Contingency and Latency in Associative Learning: Computational, Algorithmic and Implementation Analyses

Richard H. Granger
Michal T. Young
Jeffrey C. Schlimmer

Irvine Computational Intelligence Project
Computer Science Dept.
Cognitive Sciences Program
Center for the Neurobiology of Learning and Memory

Technical Report #85-10
1985

This research was supported in part by the Office of Naval Research under grant N00014-84-K0391, the National Science Foundation under grant IST-81-20685, and by the Naval Ocean Systems Center under contract N66001-83-C-0255.

Contingency and Latency in Associative Learning: Computational, Algorithmic and Implementation Analyses

Richard H. Granger, Jr.
Jeffrey C. Schlimmer
Michal T. Young

Irvine Computational Intelligence Project
Computer Science Dept.
Cognitive Sciences Program
and
Center for the Neurobiology of Learning and Memory
University of California
Irvine, California 92717

Abstract

Contingency (the learned relative salience of environmental features) and latency (the learned timing of responses to stimuli) are central phenomena of learning and memory. This paper provides a computational analysis of, and algorithms for, a set of empirical data on contingency and latency in classical and instrumental conditioning. These analyses are presented within the framework of an information-processing architecture that describes a set of modules which operate in parallel and asynchronously to store, retrieve and modify experiential information. The architecture (called 'CEL', for 'Components of Experiential Learning') provides a way of making explicit the interactions among a number of otherwise separate algorithms for related phenomena. The modules comprising the architecture each emerge from the operation of an indexed network memory. The algorithms presented are also implemented in working computer programs that interact with a simulated environment to produce contingent associative learning and differential response latencies that correspond to the relevant behavioral data. The model makes a number of specific empirical predictions that can be experimentally tested.

This research was supported in part by the Office of Naval Research under grant N00014-84-K-0391, the National Science Foundation under grant IST-81-20685, and by the Naval Ocean Systems Center under contract N66001-83-C-0255.

Contents

1 Introduction

A rat in a laboratory cage hears a tone. It also hears the air conditioning system start, and sees a lab assistant taking notes. Shortly afterwards, it feels an unpleasant electric shock. What does the rat learn? Extensive empirical work in experimental psychology tells us that the rat will associate with shock exactly those cues which indicate an increased probability of shock, and that the timing or latency of its response will depend on the extent to which the response can prevent the shock.

This paper provides a set of theories at the computational, algorithmic and implementation levels that provide partial accounts of phenomena of contingency and latency in classical and instrumental conditioning. Our approach focuses first on analyses of the necessary computations that must be performed by an animal, based on empirical data, independent of *how* they may be performed. The resulting computational theory is the basis for the development of detailed algorithms that conform to the identified computational-level constraints. These theories must also be grounded in the operation of an implementation-level substrate.

All of our algorithmic results are stated in the language of a unifying framework or 'architecture' we have termed 'CEL' (Components of Experiential Learning). This architecture provides explicit algorithm-level representations of crucial processes that emerge from the operation of a low-level parallel, distributed memory; making these explicit serves to tie together a set of otherwise independent algorithms for the computation of various phenomena in the areas of contingency and latency.

The rest of this paper is organized as follows: Section 2 describes the theoretical framework of our approach, and outlines the need for and details of our algorithmic architecture, which provides for the identification of computational and algorithmic interactions that may, in turn, arise from implementation-level considerations. Sections 3

and 4 go into detail on our analyses (at all three levels) of empirical data on contingency and latency in classical and instrumental conditioning. Section 5 gives a discussion of some related work, and section 6 points out some potentially interesting conclusions and generalizations that can be drawn from our work.

2 The CEL framework for information storage and retrieval

2.1 The Computational, Algorithmic and Implementation Levels

Animals can be said to perform identifiable computations, that is, to transform inputs to outputs in a principled way. For instance, in order to learn which of many possible cues (tone, light, air puff) reliably predict the occurence of an aversive stimulus (shock), a rat in a classical conditioning situation must 'compute' the relative salience or predictiveness of each of the possible cues with respect to the occurence shock.

We assume three levels of analysis that are necessary for a complete theory: the computational, algorithmic and implementation levels [see e.g., Marr 1982]. At the computational level, we may speak simply of the computations that must somehow be performed (such as the contingency or salience relationship between cues and results), without reference to *how* those computations may be carried out. The algorithmic level constitutes the level of mathematical function that performs the necessary computations. Finally, these algorithms may be instantiated in hardware (neural or computer) at the 'implementation' level.

These three levels are not wholly independent. In particular, the algorithm level must conform to the constraints provided by each of the other two levels: it must compute

all and only those things that have been identified as actually occuring in humans and animals (at the computational level), and the proposed algorithms must be able to be carried out somehow given the tools and limitations provided in the substrate (at the implementation level).

Bridging the gaps among these levels is a difficult problem, but one that must be addressed. For instance, when there are multiple constraints to be satisfied at the computational level, it is usually possible to derive isolated algorithms, one for each constraint. However, a complete analysis would also be able to represent the *interactions* and relations among otherwise disparate algorithms. In order to find and represent such relationships among multiple constraints, the algorithm level must include a unifying *architecture* or framework that effectively provides a 'process language' within which the algorithms for each of the multiple constraints must be stated.

2.2 The CEL architecture for learning and memory

At the computational level, a good deal of work on animal learning and memory (e.g., [Kamin 1968, 1969], [Mackintosh 1983], [Rescorla 1980]) has identified necessary constraints on behavior, i.e., input-output specifications that must be accounted for by any algorithms intended as theories of learning and memory. The two sets of constraints focused on in this paper are the 'salience assignment' task described above for identifying which of several features are predictive of a particular result, and the problem of response latency in instrumental and classical conditioning. Many of these empirical results are interdependent. The data on latency depend on the existence of a salience assignment (contingency) algorithm, and within the latency data, there are two sets of related and apparently contradictory empirical results that require conciliation.

This paper presents computational theories of both contingency and latency in classical and instrumental conditioning tasks. We also present a theoretical framework or

architecture for the description of algorithms, with the intent of continuing to use the framework as a way of unifying otherwise disparate results. This framework arises from the identification of a set of computationally primitive functions that can be used as building blocks for the construction of useful 'functional modules' which are used as the main unifying language of presentation of the algorithms. These functional modules interact together to effect the storage and retrieval of information.

The level of description of these primitives and modules is, therefore, the algorithmic level, rather than a neurobiologically-inspired one (consisting, for instance, of a net of neuron-like nodes), or a psychologically-inspired one (which might consist of processes like 'recall' or 'attention'). The language of this algorithm level is, rather, that of information-processing functions, arising from abstract considerations of the recording and retrieval of represented information. This level has the advantages that (a) it can be descriptively rich enough to represent functions that must occur in any storage and retrieval system (including both nodes-and-links models and psychological models), and (b) it can be used to analyze the computations that such schemes actually carry out, without the details of the implementation (e.g., the properties of the neuronally-inspired nodes) intruding. Our goal is to characterize the processing which must take place during learning, independent of the actual implementation of processes at the neural level.

2.3 The Functional Modularity hypothesis

Viewed at the algorithm level, there exist abstract computations that must be carried out at each particular step in the creation, retrieval or modification of memories. For example, 2D visual input must be transformed into a representational 'code' as a necessary module of the task of memory storage; the output of this module is made available as readable input to other modules. Regardless of what specific algorithms are proposed for performing this job, the fact that it must be done, and the ways in which it may be

used by other equally necessary modules can be specified in this abstract way.

The idea of 'modules' is usually associated with observable behaviors such as vision, language or motor movements. Our view of modularity arises, rather, from abstract functional considerations of information storage and retrieval. Visual processing, as an example, emerges as one of many special cases of the general 'functional module' of 'sensory input coding' (along with audition, olfaction, proprioception, etc). We term this functional module 'DETECT' – such a function *must* be performed, and must interact with other functional modules, regardless of the specifics of its algorithmic formulation. (This also implies that the functional modules presented here can not necessarily be mapped in any simple or straightforward way onto brain structures; the functions of these information-processing modules may well be distributed among many different such structures). To give a further example, the ability to *recognize* an event or state that has occurred before might be performed via any number of algorithms, to compute whatever turns out to be the correct set of computations. Regardless of how it is performed, however, it must exist (it might be termed 'REMIND' or simply 'Recognize'), and the ways in which it interacts with other functional modules (e.g., sensory input DETECTion) can be specified within the architecture.

We have identified twelve such functions, and are investigating their natures and interactions in an ongoing research program [Granger 1982, Granger and McNulty 1984, McNulty and Granger 1985]. Any algorithm for recognition may be viewed in the framework provided by the architecture, and hence constraints on such algorithms can be predictively stated independently of the algorithms themselves, via the architecture. That is, even in the absence of a complete computational analysis or detailed algorithm for, say, a recognition-matching (or 'REMINDing') process, the identification of this process as a necessary functional module enables us to identify its interactions with other such modules.

The rest of this section outlines each of the twelve functional modules so far identified within the CEL architecture, classified by five basic subdivisions of the storage and retrieval of information. Each of these 12 functional modules (or simply 'operators') denotes the need for (eventual) development of computational theory and algorithms for particular information-processing tasks; the operators as described here can be viewed as 'placeholders' in the framework.

2.4 The Functional Modules of the CEL Architecture

2.4.1 Information-processing subdivisions of learning and memory

The overall functions of learning and memory are divided into five basic categories, as follows:

1. *Reception:*

   Establishing a temporarily-active memory trace from incoming sensory data, that is, creating a short-term memory trace.

2. *Recording:*

   Incorporating a temporary memory trace into a permanent memory trace, allowing for the subsequent effective retrieval of that memory in appropriate situations.

3. *Retrieval:*

   Activating existing permanent memories when and if they are appropriate to the processing of incoming sensory data.

4. *Reconstruction:*

   Using existing memories to process incoming data, for example, to predict and react to new experiences on the basis of previously recorded and retrieved memories.

5. *Refinement:*

   Altering memory traces on the basis of successes, failures, and differences between recorded memories and new incoming experiences, to strengthen and weaken associations, reinforce, extinguish, and differentiate representations of experiences.

Within this functional subdivision of memory processing, we identify a set of twelve functional modules. (Each module is in turn composed of primitive computational functions, derived from those used in theoretical computer science [Schönhage 1980, Knuth 1968] including 'match' (the comparison of two items), 'traverse' (the excitatory propagation of information), 'prune' (the inhibition of traversal), 'gate' (the modulation of the operation of other computational functions), 'establish' (the laying down of a representational trace), 'link' (the establishment of a connection between representational traces), 'input' (sensory inputs from the external world) and 'output' (motor outputs to the external world)). The twelve functional modules are viewed as memory operators which, taken together, perform the five classes of memory manipulation. The twelve operators act in parallel and asynchronously. The CEL framework consists of the interactive operation of these modules on memory traces.

