Expert control for a robot body

John H. Andreae, senior member
Bruce A. MacDonald, member

ABSTRACT

Mobile robots with dextrous hands and sophisticated sensory systems will require intelligent, knowledge-based, expert controllers. In this paper we develop a design for a robot controller which can acquire task knowledge as it interacts in the world with its human users. The design is based on four reasonable assumptions which lead us to a theoretical framework for robot learning systems. The framework is called a multiple context learning system. It is a production system with multiple templates for forming productions as the system interacts with the world. The paper discusses elaborations of the framework and experimental tests of the system.

I. INTRODUCTION

In the future, robots will have sophisticated sensory systems, dextrous hands, and mobility. These robots will need intelligent, knowledge-based expert controllers. Most robots in use at the present time are fixed to a factory floor. They are flexible machine tools carrying out a limited range of tasks in a limited environment. Since they are often used by people untrained in programming, manual techniques have been developed to enable an operator to "lead" them through tasks [1,2,3]. However, as new facilities have been added to the robots in the form of simple computer vision and other sensors, more advanced programming methods have been introduced and special high level programming languages have been developed [4,5]. The factory robot will perform increasingly complex tasks as the art and science of robot programming advances.

Mobility is going to change future robots in a dramatic way. They will come out of the factories to share the environment of ordinary living and working humans and will be available to do tasks for humans in the home, office, street and elsewhere. They will be instructed and commanded by humans who have no special training. The robots will need expert controllers that cope with complex tasks and understand the rich sensory information to be gleaned from their environmental sensors. Brooks [6] describes in more detail the complexity mobile robots must face in the real world, and the resulting difficulties in designing them, and these difficulties are illustrated in [7-10] where current advanced mobile robots are described. Minsky [11] has distinguished "expert control", which is based on knowledge, from "adaptive control" and classical "feedback control". The controller of a mobile robot will have much in common with an "expert system" [12] and it will inherit many of the problems associated with expert systems. In particular, it will have to be able to

acquire new knowledge about new tasks in new environments. Knowledge acquisition is recognized to be a key research problem in the development of expert systems [13-15], and it will be no less of a problem for the expert controller of a mobile robot. It will have to acquire much of its knowledge from the ordinary humans who need its help to do the tasks they know how to do themselves.

The expert controller will have to acquire the knowledge for some tasks from its users, since the designers of a robot cannot anticipate all the tasks it may have to perform, nor all the environments in which it may have to perform its tasks. We envisage that many of the tasks performed by the mobile expert robot will be ones that humans can do themselves, such as household chores. Unfortunately, humans cannot specify explicitly how to perform tasks, even when they themselves perform the tasks regularly [1,2,16,17]. Most of our knowledge of skills is tacit or subconscious [18]. To instruct other people in a skill, we give examples or demonstrations, add verbal explanations and then, when the pupil attempts the task, point out inadequacies. The expert controller will need to acquire task knowledge from the examples, demonstrations and explanations provided by its human users.

We see the development of a robot as occurring in three stages.

General framework: First, in this paper we develop a design for the general framework of a robot controller---a sort of robot controller shell---which can acquire task knowledge as it interacts in the world with its human users. The design is based on four reasonable assumptions which lead us to a theoretical framework for robot learning systems. The result can be seen as a production system---which we discuss here---or as a neural net [19-21]. The paper also discusses elaborations of the framework and experimental tests of the system carried out to date. This design would be implemented by a programmer, without regard to specifics of any robot body.

Body: Second, a particular robot body would be designed and built, including a set of input and output channels through which the body and controller communicate. At this stage the body designer must also specify a set of "production templates", which make the general controller form associations appropriate to the body. The nature of these templates is explained in detail in section II and examples are given in section IV.

Teaching: Third, the users or teachers of the system will provide environments and tasks within the capability of the robot, and they will actively help it interact with its environment to acquire expertise and skills for those tasks.

We begin by making four assumptions---about (a) input and output, (b) memory, (c) internal structure and (d) performance---and giving our reasons for making them. The reasons for all but the third come from the classic work of Newell and Simon [22], where the characteristics of human information processing are expounded.

A. Motivation for Four Assumptions

Figure 1 depicts the general form of a controller in a robot body.

