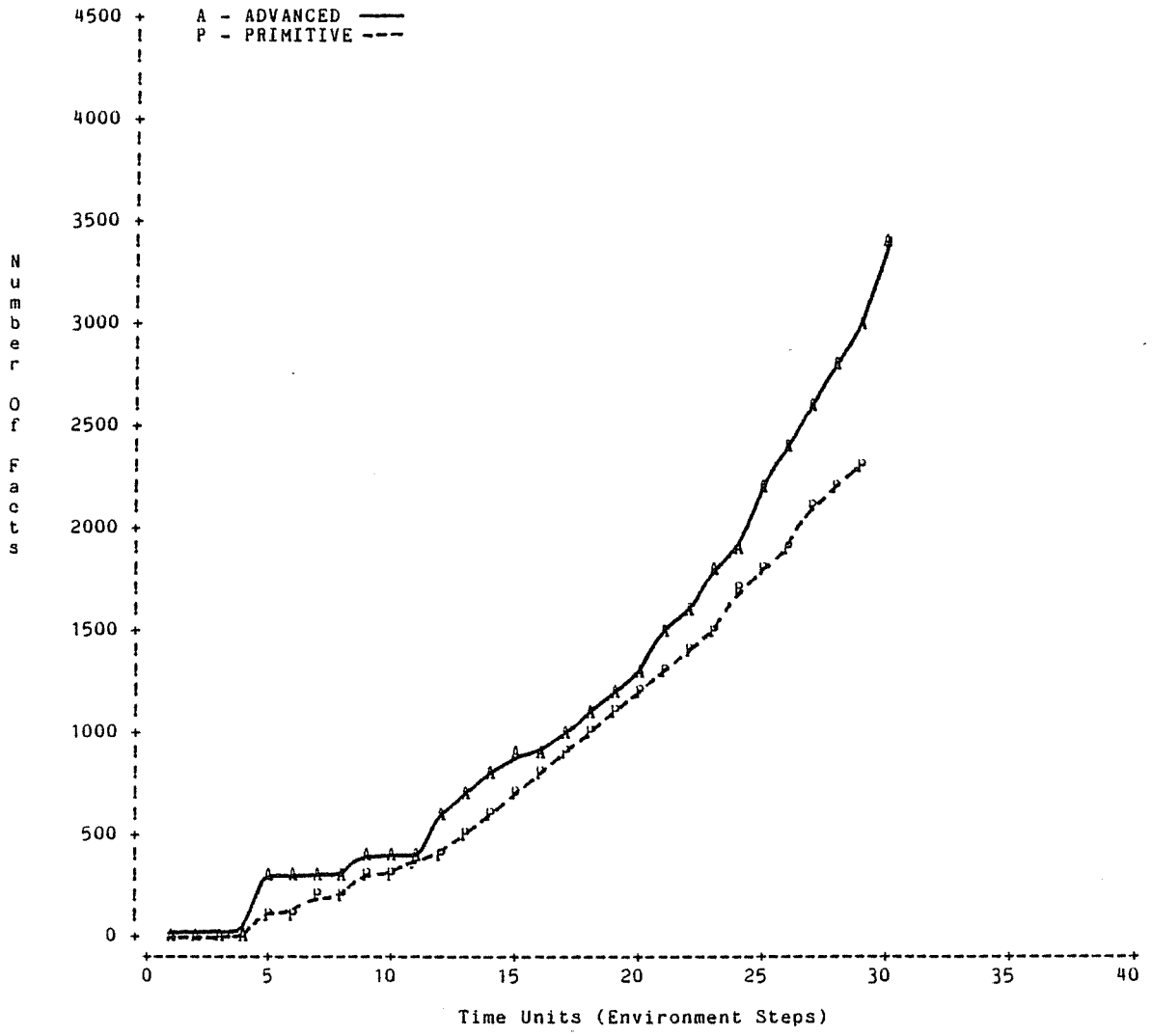
Graph II: Average Importance Of Defaults



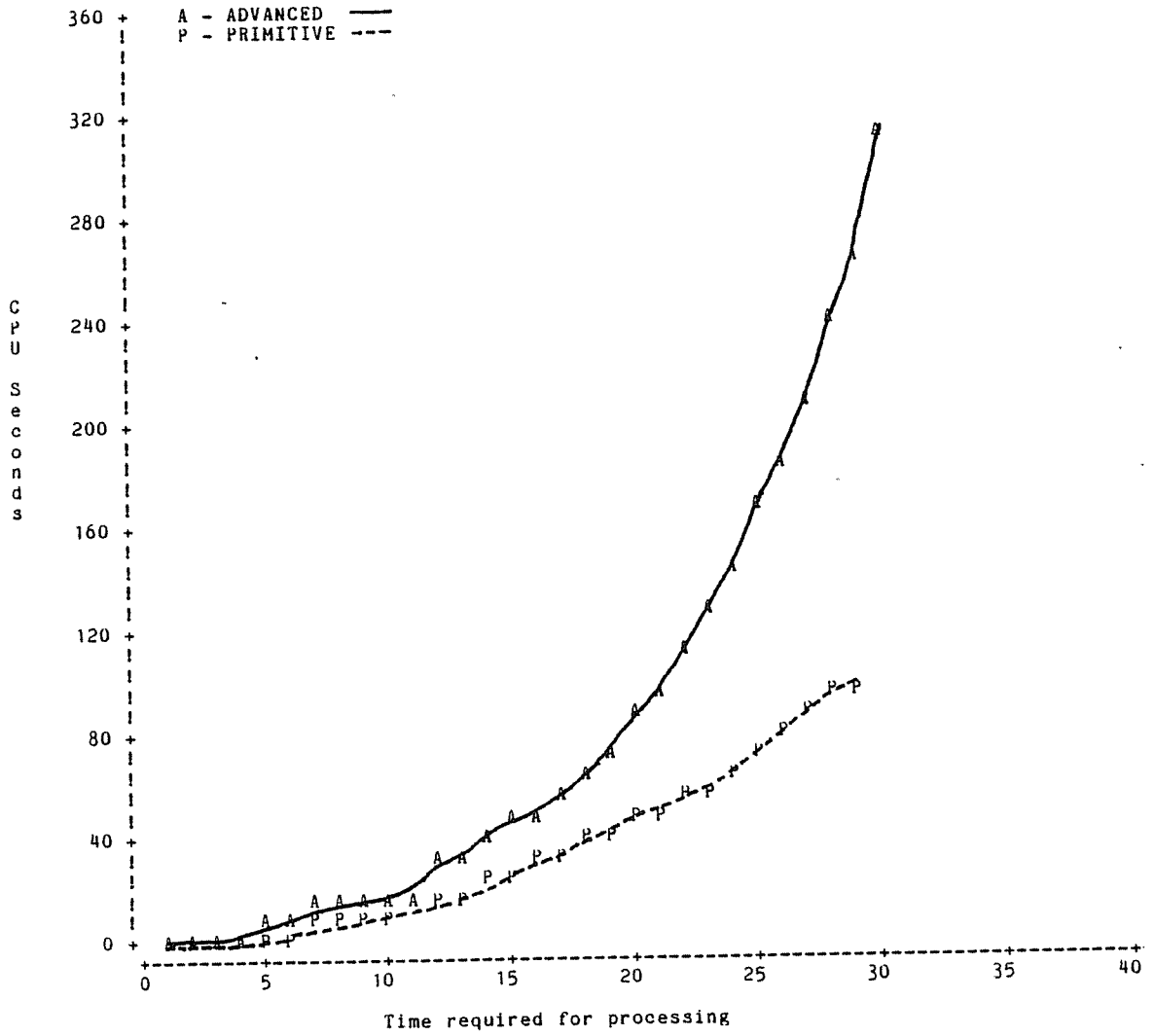
Graph III: Average Importance Of Conflicting Defaults



Graph IV: Number Of Kernels In Episodic Memory



Graph V: Elapsed Time (In CPU seconds)



All these graphs cover only MUL's first 30 time units of existence. Obviously 30 time units is a short time on which to base conclusions about RP/pattern-induction properties, but computing resource limitations discouraged attempts at longer runs. Graphs IV and V tell the story. The number of kernels in memory is limited (by the physical storage available) to a maximum of about 4000. As more RPs are learned, this maximum decreases to about 3000 when 500 RPs have been learned. Storage limits, then, can be expected to set a maximum on the time spent for each time unit: about 50 computer seconds were allocated to one MUL time unit as memory became filled. To run ten times longer (still only 300 MUL time units) would require over four hours of computing time on the UNIVAC 1110, and would require that at least 90% of all episodic memory be forgotten.

This sheer cost has been a major and unwelcome influence in MUL design. For example, the pattern induction process is allowed to introduce RPs on minimal evidence. In addition, many RPs may be introduced on the basis of one pattern induction attempt. Worse, 30 time units was found to be too short a time for inhibitions to be incorporated extensively in MUL. In order to study some ideas concerning long-term behavior of MUL, an RP mass was hand-coded.

4.8 Problems revealed by hand-coding RPs

The power of RPs for knowledge representation was put to the test by hand-coding complete understanding of the shelf environment. The result consisted of 233 two-component RPs and 29 inhibitions. The hand-coding became more than a test of representation as it revealed an interesting problem.

Suppose a two-unit-tall black block stands isolated on the shelf and a one-unit-tall black block stands alone in some other region of the shelf. Suppose also that a knowledgeable MUL awakens in front of one or the other of the blocks. If its hand is centered, MUL has no way of knowing whether the block it sees is tall or short. If MUL should then raise or lower its hand, the previously obscured spot will become visible, and MUL can then detect whether the block is one or two units in height. If MUL recenters its hand, MUL should be able to predict what it will see if it again moves its hand. But how can such information be stored as RP activity?

One possible answer lies in the "reverberatory" behavior possible among RPs. The following RP mass starts and maintains a reverberation (that is, repeated activation of an RP #300) whenever activated by the sight of black in the middle of the retina.

#23: componentless, sensing "color middle middle black"

#300: componentless

#301: (<#23,0.0> <#300,1.0>)

#302: (<#300,0.0,previous> <#300,1.0>)

When #23 is activated at some time t, then #301 will cause the activation of #300. #302 will then cause the default activation of #300 at t+1, t+2, and so forth until the importance of the unsupported defaults dwindles sufficiently. If at t+1 the center retina spot still reports black, the activation of #300 at t+1 will be strengthened by the activity of #301. Suppose that MUL obscures that middle position by moving its hand. #23 will no longer be active, but #300 will still be reactivated for several, perhaps many, time units by #302. The presence of an activation of #300 remembers that MUL has seen black. MUL anticipates what color will be seen upon lowering its hand with the following RP mass:

#4: componentless, requesting "lower arm"

#25: componentless, sensing "color middle middle red"

#400: (<#300,0.0> <#25,0.0>)

#401: (<#400,0.0> <#4,0.0>)

#402: (<#401,0.0,previous> <#23,1.0>)

A problem arises if MUL moves left or right before moving its hand. Unless some mechanism interferes, #301 will go right on reactivating #300, leading to unfounded

expectations of seeing black after a hand move. Fortunately, an inhibition could turn off the reverberation:

#303: componentless, inhibiting #300

#2: componentless, requesting "move left"

#3: componentless, requesting "move right"

#304: (<#2,0.0,previous> <#303,1.0>)

#305: (<#3,0.0,previous> <#303,1.0>)

This mass of RPs can extinguish the reverberation as follows: if a <move left> (or <move right>) is done at time t, then #304 (or #305) will result in the activation of #303 at t+1. The presence of #303 inhibits the default of #300 by #301, ending any reverberation in progress.

To be accurate, #303 must itself be inhibited when the hand is not centered, as a <move left> could move MUL in front of a tall black block. #300 should be activated by #302 in this situation.

No reliable method has yet been discovered to detect opportune times to introduce reverberatory RP masses. Their utility seems great, however, in MUL-like systems. For example, reverberatory circuits have been suggested by neural modelers [cf. Hebb 49, Whitehead 78]. Reverberator subsystems have also been detected in the cortex, as discussed in Section 5.10. Reverberatory masses may have a role in short-term memory, an idea explored below.

The STM described previously as part of the RP activity cycle acts like a buffer. Specifically, the MUL perception activity seems to be a parallel process forced into a sequential model, with the "STM" buffer merely an instrument of that translation. In MUL the function of STM as a temporary knowledge store is eliminated, as kernels leave STM when processed. Besides, as Graph IV shows, hundreds of kernels traverse STM in each time unit, even for a simple environment. It is difficult to rationalize why any particular kernel should remain long in STM, especially for the hundreds, even thousands, of time units over which the human STM seems to operate.

The reverberatory RP mass, in contrast, does seem to possess many characteristics of human STM. It could remain active over many time units. It can be "boosted" by exterior excitement, as with #302. Harder to explain is the size limitation, the famous 7 plus-or-minus 2 size, postulated for the STM mechanism [Coltheart 76]. It is conceivable that the size limit is a function of retrieval: only about seven reverberating masses might be accessed before remaining masses die out. More likely, reverberating RP masses interfere with one another so that limitations arise from exponentially intensifying competition for activation resources (attention?).

Hand-coding RPs and inhibitions for the shelf environment exposed the need for a short-term memory apart from the STM buffer used in MUL. Hand-coding also demonstrated that, given reverberation, the RP knowledge representation could provide a means for MUL to anticipate activity in the shelf environment.

Tests of the hand-coded RPs emphasized the need to anticipate future motions in order to fully anticipate environment conditions. Instead of inventing some arbitrary schemes for motion control, MUL was moved to an environment where particular motions would have reasonable motivation. The environment change also facilitated study of some aspects of behavior not required in the shelf environment, particularly motivation itself.

4.9 Motivation and the bottle-filling environment

In theory, MUL was to learn to control its own activities in the shelf environment. In practice, the costs of long runs prohibited fair investigation of that theory. Two steps were taken. First, an environment of reduced complexity was constructed. Second, MUL was given a "teacher" to help it learn proper motor activity.

The simplified environment is called the bottle-filling environment. MUL is given control of a valve which it can open and close. When the valve is open, a liquid flows from

a nozzle onto a platform where a bottle should receive it. MUL can sense the status of the valve and whether or not a bottle is on the platform. MUL also can sense what percentage of the bottle is filled. A teacher sits with MUL, putting empty bottles on the platform and taking full bottles off. MUL is to learn to open the valve only when an empty bottle is presented and to close the valve only when the bottle is full.

In this environment virtually all activity is predictable. When the valve is open the bottle fills steadily because of the physics of the environment. The teacher always removes a full bottle as soon as MUL turns off the valve, but never removes a partially empty one. Also, the teacher always places an empty bottle on the platform within about four time units of removing a full bottle. Only MUL's valve operations are unpredictable. MUL is to learn to manipulate the valve to fill bottles but not spill liquid.