2.4.2 Operators as emergent properties of memory

All twelve operators are viewed as emergent properties of the implementation level, that is, they arise from the functioning of a parallel, distributed memory, and are not 'homunculi' that oversee or control the processing of memory. We have already argued that the simple statement of *independent* algorithms for each of several computations is an insufficient analysis for algorithms that *interact* with each other. Merely turning to the implementation level (e.g., neurally-inspired models) to look for emergent properties that might account for algorithmic interactions loses the valuable expressive power of the algorithm level. Rather, providing an architecture at the algorithm level itself provides

an expressive language within which to make these interactions explicit. Hence, we present the framework at the algorithm level, while acknowledging that the gap between the implementation level and the framework must eventually be bridged.

Therefore, concepts that are perfectly sound within the framework at the algorithm level, such as 'short-term trace activation' vs 'long-term' or 'permanent' trace encoding, are intended only to exist at the algorithm level, as are the proposed representations that encode feature, event and temporal information in our theory. These are all, at bottom, emergent properties of the operation of the substrate. A complete theory must of course eventually provide a complete bridge between the algorithm and implementation level; our representation of a parallel, distributed memory of nodes and links that pass numeric (non-symbolic) values is meant as a first step towards such a bridge.

2.4.3 Reception Operators: DETECT and SELECT

There are two reception operators: DETECT and SELECT. DETECT performs sensory input and special preprocessing. For example, specialized processing performed by the visual system would be encapsulated within DETECT. The CEL framework doesn't focus on problems of nonspecific attention or arousal, and hence in this paper we make the simplifying assumption that the organism will reliably DETECT all incoming sensory stimuli. SELECT, however, acts as an active filter on the sensory information processed by DETECT, controlling which sensory inputs are maintained as temporarily active (those inputs may then be thought of as being in 'short-term memory').

2.4.4 Recording Operators: NOTICE, COLLECT, DETOUR, and INDEX

Once representations of external experiential events have been activated as a temporary memory trace, the recording operators may act to establish parts of that trace as a permanent memory.

CEL's first recording operator, NOTICE, monitors the characteristics of temporary-memory events (which were activated by the SELECT operator), checking those characteristics against an internal (e.g., hypothalamically-regulated) set of known desirable and undesirable features. NOTICE maintains the hedonistic state of the organism, and also adds the current hedonistic value to the information stored in event descriptions in the short-term memory store. If the hedonistic value reaches a threshold, either positive or negative, then the rest of the recording operators will be triggered, initiating the transformation of the temporary trace to a permanent or long-term memory trace.

A long-term memory trace, or schema, is represented in CEL as a sequence of events, and each event is represented as a set of features. Actions performed by the organism and features perceived from the outside world are represented in a similar manner; i.e., there is no separate representation language for actions. The COLLECT and DETOUR operators denote the packaging of short-term memory traces into schemas. COLLECT packages a desirable schema; DETOUR, an undesirable one.

COLLECT and DETOUR 'hand' their newly created schemas to INDEX for storage in long-term memory. In order for a memory trace to be retrieved in appropriate circumstances, it must also include information for matching against future sensory input. INDEX provides the appropriate keys, and maintains an organization in memory which leads to efficient retrieval.

Before establishing a schema as a long-term memory trace, INDEX scans the hedonistic value tagged by NOTICE to each event. INDEX assigns the schema as a whole the most extreme value present, so that retrieval operators can efficiently compare the desirability of schemata. INDEX also promotes plausible experimentation by generating alternative indexing schemes and creating variations of schemata [Granger 1982, Granger and McNulty 1984, McNulty and Granger 1985].

2.4.5 Retrieval Operators: REMIND and ACTIVATE

When an experiential schema has been indexed according to some particular event in the sequence, then the next time the event occurs, the entire schema is recalled or triggered. This 'reminding' phenomenon is a result of the schema's index being matched against the new occurrence of the event. The retrieval operator REMIND denotes the matching of the last event specification in short-term memory against the indexing structure built by INDEX in long-term memory. Schemata which share significant features with the current event are 'placed' in intermediate-term or 'reconstructive' memory. ACTIVATE denotes the process of examining each of the schemata in intermediate-term memory and choosing one for reenactment on the basis of a number of metrics. This chosen schema is called the current predictive schema (CPS). The competition between these REMINDed schemata bears some similarity to 'conflict resolution' in a production system architecture [see e.g., McDermott and Forgy 1978, Anderson 1983].

2.4.6 Reconstruction Operators: SYNTHESIZE and ENACT

Once an experiential schema has been found and retrieved via the REMIND and ACTIVATE operators, the organism will 'reconstruct' the memory as though the current episode were a new instance of the remembered one. This reconstruction process has two components: (1) monitoring the similarities and variations between features of the current episode and those in the retrieved schema; and (2) actually producing the efferent actions corresponding to those which were recorded as having occured in the original episode. That is, the organism re-creates the mental and physical states associated with each of the events in the schema.

The SYNTHESIZE module denotes the matching of the CPS to events appearing in short-term memory. While event descriptions in the CPS and short-term memory match, the ENACT operator attempts to perform any actions in the next event of the

CPS.

2.4.7 Refinement Operators: REINFORCE and BRANCH

The refinement operators, REINFORCE and BRANCH, are triggered by SYNTHESIZE. If SYNTHESIZE is not able to match an event description in the CPS with the most recently experienced event in short-term memory, it triggers the refinement operator BRANCH. If SYNTHESIZE has been able to successfully match each event description in the CPS with sequential event specifications in short-term memory, then it will trigger the refinement operator REINFORCE. REINFORCE incrementally strengthens a schema. BRANCH packages event descriptions into a new schema for indexing, effectively reducing the strength of the current predictive schema (CPS) by adding a competing schema to long term memory.

The CEL operators and their paths of interactions are sketched in figure 1.

2.5 Organization of indexed memory

Long-term memory in CEL is organized as a network of nodes and links accessed via parallel matching processes. The REMIND operator runs as a distributed process in the network, as to some extent do other operators such as BRANCH and REINFORCE. Besides providing a degree of parallel processing our model provides a principled method of limiting the spread of activation during retrieval, by 'diagnosticity' of tests applied during activation (see section 3.6).

Links in our model are simple one-way transmission channels capable of communicating a single non-symbolic value (magnitude) between nodes. These are categorized according to the interpretation of the signal they carry as one of four types of signal: probe links, trigger links, expectation links, or confirmation links. Nodes are of two types: feature nodes (henceforth nodes) and intermediate nodes (henceforth internodes)

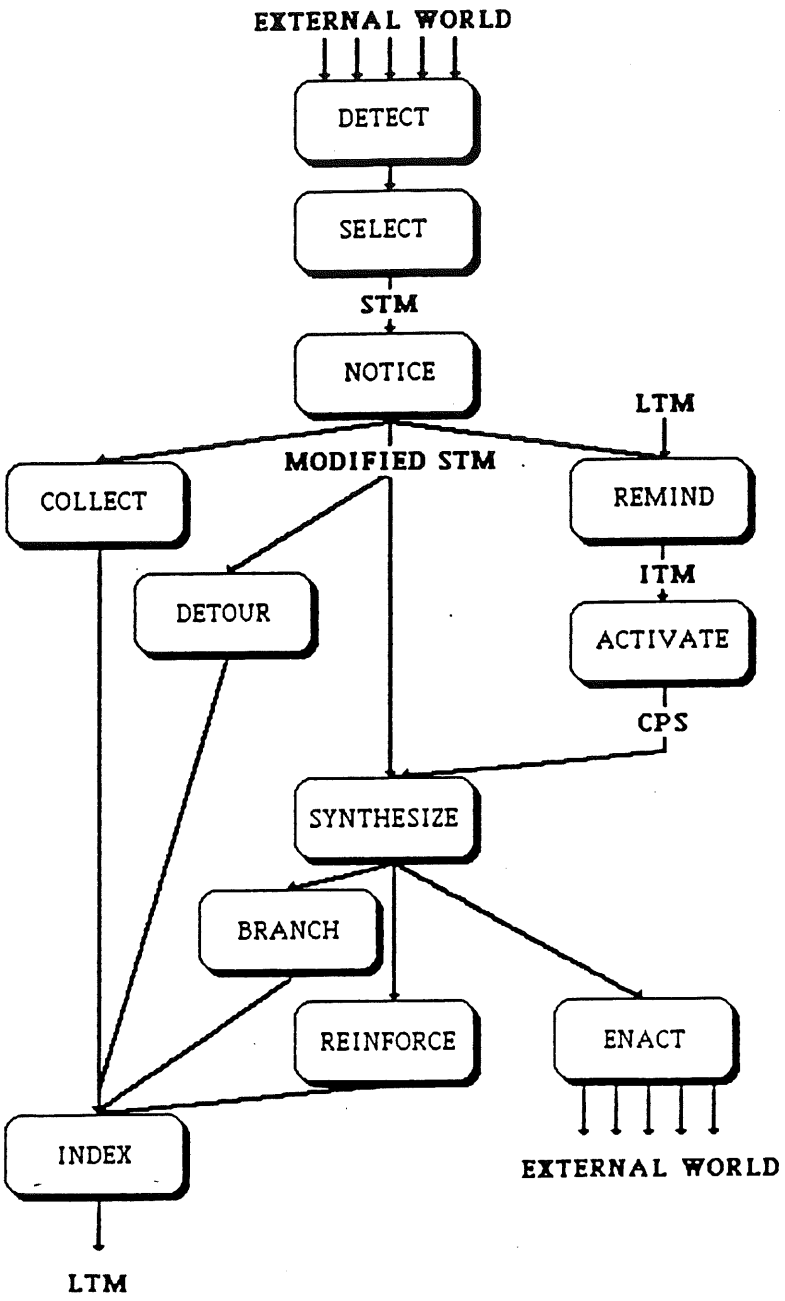

Figure 1: Outline of CEL operators

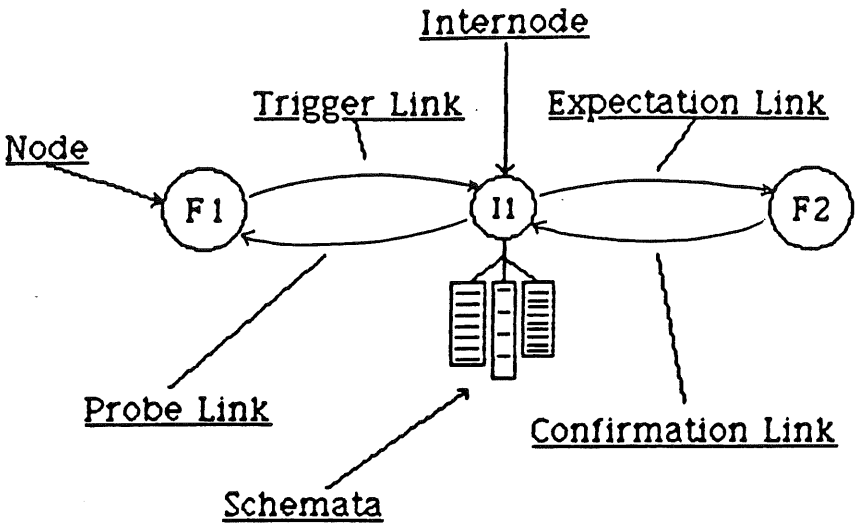

Figure 2: An example memory index structure.

which record relationships between feature nodes.

If a feature F1 is indexed as a cue which may lead to an expectation of feature F2, then an internode will lie between node F1 and node F2, as illustrated in figure 2. Long term memory traces which record occurrences of both F1 and F2 are accessed via the internode between them. Feature F1 may be triggered by incoming sensory data (i.e., via DETECT and SELECT) and in turn send a signal to I1, which adjusts the signal strength based on the predictive value of F1 for F2, and passes it on. A detailed description of both the organization and activation of memory in CEL is given in section 3.6.

3 Contingency and Salience Assignment

3.1 Contingency: The data

Imagine again a rat attempting to decide, over trials, which of several environmental features or events should be learned to be a predictor of a recurring shock event. We can view the animal's task as having to hypothesize potential prediction relationships between the shock and the various features, individually and in various combinations, and to weigh these hypothesized relations against each other.

We might initially assume that a particular feature or event would be inferred to be the predictor of the shock depending on the number of times that feature actually occurred immediately before the shock. Each time the feature is paired with the shock, the 'association' between the feature and the shock might be 'strengthened' (see e.g.,[Anderson 1983]). Extensive experimental evidence in the psychological literature shows this to be false. Over time, a particular feature, say the tone, may be paired with the shock more often than is some other feature (e.g., light). However, this condition is not by itself sufficient to warrant the animal's inference that the tone is more likely to predict the occurrence of shock. In particular, even if it happens over a number of trials that the tone precedes the shock more often than the light precedes the shock, say 7 times versus 4 times, *but* the shock also occurs a large number of times without having been preceded by the tone (say 8 times), while the shock only rarely occurs without the light (say 3 times), then an animal will learn to predict that the light, and not the tone, is the better predictor of the occurrence of shock. Hence, the naïve idea that the number of pairings alone determines the predictiveness (or salience) of candidate predictive features, or that 'strengthening' alone could be the mechanism for learning associations, is false.

3.2 Contingency vs number of pairings

This result requires a somewhat counterintuitive computation on the part of the animal: the animal must be computing, somehow, the probability of the shock occurring given the light and given the tone. (Again, our saying that this 'must be computed' speaks only of a behavioral constraint, but does not yet address the question of the *algorithm* that the animal might be using to arrive eventually at behavior that conforms to the constraint.) Rescorla's [1966, 1967, 1968] formulation of the necessary computation is that the probability of the shock outcome (the US) given the conditional stimulus feature (the CS, e.g., the tone) must be greater than the probability of the outcome occurring without that feature having occurred, or $p(US|CS) > p(US|\overline{CS})$.

This measurement of relative probabilities is referred to as ***contingency***; animals, and humans in analogous circumstances, exhibit ***contingency-driven learning*** in the sense that they somehow maintain incrementally-updated knowledge of the relative predictiveness or *salience* of features. Through experience the animal must pick out the relevant (salient) features from the background of uncorrelated features and use only these salient features to predict future events.

3.3 An algorithm for salience assignment

3.3.1 What is computed

We term the problem described here as *salience assignment*, i.e., the differential assignment of predictive value to the candidate predictive features (or combinations of features). All that the animal has available to it as input from the environment is the presence of features sensed over time. What it must compute from these inputs is the relative probability of some features relationships to others over time.

There are four logical categories of these relationships that can be computed: positive

predictions, negative predictions, uncorrelated cues, and context; each has a behavioral correlate in animals. First, there are two types of what we term *predictive* cues: positive and negative predictions. Positive predictive cues are those that accurately (within the bounds of relative probability as described above) predict the occurrence of an outcome. Negative cues accurately predict the absence of an outcome (e.g., these 'safety signals' might predict that the shock will not follow the cue, and therefore that the animal need not fear its coming). *Uncorrelated* cues are irrelevant and therefore not necessary for prediction of an event. Finally, an animal cannot readily evaluate the predictive importance of a *context* cue, i.e., one that occurs constantly in the background of a training session. It is impossible to know whether such a cue is a necessary precondition for predicting a shock, unless the shock has been predicted a few times in the absence of the context cue. When this happens, either the context cue will become a positive predictive cue (if the prediction was successful), or an uncorrelated cue (if the prediction failed).

The rest of this section is devoted to the presentation of an algorithm that assigns feature cues to one of these four categories on the basis of experience.

3.3.2 Partial analysis of the inputs

The inputs to be categorized as positive, negative, context or uncorrelated are occurrences of features. For simplicity, we can categorize the logically possible pairwise combinations of two feature events F1 and F2: either F1 occurs and then F2 occurs (*prediction*), F1 occurs and then F2 does not occur (error of *commission*)[1], F1 does not

[1]We classify these events according to the types of error an animal can make. An error of *commission* occurs if the animal 'commits' an erroneous prediction of F2, while an error of *omission* occurs if the animal mistakenly 'omits' a prediction of F2 when one should have been made. These terms might be assigned in the opposite sense if events were classified in terms of how the world differed from expectations, e.g. omission of an expected event, as in an omission schedule.

occur and then F2 does occur (error of *omission*), or F1 does not occur and neither does F2 (*non-prediction*). The first and last of these combinations strengthen the predictive value, or association, between F1 and F2, while errors of commission and of omission weaken the association.

	F2 present	F2 absent
F1 present	++ Prediction	+− Error of Commission
F1 absent	−+ Error of Omission	−− Non-prediction

Table 1: Possible combinations of F1 and F2

(Note that non-predictions are implausible to describe as 'events'; the absence of F1 combined with the absence of F2 happens almost all the time, i.e., all times except when either F1 or F2 occurs).

The proposed algorithm for the calculation of contingent predictiveness essentially just keeps a running count of each of these four categories of pairwise events;[2] these counts are used to calculate an estimate of the likelihood that one of these two features predicts the other.

3.3.3 The basis of the algorithm: *Sufficiency* and *Necessity*

Bayesian statistics (see e.g., Duda et. al. [1979]) provide formulae for the calculation of two values in inductive logic: *Logical Sufficiency (LS)*, which indicates the extent to which the presence of one event predicts, or increases the expectation of, another

[2]Although a non-prediction will *only* be considered to happen when either F1 or F2 has been predicted, and then neither occurred. This is because, as noted above, all non-predictions would otherwise give rise to a huge, ongoing number of counts. Hence, in this algorithm, non-predictions are systematically 'undercounted'.

particular event; and, reciprocally, *Logical Necessity (LN)*, which represents the extent to which the *absence* of an event *decreases* expectation or prediction of the second event. LS and LN are each calculated by a simple formula composed of precisely the four possible categories of pairwise feature occurrence given above.

$$\mathrm{LS} = \frac{s(n+o)}{o(s+c)} \qquad\qquad \mathrm{LN} = \frac{c(n+o)}{n(s+c)}$$

where s is the count of successful predictions, c is errors of commission, o is errors of omission, and n denotes non-predictions.

Our proposed algorithm makes use of the calculation of LS and LN values to categorize the relationships between a pair of cues. The categorization is based on the interpretation of LS and LN values. LS values range from 0 to ∞, with high LSs corresponding to a feature F1 strongly predicting a second feature F2, (since high LS implies a high ratio of successes to errors of commission); and very low LSs corresponding to the case where F1 implies that F2 will *not* occur (low ratio of successes to commissions). Hence, for a high LS value, F1 is a positive predictor of F2; for low LS, F1 is a negative predictive cue, i.e., the presence of F1 predicts that F2 will not occur. An LN value near 1 indicates that the absence of a cue may be ignored, while a low LN value (near zero) indicates that a particular cue is quite necessary for prediction. When the value of LS is approximately 1, i.e., neither very high nor very low, then the cue F1 may be either a context cue or uncorrelated. In such a case, if there are more errors of commission than of omission, i.e., more failed predictions than unexpected shocks, then the cue is categorized as a context cue, since it is often present, but often fails to predict the shock; yet the shock doesn't often occur in its absence.

If there are more errors of omission than commission, however, then the cue is categorized as uncorrelated.[3]

Positive cue	$ls >> 1$
Negative cue	$ls << 1$
Context	$ls \approx 1, omissions < commissions$
Uncorrelated	$ls \approx 1, omissions \geq commissions$

3.4 CAP-CEL: An implementation of the salience assignment algorithm

3.4.1 Introduction to the detailed algorithm

We have designed a computer program called CAP-CEL (Contingent Associative Process in CEL) that is a detailed implementation of the salience assignment algorithm outlined above, thereby simulating contingency-based learning. The major processes and data structures of CAP-CEL are those of CEL, although some details of CEL have been simplified or omitted. In response to a pleasant or unpleasant stimulus (F2), CEL forms a memory trace, or schema, containing several events that lead up to it (see section 2.4.4 or [Granger and McNulty 1984] for details of trace-establishment processes in CEL). CAP-CEL operates by establishing memory traces of these 'streams' of input features, keeping track of successful predictions, omissions, commissions and non-predictions and calculating logical sufficiency and necessity as part of the internal memory-indexing process of the model. CAP-CEL continually categorizes feature clusters as either predictive

[3] As stated earlier, it's impossible to know *a priori* whether or not such a cue is a necessary precondition for predicting a shock, unless the shock has been predicted a few times in the absence of the context cue. When this happens, either the context cue will become a positive predictive cue (if the prediction was successful), or an uncorrelated cue (if the prediction failed).

cues, context cues or uncorrelated features, and exhibits behavior based on retrieval of memorial schemata according to this indexing classification. CAP-CEL uses a parallel, distributed, indexed memory organization, described in section 3.6.

3.4.2 Gathering evidence

All counts in CAP-CEL's memory are initially 1. These counts will be updated only when an index node (corresponding to a feature complex) is triggered by matching cues in the environment (REMIND), at which point one of the schemas organized below this index node is chosen (ACTIVATE) to become the 'current predictive schema' (CPS); i.e., the current memory that contains predictions of what will happen and what behavior is associated with these predictions.

From this point on, the CPS is continually matched against new events as they occur, via the SYNTHESIZE mechanism, which eventually will trigger REINFORCE or BRANCH, depending on whether the match between the current episode and the CPS is successful or unsuccessful.

REINFORCE comes into play when a prediction has succeeded. Cues in the index node which match features in the environment during the current experience are called successes, and their success scores are incremented. Cues in the index node which did not match features in the environment are called omissions, and their omission scores are incremented.

BRANCH is responsible for contingency assignment when predictions fail. BRANCH scores a commission for each cue feature that matched the environment, and a non-prediction for each cue feature that was absent from the environment. Features present in the environment but not present in the schema are added to the index with an initial score of 1 commission, 1 prediction, 1 omission, and 1 non-prediction.

	++	+−	−+	−−	LS	LN
Cage	9	9	1	1	1.0	1.0
Tone	4	4	6	6	1.0	1.0
Light	4	4	6	6	1.0	1.0
Buzz	4	5	6	5	0.81	1.22
Whrr	4	4	6	6	1.0	1.0

Table 2: Random Contingency

3.4.3 A detailed example

Assume CAP-CEL is simulating a situation where tones, lights, noises, and shocks are occurring. CAP-CEL's task is to construct a memory record that allows it to learn which of the many features of the environment predict the occurrence of shock (so that it can avoid the shock). In this section we will illustrate what a 'snapshot' of CAP-CEL's memory would look like in three circumstances: (1) where shock is randomly paired with a number of environmental features (*random contingency*); (2) shock is reliably preceded by a conjunction of predictive features (e.g., tone and light) (*positive contingency*); and (3) shock only occurs in the absence of a particular feature cue (*negative contingency*), so that the cue becomes a 'safety signal'.