To motivate MUL, the teacher possesses a "pain" communication channel to MUL. When MUL opens or closes the valve at an inopportune time, the teacher signals with pain. The pain signal is received by MUL as the activation of its pain RP. By arranging to have the pain RP inhibit itself, any RPs that learn to default pain will cause conflicts. These conflicts cause the formation of inhibitions to prevent the default of pain by inhibiting the occurrence of

situations leading to pain. For example, suppose some RP x defaults pain:

RP x: (<y,0.0> <pain,1.0>)

When RP y is activated, RP x will probably also be activated, defaulting pain. Because pain inhibits itself, a conflict arises. To resolve the conflict, MUL must prevent the activation of RP x. (The <pain> RP is treated exceptionally to insure that it cannot be inhibited by new RPs.) Inhibition of RP x will not be attempted since its activation is a result of recognition and therefore is not identified as a culprit in the conflict. Thus, MUL tries to learn grounds for inhibiting y or something in y's support. RP y recognizes features of a painful situation; prevention of y's activation may avoid the painful situation.

MUL also has a "pleasure center", to be activated by the teacher as a reward for good behavior. Pleasure influences MUL in a way similar to that of pain. MUL learns RPs that inhibit pleasure. These RPs act by defaulting the pain RP, specified by preprogramming as the inhibitor of pleasure.

Mixed results were obtained from the bottle-filling task. MUL is able to anticipate environment activity very well. MUL does not, however, learn to fill the bottles correctly. That is, MUL can anticipate that pain is coming, but does not learn to avoid it. The failure apparently

stems from over-caution in creating inhibitions. Performance might be improved by adopting less exacting requirements for predictive pattern induction. MUL requires that an inhibition never be activated when it should not be. This rigid requirement is a carry-over from early inhibition experiments when inhibitions could not themselves be inhibited, and should probably be relaxed.

Predictive pattern induction also requires that the inhibitions be activated before the event to be inhibited. While relaxing this requirement may not be advisable, it can be made easier to meet. That is a goal of the level/area perception cycle discussed in Section 6.3.

A final approach to improve performance is the introduction of RPs having "differential" behavior - that is, RPs that are activated by a change in RP activity patterns. For instance, in the shelf environment when MUL moves its hand to a centered position, the <color middle middle red> RP is activated and will continue to be activated until MUL again moves its hand. An RP exhibiting differential behavior is #3 below:

#1: componentless, sensing "color middle middle red"

#2: componentless, inhibiting #1

#3: (<#2,0.0,previous> <#1,0.0>)

RP #3 is activated (by recognition) only by the first appearance of the centered hand. The activation of #3 is easily associated with the <raise hand> or <lower hand> RP

that caused it, because those arm-move RPs will always be active in the time unit preceding activations of #3. Pattern induction mechanisms should easily discover this simple relationship, while discovering a relation between arm-move RP activations and activations of #1 is difficult. RP #3 has provided an intermediary between activations of #1 and the causes of those activations.

RPs exhibiting differential behavior are helpful whenever MUL observes that an RP tends to be continuously activated over several consecutive time units. In the bottle environment, such RPs would center attention on the events that initiated spillage, rather than on events during the spillage. Some differential RPs might also be preprogrammed for some environments, motion detectors, for example.

4.10 Comments on computer modeling

Currently affordable computer simulation and hand simulation has not verified the model but that has not been the point of it. The value of the computer runs and the hand-coding has been in pointing out problems in the model, and in forcing the model to be coherently and clearly stated.

The worth of computer modeling is magnified by the use of TELOS in model specification: the TELOS language

facilitates extensive program modifications to accommodate model changes. Models programmed in a less flexible languages are difficult to modify extensively, encouraging the modeler to make "local fixes" to remedy model insufficiencies - with the result being a kludge with little conceptual coherence.

How is MUL to be evaluated? It may be viewed as a gedanken model, as was Becker's, to be evaluated by argument and analysis. Eventually, it, or models like it, may suggest simulations or empirical experiments which are more directly verifiable. Such indirectness of verification is unfortunate, but is to be expected in scientific theorizing. MUL may also be evaluated in its compatibility with other evidence and theorizing. Toward this end, we now turn to comparing the structure and behavior of MUL with those of the human cerebral cortex, as they are conceived and hypothesized in the partly empirical but also largely gedanken theorizing of neurophysiologists.

Chapter 5: MUL as a model of the cortex

MUL was inspired by observations of early-stage human intellectual development. It becomes of increasing interest to the extent that it can be interpreted in quite different areas of observation and theory. Toward this end, this chapter will discuss possible correspondence between MUL construction, behavior and problems and that of the human infant.

Specifically, RPs are compared to cortical neurons. The idea of neuron fixing is introduced, corresponding to the formation of a new RP. It is proposed that neurons can be directed to record the synaptic activity they receive as episodes, forming a neuron-local episodic memory. Further proposals suggest that neural fixing is a response to regularities discovered by the neuron in its local episodes, thus introducing pattern induction. Discussions follow on defaulting and inhibition in the cortex.

Later sections of the chapter relate the behavior of the modified MUL model with observations about human infants. Subcortical elements are discussed in order to put the cortical model and the difficulties inherent in brain-studies in perspective. Section 5.9 summarizes the four forms of memory in the MUL model: echoic, short-term, perceptual, and episodic. Finally, some insights arising

from the model are presented concerning possible future progress.

5.1 The RP as neuron

The neuron and surrounding glia cells seem to be the basic functional unit of the central nervous system [Hyden 73]. The RP as the basic unit of knowledge representation in MUL exhibits functional similarities with neurons.

A sending neuron communicates information to a receiving neuron by emitting chemicals (called neurotransmitters) into confined spaces (called synapses) next to the receiving cell's covering (called its membrane). Neurons generally establish synapses at the termination of a long branching extension called an axon, although neurons exhibit wide variety in structure. When a neuron fires, it changes the membrane of its axon where the axon connects to the cell body. The membrane change propagates down the axon, triggering the release of neurotransmitters at the synapses as they are encountered.

Current theory suggests that neurons fire when ion concentrations within the cell body reach certain threshold values. The ion concentrations are altered by the action of neurotransmitters at the neuron's post-synaptic receptor sites. These sites are usually located on the neuron cell body or on non-axon branching structures called dendrites.

The effect of the receipt of neurotransmitters at a particular post-synaptic site seems to depend on factors like the distance of the site from the cell body; the size, structure, and efficiency of the site; the type of neurotransmitter; and the metabolic conditions in the neuron. Normally, action at a single synapse will not alter ion concentration enough to cause neuron firing. Instead, harmonious effects at several synapses are thought to integrate, combining to push ion concentrations over the required thresholds.

Recognition by RPs is similar to firing by neurons in that several inputs of at least some threshold strength combine to cause activation of the RP or neuron. Many differences exist, however, between the RP and the hypothesized neuron just described. Instead of cataloging these differences, a somewhat different theoretical neuron will be presented.

Central to the theoretical neuron postulated in this dissertation is the concept of <u>fixation</u>. Each cortical neuron is initially unfixed; the firing (if any) of an unfixed neuron is not meaningful. Each neuron eventually becomes fixed; the firing of a fixed neuron means that the neuron has recognized a specific pattern in the activities at its post-synaptic sites. A fixed neuron is hypothesized

to correspond to an RP. These post-synaptic sites involved in the specific pattern recognized by a neuron correspond to the components of the RP. Only a small percentage of the roughly 2500 post-synaptic sites of any fixed neuron are likely to be components. However, fixed neurons probably have more than the two or three "components" associated with RPs. In addition, while RPs specify only that components be active (or have been active just previously), neurons might be able to specify more complex requirements.

Neurons begin as unfixed, and at some point become permanently fixed; pattern induction is the mechanism of fixation. We postulate that each unfixed neuron formulates episodic memories of the post-synaptic activities it experiences, probably first as RNA molecules, and later as corresponding peptide chains. Each neuron performs a chemical equivalent of pattern induction on its episodic memories to discover regularities in post-synaptic activities. When an unfixed neuron discovers a regularity, it fixes itself to recognize future instances of that regularity.

Each neuron performs pattern induction in parallel with other unfixed neurons. However, every neuron in the brain does not fix at once. Only the activity of fixed neurons is meaningful, so neurons record episodes only for post-synaptic activity caused by fixed neurons. Initially, only subcortical (in particular, sensory/motor) neurons are

fixed. In the neonate, then, only neurons receiving substantial input directly from fixed (sensory/motor) neurons may compile sufficient episodic memories to enable fixation. As these cortical neurons fix, neurons that receive output from them may begin the fixation process.

Each cortical neuron has thus far been regarded as an independent processor, influenced only by the activity of the neurons making synaptic contact at its post-synaptic sites. It is unlikely, however, that each neuron alone selects the regularity upon which it fixates. Many, if not most, regularities are not useful for recognition. The "blades of grass" example of Section 2.8 illustrates the corresponding problem for RPs. MUL solves the problem by introducing an instance selection mechanism to define what events are worth pattern induction. We postulate the existance of a subcortical instance selector, called UNEX, capable of detecting ‹UNEXpected neuron activity and of assessing the importance of such activity. When ‹UNEX detects unexpected neuron activity, it alerts unfixed cortical neurons in the region of that activity. Only when alerted by ‹UNEX do unfixed neurons make episodic recordings. Thus, any regularities the neurons discover will relate to occasions when some important and insufficiently anticipated event was occurring.

Some neurological evidence supports this proposed neuron activity:

- Cortical neurons apparently have a genetically (rather than experientially) defined synaptic structure, implying that synaptic distribution is probably not a result of learning. This conjecture is strongly supported by studies of *in vitro* neuron cultures. Cortical neurons can be removed from developing brains and grown in cultures. Requiring only the timely addition of a chemical known as the Nerve Growth Factor, these cultures develop in remarkable mimicry of their *in situ* counterparts [Crain 76]. Genetic control of general synapse placement implies that evolution has solved one difficulty with the MUL computer model: the formation of RELATED_RPS. For each unfixed neuron, the subset of its post-synaptic sites that receives inputs from fixed neurons corresponds to the RELATED_RPS set in MUL pattern induction.
- Myelination, a process where neurons have their axons insulated by glia cells, may indicate neural fixation. That is, a cortical neuron is fixed if and only if it is myelinated. The myelination process follows a course similar to that expected for the fixation process postulated above, and has similar results. When an axon becomes myelinated, its conductive properties are altered: the membrane alterations of nerve firing are, perhaps for the first time, quickly and reliably propagated throughout

the axon structure. This myelination is regarded as the last step in the development of neurons. It is generally completed for subcortical neurons before birth. It proceeds in the inner layers of the cortex, where subcortical inputs are directly received, before continuing to the outer layers (in any particular cortical region). It proceeds in sensory projection areas and the motor area long before beginning in the "association areas" of the cortex, or regions presumed to be responsible for higher mental functions. Myelination continues through puberty, possibly into adulthood in the frontal lobe of the cortex. Myelination is carried out by glia cells, meaning that the fixation of a neuron can communicate be detected by other cells. It seems reasonable to suggest that the neurons receiving output from the fixing cell can also detect the fixation [Altman 67]. Also, glial proliferation is slowed in the visual cortex of visually deprived rats [Davison 77], a fact that could indicate slower fixation rates.

- There is certainly a subcortical mechanism corresponding to UNEX, at least in part. Infants show distinct alertness in response to changes in their environment. When the changes represent totally new phenomena, the infant's alertness may be intense and prolonged. Alertness can be measured externally by eye and head movement, heart rate, and respiration rate [Reese 77].

More specific to learning, alertness is marked by preparation for heightened protein manufacture, as discussed below.

- The neurons in areas thought by neurophysiologists to be engaged in learning show heightened levels of RNA. At first the gross substructure of the new RNA reflects that of cell genomes. That is, the RNA produced seems to be messenger RNA from the cell's DNA. Later, as learning progresses, the RNA shows unique composition indicating that it may be forming in response to learning. The effects of the learning will not appear if protein synthesis is chemically prevented, but will appear sometime after the synthesis is allowed to resume as long as the unique RNA is still present [Hyden 67]. This observation indicates that RNA formed in response to experience, perhaps representing episodes, is used to create specific polypeptides, perhaps permanent episodic memories.
- Evidence now suggests that synaptic activity may have impact upon neuron mechanisms other than the membrane ion permeability. In particular, the effect of at least some neurotransmitters appears mediated by the construction of cyclic AMP at the post-synaptic site. Cyclic AMP has been implicated in control of diverse intracellular activities, as it also mediates intracellular effects of hormones. Most important, cyclic AMP apparently initiates a specific

series of reactions resulting in protein synthesis [Nathanson 77]. Further, another chemical, called GMP, is apparently produced when synapses are not active. These chemicals, cyclic AMP and GMP, may act to direct episodic memory construction.

- The number of dendrites increases markedly throughout cortex development. This increase. and the increase in glia cells, accounts for brain size increases. (The number of neurons is apparently constant.) Dendrites may increase in order to provide storage facilities for the protein/RNA episodic store postulated above, as well as to extend synaptic input. No reliable evidence suggests that new synapses form on any large scale for learning rather than developmental reasons.

5.2 Defaulting among CNS neurons

The concept of defaulting is important in the effective use of knowledge as represented by RPs. If fixed neurons act like RPs, then neurons also should exhibit defaulting behavior.

Suppose a fixed neuron receives activations at most of its required input synapses, but some required synapse(s) remains unactivated. For defaulting to occur, the neuron must encourage, even force, the inactive neuron(s) to fire.

One possible mechanism might be some sort of reverse communication across synapses. No neurophysiological evidence supports the existence of this mechanism, but none denies its existence either.

Another possible mechanism relies only on normal synaptic mechanisms themselves. Research has indicated that synapses are not always axon-to-dendrite but may be dendrite-to-dendrite, axon-to-axon, or dendrite-to-axon [Shepard 74]. In particular, structures called "reciprocal" synapses consist of a pair of synapses between two neurons. The two synapses are oriented in opposite directions, so that activation in either cell should activate the other. Defaulting, then, might conceivably be implemented by reciprocal synapses. However, reciprocal synapses appear infrequently, while defaulting seems to be required quite often.

In still another possible mechanism, a neuron could cause default activity by growing a branch from its axon to the component neuron to be defaulted. Axonal growth has been observed after damage to an axon and during maturation, but has not been observed in mature cortical tissue.