After training in the random condition, i.e., with tone, light, shock, etc., *independently* occurring at regular intervals, CAP-CEL will have a memory node similar to table 2 (note that successes are indicated by '++', commissions by '+−', omissions by '−+', and non-predictions by '−−'. The figures in tables 2, 3, and 4 are taken from runs of an early version of our computer model.) The LS (logical sufficiency) value indicates the degree to which a cue is sufficient to cause expectation of a result feature, with values greater than 1 indicating a positive contribution to expectation. The LN (logical necessity) value indicates the degree to which absence of a cue precludes expectation of

	++	+−	−+	−−	LS	LN
Cage	52	11	1	1	1.65	0.35
Tone	52	7	1	5	5.29	0.14
Light	52	8	1	4	4.33	0.17
Buzz	19	4	34	8	1.02	0.91
Whrr	43	10	10	2	0.97	1.13
And[Tone,Light]	48	3	1	5	5.65	0.07

Table 3: Positive Contingency

a result feature. An LN value near one indicates that absence of a cue may be ignored, while an LN value near zero indicates that a cue is quite necessary for expectation.

CAP-CEL in a positive continency condition, on the other hand, would have a memory node like table 3. The conjunction of light and tone has been proposed by the CAP-CEL program itself (see discussion in section 3.5). This chart illustrates important differences between contingency learning and more intuitive notions of strengthening based on number of pairings. Cage and tone receive the same number of pairings with shock, but tone is a much better predictor of shock. Moreover, tone was involved in a greater number of mistaken predictions (errors of commission) than was buzz, but tone is still recognized as the better predictor.

Finally, in a negative contingency situation, CAP-CEL's memory would have gathered statistics like table 4. In training situations with aversive stimuli, negatively contingent cues have been shown to increase appetitive behavior and decrease avoidance behavior (even when the avoidance or appetitive behavior is not related to the stimulus whose absence is predicted). Negatively contingent cues may also serve to suggest particular avoidance or escape reactions.

	++	+−	−+	−−	LS	LN
Cage	22	6	1	1	1.57	0.43
Tone	1	3	22	4	0.30	4.88
Light	12	3	11	4	1.09	0.75
Buzz	11	5	12	2	0.80	2.19
Whrr	11	4	12	3	0.92	1.33
Not[Tone]	22	4	1	2	2.54	0.23

Table 4: Negative Contingency

3.5 Combining features

It is not sufficient to note relations between individual features; it is also necessary to note useful *combinations* of features. We will use the term 'clause' to refer either to a memory node representing a single feature or a node representing a boolean combination of features. The CAP-CEL model uses current associations between clauses to suggest new combinations. Clauses containing boolean AND or NOT are introduced to discriminate between positive and negative instances.[4] Generation of new clauses is guided by current relations between clauses and the features to be predicted. When a clause is satisfied in a negative instance but not in a positive instance, and that clause has an LS value less than 1, it is a candidate for negation. When a clause with a LN value below 1 is present in a negative instance it is a candidate for conjunction with a clause that is unsatisfied in the negative instance and also has a LN value less than 1. We can think of the value of clauses as guiding a plausible move generator for searching the space of discriminated conditions.

[4]It is desirable to allow both discrimination (through conjunctions of clauses) and generalization (through disjunction). At this time, however, the CAP-CEL model proposes only conjunctions and negations. A weaker form of generalization is achieved by dropping clauses with low predictive value.

Propose	When (error of commission)
Not[A]	LS(A) << 1, A satisfied
And[A,B]	LN(A) << 1, A satisfied LN(B) << 1, B unsatisfied

3.6 Details of CEL's indexed memory operation

The algorithms described here for the incremental calculation of LS and LN and the use of those values to successfully categorize feature cues are all grounded in the operation of an indexed network memory (introduced in section 2.5). This section provides a bit more detail as to how this memory is organized, and how it operates to give rise to these calculations.

Recall that the memory is composed of nodes and links; links are simple one-way transmission channels that transmit non-symbolic values between nodes. Links are categorized as being one of four types, according to the interpretation of the signal they carry: probe links, trigger links, expectation links, or confirmation links. Nodes are of two types: feature nodes (termed simply 'nodes') and intermediate nodes ('internodes') which record relationships between feature nodes.

If a feature F1 is indexed as a cue which may lead to an expectation of feature F2, then an internode will lie between node F1 and node F2, as illustrated in figure 3. Long term memory traces which record occurrences of both F1 and F2 are accessed (REMINDed) via the internode between them. Feature F1 may be triggered by incoming sensory data (via DETECT and SELECT) and in turn send a signal to I1, which adjusts the signal strength based on predictive value of F1 for F2, and passes it on.

Internodes store two different expectation values relating F1 to F2. The first is

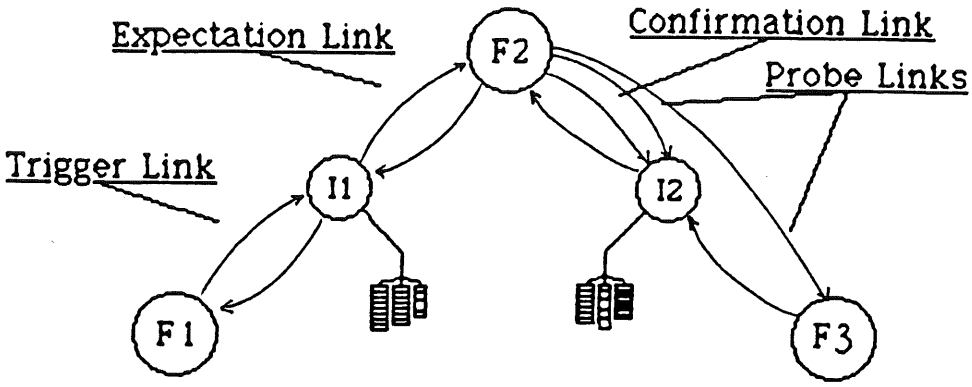

Figure 3: An example memory index structure.

sufficiency (LS), which indicates the extent to which presence of F1 increases expectation of F2. When a signal passes from F1 to I1 via the trigger link, the value passed from I1 to F2 on the expectation link is scaled by LS. The second expectation value stored at an internode is necessity (LN), the extent to which absence of F1 decreases expectation of F2. (LN is expressed as a multiplier between 0 and 1, with smaller values indicating a greater degree of necessity.)

To understand how this LS and LN are used, refer again to figure 3. Suppose that F1 matches an item in short term memory but F3 does not. Further suppose that F1 is moderately suggestive of F2, and F3 is quite necessary for an expectation of F2. Activation from F1 will be scaled to a moderate value by internode I1 and passed to F2. F2 will then send signals (of moderate strength) along each of its outgoing probe links. I2 receives a probe and, finding the LN value relating F3 to F2 is quite small (i.e., the presence of F3 is quite necessary for any expectation of F2), I2 sends a strong

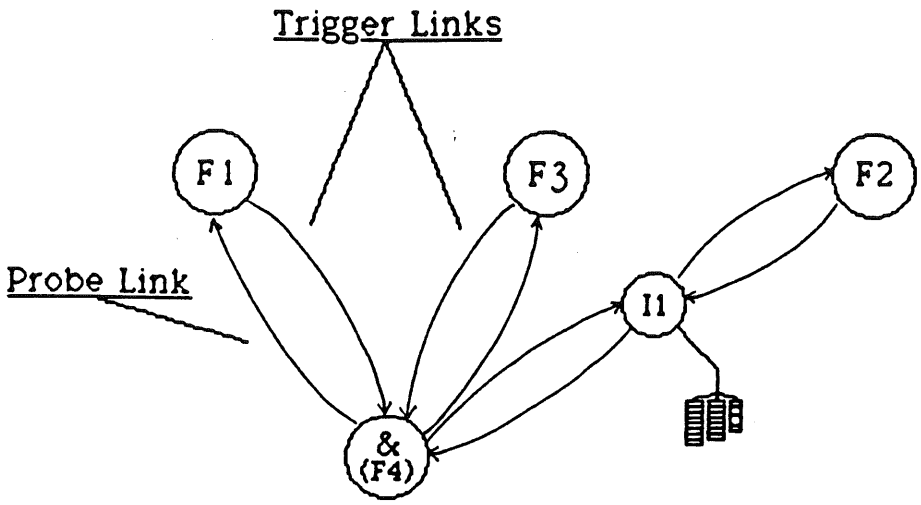

Figure 4: A memory index structure with combined features.

probe signal to F3. Since F3 does not return a signal along a trigger link, I2 sends an inhibitory signal along the expectation link to F2, reducing the expectation of F2. (Values are multiplied together by the receiving node; hence, an inhibitory signal is simply one that has a value less than 1; any such value will lower the result of the multiplication).

Feature nodes may represent combinations of features, as illustrated in figure 4. Node F4 is triggered when F1 and F3 are both present, so F4 functions as the boolean 'and' operator. If F4 is triggered by either F1 or F3, it sends probe signals to both. Other boolean combinations can be similarly represented.

The principle advantage of the index network as described above is that *test diagnosticity*, as expressed by LS and LN values stored in internodes, provides natural control for the parallel matching process of REMIND; the spread of activation is limited in a principled way by these tests on links between nodes, rather than by a 'decay' of

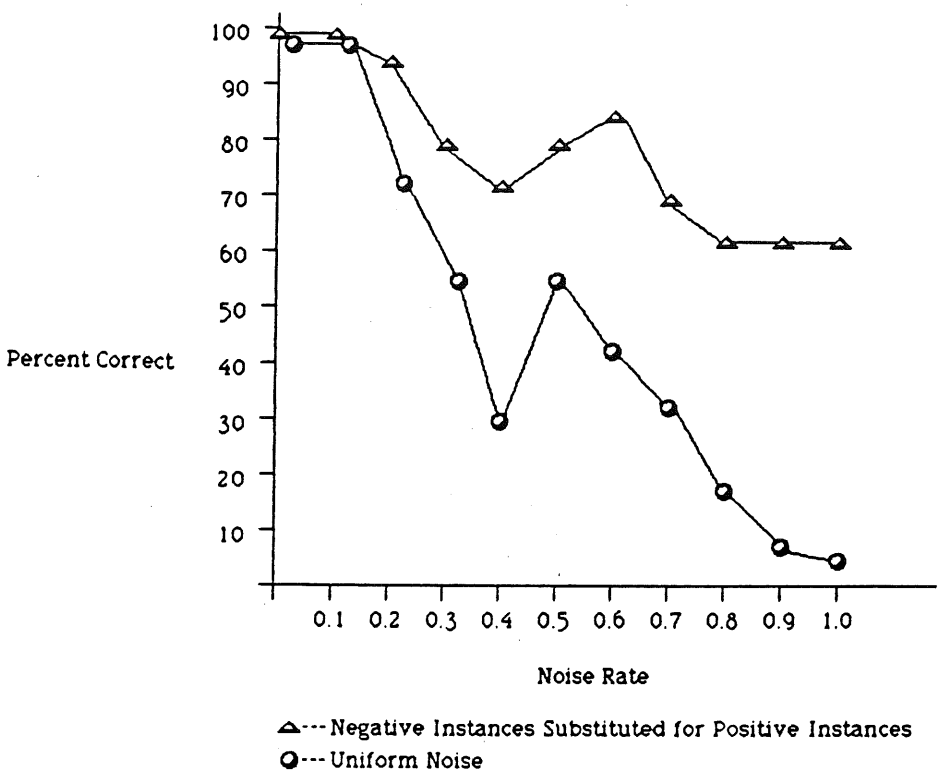

Figure 5: Performance of CAP-CEL as a function of noise.

strength of pursuit of links over time. The number of nodes needed to represent complex patterns is moderate, and the network can be grown incrementally as new relations between features are discovered.