Defaulting was useful in the MUL model as a means of anticipating future input among other things. Anticipation was accomplished by keeping track of RP activations and their relative times of occurrence in a global episodic

memory. The same technique is not easily applied to neuron-local episodic memories of the type proposed for pattern induction to attain fixation; maintaining such a global episodic memory would require that each neuron communicate not only its current activity status, but its expected status to a global storage place. In our discussions above, we have not postulated chemical or structural mechanisms adequate for such communication. Instead, the mechanism of anticipation probably involves the as yet unexplored UNEX, since, if UNEX is to detect unexpected neuron activity, it must have a means of representing expectations. In Section 5.8, an attempt is made to identify UNEX with the hippocampus, a subcortical element with which the cortex has much direct and indirect communication. Anticipation may be accomplished through some representation of expectations in the hippocampus. The lack of even rudimentary understanding of the many subcortical elements communicating with the neocortex prevents effective attacks on this problem.

5.3 Inhibition in the cortex

Inhibition is another crucial mehcanism in MUL. Inhibition serves both as a mechanism to refine behavior (by preventing improper defaulting) and as a mechanism involved in behavior evaluation (through the conflicts it causes).

Inhibition is also a main feature of neuron activity, and its purpose may be similar.

The activation of a neuron may be inhibited, or at least strongly discouraged, by moving the neuron's ion concentrations away from their threshold-for-firing values. Apparently such moves can be brought about by activity at inhibitory synapses on the neuron. These inhibitory synapses are usually found on neuron cell bodies. If located on a cell body, a single synapse can have a powerful effect on ion concentrations at the axon base (where the thresholding is apparently measured).

As in MUL, inhibitions seem to develop in a given area of the cortex after neuronal fixing has begun in that area. Biochemical evidence for this observation is obtained by measuring the amount of inhibitory neurotransmitters present in brain areas as they develop. These measurements show a steady rise in relative amounts of inhibitory neurotransmitters with respect to excitatory neurotransmitters. Further evidence arises from studies of the effects of inhibition-blocking. In these studies, drugs that prevent inhibition are injected into animals: Young animals show little effect while older animals are mentally incapacitated, indicating a shift toward reliance on inhibitory mechanism with age [Reese 76].

Inhibition is so centrally important to MUL that a neurophysiological basis for inhibition is strongly

suggested if the MUL model is to be substantiated. Unfortunately, no neuroscientific evidence has yet been found other than the indirect evidence mentioned above.

To continue by way of speculation, suppose that cortical neurons do employ a MUL-like inhibition. The cortical equivalent of a conflict, then, would occur when a fixed neuron (possibly subcortical) receives activations on all of its components (thus recognizing the regularity for which it is fixed), but also receives activation at one or more inhibitory synapses.

Inhibitions could prevent defaulting if neurons are capable of distinguishing between ion concentration changes caused by neurons trying to activate them by default and those caused by activity in their "component" synapses. Such a discriminitive capability would appear to require information other than concentrations alone. Perhaps ion concentration changes are not the only mechanism used to decide when cortical neurons are to fire.

Inhibitions could be learned by a process similar to MUL's predictive pattern induction. The process would begin with the detection of a conflict: a neuron finds itself inhibited but has received activations at all its "component" synapses.

First, the cause of the conflict must be determined. In MUL a likely culprit is selected from the set of defaulted RPs that support either the RP activation that

defaulted the inhibitor or the inhibitee. If a similar mechanism is employed in the cortex, then a mechanism to communicate backwards across synapses seems required. Given such a mechanism, the occurrence of a conflict could trigger a backward signal across all currently active synapses of the neuron containing the conflict.

When such a backward communication is received by a neuron, the signal could be communicated to all the active synapses of that cell. If the cell was activated as a default, then it could consider itself a culprit. It would be able to tell if it was activated by default (because, by definition, that means that at least one of its "components" will be unactivated). If the cell was not activated by default, the backward communication might be propagated on to all the neurons whose firing activated "components", although this backward propagation might reach avalanche proportions if it were allowed to continue indefinitely. In MUL such an avalanche is stopped simply by limiting the propagation to four steps. It is doubtful the neurons could keep count, though a similar effect might be achieved by some kind of diminishing signal intensity.

Neurons that have designated themselves as culprits are to be inhibited. If a culprit possesses an inhibitory synapse from an unfixed neuron, the new inhibition could be implemented by fixing that neuron to fire upon the required occasions. If the unfixed neuron has preserved its recent

input patterns then upon some kind of triggering signal it can fix itself in the way suggested in Section 5.1 (how unfixed neurons transform themselves into RPs capable of recognizing a particular input pattern).

The concept of the componentless inhibitory RP can be introduced to advantage at this point. There is quite strong empirical evidence that only a small number of inhibitory synaptic sites occur directly on neuron cell bodies. Only sites on the cell body are likely to have a sufficiently powerful effect to single-handedly inhibit a neuron, so one might suppose that for any given neuron there is by only a small number of potential inhibitors. The inhibitors might be cells that act like componentless inhibitory RPs. A componentless neuron might be activated by <u>any</u> activation from a synapse with a fixed neuron. Alternatively, such a neuron might preserve several sets of input requirements and fire if any were met. In either case, the inhibiting neuron would not become fixed as requiring a certain unique, definite patterns of inputs.

Current orthodoxy suggests without proof that neurons either inhibit or excite at all tbeir synapses. If so, then neurons corresponding to componentless inhibitory RPs must have only one (operative) output synapse, namely the one to the cell to be inhibited. If instead a componentless "RP" neuron can designate a particular output synapse as inhibitory then the inhibition learning process might

involve any unfixed neuron contacting the culprit's cell body. Any such neuron that learns to fire at appropriate times to inhibit the culprit can fix itself, making its synapse to the culprit inhibitory. The other outgoing synapses would remain excitatory. In this way inhibiting neurons would behave the same as excitatory neurons during both learning and excitement phases, except that some synapses may become inhibitory during fixation.

5.4 Problems with neuroscientific studies

The proposals of neocortical operations made in this dissertation are just that, only proposals. No evidence directly confirms or refutes them. But the neurological study of cortical neurons in general is restricted to compiling circumstantial evidence. Neurological evidence is accumulated by three methods: microscopic study, electronic study, and chemical study. Each method has its problems.

Neurons, if stained, may be seen with light microscopes. Neurons often have very complex structures with hundreds of dendrites intertwined among glia cells and among the dendrites and axons of hundreds of other neurons. The Golgi stain is used to study neuron structure because it capriciously stains about one percent of the neurons (for unknown reasons). The affected neurons are stained almost completely: only ultra-small "dendritic spines" may avoid

the stain. The structures of individual neurons are insufficient to determine neuron connectivity, however.

Electron microscopic techniques have been helpful in determining neuron structure. Unfortunately, neurons must be fixed (killed), sliced, and "stained" with electron-dense chemicals before they can be studied. Such treatment may introduce artifacts that mislead researchers. Further, it is difficult to reconstruct the overall structure of the neurons from slices, even to determine whether the cell parts seen are axons or dendrites [Cottrell 77].

Electronic measurements can indicate neuron activity. EEGs indicate gross, aggregative neuron activities. Measurement of potentials within individual neurons can be accomplished by microelectrodes inserted within the brain. Microelectrodes are not small enough to record activity in small dendrites, but can be positioned within large axons and dendrites, and within cell bodies. Unfortunately, it is not possible to select particular neurons in a cortex for study; this capability would permit the insertion of more than one electrode in a single cell so that activity within a single neuron could be more precisely monitored. Also, if particular neurons could be selected, electrodes in several dendrites, cell bodies, and axons of interconnected nerves could resolve questions about relations between input synaptic activity and neuron firing. In any case,

microelectrodes may alter the electrical behaviors of the cells they penetrate, making such studies suspect.

Chemical analysis of neuron activity is generally hampered because cells cannot be individually examined. Instead, groups of cells are assessed for a composite chemical property or behavior. One exception may be important: the distribution of radioisotopically tagged chemicals can be determined by a microscopic examination. Here, too, problems prevent straightforward interpretations of results. For instance, the tagged chemicals can enter a neuron only after diffusion or transport across the neuron membrane and can approach a neuron membrane only after traversing the glial cell "blood-brain barrier". This complex path leaves much room for interference by extraneuronal factors [Rose 77].

In short, every study of cortical neuron behavior is necessarily indirect, providing only circumstantial evidence about that behavior. Even the circumstantial evidence is often suspect because all factors could not be controlled by the experimenters. Finally, the evidence is often ambiguous and open to a wide range of interpretations.

5.5 Neurological questions raised by MUL

Given the difficulties inherent in neurological study, it is not surprising that no neurological model has gained

universal support. Certainly the proposed model raises more neurological questions than can yet be answered. This section briefly discusses some of those questions.

The MUL model requires the recording of episodic memories. In the MUL program a global history of all RP activities is maintained. Global recording is convenient on a single-processor computer. In the cortex, neuron-local recording by RNA and protein formation has been proposed. Local recording of episodes relieves the need for sophisticated communication mechanisms between neurons and a central recording site. While circumstantial evidence in the form of generally elevated RNA and later protein formation among learning neurons supports the local episodic memory theory, the evidence also supports a variety of altogether different theories. Episodic memory does require some distinctive mechanisms, as listed below. Neuroscientific research may confirm or infirm the presence of such mechanisms. (Speculative activity like this dissertation can play an important role in suggesting what the research might look for.)

- Mechanisms must exist for distinguishing the episodic recordings from different synapses, and from a given synapse over time. That is, recordings must be somehow identified as to when and where they were made.
- Pattern induction over episodic memory recordings requires comparison of episodic recordings. If recordings are to

be compared, they must be comparable. Specifically, they must be locatable within a common time frame. The synaptic recorders must have a standard "recording speed".

- When episodic memories exhibit regularity, some mechanism must "fix" the neuron, presumably by adjusting synapses to detect that regularity. Note: MUL as a program did better with RPs having a small number of components, because small fixed-size data structures are more easily programmed and because smaller RPs were less likely to be redundant. Neurons might have many components, however. With neurons, the redundancy problem seems to be avoided because each neuron has a unique set of connections to other neurons. In MUL terms, each neuron has a unique RELATED_RPS set. Cortical neurons have about 5000 synapses on the average. It is likely that fixed neurons would place requirements on substantially more than two or three of their synapses.

Episodic memory is not the only unsubstantiated mechanism suggested by the MUL model. Backward communication across synapses needed for learning inhibitions (and possibly for defaulting) has not yet been observed in neurological studies, nor has a UNEX mechanism that detects unexpected events been located or explicated with any great plausibility.

It may seem unreasonable that the MUL model presumes so many novel and complex activities in neurons. However, the

complexity required does not seem incommensurate with known cell operations of metabolism and reproduction. Also, the proposed activities operate at the molecular level, a level that has not been amenable to study and understanding.

At least we can conclude that the MUL model is not incompatible with known neuroscientific evidence concerning cortical neurons, after allowances for distortions in MUL caused by its implementation on a single-processor computer.

5.6 Evidence from experimental psychology

This section will discuss some results from psychological studies in relation to the proposed MUL cognitive model.

Piaget studied the growth of young children and organized the many changes he observed into a set of stages in intellectual development [Piaget 54]. The stages he observes seem compatible with the hypothetical development of a MUL-model intellect.

The new-born infant, until about two months, stares at new scenes for long periods. In the MUL model such behavior is expected: the infant would not yet have learned any voluntary motor behavior and so his vision must be under control of lower "hard-wired" brain centers. In the infant, lower brain centers are apparently responsible for quite complex behaviors, notably the orienting response. When a

new object is presented to a newborn, the baby turns and fixates on the first edge of the object. In addition, the baby's heart rate increases, its respiration rate is altered, and its sensory receptor sensitivity is enhanced [Reese 76]. It is not unreasonable to speculate that this orienting reflex is, at least in part, an externally observable effect of the proposed UNEX mechanism.

A four-month old baby can track moving objects but does not seem to recognize them as specific entities. For example, when a moving object is altered during a brief passage behind a screen, the four-month infant shows no excitement. However, he can visually scan new objects. For example, when presented with a new object, he scans the object rather than fixating on the first edge of the new object as would a neonate. It seems reasonable to expect that a MUL-modeled intellect would develop in a similar fashion. That is, as UNEX repeatedly signals the importance of a moving object and generates movements to look toward the movement, the cortical neurons would record episodes of moving objects and efforts to orient toward them. Such events must show regularities among some neural firings, so neurons might begin to fix to recognize them. The young infant cannot yet make a thorough study of objects if only because he lacks sufficent motor control to manipulate the object. Therefore, it seems unlikely that his object

characterizations would be complete enough to differentiate objects.

By six months, the infant can pick up objects. He can also track erratically moving objects. Also, if he is physically moved, he can follow a stationary object.

As the infant approaches one year of age, he can search for an object hidden under a cloth, if he sees the object being hidden. If the object is moved while under the cloth, the infant will still search where he saw the object placed even though the cloth there lies flat.

By a year and one half, an infant will find an object under a cloth regardless of how it was hidden.

Literally hundreds of experiments and observations have been made with infant development. The above are fairly representative. The responses observed are probably not uniquely the result of development in any particular area of the brain. For example, learning to scan objects probably involves development in the visual cortex, the motor cortex, the association areas of the cortex, and subcortical brain elements such as the cerebellum and the collicullus superior. It is not yet possible to separate the contributions made by development in these and other areas.

5.7 Estimating the size of episodic memories

Not only does development proceed simultaneously at many points in the central nervous system, but that development may be exceedingly complex. Suppose the MUL model is valid. This section estimates a lower limit on the amount of episodic recording that occurs in the visually-involved cortex of the neonate.

The neonate brain receives about two million vision input nerves at the visual cortex. We estimate that each of these inputs has more than fifty synapses with cortical neurons. Since sensory inputs are considered fixed and cortical neurons are unfixed, the neonate would by this estimate make episodic recordings of roughly 100 million fixed-to-unfixed synapses. Optical receptors can respond within a tenth of a second to new stimuli, so, as an estimate, suppose that episodic recordings record at roughly 10 bits (synapse active or not) per second. When a neonate is presented with a new scene, he can stare at the new scene for many seconds before becoming habituated: suppose, then, that episodes last 10 seconds. The volume of information recorded by the neonate in its first 10 seconds of contact with a new scene is thus roughly estimated as (10 seconds) X (10 bits/second/synapse) X (50 synapses/optical input) X (2,000,000 optical inputs) = 10 billion bits of information.

In two months time, if the infant is excited only one percent of the time, it would still be excited for 50,000 seconds. Assuming that each neuron has at least 2000 output synapses, then, as each neuron fixes, the number of fixed-to-unfixed synapses is likely to rise exponentially (at least so long as the number of unfixed neurons outnumbers the number of fixed). As a rough guess, suppose that on the average about one half, or 1000, synapses of each newly fixed cell will impinge on unfixed neurons. If we estimate that about 100 million (1% of the roughly 10 billion) cortical neurons will fix in the first two months, then about 100 billion new fixed-to-unfixed synapses will exist. All in all, we can estimate that in two months time, episodes will be recorded for 50,000 seconds at an average of 50 billion synapses at a rate of 10 bits per second, so that by age two months, the infant has likely recorded more than 25,000 trillion bits of information. Further, these records probably include more information than an on/off synaptic activity pattern. (The identity or location of the synapse is needed, for example.)

This staggering amount of information makes clear that MUL-like models based on normal computer architectures will not be effective in any but toy environments. It also helps explain why the brain is the most active protein-producer in the body, and the largest single user of energy.

5.8 Subcortical elements

Learning in a human infant is a slow and complex process, as it is in the MUL model. As has been discussed, it is not yet possible to judge where or when or how an infant learns particular knowledge. One source of difficulty is the complexity of operations in the cortex and the inability to observe them directly. Given that indirect observations must be used, further difficulty arises from interference from the activities of various subcortical elements, a complication not dealt with in the program version of MUL.

Many behaviors have been shown to arise in subcortical areas. For example, the orienting reflex described in Section 5.6 must be subcortical in origin because it is present in infants having no cortex. Several other high-level functions are apparently "hard-wired" into subcortical elements of the brain. For instance, infants are capable of short but complex motions like grasping. They are also capable of depth perception, including focussing their eyes on a given distance and coordinating eye movements.

In addition to complex behaviors performed independently of the cortex, the inputs to and outputs from the cortex may be complex. Visual input from the retina is not merely a transmission of rod and cone outputs to the

visual cortex. For one, inhibition-centered systems in the retina and retina ganglia apparently perform edge-detection, image sharpening, and motion-detection. At an even more complex level, the retinal ganglia can apparently do spatial frequency analysis akin to fourier analysis. This type of analysis is very important because its results are image-size independent. Size-independent input may allow easy recognition of shapes at various distances from the observer [Granitt 77].

In addition to the cortex-independent reflexes and input preprocessors, the brain evidently contains several subcortical elements that can learn, further obscuring the contributions of the cortex. Among these are the cerebellum and the limbic system.

The cerebellum apparently learns often-used motor output sequences. Its main inputs and outputs are very high-speed muscle status sensors and muscle activators. When the cerebellum has learned a motor pattern, and that pattern is activated, the cerebellum apparently acts to oversee the execution of the pattern by monitoring progress and issuing corrective orders [Witte 73].

Of all the subcortical elements, the limbic system is probably the most complex and most influential. It has been hypothesized to be evolutionarily older than the cortex, related to the so-called basic drives: the quest for food and water, self-protection, territoriality, and

reproduction-directed behaviors. In general, the limbic system seems to be responsible for deciding when sensory inputs describe a pleasurable situation, or a situation to be feared, or an unusual situation worthy of more study [Sagan 78, Quarton 67].

Of particular interest in the limbic system is the hippocampus. It is thought that the hippocampus stores or at least evaluates expectations about near-term sensory activities [Quarton 67]. That is, it stores expectations about future experiences. The hippocampus, then, may be the organ (UNEX) that can detect unusual or unexpected events. Other parts of the limbic system might then assess the goodness or badness of the new events and perhaps initiate emotional responses to them.

Many subcortical elements will not be discussed, but the thalamus deserves brief consideration. This element has been called the gateway to the cortex because most inputs to the cortex actually originate in the thalamus. It is hypothesized that the thalamus acts as a switching center. It may act as the "importance assessor" of the MUL model, by sending through input to be considered and filtering out input to be ignored. On what basis and by what means the selection might be decided remain open problems.

This section has touched on only the more important of the many subcortical elements. It is thought that the brain

may actually consist of three layers: the primitive, reptilian reticular-formation responsible for involuntary motor activities; the limbic system responsible for ascribing value to experiences; and the cortex responsible for thought and voluntary activity. In addition, a variety of special purpose elements may perform specific tasks in the brain, as for instance, eye focusing. Further, each of these layers seems to be able to do some learning.

The following observations put the MUL model in perspective, as a model for cortical operation (rather than a model for the entire human brain).

- The behavior of the human is likely a function of activity in all of these brain layers and elements. It is not likely that any behavior can be isolated as purely cortical.
- Neurons of the cortex receive inputs from and produce outputs to the lower brain elements. When neocortical neurons can recognize and anticipate activity, they may begin to shape that activity, for example, by creating expectations in UNEX, or by taking partial control of or acting to initiate various motor tasks. Subcortical elements act not only as preprocessors for and controllers of the cortex, but also as <u>teachers</u> for the cortex. Cortical neurons learn from regularities in the behaviors of subcortical elements.

5.9 Long-term memory

This dissertation describes learning as a process of cortical neuron fixation. For several reasons, it appears that another sort of learning occurs in the cortex (or at least in some cortical areas). Specifically, it is hypothesized that the episodes recorded within certain neurons may be recalled from outside the neurons.

In the MUL model, information is represented in four different memories, although in only three has an access mechanism been provided to reach the information.

1. Echoic memory

Current experience is represented by neural firing patterns or, in MUL, by RP activations for a particular time unit. Neural firing patterns rapidly decay (in the absence of reverberation). If two different visual stimuli are presented in quick order, then we would expect the patterns to co-mingle as neurons fired from the first stimulus will still be active when neurons begin firing from the second stimulus. If the first stimulus is presented just long enough for the cortical neurons it initially fires to become inactive, and then a second stimulus is presented, then we might expect that the neural firing pattern from the second stimulus might closely follow the pattern of the first, overriding and destroying it. These phenomena - decay,

integration, and erasing - are observed properties of human "echoic" memory [Coltheart 76].

2. Short-term memory

It was hypothesized in Section 4.8 that RPs could be re-excited in reverberatory feedback loops. Reverberation in such circuits can preserve information as long as nothing interferes with the reverberations. No mechanism has been described that could learn (that is, establish appropriate synaptic arrangements for) purely cortical reverbatory microcircuits, although apparently such circuits may exist in the cortex. Feedback loops may occur as a result of neural development, or they may arise from the action of subcortical elements, notably the thalamus. Competition for thalamic attention may limit the amount of short-term memory to about seven items for any one cortical area. Memory by reverbatory excitement seems to possess characteristics ascribed to short-term memory in humans. Short-term memory is a general memory [Coltheart 76]. It must be rehearsed. It seems to exist independently for different cortical regions. For each cortical region, it can preserve only a small amount of information. The information may be complex, but apparently contains only information local to the processing done in its particular cortical region. Also, humans apparently can choose what they preserve. These properties are compatible with a

re-excitement memory that must be rehearsed through thalamic interaction.

3. Perceptual knowledge

Neural fixation, as a permanent change in response to experience, is also a form of memory. It can properly be called perceptual knowledge. It seems unlikely that neural fixation is the only mechanism for the long-term preservation of knowledge, because the number of neurons is fixed and fixation must eventually deplete the supply. Cortical neurons are almost completely myelinated by the end of puberty, so that if fixation does entail myelination, then no further significant learning of this kind is done past about the age of 20.

4. Episodic memory

It has been suggested that neural fixing patterns are preserved as molecularly recorded episodes within individual neurons. Until now in our speculation, those episodes were used only in neural fixation. If by some mechanism the information encoded within episodes could be made available outside the neuron that contains them, we would have the basis for a powerful long-term memory.

Episodic memory may not be accessible in fixed neurons of the kinds so far hypothesized. There are, however, several different cell types distributed throughout the cortex. While different cell types may exist to provide different classes of RELATED_RPS or different functions

within the fixing model (e.g., inhibitory cells), it is possible that some cell types do not fix in the suggested way but instead have the different role of collecting accessible episodic memories. Or, accessible episodic memory may reside in a subcortical element. Wherever it resides, several properties might be ascribed to accessible episodic memory.

- it would be of large capacity
- it could continue throughout the life of the cortex/brain, although forgetting could easily occur if episodes decay
- it would preserve portions of particular experiences, or pattern induction's generalizations of them

Some general evidence supports an accessible episodic store. For one, people do remember episodes! The usual sketchiness of those memories might result because episodes are recorded within a particular neuron and the neuron could record only the activity that it received directly (necessarily a rather local view). Further, people generally remember little of their experience before the age of about 5 years, and their memory for episodes improves asymptotically up to a level usually reached at puberty. If accessible episodes record the activities of synapses with fixed neurons, then the delayed onset and steady improvement in episodic memory roughly parallels the neural fixation of

neurons capable of perceiving meaningful "global" qualities of situations. Finally, electronic stimulation in the temporal region of the cortex seems to trigger the re-experience of previous episodes in the subject's life.

This idea of accessible episodic memory has not been explored in a MUL program model.

5.10 MUL as a cognitive model, in summary

The MUL model has been extrapolated to become a model for the human cortex. Too little is known about the cortex for the model to be declared good or bad. Additionally, the model addresses such low-level activities of the cortex that little can be said about the combined activities of millions of cortical elements (at the level mainly addressed by MUL) and dozens of ill-defined subcortical elements.

In all, the implications of the MUL model are somewhat discouraging. It suggests that sophisticated information processing is done at a subcellular level, at least for learning and memory. It suggests that the meaning of the firing of a particular neuron will probably not have an easily defined meaning, such as "the sky is blue". We may conclude that computer models for human cognition may be a long way off, and that neuroscience must solve many subcellular puzzles before neuron behavior can be understood. In general, the intellectual development in a

particular individual may always be nearly indecipherable to external observers, not easily interpreted, and not easily influenced by any but crude ways.

Chapter 6: Conclusion

This chapter summarizes first the concept of pattern induction as we have postulated it and then the RP structure suggested for the representation of perceptual knowledge. The final section suggests directions for possible future research.

6.1 Pattern induction

Pattern induction is the analysis of episodic memory to discover recurrent relationships among events. This research has not suggested, nor is it intended to suggest, the definitive program-model implementation of pattern induction. However, the algorithms and data structures that are presented (for both predictive and non-predictive pattern induction) are a useful step toward mechanisms at least sufficient for an implementation.

The general concept of pattern induction, as opposed to the specific algorithms and data structures proposed to accomplish it, has several attractive characteristics. These characteristics are discussed in Section 3.9 but are summarized below:

- Pattern induction is cautious. It will normally learn only after several similar experiences (although the experiences might conceivably be imaginary).
- Pattern induction is self-correcting. New RPs are never incorrect, and are never modified once learned. Instead, pattern induction modifies RP use in defaulting: new RPs activate inhibitory RPs to prevent improper default activation. Inhibitory RPs may be activated only by default; any inhibition (correction) may itself be inhibited (corrected) by new RPs introduced by pattern induction.
- Pattern induction is universal. The interpretation of preprogrammed RPs is not important to the learning process. A given RP may mean <color middle middle black> or <bottle filled>: pattern induction will attempt to anticipate its activations in any case. In particular, new RPs can be introduced and their activity patterns analyzed in turn by the same pattern induction algorithm.

The particular pattern induction mechanisms developed for this research possess the above characteristics in varying degrees. These mechanisms are universal. They are cautious but have difficulty in defining and locating similar experiences. They have at least shown a potential for self-correcting behavior.

The rough calculations presented in Section 5.7 indicate the enormity of resources required for pattern

induction in the human cortex, if something like what we have postulated does occur there. Even with vastly improved computational speeds and memory sizes, it is difficult to conceive of pattern induction proving useful in the human environment on a single-processor machine. Something approaching a neural-net like architecture, perhaps of millions of processors, would appear to be required.

6.2 Representing knowledge: RPs and inhibitions

The MUL program led to a new knowledge representation, the RP. In final form, RP structures are simple:

- Each RP typically contains three components. (Componentless RPs are exceptions.) Each component references another RP and specifies a time for that RP's activation relative to the times of activation of the RPs referenced in the other components. Each component provides information (a weight) concerning whether that component's RP should be defaulted.
- Componentless RPs are used for sensory/motor interfaces, for generalization, for inhibition, and for reverberatory short-term memory.
- Inhibition is an effect of activations of inhibitory RPs. Inhibitory RPs are always componentless, and have the interpretation of "NOT x" for the single, unique x they inhibit.

The operation of RPs is also simple. At any time unit t, an RP x can become activated in any of four ways:

1) Recognition - when activations can be found for each component RP of x at their proper time units in relation to t.
2) Defaulting - when x is referenced as a component of some RP y and an activation is found for the RPs referenced in y's other components in the proper time unit relative to t, and x is not inhibited for time t.
3) RFI supporting - when x is referenced as a component of some RP y, and y has been defaulted with a sufficiently great importance for the appropriate time unit with respect to t, and x is neither already active nor inhibited.
4) Sensation - when x is a componentless sensory/motor RP and conditions in the environment at time t cause x's activation.

Inhibitions play a direct role in RP activation only by preventing RP activation by default. Here again the mechanics are simple: a default of RP x is not performed if the RP inhibiting x is activated in the time unit of the default.

Inhibitions also play an indirect role in modifying RP behavior. Whenever an RP is inhibited yet activated at the

same time, a conflict is recorded. These conflicts are later examined by a pattern induction mechanism that produces new inhibitions to prevent the conflicts in future activities.

RPs have several attributes that recommend their use for knowledge representation and learning.

- They are easily applied. There is no need to perform complex "closeness" evaluation to determine if an RP should be activated. Even the defaulting process can take only one well-defined action (namely, to default an activation of a specific RP). Defaulting also documents the action it takes: the default RP activation is preserved in episodic memory, allowing future pattern induction to adjust the defaulting action. Finally, RPs are applied without backtracking.
- They are easily corrected. RPs can be wrong only in their defaulting activity: RPs can not be incorrect as recognizers except perhaps for irrelevancy or redundancy. Improper defaulting is corrected by inhibitions without modification of the meaning of existing RPs. Inhibitions change only the applicability of RPs, not their meaning. Since each RP may be referenced by other RPs, changing the meaning of one RP would alter the meaning of all RPs that reference it. Such wide-spread alteration seems impossible to control. Further, changing the meaning of

an RP invalidates the past history of that RP in episodic memory.

- RPs are uniform and small. Learning algorithms are thus more easily programmed.

Knowledge representation by RPs is not without potential problems, specifically concerning its expressive power. An important question is whether such a simple structure represents all knowledge. The answer is probably no.

First, RPs seem weak in representing slow events. Each RP can represent activity across only one time unit. Events occurring across three time units must be represented by at least two RPs. In general, 'n' RPs can represent events across 'n+1' time units. Events separated by an indefinite but short time can be represented by the reverberatory RP masses suggested. However, RPs do not seem suited to representing long-term events, occurring over hundreds or thousands of time units.

Second, RPs seem weak in supporting generalization. While generalizations can be represented by introducing disjunctive componentless RPs, the only mechanism thus far used to introduce these disjunctive RPs is inhibition. Activation of an inhibitory RP denies the activation of the RP it inhibits, forming the practical "negation" of the concept recognized by the inhibited RP. It seems doubtful that all generalizations can be easily represented or discovered as negation.