3.7 Experience with the CAP-CEL System

3.7.1 Robustness

Real-world environments invariably entail some degree of noise, so a learning engine must be able to tolerate erroneous training instances. We have been pleasantly surprised by the performance of the CAP-CEL program in these circumstances. Figure 5 depicts the

performance of CAP-CEL when trained with various rates of erroneous instances.

The line plotted with circles in figure 5 shows performance under conditions of 'uniform' noise, that is, positive instances have been substituted for negative instances and vice versa, with equal likelihood. As the error rate approaches 0.3, the CAP-CEL's performance falls toward a chance level (50%). As one would expect, error rates in excess of .5 cause CAP-CEL to acquire the opposite of the training concept and to perform at less than chance level.

The triangles in figure 5 plot the performance of CAP-CEL when negative instances are substituted for positive instances, but there are no spurious positive instances. This is similar to partial reinforcement in conditioning. CAP-CEL's performance remains well above chance in this case even for levels of noise in excess of 50% because the tone and light are in positive contingency relation with shock even when the absolute probability of shock following the cues is low. CAP-CEL's level of performance is similar when the only noise is positive instances substituted for negative ones.

CAP-CEL's tolerance of erroneous training instance is partly due to the smooth weighting function. In addition, though, we found that robustness depends critically on the introduction of combined features (section 3.5). With no noise in the data, CAP-CEL can achieve perfect performance for simple conjunctive classifications even when the combination proposer is disabled, since the extreme values of LS and LN are sufficient to express logical necessity and logical sufficiency. But when even a small number of erroneous instances are introduced, performance falls off precipitously unless combinations are proposed. The predictive value of a combination of features is higher than that of any of its component features, and the influence of erroneous instances on that value is correspondingly less.

3.7.2 Annotated run-time output of the CAP-CEL program

We have implemented CAP-CEL in Franz Lisp on a VAX 11/750 running under Unix. In the following transcript, CAP-CEL learns that the conjunction of tone and light is a positively contingent predictive cue for the onset of shock in a classical conditioning setting. Annotations are separated from actual program output by semicolons.

⋮

```
Detecting: cage, light, tone, whrr      ; these are external cues.
    doubtful of shock occurrence (odds = 0.3 < 1).
Detecting: shock                        ; CAP-CEL doesn't predict the shock,
Updating Expectations                   ;  but gets one.
    marking successes                   ; satisfied cues get a success.
    marking omissions                   ; unsatisfied cues get an omission.

Detecting: cage, light, tone, whrr, buzz
    strongly expecting shock (odds = 3.25 >> 1).
Detecting: shock                        ; CAP-CEL successfully predicts the shoc
k.
Updating Expectations
    marking successes                   ; satisfied cues get a success.
    marking omissions                   ; unsatisfied cues get an omission.

Detecting: cage, light
    doubtful of shock occurrence (odds = 0.64 < 1).
Detecting: nothing                      ; CAP-CEL doesn't predict the shock
                                        ;  and doesn't get one.  This is a
                                        ;  successful prediction, but the counts
                                        ;  are not updated when a node isn't
                                        ;  activated.

Pattern                                 ; Long-term memory looks like this now:
    ++  +-  -+  --     ls   ||   ln   |
    ------------------------------------
not[whrr]                   ||        |
    2   1   3   1     0.89  ||  1.33  |; a mistakenly introduced clause.
    ------------------------------------
tone                        ||        |
```

```
    4   2   1   2     2.0   ||   0.5  |; tone has a low LN.
    ------------------------------------
light                       ||        |
    4   2   1   2     2.0   ||   0.5  |; light also has a low LN.
    ------------------------------------
buzz                        ||        |
    3   1   2   3     1.88  ||  0.42  |
    ------------------------------------
whrr                        ||        |
    3   3   2   1     0.75  ||   1.5  |
    ------------------------------------
cage                        ||        |
    4   3   1   1     1.14  ||  0.86  |
    ------------------------------------


Detecting: cage, light, whrr, buzz
    strongly expecting shock (odds = 2.14 >> 1).
Detecting: nothing                      ; CAP-CEL makes an error of commission.
Updating Expectations
    marking commissions                 ; satisfied features get a commission.
    marking non-predictions             ; unsatisfied features: a non-predictio
n.
Suggesting Possible New Feature
    and[light,tone]


        ;  The light cue is satisfied (present) in this instance
        ;  and has a LN << 1.  The tone cue is not satisfied
        ;  (missing) and also has a LN << 1.  Their conjunction
        ;  is suggested as a new clause.
```

3.7.3 Explanation of program behavior

Confidence in-CAP-CEL is calculated by multiplying together the LS values of each satisfied feature and the LN values of each unsatisfied feature. This confidence measure is then interpreted in terms of odds: much less than 1 indicates that F2 is not expected; about 1 indicates uncertainty; much greater than 1 indicates that F2 is expected.

CAP-CEL introduces new clauses only on errors of commission. By restricting the

proposal of new clauses to immediately after errors, CAP-CEL avoids endlessly making proposals when it reaches proficiency. Possible new clauses are proposed from the satisfied and unsatisfied features with LN values below a bound of 1. In the example above, light has an LN value below well below 1, as does tone, and an error of commission occurs where light is present and tone is absent; CAP-CEL hypothesizes that light *and* tone together might be a better predictor than either alone.

4 Latency: Learning When to Respond

4.1 The problem of timing a response

Again let us consider an animal (this time a dog) who hears a tone stimulus and several seconds later receives a shock. The dog jerks its leg in response to the shock. If the tone is in a positive contingency relationship with shock, then after many trials the dog will jerk its leg in response to the tone alone, whether it receives a shock or not. Suppose in every trial the tone and shock are separated by an interval of twenty seconds; does the dog jerk its leg immediately upon hearing the tone, or does it wait until the shock is 'due?'

The answer, as it turns out, is that the dog will jerk immediately if jerking can prevent the shock, but will wait if the shock is not controllable. In this section we show how this difference in *response latency* in classical conditioning and instrumental conditioning procedures can be accounted for in a unified way within the CEL framework.

4.2 The experimental data

A representative experiment performed by Wahlsten and Cole [1972] demonstrates the difference in response latency between animals in a classical learning situation and ani-

mals in an instrumental learning situation. The experimenters divided a population of dogs into two groups: classical (C) and instrumental (I). Both groups of dogs received tone in a positive contingency relation with an electric shock applied to each animal's leg. Animals in group C received shock with equal probability regardless of response. Animals in group I could terminate or prevent the shock by jerking their legs.

Animals in group C, who had no control over the shock, waited after the onset of the tone and jerked their legs immediately before the scheduled shock. Animals in group I jerked immediately after hearing the tone.

4.3 Computational requirements

We call this problem learning *effective timing*. The animal distinguishes between actions which may instrumentally affect the environment from responses to uncontrollable events. Since the same response (e.g., jerking a leg) may be placed in either category, classification must be based on experience. Moreover, if the animal always responded in the same way to the tone, (i.e., always waited the same amount of time before jerking), then experience would provide insufficient information for choosing between a quick instrumental response and a delayed classical response. For instance, if the dog always barked and jerked its leg exactly four seconds after hearing the tone, it could never decide whether the shock was prevented by jerking or by barking, and whether the timing of its actions was relevant or irrelevant. Thus the animal must learn effective timing based at least partly on *variations* in its own response.

4.4 A hypothetical account of response latency

We can account for the response latency in both classical and instrumental learning procedures with a single set of processes in the CEL framework. We hypothesize that memory traces are imperfect records of actual events, and that a common discrepancy

between a memory trace and the episode it records is that the trace omits certain unimportant events. Given this hypothesis, the observed difference between response latency in instrumental and classical learning situations arises naturally from the interaction of the CEL operators.

4.4.1 Shortening memory traces

As we noted above, an animal can only discover which of its actions are effective in controlling the environment if its responses are varied. An animal that always responds in the same way cannot tell what aspects of its actions are effective, or even if its actions are effective at all.

Simply omitting certain events from long term memory traces is sufficient to generate variations in response timing and enable the animal to learn whether timing is relevant. In fact, even completely random omission of events will eventually cause the animal to jerk its leg immediately after hearing the tone if doing so prevents the shock.

When the animal first hears the tone, and later receives a shock, a trace in is established in long term memory. This trace records both the tone and the shock, and may also record the action of the animal in response to the shock.[5] Over several trials the animal learns that tone is a good predictor of shock, as described in section 3. During this time a large number of memory traces, each recording the tone and shock and many recording the jerk response, will be stored. These traces will have many features in common, but also will differ in many ways, including timing.

[5]As mentioned in section 2.4.4, long term memory traces in CEL record both features perceived from the outside world and actions performed by the animal. See also section 4.4.4 for some justification and implications of this representation.

4.4.2 Small latencies in instrumental conditioning

Consider first an animal in an instrumental learning situation: If it jerks before the shock is delivered, the shock is prevented. We have said that a common variation in memory traces is the omission of some events, i.e. a schema which represents an episode shorter than an episode that really happened. If one of these shortened schemata becomes the current predictive schema (CPS; see section 2.4.5), then the animal will make an earlier response. Since the response still has precisely the predicted effect, i.e. the shock is prevented, the refinement operators of CEL will strengthen this trace in memory and make it more likely to be selected (and shortened) again.

4.4.3 Larger latencies in classical conditioning

It is natural enough that the schema shortening process leads to quick instrumentally conditioned responses, but why doesn't the same thing happen in classical conditioning? According to our hypothesis, schemas are shortened in either case. The difference is that, in a classical conditioning situation, a shortened schema will make an erroneous prediction and will therefore be weakened by the CEL refinement operators. In the case of the dog in an unavoidable shock situation, a schema that has been shortened will predict the shock earlier than it actually occurs. Thus, even if there is a systematic bias to shorter schemata, the full length schemata with the shock predicted at the proper time will remain dominant.

4.4.4 Omission schedules vs instrumental conditioning: A prediction

A representative example of an omission schedule is classical conditioning of the eyelid of a rabbit. The experiment uses a tone as a cue and an air puff to the eye of the subject which causes the rabbit to blink. When the blink precedes the air puff, the air puff may be omitted (thus the term omission schedule). Instead of responding in an instrumental

manner, i.e., learning that his blink prevents the air puff, the rabbit begins to show extinction of the eye blink response ([Sheffield 1965], [Mackintosh 1983]).