6.3 Directions for future research

A major direction for future effort is to apply the lessons learned by interpreting the MUL program as a cognitive model. This interpretation not only suggests new directions for an improved and extended program, but it also suggests where the MUL model has been distorted to accommodate a desire to program it for a particular computational architecture and within existing computational resource limits. Conversely, programming the MUL model has produced a detailed, although tentative, model of the operation of the human cortex.

The importance of MUL as a cognitive model will not be evaluated in this dissertation. We leave that to others.

To enhance the value of MUL as a cortical model, several specific alterations are needed:

- The primitive level/area hierarchy used to define RELATED_RPS must be improved. Levels and areas are used to mimic the effects of particular dendritic and axonal interconnections in confining the areas of activity of neurons in the cortex. The effects of these interconnections should be representable in some better way.
- The MUL program had difficulty learning and using inhibitions. A reason for the difficulty was the lack of orderliness in RP activations. To accommodate the single

processor, the program allocates resources to the most important RP. It now seems preferable to at least simulate parallelism, even if only a single actual processor is available. RPs may be treated as being imbedded in a distributed structure all parts of which are to be examined for activity in parallel. Instead of immediately activating an RP when it is discovered to be activatable (by recognition) or when it is to be activated by default, the program would wait until all currently active RPs have been examined. Then, all activatable uninhibited RPs would be activated at once. These new activations would make still other RPs activatable, but again, no RP would be activated until all current activations are processed. This strategy more closely mimics neuronal activation timing. It also allows more time for inhibitions to become activated. Finally, the more predictable sequencing should make inhibitions easier to learn.

- The neural model suggests that neurons might have more than two components, possibly with more time specificity than one time unit. Neurons as postulated can apparently maintain this flexibility because they do their own pattern induction in parallel over hundreds or thousands of inputs. The MUL program as it stands must do its induction serially, for one input of one RP at a time. Further, neurons apparently have their RELATED_RPS set up

by the genetically-directed development of their dendritic and axonal structures. The MUL program must continue to calculate RELATED_RPS for each RP.

- Another idea suggested by brain structure regards the introduction of reverberation. The thalamus has been hypothesized to re-excite neurons as a mechanism for short-term memory. Further work on the MUL model might investigate how a hypothetical thalamus "importance analyzer" decides which neurons to re-excite, by exploring similar activity with RPs.
- An important development in the cortical model is the supposition that episodic memories could be generally or globally accessed in some way. Access to episodic memories provides a long-term memory for particular occurrences that are not available if only RPs (or neurons) encode long-term memories. Accessible episodes show promise in the representation of complex events, an area of weakness in knowledge representation by RPs and neurons. An access mechanism must be proposed and supported both in the MUL program and in neural models.
- Many important brain centers and functions are ignored or trivialized in MUL, both as a program and as a model, even many that directly influence or are influenced by the cerebral cortex. Many of these centers manifest construction that surpasses the cerebral cortex in complexity. Of particular importance are the centers that

give input to the frontal lobes of the cortex, thought to be the conscious planning center of the brain. The notions of planning and hypothesizing at a conscious level have not been explored in this research.

- Subcortical brain elements may act as "teachers" for the cortex. That is, the cortex "observes" the inputs to these centers and their responses to the inputs, and these centers possibly provide indicators of relative importance (say for the survival of the organism). For example, the orienting response may "teach" the cortex how to move the eyes to scan objects.

The main purpose of further work on MUL programs must remain to formulate ideas about cortical structure and function. It should by now be clear that MUL simulations require far too much from current machines to duplicate human behaviors, even in drastically simplified environments. The model does not compete with other representation models like the frames of Minsky, the schemas of Norman and Rumelhart, or the scripts of Shank. Nor is it in conflict with the neural models for cell assemblies inspired by Hebb. It applies to operations at the level of single neurons; as such, it complements these higher-level approaches. They are still needed for understanding and explaining aggregative activities of hundreds of millions of neurons of the brain.

Appendix 1: The TELOS programming language

TELOS is a programming language intended for the complex applications common in artificial intelligence [Travis 77, LeBlanc 77]. Based on PASCAL, TELOS is designed to extend that language by providing facilities for modularization (by data and control abstraction), for pseudo-parallelism (by coroutines and generators), for inter- and intra- process communication (by an event mechanism unifying messages and software interrupts), and for an associative data base. Only the data base related facilities were implemented for the TELOS compiler used in MUL research.

Even without modularization, coroutine, and event mechanisms, TELOS was an invaluable aid to MUL research. Without TELOS, MUL research could not have avoided costly investment in software specific to particular MUL versions, investments that would have to be repeated with each major change in the MUL model. It is unlikely that the MUL model could have evolved in the same way had the use of TELOS not diminished software investment to acceptable levels. The remainder of this appendix discusses aspects of TELOS important to MUL research. (The discussions assume knowledge of PASCAL [Wirth 71].)

The heaviest investments required in a MUL model involve the construction of knowledge stores. Mechanisms must be provided to store knowledge, alter knowledge, and retrieve knowledge (often associatively). Any change in knowledge representation requires alteration of all these mechanisms. A major contribution of TELOS has been to provide these mechanisms automatically and efficiently, via the TELOS data base.

The TELOS data base is composed of data base objects (DBOs). DBOs are dynamically created by user calls to the system STORE function. (TELOS data base functions will be distinguished by capitalization.) The call STORE(<ptr>) will cause the creation of a new DBO having as its value a transcription (into the data base) of the object referenced by <ptr>. The only constraint on <ptr> is that it must be a pointer-valued expression: any objects referenced by pointers may be transcribed into the data base. STORE returns a reference called a DBOP (for DBO pointer) to the new DBO. User programs may use DBOPs as they would pointers, to obtain access to DBO values. DBO values may also be accessed by associative lookup, as will be explained later.

The following is the declaration of the data structure representing kernels in the final MUL version. Its semantics with respect to MUL will be discussed in Appendix

2; only its TELOS data base properties are important here. Numbers in parentheses refer to notes found below.

```
kernel_type = Packed Record (1)
    importance_factor : Real Sequencer; (2)
    activation_time : Integer Indexed; (3)
    activated_rp : DB -> rp_type Indexed; (4)
    support : Array [1..max_components] Of DB ->
                                        kernel_type; (5)
    defaulter : DB -> kernel_type;
    caused_conflict : Boolean;
    End; (*of kernel_type*) (6)
```

(1) TELOS records have the same structure as PASCAL records, and may be packed to conserve storage (at the expense of time). [TELOS keywords are capitalized for readablity only.]

(2) When record objects are stored into the TELOS data base, keywords in the field declarations direct the data base routines. A field marked 'Sequencer' determines the order in which data base objects (DBOs) of this type are returned by associative retrieval operations. A field marked 'Indexed' has an inverted index list constructed for values found in that field (used to meet Requirements: see below). A field marked 'Unstored' is not placed in the data base, and its value is ignored.

(3) Values in the field named 'activation_time' are placed in an inverted-index structure. Later associative retrievals may access kernel_type DBOs having a particular value in their activation_time field.

(4) With the DBOP (for database object pointer) TELOS allows direct access to specific objects in its data base. The field 'activated_rp' provides a cross reference from each kernel directly to the DBO representing the recognizer/predictor (RP) for which the kernel is an activation. The DBOP facility allows direct read access to and write access via a CHANGE function. In general, DBOP types are declared by:

```
dbop_type = DB -> object_type
```

A field or variable of type dbop_type can reference an object of type object_type in the data base.

(5) Though TELOS provides for variable-sized arrays in certain circumstances, the 'support' array has a fixed size of max_components (=3). Since the number of components of RPs is variable, some elements of 'support' may not be used. For each specific kernel, the number of components (equal to the number of 'support' elements used) is given by accessing the RP referenced by the 'activated_rp' field. In particular, given a kernel referenced by a DBOP 'K', the number of components is given by:

```
K->.activated_rp->.num_components
```

```
        (*'num_components' is a field in an rp_type
          record*)
```

The dereference operator, '->', accesses the object named by a pointer or DBOP, as in PASCAL.

(6) Comments within TELOS may appear anywhere, and are demarcated by '(*' and '*)'.

As an example, suppose RP #10 is activated by sensation at time t, where:

RP #10: componentless, sensing "color middle middle red"

Given that the variable "rp_to_activate" (declared as type "DB -> rp_type") contains a reference to the DBO representing RP #10, the following code activates RP #10 at time t by storing an appropriate kernel in episodic memory:

```
(*'new_kernel' is a pointer to a PASCAL-type heap object
   of type 'kernel_type'.  The following code sets up the
   desired kernel in the heap, for later transcription
   into the data base*)
With new_kernel -> Do   (*the PASCAL With-statement
                          opens the fields of the object
                          referenced by new_kernel for
                          direct access*)
   Begin
   importance_factor := 0.9;  (*the importance of
                                sensory input*)
   activation_time := t;
```

```
    activated_rp := rp_to_activate;
    defaulter := NIL;  (*this kernel is not a result of
                                                defaulting*)
    caused_conflict := False;  (*discussed in Appendix 2*)
    (*note: since RP #10 is componentless, rp_to_activate
      -> .num_components is zero, so that no elements of
      'support' are used*)
    End;
  (*the following statement stores a copy of the desired
    kernel into the data base, joining other kernels for
    time t of episodic memory*)
  K := STORE (new_kernel);
```

Associative access to the TELOS data base is accomplished by user routines, called "Match_Routines", in the control of TELOS associative retrieval routines. When a user program requests associative retrieval, it provides a pointer to a PASCAL-like heap object called a parameter record. Parameter records look and act like PASCAL records. They always have a field named 'Routine' for reference to a Match_Routine. Remaining fields are named and typed the same as the formal parameters of the referenced Match_Routine. TELOS associative retrieval routines use the provided parameter record to obtain a reference to and parameter values for the Match_Routine to be used in the retrieval.

For example, MUL versions often need to know whether a particular RP was active at some particular time, that is, whether a kernel is present at that time for that RP. The presence of such a kernel is detected by attempting to retrieve the kernel from the data base. This Match_Routine is used (the numbers in parentheses refer to notes that follow):

```
Match_Routine find_activation_rtn (rp: DB->rp_type;
                                   time: Integer)
      Matching K : DB->kernel_type; (1)
   Begin
   find_activation_rtn := (K -> .importance_factor >
                          minimum_importance); (2)
   End; (3)
Requirements (4)
   Begin
   Require (rp,time); (5)
   End; (6)
```

(1) Match_Routines are like PASCAL functions in several ways. When executed, they communicate with TELOS system routines by returning a value of True or False through their name (in this case, 'find_activation_rtn'). This Match_Routine has two formal parameters, 'rp' and 'time'. Every Match_Routine has an additional special parameter, ('K' in this case). The type of the special parameter indicates the type of the DBOs to be retrieved

from the data base ('kernel_type' DBOs in this case). In operation, TELOS routines use the Requirements section (see notes 4 and 5) to extract candidate DBOs of this type for the retrieval. Each candidate is tested to determine if it should actually be retrieved by calling the body of the Match_Routine with a reference to the candidate assigned in the Match_Routine's special parameter. An example of the use of 'find_activation_rtn' appears after these notes.

(2) When executed for a candidate DBO, Match_Routines specify that the candidate should be returned by assigning a value of True to their name before returning control to TELOS routines. In this case, a candidate kernel is acceptable if its importance_factor is greater than some 'minimum importance'. Match_Routine bodies may contain arbitrary TELOS code, possibly involving other data base operations. Appendix 2 presents a complex Match_Routine body implementing predictive pattern induction.

(3) This End closes the body of find_activation_rtn. When the body is executing to test a candidate DBO, encountering this End returns control to TELOS routines.

(4) Execution of the Requirements section of a Match_Routine gives the TELOS associative retrieval routines information about how candidate DBOs should be selected.

TELOS currently uses inverted-indexing to identify candidates.

(5) Calls to the TELOS Require procedure require that any candidates contain the values specified as actual parameters, in fields that have inverted index lists. By executing the requirements section of find_activation_rtn, TELOS associative retrieval routines learn that candidates for the retrieval must contain the value in the parameter rp, and the value in parameter time, in some Indexed field. Requirements sections may contain arbitrary TELOS code, possibly involving other data base operations.

(6) This End indicates the end of the Requirements section, and of the Match_Routine declaration.

Match_Routines can be executed only system routines, for example, by the TELOS data base routines, in turn called by users to perform data base operations. Parameter records supply the information needed by the data base routines. A parameter record for find_activation_rtn can be declared by:

```
find_activation_prec :->
    Parameters Of find_activation_rtn:
```

Given the following additional declarations:

```
kernel_array = Array [1..5] Of DB-> kernel_type;
number_of_kernels : 0..5;
```

The following code would return, in 'kernel_array[1]', a DBOP to the kernel for the activation of the RP named by 'rp_in_question' at episodic time unit 'time_in_question'.

```
New (find_activation_prec); (*create a parameter record
                                in the heap*)
With find_activation_prec -> Do
   Begin Routine := Routine_Ref (find_activation_rtn);
      (*assign a reference to find_activation_rtn to
        the Routine field of the parameter record.  Any
        Match_Routine having a parameter structure
        identical to that of find_activation_rtn may be
        assigned to Routine.  The TELOS function
        Routine_Ref returns a reference to a routine*)
   rp := rp_in_question;
   time := time_in_question;
   End;
number_of_kernels := 1;
Find (find_activation_prec, kernel array,
                                number of kernels);
```

The procedure FIND is provided by TELOS for associative retrieval. If the above code were executed, FIND would perform several activities. First, FIND determines the type of the DBOs it is to find by examining the implicit parameter to the Match_Routine named by find_activation_prec -> .Routine; it could just as well examine the component types of the array provided as the second parameter.

(Actually, these types are known and checked at compile time.) Second, FIND calls the Requirements part of find_activation_prec -> .Routine, giving values to actual parameters from the remaining fields of the parameter record. In the above case, Requirements for find_activation_rtn is executed with parameters rp = rp_in_question and time = time_in_question. That execution results in a call to the TELOS Require function for the values of rp_in_question and time_in_question. Third, FIND searches the inverted index lists of DBOs of type "kernel_type" for DBOs containing the values rp_in_question and time_in_question. Fourth, for each DBO meeting the indexing requirements (a candidate) FIND executes find_activation_rtn with parameters rp and time bound as before and the implicit parameter, K, assigned a reference to the candidate. If find_activation_rtn returns True then kernel_array [foo] (foo is initially 1) is set to reference the candidate, foo is incremented, and if foo exceeds number_of_kernels (the third parameter to FIND), FIND terminates. When no candidates can be found, or all candidates have been examined, FIND sets the third parameter, number_of_kernels, to the number of DBOPs that were inserted into kernel_array (=foo-1).