A problem for theoretical accounts of instrumental and classical conditioning is the question of why some animals in some circumstances learn to make instrumental responses to omitted stimuli, while at other times they simply extinguish the learned response as a result of the omission. The animal could be viewed as 'trying' to decide whether its own actions are responsible for the non-occurrence of the omitted air puff (or shock), in which case a continued instrumental response is appropriate, or whether the world has simply stopped behaving that way, in which case extinction of the response is appropriate.

The problem is that omission schedules and instrumental conditioning situations are indistinguishable from each other, except for the difference in animals' responses; hence, the difference must be in the animal, not in the situation. One hypothesis to account for this is based on the idea of 'preparedness'. Some animals are apparently better 'prepared' than others to learn certain associations (see e.g., [LoLordo 1979]); for instance, cats have difficulty learning that the action of licking their paws leads to their release from a cage [Thorndike 1911]. It would be reasonable to suppose that the cats may simply not record the action of licking their paws, if that is a low-level procedural task. Failing to record (COLLECT) the event would certainly predict that the association would not be learned.

To elaborate on this hypothesis, it is necessary to illustrate the connection between what events actually get stored, and what behaviors will be possible to learn as a consequence of either storing or not storing. Rats are capable of storing the *results* of conditioned associations, not just the responses that may lead to those results.[6] In

[6]This is elegantly and clearly demonstrated by Rescorla [1980]: animals are first trained that a tone means flavored water will be found at the end of a hallway. When that flavor is later associated with

many circumstances, however, it is possible that the *response* (e.g., blinking an eye) leading to a result is *not* stored, even if the result itself (e.g., avoid air puff) is stored. In such a case, classical conditioning would still be possible (though it might take many more trials than otherwise), as long as the expectation of the result (e.g., shock, air puff) could cause ENACT to generate an appropriate response. (This would imply that ENACT had available to it some purely 'procedural' way of causing a motor movement, even if that motor movement could not be declaratively represented in a memory trace).

In such a case, then, the response action itself would not have been declaratively encoded, but that action would be produced nonetheless by the ACTIVATEd expectation of the shock, via this procedural knowledge within ENACT. This would enable classical conditioning to occur over trials, but would NOT permit instrumental conditioning to occur; rather, an omission schedule would emerge. A declarative encoding of the response action would have to be present in order for that response to be able to be generated earlier via a mechanism like our variation-generator, as already described in this section. Hence, we may predict that in precisely the cases where an animal is not 'prepared' to readily learn an association (as in rabbits and eye-blinks), then also instrumental conditioning will prove impossible, and omission-schedule extinction behavior will result instead.

illness, the animal will not go to the water on hearing the tone. The animal knows that running down the aisle will get him to the water, but he now also knows that that water is an undesirable result.

4.5 CEL-TIC: A computer model of response delay

4.5.1 Description of the program

We have constructed a computer model called CEL-TIC (CEL with Temporal Information Coding) embodying the hypothesis outlined above.[7] Part of the computer program simulates the environment of the dog, and this part may or may not make adjustments depending on the actions of the part of the model which simulates the dog. By changing only the part of the model which simulates the environment, we can ascertain that the part which models the dog reacts differently depending on whether its actions prevent the (simulated) shock.

The CEL operators are simulated by several independent processes running asynchronously (i.e., there is no set order governing the actions of the operators). The INDEX operator is given responsibility for generating variations in memory traces. This is accomplished in simple manner which does not assume any special knowledge or guidance in INDEX: it simply omits events at random from memory traces.

Despite this simple approach to experimental behavior, the program develops a quick response to the simulated tone when the simulated shock can be prevented; it waits almost until the shock is scheduled when shock is unavoidable.

[7]The CEL-TIC computer model described here is distinct from the CAP-CEL program described earlier in this chapter. Each program contains simplifying assumptions about some aspects of the CEL architecture. For instance, the representation of sequences of events is irrelevant to the issues explored with CAP-CEL, while the generation of new index nodes for combinations of features is irrelevant to issues of response latency. Our longer-term goal is to build a single computer program that combines what we have learned in these and other projects.

4.5.2 Annotated run-time output of the CEL-TIC program

CEL-TIC, like CAP-CEL, is written in Franz Lisp on a VAX 11/750 running under Unix. The following output from a CEL-TIC run, begins after CEL-TIC has acquired the contingencies between the cue and the result (tone and shock respectively) and learns to make a conditioned response by moving its leg in anticipation of the shock's onset. Annotations are separated from actual program output by semicolons.

⋮

```
-- 91 --      ; Time clock.  CEL-TIC divides the stream of input events
              ; into time 'tics'; multiple events may happen at a single
              ; time tic, and a single event may last for many tics.


        ; The CEL operator DETECT perceives the status of features in the
        ;   external environment.  The leg is relaxed, there isn't a shock,
        ;   but the tone is on.  This is the beginning of an aversion trial.

Detecting
    no move-leg, no leg-shock, and tone

        ; The CEL REMIND operator denotes the retrieval of schemata
        ;   from long-term memory that match with DETECTed features.

Reminding
    of leg-shock in s-2, s-14, s-12, s-8, and s-0
    of move-leg in s-2, s-13, s-10, and s-0

        ; Each REMINDed schema (s-2, s-12, etc.) contains an occurrence
        ;   of the tone which leads either to the leg-shock or movement
        ;   of the leg (in two lines, above).

        ; The CEL operator ACTIVATE denotes the choice of a single schema
        ;   to recreate from those retrieved from long-term memory,
        ;   based on the significance of their outcomes, their predicted
```

```
        ;   desirability and the number of times each has occurred.
        ;   In this case, a schema whose outcome is the shock (i.e.,
        ;   which predicts anticipating the shock), and which has
        ;   occurred frequently, is chosen.  This schema becomes the
        ;   current predictive schema (CPS).

Activating new schema s-12
    tone leads to move-leg and leg-shock in s-12
      (occurred 5 times, hedonistic value is low)

        ; In the following sequence, the 4-'tic' schema will be
        ;   ENACTed; all the while, SYNTHESIZE will be matching actual
        ;   world events against those that are predicted (i.e., those
        ;   that occur in the schema.  As long as the two match,
        ;   ENACT will be triggered by SYNTHESIZE to perform any
        ;   efferent actions in the coming event specification.
        ;   (An arrow [<---] denotes a match; [<-X-] denotes mismatch).
        ;   Simultaneously, ACTIVATE will be re-evaluating whether
        ;   to retain (continue to pursue) this predictive schema,
        ;   or to switch to another.

Synthesizing
    no move-leg and no leg-shock <--- no move-leg, no leg-shock, and tone
    no move-leg and no leg-shock
    no move-leg and no leg-shock
    move-leg and leg-shock

        ; ENACT performs any efferent actions in the next event
        ;   specification.  Here the only efferent is to not move the leg.

Enacting
    relaxing move-leg                          ; Don't move the leg yet.

-- 92 --
Detecting
    no move-leg, no leg-shock, and tone        ; No shock yet.
Reminding
    of leg-shock in s-2, s-14, s-12, s-8, and s-0
    of move-leg in s-2, s-13, s-10, and s-0
Activate retaining previous schema as CPS.

        ; ACTIVATE continues to prefer its choice of schema, since
        ;   the CPS predicts the most significant result, has the
        ;   most favorable outcome, and is a frequently occurring schema.
        ;   (If ACTIVATE had changed its selection of a CPS, then
        ;   SYNTHESIZE would begin following a new sequence of event
        ;   specifications and the leg movement would be improperly timed).
```

```
Synthesizing
    no move-leg and no leg-shock
    no move-leg and no leg-shock <--- no move-leg, no leg-shock, and tone
    no move-leg and no leg-shock                ; Another successful
    move-leg and leg-shock                      ; SYNTHESIZE match.
Enacting
    relaxing move-leg                           ; No efferent action yet.

-- 93 --
Detecting
    no move-leg, no leg-shock, and tone         ; Still no shock.
Reminding
    of leg-shock in s-2, s-14, s-12, s-8, and s-0
    of move-leg in s-2, s-13, s-10, and s-0
Activate retaining previous schema as CPS.      ; Still sticking with
                                                ;  this schema.
Synthesizing
    no move-leg and no leg-shock
    no move-leg and no leg-shock
    no move-leg and no leg-shock <--- no move-leg, no leg-shock, and tone
    move-leg and leg-shock

        ; SYNTHESIZE anticipates the onset of the shock.  The next event
        ;   specification which is passed to ENACT specifies that the leg
        ;   should be moved.

Enacting
    attempting to move-leg                      ; Move the leg now.

-- 94 --
Detecting
    move-leg, leg-shock, and tone

        ; The shock did occur when CEL-TIC predicted it would, the subject's
        ;   leg is moved: a conditioned response.

        ; The CEL operator NOTICE tags events with a hedonistic value.  It
        ;   notices the occurrence of the motivationally significant feature
        ;   'shock' and marks it as undesirable.  NOTICE also triggers
        ;   COLLECT; see next time tic from now (95, below).

Noticing
    leg-shock (which has a poor hedonistic value)
Reminding
    of leg-shock in s-2, s-14, s-12, s-8, and s-0
    of move-leg in s-2, s-11, s-0, s-2, s-13, s-10, and s-0
    of tone in s-2, s-9, and s-0
```

```
        ; A few new schemata are retrieved.  These are reminded by the shock
        ;   and leg movement.  Note that the tone is now a possible result.
        ;   CEL-TIC does not a priori determine which features are cues and
        ;   which are results.

Activate retaining previous schema as CPS.
Synthesizing
    no move-leg and no leg-shock
    no move-leg and no leg-shock
    no move-leg and no leg-shock
    move-leg and leg-shock <--- move-leg, leg-shock, and tone

        ; The CEL operator REINFORCE is triggered by SYNTHESIZE since all of
        ;   the event specifications in the CPS matched sequential events in
        ;   the external world.  REINFORCE will strengthen the CPS in
        ;   long-term memory.

Reinforcing CPS (s-12).

        ; Because the INDEX operator was triggered by REINFORCE, in addition
        ;   to strengthening the CPS in long-term memory, it will generate
        ;   variations in response and store them as well.

Indexing s-12.

-- 95 --
Detecting
    move-leg, leg-shock, and tone
Noticing
    leg-shock (which has a poor hedonistic value)
Reminding
    of leg-shock in s-20, s-2, s-14, s-12, s-8, and s-0
    of move-leg in s-2, s-11, s-0, s-21, s-2, s-13, s-10, and s-0
    of tone in s-2, s-9, and s-0
Activate retaining previous schema as CPS.

        ; The CEL COLLECT operator packages event specifications drawn from
        ;   short-term memory into a schema for subsequent placement into
        ;   long-term memory.

Collecting new schema s-22
    no move-leg, no leg-shock, and no tone
    no move-leg, no leg-shock, and no tone
    no move-leg, no leg-shock, and no tone
    no move-leg, no leg-shock, and tone
    no move-leg, no leg-shock, and tone
    no move-leg, no leg-shock, and tone
    move-leg, leg-shock, and tone
```

```
    move-leg, leg-shock, and tone
Synthesize already finished following CPS.
Indexing s-22.

-- 96 --
Detecting
    move-leg, no leg-shock, and no tone
```

.
.
.

```
; After a number of trials, CEL-TIC reaches asymptotic latency
;   occurring between a simultaneous response and one immediately
;   preceding the unconditioned stimulus (the shock).  The
;   interaction of the CEL operators ensures that unpredictive
;   schemata are not shortened.
```

4.5.3 Performance of the program

Figure 6 shows the performance of the CEL-TIC program appropriately scaled[8] and compared to the data of Wahlstein and Cole [1972]. The simulated dog, like the real dogs used by Wahlstein and Cole, shortens its response latency if and only if the shock can be avoided. However, the computer program shortens its response at a steady rate, whereas Wahlstein and Cole observed a sudden decline in response latency among instrumentally conditioned dogs. (This is verbally reported by Wahlstein and Cole [1972], but not recorded in their actual graphs – hence it also does not appear in figure 6, which is taken directly from their graphs). Assuming their verbal report is accurate, then CEL-TIC's steady response shortening would seem to indicate that the simple strategy of removing events at random from memory traces, while sufficient to achieve the correct end result, may not adequately capture the process by which dogs learn to respond more

[8]It is necessary to set parameters in the computer program in such a way that its performance changes over far fewer trials than real animals. The computing resources needed to simulate an animal learning at a normal rate, even in a vastly simplified model, would be prohibitive.

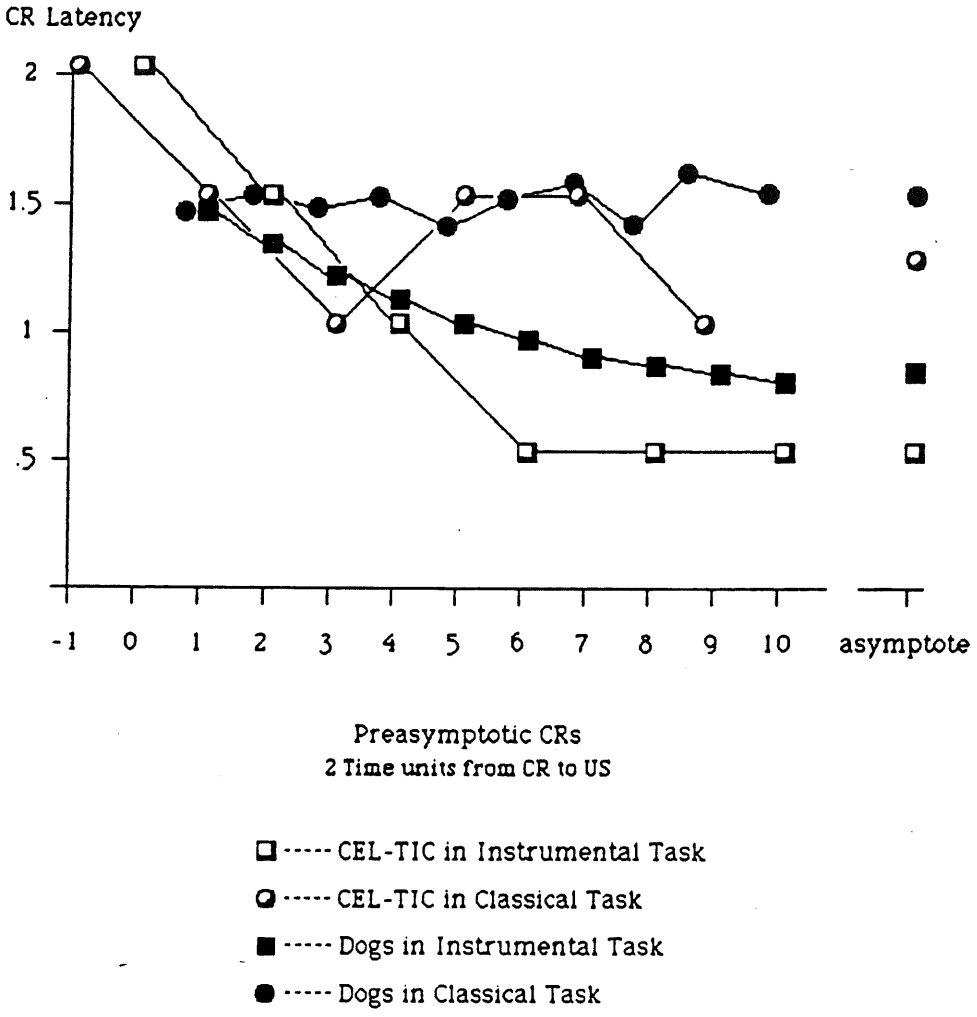

Figure 6: Performance of CEL-TIC compared to experimental data.

quickly in instrumental learning situations. We suspect that this discrepancy may be due to our still rather weak representation of time, which is implicit in the ordering of events. An explicit representation of time would allow experimental variation in timing to occur more suddenly, rather than slowly by of incremental experimentation with schema size.

5 Related Work

Each development and refinement of the CEL framework is driven by an attempt to accurately model experimental data. The models of contingency and response latency described in this chapter were motivated by results from animal behavior experiments. Animal learning theorists, notably Rescorla and Wagner [1972], Wagner [1978], Mackintosh [1975], and Pearce and Hall [1979], have proposed models of associative learning which account for contingency. These theories predict the strength of associative learning as a function of experience, but do not describe the detailed processing necessary to form and modify associations. Our model is intended to supply this further level of detail. The current computer implementations of the model (CAP-CEL and CEL-TIC) are most closely related to the Rescorla-Wagner model, since changes in attention to the conditional stimulus is not modeled, but more aspects of the other theories may be adopted as CEL is extended to model a greater range of associative learning phenomena.

Wagner [1981] has proposed a memory model, SOP, which accounts for a variety of associative learning phenomena. SOP is a nodes-and-links model with spreading activation, but with the interesting property that every memory node is always in one of three distinct states and transition from state to state is stochastic. Since there is no representation in SOP for *sequences* of events, it is not clear how problems like response latency could be explored in SOP. On the other hand, the SOP memory model solves

some problems of spreading activation networks (for instance, the problem of infinite reverberation in a positive feedback loop) in a novel and elegant manner; SOP will certainly influence the further development of CEL's memory structure.

Production systems can also be used as a framework for describing the processing underlying complex behavior. The most comprehensive production system model of learning to date is the ACT* family of programs developed by John Anderson [Anderson 1983]. ACT* creates new production rules through processes of composition, proceduralization, generalization, and discrimination. Of these, generalization and discrimination address the problem of discovering which features are relevant for determining when an operator should be applied. ACT* creates a generalized rule by omitting a condition from the antecedent part of another rule. Discrimination adds a new clause either to the antecedent or to the consequent part of a rule. Rules in ACT* are strengthened when they are reinvented and when they are activated through the spread of activation in memory. They are weakened by negative feedback.

The ACT* framework has been used to account for a wide variety of data from the psychology of human learning. The scheme for strengthening and weakening rules, however, does not appear to be consistent with the basic psychological data concerning contingency. The processes of composition and proceduralization provide a model of performance gains through practice, but do not address the question of learning when to respond, i.e., response latency and temporal coding.

In contrast to the algorithm level descriptions of CEL, SOP, and ACT*, there are a set of 'neurally inspired' models (e.g. [Rumelhart and Zipser 1985], [Anderson and Hinton 1981], [Feldman 1981], [Kohonen et. al. 1981]) that address implementation-level problems of learning and memory. These models, although suggestive and intriguing, are still quite limited in their scope. For example, none of the models is capable of accounting for any sequences of events; only static phenomena (e.g., reconstructive recognition

of single states) have so far been studied, and the current approaches clearly do not 'scale up' in any clear way to the problem of sequences of states or events. Furthermore, these models will not be a substitute for algorithm-level descriptions, but rather a supplement. The models "won't do much unless programmed by a clever person.... What any particular network can do is dependent on its structure" [Rumelhart and Zipser 1985]. This is precisely because these 'programmed' network structures simply correspond to network-level encodings of algorithms. Hence, the algorithm level is still the most concise and expressive way to state an algorithm; the fact that they can also be stated at the implementation level is a useful supplement, and of course will have to eventually be part of any truly complete theory.

Finally, it is misleading to imagine that network models are somehow more accurate models of actual brain phenomena. Since many aspects of neural functioning are only poorly understood, and since in any case it would be prohibitively expensive to simulate real neurons in detail, such models typically simulate simple quasi-neurons which, it is hoped, capture enough of the essential characteristics of real neurons to shed light on the way algorithms must be implemented in the brain. Such explorations are interesting, and may eventually provide valuable constraints for models at the algorithmic level. At present, though, the connection between 'neurally inspired' models and real brains is too tenuous for the former to provide much guidance.

6 Conclusions

6.1 The CEL architecture

The twelve functional modules of the CEL architecture provide a framework for the characterization of computationally-necessary modules of the storage, retrieval and modification of behavior. Within this framework, specific algorithms can be proposed for

the computation of particular known constraints in learning and memory. CEL is not the only architecture proposed for this purpose (production systems being the principal alternative). We intend the CEL architecture, like any other, to be judged by how well it captures constraints from both the computational (top-down) and implementation (bottom-up) levels simultaneously; how well it predicts interaction effects among algorithms; and how useful it is as a language for the statement of problems at the algorithm level, without either obscuring important high-level questions or forcing premature implementation decisions based on details of the substrate.

Each functional module represents a specific necessary information-processing function that must be performed during learning. For example, a comparison of what was predicted to happen (e.g., from some action) against what actually occurred in the outside world must be computed by some algorithm. In CEL, this necessary comparison function is represented by the SYNTHESIZE module, and the function results in a new necessary distinction between two types of working memory: a 'retrieval-side' working memory store (which we've termed intermediate-term or reconstructive memory) and a 'recognition-side' working memory store (temporary or short-term memory), since these represent the two stores that must be compared against each other in order to test predictions. Other architectures such as production systems are often missing these crucial components of an information-processing architecture of learning and memory (Anderson's [1983] ACT*, for instance, has no provision for comparison of predictions against actual occurrences, and hence no provision for modifications to existing memories based on mismatches.)

Each of CEL's modules also conforms to emergent properties of low-level memory interactions. (For example, REMIND denotes the spread of activation through memory according to particular algorithms; ACTIVATE denotes inhibition and competition among 'competing' activated patterns of memory, etc.). We make explicit our current

model of the indexed organization of nodes and links in memory (though this is evolving with our ongoing research), and we show how, for instance, spread of activation occurs, how tasks that typically appear serial in standard symbolic models may be implemented in parallel via CEL's memory. Hence, CEL should be capable of representing the same algorithms that would emerge from any straight nodes-and-links level description (e.g., [Hinton and Anderson 1983]) – there is no loss of specificity, since CEL-level algorithms emerge as consequences of nodes-and-links operation; however, the CEL algorithm-level architecture has the advantage of providing expressive descriptions of these algorithms without the details of the implementation level intruding.