TELOS provides several other operations on the data base, besides STORE and FIND. Field values within DBOs can

be altered with the CHANGE operation. For example, CHANGE (rp_in_question -> .caused_conflict, True) assigns the value True to the caused_conflict field of the DBO referenced by rp_in_question. DBO values are completely altered by the REPLACE operation, that transcribes a heap object over the old value of the DBO. (Any DBOPs to the DBO now access the newly transcribed value.) DBOs can be entirely removed by the REMOVE operation. (Any DBOPs to the removed DBO become invalid.) Additional capabilities involve the context-dependency of DBO values; as contexts were not used in MUL research, these capabilities are not discussed [see Honda 77].

Early MUL versions made use of another unique capability of TELOS: its Pattern_Get procedure. The Pattern_Get procedure is a sort of inverse to the FIND operation: FIND takes a description of objects (as a parameter record for a Match_Routine) and searches the data base for DBOs fitting that description; Pattern_Get takes an object and searches the data base for DBOs that contain a description (again as a parameter record for a Match_Routine) that describes the object. Early MUL versions used this pattern retrieving mechanism to implement partial component descriptions (discussed in Section 2.5 and criticized in Section 2.11). By using Pattern_Get, these versions could find those RPs with components that matched a given kernel.

The properties of TELOS are not fully conveyed in the above brief discussion. Below, TELOS properties are listed to further understanding of that language and its importance to MUL research.

- TELOS extends PASCAL, inheriting both program structure and basic capabilities from PASCAL. TELOS programs for MUL research are, if examined cursorily, indistinguishable from PASCAL programs.
- The TELOS implementation built on the University of Wisconsin UW-PASCAL compiler. Thus, TELOS is compiled and executes just as does PASCAL unless data base operations are employed. In addition, TELOS adopts the extensive compile- and run-time checking used by UW-PASCAL: all operations are type-checked; array references are checked, at run-time, if necessary; pointers (and DBOPS in TELOS) are checked to insure that they reference valid objects; and tags of variant records are checked. The future will see the completion of interactive debuggers for both TELOS and UW-PASCAL.
- TELOS (and UW-PASCAL) provide an external compilation facility so that parts of programs may be compiled separately. References to global objects, types, and routines are type-checked from information in an "Environment" section [LeBlanc 78].
- TELOS provides powerful tools for building a customized data base:

A. Any data structure from the heap can be put in the data base. That is, TELOS does not force users to employ particular data structures, but allows them to compose data structures to fit their needs.

B. TELOS automatically adjusts data base operations to deal with any structures the user desires in the data base. The details of data base operations are hidden from the user.

C. The user has simple, if coarse, control over the expensive indexing operations used for associative retrieval. The user may specify exactly which fields (and which values within fields) are to be Indexed, and which are to be Unstored.

D. The user is given direct access to objects in the data base, via DBOPs. DBOPs are useful in several ways:

   i) With direct access to DBOs, the expense of associative retrieval is often avoided entirely.

   ii) DBOs need not be copied from the data base in order to be examined.

   iii) Specific DBOs can be altered, replaced, or removed directly.

   iv) DBOs may cross-reference one another, so that net structures can be constructed.

E. The user is given control during associative retrieval operations, allowing those operations to be arbitrarily complex to suit the user's needs.

F. With parameter records, the user can construct calls to procedures, functions, and Match Routines at run-time. With these parameter records it is possible to implement program-constructed programs. Since parameter records are data structures, they may be created, destroyed, placed in the data base, or manipulated in any other way that a data structure can be manipulated.

The evolutionary development of MUL may not have been realized without TELOS. Of particular importance was the extensive checking performed by TELOS, checking that could often detect the errors that accompany extensive program modifications. Equally important was the relief TELOS provided from details of data base manipulation, without sacrificing access to that data base. Finally, the efficiency of TELOS made at least partial exploration of MUL behavior feasible. (MUL runs sometimes involved a quarter of a million separate associative retrievals from a data base containing thousands of objects.)

Future development of TELOS will include the implementation of the designed abstract data type capabilities, coroutines and generators, and event

mechanisms. It is to be hoped that MUL research will benefit from the availability of full TELOS as much as it has benefited from the TELOS data base facility.

Appendix 2: MUL programs

MUL, in its ADVANCED versions, is implemented as a 2000 line TELOS program. The program is composed of five modules, performing environment simulation, RP introduction (by non-predictive pattern induction), inhibition introduction (by predictive pattern induction), perception, and control. These modules manipulate two data bases; a knowledge store containing RP representations and an episodic store containing kernel representations. This appendix discusses the data structures and algorithms of the five modules and two stores.

The data structure for RP representation, named the "rp_type", is declared as follows (numbers in parentheses reference later notes).

```
rp_type = Packed Record
   identifier : id_record;    (*used for RP output only*)
   num_components : 0..max_components;    (1)
   components : Array [1..num_components] Of
      Record    (2)
      rp : DB -> rp_type Indexed;
      is_previous : Boolean;
      failure_count : 0..1000;
      End;
```

```
inhibitor : DB -> rp_type Indexed;    (3)
area : area_descriptor    (4)
uses : Integer;
End;
```

(1) 'max_components' is 3. Componentless RPs have zero in their 'num_components' field.

(2) The 'components' array contains elements for each component (if any) of the RP. Each component has an 'rp' field containing a DBOP to the component RP, an 'is_previous' field set True if the component RP must be active in the previous time unit, and a 'failure_count' field recording a count of the number of times the component was incorrectly defaulted. 'failure_count' is used in conjunction with 'uses' to determine whether, and with what confidence, each component may be defaulted.

(3) The 'inhibitor' field contains a DBOP to the RP whose activation is to inhibit default of this RP. An inhibitor RP is created only if this RP has been improperly defaulted and must be inhibited. If no inhibitor has been created, the 'inhibitor' field contains the value Nil.

(4) The 'area' field contains two sub-fields, 'region' and 'level'. As discussed in Section 3.5, 'area' is used for the construction of related_rps, and for resource

allocation within MUL programs. To illustrate the representation of RPs within the knowledge store, consider this RP:

RP #10: <#5,0.0,previous> <#4,0.8>

RP #10 might be represented this way in the knowledge store:

```
identifier          : #10 (and some other information)
num_components      : 2
components[1]
   .rp              : DBOP to RP#5 in the knowledge
                      store
   .is_previous     : True
   .failure_count   : 9999, indicating that over 75% of
                      defaults of this component were
                      invalid.  No further defaults of
                      this component will be attempted.
components[2]
   .rp              : DBOP to RP#4 in the knowledge
                      store
   .is_previous     : False
   .failure_count   : 5
components[3] (unused. RP#10 has only two components.)
inhibitor           : DBOP to the RP inhibiting
                      RP#10, if any
area                : (values for region and level
                      might be 6 and 2 respectively)
```

uses : 25 (this RP has been activated 25 times)

The data structure for kernel representation, named the "kernel_type", is declared as follows:

```
kernel_type = Packed Record
    importance_factor : Real Sequencer;
    activation_time : Integer Indexed;
    activated_rp : DB -> rp_type Indexed;
    support : Array [1..max_components] Of DB ->
                                        kernel_type;
    defaulter : DB -> kernel_type;
    caused_conflict : Boolean;
    End;
```

Suppose K is a DBOP to some kernel in the episodic store. K -> .activated_rp contains a DBOP to the knowledge store representation of the RP whose activation is recorded by kernel K (the kernel referenced by K). K -> .activation_time specifies the time unit of episodic memory in which the kernel K exists. The support array contains DBOPs to the kernels recognized by kernel K (if any). K -> .defaulter is Nil unless kernel K appeared by default; in that case, K -> .defaulter contains a DBOP to the kernel recording the activation of the RP responsible for the default of kernel K. K -> .caused_conflict is set True if kernel K is ever marked as a culprit having caused a

conflict. The kernel_type structure is further discussed in Appendix 1.

The general operation of the MUL program is specified by the control module, called Main. The Main module is coded roughly as follows:

```
(* Main programs *)
initialize;   (* set up environment and establish MUL's
                 sensory and motor RPs *)
While True Do
   Begin
     ...(* code for statistics on MUL behavior *)...
   update_environment;   (* alter the environment to
        reflect any physical changes precipitated by
        MUL motor activity or by other agents in the
        environment.  Also, provide new sensory input *)
   perceive;   (* perform perception on the new sensory
        kernels, any kernels remaining unprocessed from
        last time *)
   learn_new_rps;   (* perform non-predictive pattern
        induction to discover new RPs involving
        important, recently-encountered kernels *)
   learn_new_inhibitions;   (* perform predictive pattern
        induction to introduce new RPs to inhibit recent
        improper defaults *)
```

```
End;    (* Main program loop *)
```

The initialize procedure contains code to initialize an environment simulation and to establish the sensory/motor RP interface to that environment. This interface includes not only the sensors themselves (retina cells, for example), but also the "subcortical" preprocessors of inputs (and outputs). Because of environmental simplicity, preprocessing is trivial but does involve establishing inhibitions to detect defaults that contradict experience or are otherwise undesired.

As an example, the following excerpt from the initialization for the bottle-filling environment creates RPs sensing valve status:

```
(* declarations of types and variables used in excerpt *)
Type valve_status = (open, closed);
Var rp_template :-> rp_type;    (*used to set up new RPs*)
   db_rp : DB -> rp_type;    (* used for auxilliary,
        temporary purposes *)
   valve_rp : Array [open..closed] Of DB -> rp_type;
        (* used to preserve valve status sensory RPs
        for environment interface *)
Begin...(* other initializations *)
New (rp_template);
(* create RP#1: componentless, sensing "valve open" *)
With rp_template -> Do
```

```
   Begin
   identifier.number := 1;   (* "valve open" is RP#1 *)
        (* code dealing with other fields of identifier
        is uniformly ignored throughout Appendix 2 *)
   num_components := 0;   (* componentless sensory RP *)
   inhibitor := Nil;   (* set later in initialize *)
   area.region := 1;   (* valve status RPs are assigned
        to region 1 *)
   area.level := 1;
   uses := 0;
   End;
valve_rp[open] := STORE(rp_template);
(* create RP#2: componentless, sensing "valve closed" *)
rp_template -> .identifier.number := 2;   (* "valve
        closed" is RP#2 *)
(* other fields of rp are same as for RP#1 *)
valve_rp[closed] := STORE(rp_template);
(* this code creates inhibitions to cause conflicts
        whenever the valve is asserted to be both open
        and closed concurrently *)
New(inhibitory_rp);
inhibitory_rp ->:= rp_template ->;   (* most fields of
        componentless RPs are the same regardless of the
        meaning of the RPs *)
inhibitory_rp -> .identifier.number := 3;   (* NOT "valve
        open" is RP#3 *)
```

```
(* establish new componentless RP as the inhibitor of
        RP#1 *)
CHANGE(valve_rp[open] -> .inhibitor,
   STORE(inhibitory_rp));
   (* CHANGE is used to alter DBOs.  Simple assignment is
        not allowed, to protect data base integrity. *)
(* create inhibitor for RP#2, as RP#4 *)
inhibitory_rp -> .identifier.number := 4;
CHANGE(valve_rp[closed] -> .inhibitor,
   STORE(inhibitory_rp));
(* finally, create RPs such that each valve state
        inhibits the other valve state *)
With rp_template -> Do
   Begin
   num_components := 2;
   With components[1] Do
      Begin rp := valve_rp[open]; is_previous := false;
      failure_count := 0;   (* should remain zero in
        later usage *)
      End;
   With components[2] Do
      Begin rp := valve_rp[closed] -> .inhibitor;
      is_previous := false;  failure_count := 0;
      End;
   identifier.number := 5;
   End;
```

```
db_rp := STORE(rp_template);   (* creates RP#5:
        <#1,1.0>  <#4,1.0>  the default coefficients of
        1.0 will appear with use *)
With rp_template -> Do
   Begin
   component[1].rp := valve_rp[closed];
   component[2].rp := valve_rp[open] -> .inhibitor;
   identifier.number := 6;
   End;
db_rp := STORE(rp_template);   (* creates RP#6:
        <#2,1.0>  <#3,1.0> *)
```

The effects of the code segment just presented is to create the following RP mass for sensing valve status:

RP #1: componentless, sensing "valve open"

RP #2: componentless, sensing "valve closed"

RP #3: componentless, inhibiting #1

RP #4: componentless, inhibiting #2

RP #5: <#1,1.0> <#4,1.0>

RP #6: <#2,1.0> <#3,1.0>

The update environment procedure maintains, inaccessible to the rest of the program, information on the current state of the environment. It detects any motor activity MUL has indicated, tries to perform that activity (or flail if no activity is specified), and then sends

appropriate sensory information to MUL, to reflect the new environment state.

Update_environment detects desired motor activity by searching episodic memory for activations of motor RPs for the time unit just experienced. (Motor RPs are activated only by default, or by flailing.) The default of a motor RP is interpreted as a request that the associated motor activity be performed. If, for example, the "close_valve" RP was active at the "current" time, update_environment would change its environment information to indicate a closed-valve state; if the valve could not be closed (because it was already closed, perhaps) the update_environment indicates a problem by (depending on the version) either activating NOT ("close valve") or activating the "pain" RP, or both.

If no motor RP is activated for the time unit being experienced, then update_environment causes a random flail. Usually, a "rest" event is chosen so that MUL does nothing.

Once the environment simulation has been altered to reflect movement on the part of MUL and on the part of other agents (the ongoing processes, like the bottle filling, or the actions of the teacher), update_environment sends new sensory information by activating sensory RPs. For example, it knows the status of the valve, and has access to the two RPs indicating the valve state : if the valve is open, it activates (if necessary) valve_rp[open], otherwise it

activates valve_rp[closed]. The RP may already be active by default; if so, the importance of the kernel is adjusted reflecting the verification of the default.

The perceive procedure operates on the buffers STM and RFI to activate any RPs that can become active, if possible within certain resource limitations. It selects the most important item in STM or RFI. If the item is a kernel reference from STM, perceive tries to activate unactivated RPs having the kernel's activated_rp as a component. If the most important item is a kernel from RFI, then perceive tries to activate (or find activations for) each component of that kernel's activated_rp.

The following code processes a kernel reference from STM; given these declarations:

```
Var num_rps, i, n, unactivated : Integer
    new_kernel, activation, kernel : DB -> kernel_type;
    has_as_component :-> Parameters Of
        has_as_component_rtn;
      (* this Match_Routine is described later *)
    rp_array : Array [1..max_rps] Of DB -> rp_type;
    kernel_array : Array [1..1] Of DB -> kernel_type;
...(* code to remove a kernel reference from STM,
        assigning it to kernel *)
(* find RPs referencing kernel -> .activated_rp as a
        component *)
```

```
has_as_component -> .rp := kernel -> .activated_rp;
num_rps := max_rps;   (* constant 50.  No more than 50
        rps are activated by any single kernel. *)
FIND(has_as_component, rp_array, num_rps);
For i := 1 To num_rps Do   (* for each RP having
        kernel -> .activated_rp as a component Do...*)
   Begin this_rp := rp_array[i];
   (* determine which component corresponds to
        kernel -> .activated_rp *)
   this_component :=1;
   While this_rp -> .components[this_component].rp <>
        kernel -> .activated_rp
      Do this_component + := 1;
   (* calculate 'time' in which this_rp
      might become active *)
   time := kernel -> .activation_time
        + Ord(this_rp -> .components[this_component].
        is_previous);
   (* see if this_rp is already active at time
      Function "is_active" uses "find_activation_rtn", a
      Match_Routine discussed in Appendix 1, to return
      True if "this_rp" is active at "time", by finding
      a kernel in episodic memory.  "activation" returns
      a DBOP to the kernel. *)
   If is_active(this_rp, time, activation) Then
      Begin
```

```
   If activation -> .support[this component] = Nil
     Then
     CHANGE(activation -> .support[this_component],
     kernel);
   Go To next_rp;   (* exit *)
   End;
(* activate this_rp, if possible *)
unactivated := 0;   (* will be set to the first
     unactivated component *)
n := 1;
(* find all unactivated components.  If more than one
     are unactivated, then rp cannot be activated. *)
While n <= this_rp -> .num_components Do
   If n = this_component Then n := n + 1
   Else With this_rp -> .components[n] Do
     If is_active(rp, time-Ord(is_previous),
     activation) Then
       Begin new_kernel -> .support[n] := activation;
       n := n + 1 End;
     Else If unactivated <> 0 Then Go To next_rp
     Else unactivated := n;
(* "new_kernel" will represent the activation of
     this_rp *)
new_kernel -> .activation_time := time;
new_kernel -> .support[this_component] := kernel;
...(* code calculating new_kernel ->
```

```
            .importance_factor, as the importance of
            the least important supporting kernel *)
        If unactivated <> 0 Then    (* try to default
            component[unactivated] *)
          If Not default(This_rp, unactivated, new_kernel)
             then
            Go To next_rp;    (* defaulting may not be
            possible, see below *)
        (* activate this_rp by creating new_kernel *)
        With new_kernel -> Do
          Begin activated_rp := this_rp;  defaulter := NIL;
          caused_conflict := False;
          End;
        (* "enter_kernel" does a STORE(new_kernel), but
            also inserts the new kernel into STM, and detects
            conflicts if this_rp is inhibited *)
        enter_kernel(new_kernel, new_db_kernel);
        (* if defaulting was necessary, mark the default *)
        If unactivated <> 0 Then
          CHANGE(new_kernel -> .support[unactivated] ->
            .defaulter, new_db_kernel;
    next_rp;
        End;    (* For i := 1 To num_rps...*)
```

The above is a somewhat simplified but basically accurate reproduction of the code executed for each kernel

referenced in the STM buffer. The "default" function is given below, to clarify that activity:

```
Function default (defaulting_rp : DB -> rp_type;
                  component_to_default : 1..max_components;
                  defaulter :-> kernel_type) : Boolean;
(* "default" returns True only if defaulting was possible *)
Const lessened_faith = 0.9;
      threshold = 0.4;   (* defaults must exceed this in
                            importance *)
Var result : DB -> kernel_type;
      default_kernel : -> kernel_type;
      i : Integer;
Begin default := False;   (* prepare for possible failure *)
With defaulting_rp -> .components[component_to_default] Do
   Begin New(default_kernel);
   (* calculate the importance of the default *)
   default_kernel -> .importance_factor :=
      defaulted -> .importance_factor * lessened_faith
      * (1 - 2 * failure_count/(1 + defaulting_rp->.uses));
   If default_kernel -> .importance_factor > threshold Then
      Begin
      default_kernel -> .activation_time := defaulter ->
         .activation_time - Ord(is_previous);
      If Not is_inhibited(rp, default_kernel ->
         .activation_time) Then
```

```
        With default_kernel -> Do
           Begin  (*default is possible*)  default := True;
           activated_rp := rp;
           (* default has no support yet *)
           For i := 1 to rp -> num_components Do
              support[i] := NIL;
           defaulter := Nil;   (* this field will be set
              later *)
           caused_conflict := False;
           enter_kernel(default_kernel, result);
           ...(* code to add new kernel into the RFI
              buffer *)
           End;   (* If Not is_inhibited *)
      End;   (* If default_kernel ->.importance_factor... *)
   End;   (* With defaulting_rp...*)
End;   (* Function default *)
```

The "is_inhibited" function examines the episodic store to see if the inhibitor (if any) is active at the time of interest.

```
Function is_inhibited (rp : DB -> rp_type; time : Integer)
   : Boolean;
   (* "is_inhibited" returns True only if rp -> .inhibitor
      is active at "time" *)
Var ignore : DB -> kernel_type;
```

```
   (* a DBOP to the activation
   is not useful *)
Begin
If rp -> .inhibitor = Nil Then is_inhibited := False
Else is_inhibited := is_active(rp -> .inhibitor, time,
   ignore);
End;   (* Function is_inhibited...*)
```

The Match_Routine "has_as_components" searches the knowledge store for RPs whose components name the specified "rp".

```
Match_Routine has_as_components(rp : DB -> rp_type)
      Matching candidate : DB -> rp_type;
   Begin
   has_as_components := (rp <> candidate -> .inhibitor);
      (* since the candidate is Required to reference "rp"
         somehow, we need only insure that the reference
         is as a component (the inhibitor field is also
         indexed) *)
   End;
Requirements
   Begin Require(rp)
   End;  (* Match_Routine has_as_components *)
```

The perceive procedure is also responsible for processing items from the RFI buffer. Suppose a default

kernel referenced by "K" is extracted from the RFI buffer. perceive will first determine if the components of K -> .activated_rp are already active; if so, K -> .support is filled in. Failing that, perceive will activate the unactivated components of K -> .activated_rp, so long as K -> .importance_factor exceeds a threshold (0.4). The importance_factors of these new defaults are set to 0.75 of K -> .importance_factor to discover unsupported defaulting.

The learn new RPs procedure is responsible for introducing new RPs by non-predictive pattern induction. learn_new_RPs selects several (up to 8) kernels to learn new RPs for, on the basis of highest values for the formula [(max_rps - num_rps) * kernel -> .importance_factor] calculated for each kernel processed from the STM buffer. Pattern induction is performed on each of these kernels, following the algorithm specified in Section 3.6. Detailed analysis of the code in learn_new_rps is not presented, but is provided for learn_new_inhibitions, discussed next.

The learn new inhibitions procedure attempts to construct new RPs to default the inhibitor of an RP under conditions that seem to have involved defaults of that RP in conflict. An overview of the mechanism used by learn_new_inhibitions is provided in Section 3.7.

Conflicts are detected and culprits determined in the enter kernel procedure. When a new kernel is created, enter_kernel determines if that kernel's activated_rp is inhibited; if so, a conflict is registered, and possible culprit kernels are marked (by having their caused_conflict fields set True). Culprits are defaults that directly or indirectly support either the kernel or the defaulter of its inhibitor (including the inhibitor activation itself). Action is not taken to inhibit future defaults of the culprits until the time unit containing the conflict has been processed. In this way, time is allowed for improper defaults to cause further conflicts; the defaults that cause the greatest number of conflicts are assumed to be improper and are provided to learn_new_inhibitions to be inhibited.

When learn_new_inhibitions is called, it is given a reference (in "bad_default") to a default that is assumed to be improper. It then examines episodic memory to discover if an RP X (or a small group of RPs that could be components of a new RP X) is consistently activated before improper defaults of bad_default -> .activated_rp but is rarely activated before proper activations of bad_default -> .activated_rp. Activations of RP X will be made to cause inhibition of bad_default -> .activated_rp, by introducing a new RP to cause activations of X to default bad_default -> .activated_rp -> .inhibitor.

The search of episodic memory (for X) is accomplished by associative retrievals controlled by Match_Routine build_inhibition_rtn. This routine will be presented following the declaration of the variables used, and description of the initialization preceding the call to build_inhibition_rtn.

```
Const max_examples = 10;
      max_rps = 8;
Var
   num_improper, num_proper, num_rps, num_compnts,
        best_cover : Integer;
   proper_times, improper_times : Array[1..max_examples]
        Of Integer;
   dbo_num : Array[1..max_examples] Of Integer;
   try : Array[1..max_components] Of Integer;
   rp_set : Array[1..max_rps] Of
      Record
      rp : DB -> rp_type;
      is_previous : Boolean;
      with_proper, with_improper : Packed Array
        [1..max_examples] Of Boolean;
      End;
```

learn_new_inhibitions begins by initializing the "proper" and "improper" arrays with samples of times when bad_default -> .activated_rp was properly activated or improperly defaulted respectively. This operation is performed by

associative retrieval with the Match_Routine proper_improper_times described below. When called, the parameter "rp" is assigned a reference to bad_default -> .activated_rp.

```
Match_Routine proper_improper_times(rp: DB -> rp_type)
      Matching K : DB -> kernel_type;
   Begin
   If K -> .caused_conflict Then
      Begin num_improper := num_improper+1;
      improper[num_improper] := K -> .activation_time;
      (* DBO_Number(<DBO>) is a TELOS routine that returns
         a count of the number of DBOs that were placed in
         the data base before <DBO>.  This identifier is
         used here as a cheap mechanism to insure that the
         RPs to cause the inhibition will be active before
         the improper default is made. *)
      dbo_num[num_improper] := DBO_Number(K);
      End;
   Else
      Begin num_proper := num_proper+1;
      proper[num_proper] := K -> .activation_time;
      End;
      (* sufficient = 4, see below *)
      proper_improper_times := (num_proper > sufficient) And
           (num_improper > sufficient);
```

```
   End;
Requirements
   Begin Require(rp) End;
```

The above Match_Routine illustrates an important use of associative retrieval. Note that the Match_Routine performs its work outside the accept/reject candidate formalism of associative retrieval. The Match_Routine is called such that accepting one candidate satisfies the retrieval (that is, the third parameter to FIND is 1). In this way, the expensive associative retrieval may be terminated when arbitrarily complex conditions are met, in this case when sufficient samples of proper and improper activations have been found.

learn_new_inhibitions initializes several other variables, and then calls for associative retrieval with the build_inhibitions_rtn Match_Routine. This Match_Routine is called twice, first with parameter time set to bad_default -> .activation_time, and then with time set to bad_default -> .activation_time -1. From each time, the Match_Routine attempts to discover RPs or combinations of RPs activated just prior to the times in "improper", but that were not activated immediately prior to the times in "proper".

```
Match_Routine build_inhibitions_rtn (time : Integer)
      Matching K : DB -> kernel_type;
Begin time_difference := bad_default ->
```

```
            .activation_time - time;
(*build_inhibition_rtn terminates associative retrieval
   only if it has found material to forge a new inhibition*)
build_inhibition_rtn := False;
(* the next spot of rp_set is used for imformation about
   K -> .activated_rp *)
With rp_set[num_rps+1] Do
   Begin rp := K -> .activated_rp; is_previous :=
            (time_difference <> 0);
   count1 := 0;
   (*is K -> .activated_rp active prior to "proper" times?*)
   For i := 1 To num_proper Do
      If is_active (K -> .activated_rp, proper[i] -
            time_difference, kernel)
         Then Begin count1 := count1+1; with_proper[i] :=
            True   End
      Else with_proper[i] := False;
   If count1 = num_proper Then Go To exit;   (* K ->
            .activated_rp is useless *)
   count2 := 0;
   (*is K->.activated_rp active prior to "improper" times?*)
   For i := 1 To num_improper Do Begin
      with_improper[i] := is_active (K -> .activated_rp,
            improper[i] - time_difference, kernel);
      If with_improper[i] Then
         If DBO_number(kernel) > dbo_num[i] Then
```

```
        with_improper[i] := False
      Else count2 := count2+1;
If count2 = 0 Then Go To exit;   (* K -> .activated_rp
        is useless *)
cover := (num_proper - count2) + count1;   (* measure
        of the usefulness of K -> .activated_rp *)
If cover > best_cover Then
   Begin   (* inhibition possible on activation of this
        RP alone *)
   try[1] := num_rps+1; num_compnts := 1;
   best_cover := cover;
   (* 'try' contains information on the best RP(s) yet
        found to activate the new inhibition *)
   If count2 = 0 Then   (* => useless to combine this RP
        in a group *)
      Begin build_inhibition_rtn := (count1=num_proper);
        (* "perfect"? *)
      Go To exit; End;
   End;
(* inhibition might be based on a new RP formed with
   K -> .activated_rp as one component, and the other
   component chosen from rp_set to decrease the number of
   occasions when activations precede "proper" times *)
For i := 1 To num_rps Do
   Begin tcover := cover;
   For j := 1 To num_proper Do
```

```
        If with_proper[j] Then
          If Not rp_set[i].with_proper[j] Then tcover t:=1;
      count1 := 0;
      For j := 1 To num_improper Do
        If with_improper[j] Then
          If Not rp_set[i].with_improper[j]
          Then tcover -:= 1
          Else count1 := count1+1;
      If (tcover > best_cover) And (count1 > 0) Then
        Begin   (* inhibition might be based on a new RP,
          with components <K -> .activated_rp,- >
          <rp_set[i].rp,- > *)
        try[1] := num_rps+1; try[2] := i;
        num_compnts := 2;
        best_cover := tcover;
        If tcover = (num_proper + num_improper) Then
          Begin build_inhibition_rtn := True; Go To exit;
          End;
        End;
    End;   (* For i...*)
  num_rps := num_rps+1;   (* add K -> .activated_rp
          to rp_set *)
  build_inhibition_rtn := (num_rps = max_rps);   (* quit if
          rp_set filled *)
  End;
exit :
```

```
   End;   (* Body of build_inhibition_rtns *)
Requirements
   Begin Require(time)   (* only RPs active at "time"
           are considered *)
   End;
```

When learn_new_inhibitions completes the two associative retrievals with build_inhibition_rtn, the "try" array should contain information for an adequate inhibition. The new inhibition is incorporated in this way:

```
   If num_compnts > 0 Then   (* "try" has meaningful
           information in it *)
      Begin
      (* if try specifies a new rp, create it *)
      If num_compnts > 1 Then With rp_template -> Do
         Begin ...(* code to set 'identifier' and 'area' *)
         inhibitor_previous := (rp_set [try[1]].is_previous
           And rp_set [try[2]].is_previous);
         num_components := 2;
         components[1].rp := rp_set [try[1]].rp;
         components[1].is_previous := (rp_set [try[1]]
           .is_previous And Not inhibitor_previous);
         components[2].rp := rp_set [try[2]].rp;
         components[2].is_previous := (rp_set [try[2]]
           .is_previous And Not inhibitor_previous);
         (* 'uses', "failure_counts" set in "add_new_rp",
```

```
      a routine that stores the new RP after putting
      it in canonical form and verifying that it is not
      a duplicate *)
   X := add_new_rp(rp_template);
   End
Else With rp_set[try[1]] Do
   Begin X := rp; inhibitor_previous := is_previous;
   End;
(* create inhibitory RP for bad_default ->
      .activated_rp, if necessary *)
If bad_default -> .activated_rp -> .inhibitor = Nil
   Then
   Begin ...(* code to set up componentless RP in
      rp_template *)
   CHANGE(bad_default -> .activated_rp -> .inhibitor,
      STORE(rp_template));
   End;
(* create RP to inhibit bad_default -> .activated_rp
      when X is active *)
With rp_template -> Do
   Begin ...(* code establishing "identifier" field
      values *)
   num_components := 2;
   component[1].rp := X;
   component[1].is_previous := inhibitor_previous;
   component[1].failure_count := 9999;   (* this
```

```
        component is not to be defaulted *)
      component[2].rp := bad_default -> .activated_rp ->
        .inhibitor;
      component[2].is_previous := False;
      component[2].failure_count := 0;
      inhibitor := NIL;
      uses := 0;
      End;
  STORE(rp_template);
  End;   (* If num_compnts...*)
```

The actual MUL programs contain code that has not been reproduced in these excerpts. The additional code performs such tasks as statistics gathering and exceptional-case handling, tasks that are not important to the basic ideas behind the programs.