6.2 Making implicit constraints explicit

Algorithmic interpretations of empirical results in neurobiology must conform both to 'top-down' constraints of behavioral data (i.e., they must compute the right thing) and to 'bottom-up' constraints of the substrate (e.g., they must be implementable in parallel networks, they must conform to the modularity of the brain, etc.). All experimental research in neurobiology is (at least implicitly) guided by some top-down hypotheses. At their simplest, these hypotheses can simply be statements like 'there are long-lasting changes in behavior during learning (the computational level), so there must be an alteration mechanism (algorithm level) for long-lasting changes in the brain (implementation level). Further constraints, of course, are provided by data on duration, or just how long-lasting these changes in behavior might actually be; onset, or how long the changes take to happen in the first place; and extinction, i.e., under what circumstances the changes might go away.

Indeed, research on the neurobiology of learning and memory went for a long time in search of a mechanism, any mechanism, that would roughly satisfy these constraints. This search for plausible implementation-level mechanisms for known computational

and algorithmic constraints is necessary for a complete theory of learning and memory. However, even the duration, onset and extinction elaborations of the basic necessary 'long-lasting trace' computation do not address any problems of *interactions* among computations. Even at this rough, almost pretheoretic level, there are factors that are known to be basic computational constraints, such as how a long-lasting trace might be stored in such a way as to be *retrievable* exactly when it is appropriate at some later time, how *sequences* of such long-lasting traces might be stored, and how *generalizations* of these traces might arise. It may still be premature to ask these questions at the implementation level; the field of experimental learning and memory is not yet ready to answer them. Nonetheless, ongoing computational and algorithmic work may serve the useful purpose of providing successively more refined and explicit predictions of necessary computations, which can help direct the search through the substrate.

6.3 Contingency vs simple strengthening of associations

The importance of contingency is well known in animal learning theory, and the extensive experimental data concerning contingency provide a clear set of computational requirements for a process model of learning. We have expressed this set of requirements algorithmically as a *salience assignment* problem, and we have shown how this problem is solved within the CEL framework, via formulae based on Bayes' algorithm. It is worthwhile to note that without the computational level being explicitly separated from the algorithmic, an approach may be elegant and elaborate, and yet *compute the wrong thing*; this happens to be true of ACT*, which simply does strengthening based on number of successful applications of productions [Anderson 1983, p.24].[9]

[9]Although it may seem as though we are picking on shortcomings of Anderson's ACT* system, it should be noted that ACT* is a well-worked-out system with many explicit hypotheses and many good attempts to account for existing data and predict new results; we applaud the depth of this

One of the advantages of a processing framework like CEL is that we can test our hypotheses by building computer models that actually perform the tasks we are attempting to account for. The CAP-CEL program is such a model, and it shows that the account of we have offered is in fact adequate to distinguish useful, predictive cues from context and uncorrelated cues. Moreover, the performance of CAP-CEL, documented in this chapter, comprises a set of detailed predictions of our hypotheses that may be confirmed or rejected on the basis of experiments.

6.4 Response latency in instrumental and classical conditioning

Since the CEL model divides learning and memory into individual functional modules, it does not explicitly provide separate processes for classical as opposed to instrumental tasks. Hence, it is incumbent upon us to explain how differences between learning in these situations can be accounted for by the same set of processes in the CEL framework. One such difference is in response latency: an animal tends to respond earlier in an instrumental conditioning situation than in a classical conditioning situation. We have shown that, provided there is some variation in the animal's response (and particularly, some variation in response timing), the difference between response latencies in these two situations arises naturally out of the interactions of the CEL operators. One way to achieve experimental variation in response is to alter traces in long term memory. We have built a computer model, CEL-TIC, which removes events at random from long term memory traces; CEL-TIC displays the difference in response latency described above.

Furthermore, because our algorithms are presented in the unified framework of CEL,

work, and acknowledge that it is in part this depth that makes ACT* vulnerable to attack for its shortcomings. Many other proposals are simply not worked out well enough to have specific hypotheses that can be tested, as does ACT* (and CEL).

we are able to make pointed hypotheses accounting for otherwise problematic differences in results in the experimental literature on response latency (see e.g., [Mackintosh 1983]). As described in section 4.4.4, a new and plausible hypothesis about these apparently conflicting data arise from the model presented here.

6.5 Future work

There is a wealth of empirical behavioral data waiting to be accounted for. We intend to continue to concentrate on basic phenomena of animal learning rather than following the current AI and cognitive science fashion of building computer models of complex human problem solving tasks, since we believe these basic phenomena shed more light on the fundamental properties of learning in humans as well as animals. We have begun work on the phenomenon of blocking in associative learning [Kamin 1968, 1969] and on habituation and latent inhibition [Mackintosh 1983].

Some important aspects of learning and memory have thus far been omitted from the CEL architecture. The inadequacy of our discrete representation of stimulus features is demonstrated by behavioral data showing a gradient response curve over a continuum of similar stimuli [Hintzman 1978]. Also, since decay of memory traces is not a part of the CEL architecture at this time, we cannot accurately model timing effects in extinction, spontaneous recovery, nor can we account clearly for the partial reinforcement effect. These and other properties of memory must eventually be incorporated into the architecture.

References

1. Amsel, A. 1972. Behavioral habituation, counter-conditioning, and a general theory of persistence. In *Classical Conditioning II: Current Research and Theory*, A. H. Black and W. F. Prokasy (eds), New York: Appleton-Century-Crofts.
2. Anderson, James A. 1981. Models of information processing in the brain. In *Parallel Models of Associative Memory*, G. E. Hinton and J. A. Anderson, New Jersey: Lawrence Erlbaum, 9–48.

3. Anderson, John R. 1983. *The Architecture of Cognition.* Cambridge: Harvard University Press.

4. Bower, G. H. and Hilgard, E. R. 1981. *Theories of Learning.* Englewood: Prentice-Hall.

5. D'Amato, M. R., Fazzaro, J., and Etkin, M. 1968. Anticipatory responding and avoidance discrimination as factors in avoidance conditioning. *Journal of Experimental Psychology* **77**, 41–47.

6. Duda, R. O., Gaschnig, J. G., Hart, P. 1979. Model design in the Prospector consultant system for mineral exploration. In *Expert Systems in the Micro-electronic Age*, D. Michie (ed). Edinburgh: Edinburgh University Press, 1979.

7. Feldman, J. A. 1981. A connectionist model of visual memory. In *Parallel Models of Associative Memory*, G. E. Hinton and J. A. Anderson (eds), New Jersey: Lawrence Erlbaum, 1981, 49–82.

8. Granger, R. H. 1982. Identification of components of episodic learning: The CEL process model of early learning and memory. *Cognition and Brain Theory* **6**, 5–32.

9. Granger, R. H. 1983. An artificial intelligence model of learning and memory that provides a theoretical framework for the interpretation of experimental data in psychology and neurobiology. Department of Computer Science Technical Report 220, University of California, Irvine.

10. Granger, R. H. and McNulty, D. M. 1984. Learning and memory in machines and animals: An AI model that accounts for some neurobiological data. *Proceedings of the 6th Annual Conference of the Cognitive Science Society*, Boulder, Colorado.

11. Hall, G. and Pearce, J. M. 1979. Latent inhibition of a CS during CS–US pairings. *Journal of Experimental Psychology: Animal Behavior Processes* **5**, 31–42.

12. Hintzman, D. L. 1978. *The Psychology of Learning and Memory.* San Francisco: W. H. Freeman and Company.

13. Kamin, L. J. 1968. Predictability, surprise, attention, and conditioning. In *Miami Symposium on the Prediction of Behavior, Aversive Stimulation*, M. R. Jones (ed), Coral Gables, Florida: University of Miami Press.

14. Kamin, L. J. 1969. "Attention-like" processes in classical conditioning. In *Punishment and Aversive Behavior*, B. A. Campbell and R. M. Church (eds), Conference on Punishment, Princeton, NJ, 1967. New York: Appleton-Century-Crofts.

15. Knuth, D.E., *The Art of Computer Programming, vol. 1, Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1968.

16. Kohonen, T., Oja, E. and Lehtiö, P. 1981. Storage and processing of information in distributed associative memory systems. In *Parallel Models of Associative Memory.* G. E. Hinton and J. A. Anderson (eds), New Jersey: Lawrence Erlbaum, 105–144.

17. Kolmogorov, A.N., and Uspenskii, V.A., *On the definition of an algorithm*, AMS Transl.2nd ser., vol. 29, 1963, pp. 217-245.

18. LoLordo, V.M., Constraints on learning. In Bitterman, M.E., LoLordo, V.M., Overmier, J.B. and Rashotte, M.E. *Animal Learning: Survey and Analysis.* New York: Plenum Press, 1979, 473-504.

19. Mackintosh, N. L. 1975. A theory of attention: Variations in the associability of stimulus with reinforcement. *Psychological Review*, **82**, 276–298.

20. Mackintosh, N. L. 1983. *Conditioning and Associative Learning.* New York: Oxford University Press.

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

22. McNulty, D. M. and Granger, R. H. 1985. A computer model of elements of latent learning in a simulated maze environment. Department of Computer Science Technical Report 85-03, University of California, Irvine.

23. McDermott, J. and Forgy, C.L., Production System Conflict Resolution Strategies. In D.A. Waterman and F. Hayes-Roth (Eds.), *Pattern-Directed Inference Systems*, New York: Academic Press, 1978, 177-199.

24. Pearce, J. M. and Hall, G. 1980. A model for Pavlovian learning: variations in the effectiveness of conditioned but not of unconditioned stimuli. *Psychological Review* **87**, 532–52.

25. Rescorla, R. 1966. Predictability and number of pairings in Pavlovian fear conditioning. *Psychonomic Science* **4**, 383–384.

26. Rescorla, R. 1967. Pavlovian conditioning and its proper control procedures. *Psychological Review* **74**, 71–80.

27. Rescorla, R. 1968. Probability of shock in the presence and absence of CS in fear conditioning. *J. Comparative and Physiological Psychology* **66**, 1–5.

28. Rescorla, R. 1980. *Pavlovian Second-Order Conditioning: Studies in Associative Learning.* New Jersey: Lawrence Erlbaum Associates.

29. Rescorla, R. and Wagner, A. R. 1972. A theory of Pavlovian conditioning: Variations in the effectiveness of reinforcement and non-reinforcement. In *Classical Conditioning II: Current Research and Theory*, A. H. Black and W. F. Prokasy (eds). New York: Appleton-Century-Crofts, 1972.

30. Rumelhart, D.E. and Zipser, D. Feature Discovery by Competitive Learning. *Cognitive Science*, 1985, to appear.

31. Sheffield, F. D. 1965. Relation between classical conditioning and instrumental learning. In *Classical Conditioning*, W. F. Prokasy (ed), New York: Appleton Century Crofts, 302–322.

32. Schöhage, A. Storage Modification Machines. *SIAM J.Comput*, vol. 9, No. 3, 1980.

33. Thorndike, E.L. 1911. *Animal Intelligence: experimental studies.* New York: Macmillan.

34. Wagner, A. R. 1981. SOP: A model of automatic memory processing in animal behavior. In *Information processing in animals: memory mechanisms*, N. E. Spear and R. R. Miller (eds), Erlbaum, Hillsdale, NJ, 5–47.

35. Wahlsten, D. L. and Cole, M. 1972. Classical and avoidance training of leg flexion in the dog. In *Classical Conditioning II: Current Research and Theory*, A. H. Black and W. F. Prokasy (eds), New York: Appleton-Century-Crofts.