The following is a list of the first 15 RPs produced by the ADVANCED MUL version for the bottle-filling environment. In 70 environment steps, a total of 243 RPs were created, enabling the program to anticipate most events in its environment. It did not, however, learn to correctly default kernels for motor RPs in order to gain control of its valve.

The episodic experience of MUL is provided for the first 20 time units. For convenience, sensory/motor RPs are described by function rather than by number. (note: the default coefficients shown are those initially given the components.)

Time Unit 1. experience: "no bottle", "valve closed"

Time Unit 2. experience: "no bottle", "valve closed"

Time Unit 3. experience: "no bottle", "valve closed"

Time Unit 4. experience: "no bottle", "valve closed"

Comment: MUL now has enough experience to learn new RPs.

RP #1000: <"no bottle",1.0,previous>
<no bottle",1.0>

RP #1001: <"valve closed",1.0,previous>
<"valve closed",1.0>

Time Unit 5. experience: "empty", "valve closed"

Comment: the teacher has placed a bottle on the platform.

RP #1002: <#1001,1.0,previous> <#1001,1.0>

Time Unit 6. experience: "empty", "valve closed"

Time Unit 7. experience: "empty", "valve closed"

Time Unit 8. experience: "empty", "valve closed"

RP #1003: <"empty",0.75,previous> <"empty",1.0>

Time Unit 9. experience: "empty", "valve closed"

RP #1004: <#1003,0.80,previous> <#1003,1.0>

Time Unit 10. experience: "open valve", "empty", "valve closed"

Comment: the teacher, tired of waiting for MUL to open the valve, is forcing the valve open.

Time Unit 11. experience: "good", "valve opened", "one third full"

Comment: the teacher tries to reinforce the valve opening by "praise" (that is, stimulating MUL's pleasure RP "good"). With the valve now open, the bottle has begun filling.

Time Unit 12. experience: "two thirds full", "valve opened"

Time Unit 13. experience: "bottle full", "valve opened"

Time Unit 14. experience: "spilling", "bad!", "valve opened"

Comment: the teacher, upset with the spillage, is punishing MUL by stimulating the pain RP.

RP #1005: <"valve opened",0.75,previous> <"valve opened",1.0>

Time Unit 15. experience: "spilling", "valve opened"

RP #1006: <#1005,0.75,previous> <#1005,1.0>

Time Unit 16. experience: "spilling", "valve opened"

Time Unit 17. experience: "spilling", "valve opened"

RP #1007: <"spilling",0.75,previous>
<"spilling",1.0>

Time Unit 18. experience: "spilling", "valve opened"

RP #1008: <#1007,0.75,previous> <#1007,1.0>

Time Unit 19. experience: "close valve", "spilling", "valve opened"

Comment: the teacher, tired of spillage, is forcing MUL's valve to close.

Time Unit 20. experience: "good", "no bottle", "valve closed"

Comment: the whole cycle now begins again. Accordingly, experience will no longer be listed.

RP #1009: <"valve closed",0.0,previous>
<"valve opened",0.0>

RP #1010: <"valve opened",0.0,previous>
<"valve closed",0.0>

RP #1011: <"one third full",1.0,previous>
<"two thirds full",1.0>

RP #1012: <#1006,0.81,previous> <#1006,1.0>

RP #1013: <"spilling",0.0,previous>
<"no bottle",0.0>

RP #1014: <#1000,0.0,previous> <#1000,0.0>

RP #1015: <#1002,0.0,previous> <#1002,0.0>

Meaningful inhibitions were not learned in the bottle-filling environment. No inhibitions could meet the timing requirement (that the inhibitory RP must be defaulted into the data base in advance of the appearance of the improper default in the data base). Section 6.3 discusses several mechanisms for making this requirement easier to meet.

References

[Altman 67] Altman, J. "Postnatal Growth and Differentiation of the Mammalian Brain, with Implications for a Morphological Theory of Memory", in [Quarton 67].

[Becker 70] Becker, J.D., An Information Processing Model of Intermediate Level Cognition, Unpbl. Ph.D. Diss., Stanford, 1970.

[Becker 69] Becker, J.D., "The Modeling of Simple Analogic and Inductive Processes in a Semantic Memory System", Proc. First International Joint Conf. on AI, 1969.

[Becker 73] Becker, J. "A Model for the Encoding of Experiential Information", in Computer Models of Thought and Language, R. Shank and K. Colby (Eds.), Freeman, 1973.

[Brazier and Petsche 78] Brazier, M. and H. Petsche (Eds.). Architectonics of the Cerebral Cortex, Raven Press, New York, 1978.

[Buchanan 72] Buchanan, B.G., Feigenbaum, E.A., and Sridharan, N.S. "Heuristic Theory Formation: Data Interpretation and Rule Formation", Machine Intelligence 7, Edinburgh: Edinburgh University Press, 1972.

[Cohen and Nagel 34] Cohen,M. and E. Nagel. An Introduction to Logic and Scientific Method, Harcourt Brace, New York, 1934.

[Coltheart 76] Coltheart, M. "Iconic Storage and Visual Masking", in [Hamilton and Vernon 76].

[Cotrell 77] Cotrell, G. and P. Underwood. (Eds.) Synapses, Academic Press, New York, 1977.

[Crain 76] Crain, S. "Development of Specific Sensory-Evoked Synaptic Networks in CNS Tissue Cultures", in Electrobiology of Nerve, Synapse, and Muscle, J. Ruebin, D. Purpura, M. Bennett, and E. Kandel (Eds.), Raven Press, New York, 1976.

[Creutzfeldt 78] Creutzfeldt,O. "The Neocortical Link: Thoughts on the Generality of Structure and Function of the Neocortex", in [Brazier and Petsche 78].

[Davis 77] Davis, R. "Interactive Transfer of Expertise: Acquisition of New Inference Rules", in [IJCAI 77].

[Davis and King 75] Davis, R., and J. King, An Overview of Production Systems, Stanford University Memo AIM-271, October 1975.

[Davison 77] Davison, A. (Ed.) Biochemical Correlates of Brain Structure and Function, Academic Press, London, 1977.

[Feigenbaum 77] Feigenbaum, E. The Art Of Artificial Intelligence: I. Themes and Case Studies of Knowledge Engineering, Computer Science Dept., Stanford University, Report STAN-CS-77-621, August 1977.

[Goldstein and Grimson 77] Goldstein, I. and E. Grimson. "Annotated Production Systems. A Model for Skill Acquisition", in [IJCAI 77].

[Granit 77] Granit, R. The Purposive Brain, MIT Press, Cambridge MA, 1977.

[Hamilton and Vernon 76] Hamilton, V. and M. Vernon. (Eds.) The Development of Cognitive Processes, Academic Press, London, 1976.

[Hayes-Roth 77] Hayes-Roth, F. and J. McDermott. "Knowledge Acquisition from Structural Descriptions", in [IJCAI 77].

[Hayes-Roth 78] Hayes-Roth, F. and J. McDermott. "An Interference Matching Technique for Inducing Abstractions", Comm. of the ACM, Vol 21, #5, May 1978.

[Hebb 49] Hebb, D. The Organization of Behavior: A Neuropsychological Theory, Wiley, New York, 1949.

[Honda 77] Honda, M., R. LeBlanc, L. Travis and S. Zeigler. An Improved Data Context Mechanism, MACC Technical Report #47, University Of Wisconsin - Madison, October 1977.

[Hyden 67] Hyden, H. "RNA in Brain Cells", in [Quarton 67].

[IJCAI 77] Advance Papers of the Fifth International Joint Conference on Artificial Intelligence, Boston, Mass. 1977.

[Kuipers 75] Kuipers, B. "A Frame for Frames: Representing Knowledge for Recognition", in Representation and Understanding: Studies in Cognitive Science, D. Bobrow and A. Collins (Eds.), Academic Press, New York, 1975.

[Langley 78a] Langley, P. BACON: A Production System that Discovers Empirical Laws, Carnegie-Mellon Department of Psychology, CIP Working Paper #360, January 1978.

[Langley 78b] Langley, P. BACON.1: A General Discovery System, Carnegie-Mellon Department of Psychology, unpbl. report, January 1978.

[LeBlanc 77] LeBlanc, Richard. "The Design and Rationale for TELOS, a PASCAL-based AI Language", PhD Diss. UW-Madison Academic Computing Center Tech Rpt 50, December 1977.

[LeBlanc 78] LeBlanc, R. "Extensions to PASCAL for Separate Compilation", Sigplan Notices, Vol. 13, #9, September 1978.

[Lenat 76] Lenat, D. AM: An AI Approach to Discovery in Mathematics as Heuristic Search, Memo HPP-76-8, Stanford University Computer Science Dept., 1976.

[MacGregor 77] MacGregor, R. Neural Modeling: Electrical Signal Processing in the Nervous System, Plenum Press, New York, 1977.

[Michalski 77] Larson, J. and R. Michalski. "Inductive Inference of VL Decision Rules", Sigart Newsletter #63, June 1977.

[Milner 57] Milner, P. "The Cell Assembly: Mark II", Psychological Review 64.

[Minsky 75] Minsky, M. "A Framework for Representing Knowledge", in [Winston 75].

[Nathanson 77] Nathanson, J. and P. Greengard. "'Second Messengers' in the Brain", Scientific American, Vol. 237, #2, August 1977.

[Newell and Simon 72] Newell, A. and H. Simon. Human Problem Solving, Prentice-Hall, Englewood Cliffs NJ, 1972.

[Piaget 54] Piaget, J. The Construction of Reality in the Child, New York : Basic Books, 1954.

[Piaget and Inhelder 69] Piaget, J., and B. Inhelder. The Psychology of the Child, New York : Basic Books, 1969.

[Quarton 67] Quarton,G., T. Melnechuk, and F. Schmitt (Eds.) The Neurosciences, Rockefeller University Press, New York, 1967.

[Reese 76] Reese, H. and S. Porges. "Development of Learning Processes", in [Hamilton and Vernon 76].

[Rose 77] Rose, S. and J. Haywood. "Experience, Learning and Brain Metabolism", in [Davison 77].

[Rumelhart and Norman 73] Rumelhart, D.E. and Norman, D.A., Active Semantic Networks as a Model of Human Memory", Proc. Third International Joint Conf. on AI, 1973.

[Rumelhart and Norman 76] Rumelhart, D., and Norman, D. Accretion, Tuning, and Restructuring: Three Modes of Learning, Tech Rep 63, Center for Human Information Processing, University of California, San Diego, Aug. 1976.

[Sagan 78] Sagan, C. The Dragons of Eden, Speculations on the Evolution of Human Intelligence, Random House, New York, 1977.

[Scott 77] Scott, A. Neurophysics, John Wiley and Sons, New York, 1977.

[Shank 77] Shank, R. "How to Learn / What to Learn", in [IJCAI 77].

[Travis 77] Travis, L. E., R. LeBlanc, M. Honda and S. Zeigler. "Design Rationale for TELOS, a PASCAL-based AI Language", Proc. of the Symposium on AI and Programming Languages, New York : ACM, 1977.

[Uhr 74] Uhr, L. "A Wholistic Cognitive System (SEER-2) for Integerated Perception, Action and Thought", University of Wisconsin Computer Sciences Dept. Technical Report #234, December 1974.

[Uhr 75] Uhr, L. "'Recognition Cones' that Perceive and Describe Scenes that Move and Change Over Time", University of Wisconsin Computer Sciences Dept. Technical Report #235, January 1975.

[Uhr 77] Uhr, L. "A Parallel-Serial Recognition Cone System For Perception: Some Test Results", UW-Madison Computer Science Tech Rpt 292, March 1977.

[Uhr 78] Uhr, L. "Parallel-Serial Production Systems with Many Working Memories", University of Wisconsin Computer Sciences Dept. Technical Report #313, January 1978.

[Uhr and Jordan 69] Uhr, L., and Jordan, S. "The Learning of Parameters for Generating Compound Characterizers for Pattern Recognition", Proc. First Joint Conference on AI, 1969.

[Uhr and Vossler 61] Uhr., L. and Vossler, C. "A Pattern Recognition Program that Generates, Evaluates, and Adjusts Its Own Operators", Proc. West. Joint Computing Conf., 1961.

[Waterman 75] Waterman, D. "Adaptive Production Systems", Proc. of the Fourth International Joint Conference on AI, 1975.

[Whitehead 77] Whitehead, B. A Neural Model of Human Pattern Recognition, PhD Dissertation, University of Michigan Dept. of Computer and Communication Sciences, 1977.

[Winston 75] Winston, Patrick. The Psychology of Computer Vision, New York : McGraw-Hill, 1975.

[Wirth 71] Jensen, K. and N. Wirth. PASCAL User Manual and Report, 2nd Edition, Springer-Verlag, New York, 1975.

[Witte 73] Witte, L. A Computer System to Model the Cerebral Cortex and Other Brain Centers, Unpbl. PhD Dissertation, University Of Wisconsin Computer Sciences Dept., 1973.

[Zeigler 78] Zeigler, S. Learning by Pattern Induction, University of Wisconsin Computer Sciences Dept. Technical Report #319, April, 1978.