Input and output: each input and output will be treated in the same general manner in the design of the controller, that is they will be **Homogeneously Encoded** [22].

Since we have argued that the designer of an expert controller cannot anticipate all the tasks and environments which the robot might encounter in the future, we must isolate the designer from such information. Homogeneous encoding does this by forcing the controller designer to treat all input channels (see section II.*A*) alike, and all output channels alike.

This has the additional consequence of isolating the designer from particular characteristics of robot bodies, so we end up with a general, robot-independent design for the controller, a sort of expert controller shell. Specific robot body characteristics are used in the second stage of development, as explained immediately above. In the third stage of development the teachers of the robot must deal with environments and tasks.

Memory: the controller will have both **Short Term Memory (STM) and Long Term Memory (LTM).**

This second assumption recognizes that the expert controller must have two forms of memory, a small, transient working memory (STM) and a vast storage memory (LTM). Most computer systems have memories of these two kinds, like the working variables and databases of programs. In Psychology, human information processing is seen as requiring these two sorts of memory, the one to represent the "here and now" and the other to hold knowledge [22].

Internal structure: the controller will have the structure of a **State-output Automaton.**

The state-output automaton assumption does little more than require the system to have inputs and outputs, which we know it must have, and state, which is equivalent to the memory it must have for the second assumption. The fact that there must be state-transition and state-output functions for a state-output automaton imposes no particular relationships on the inputs, states and outputs. The state-output automaton is, strictly, a finite state automaton [23], but we shall see in section IV that the resulting system, with its vast memory, lacks neither computational power nor symbolic processing ability.

Performance: The controller will learn by **Incremental Learning.**

This final assumption is that the system acquire knowledge in small pieces or "chunks" [22]. The controller must be able to learn in many different ways, from the interactions of its robot body with the world and from the direct influence of human instructors or teachers. When the system has

learned most of a behavior or task, it must be possible to teach it missing parts and to modify existing parts. The chunks must be small and learnable in many ways.

II. A FRAMEWORK FOR AN EXPERT CONTROLLER

A. Inferring a Multiple Context Learning System

The state-output automaton, specified in our third assumption, is characterized by three sets---input, output, and state---and two functions---the state transition function, and the state-output function---defined on these sets. It is to be understood that the input and output sets are sets of general vector quantities. Each member represents events occurring on a number of input and output **channels**; an *n* element vector representing the events on *n* channels. Each input and output event will be unstructured, and treated as indivisible by the controller. Each channel will provide events from a particular subset of the possible events, such as the set of foveal visual events, or the set of arm movement events, or the combined set of foveal visual events and hand touch events. As this illustrates, the same event can occur on more than one channel. The nature of the inputs and outputs will become more clear in section II.*B*.

The state transition function, f_1, generates a new state from the current input and state:-

$$\text{new state} = f_1(\text{input, state}) \qquad (1)$$

The state-output function, f_2, generates a new output from the current state:-

$$\text{output} = f_2(\text{state}) \qquad (2)$$

At this point, we introduce the second assumption that the memory of the system comprises STM and LTM. Since the state of a system *is* its memory, we can write

$$\text{state} = (\text{STM, LTM}) \qquad (3)$$

which can be introduced into (1) and (2) to give (4), (5) and (6) with three functions suitably derived from f_1 and f_2 (see Figure 2):-

$$\text{new STM} = f_3(\text{input, STM, LTM}) \qquad (4)$$

$$\text{new LTM} = f_4(\text{input, STM, LTM}) \qquad (5)$$

$$\text{output} = f_5(\text{STM, LTM}) \qquad (6)$$

Equation (6), which says that the output can be derived from any information in STM or LTM, must be restricted because LTM has been assumed to be vast. We cannot within a short time process information from the whole of LTM. Only a minute subset of LTM can be accessible at any time. Rejecting the possibility (a) that this subset is fixed (large parts of LTM would never be used)

and the possibility (b) that this subset is random (LTM would be accessed without concern for its relevance to the "here and now", i.e. to STM), we assume that the accessible subset of LTM is a function of STM, so that LTM is some kind of associative memory, and outputs information associated with the current STM. In other words, the information derived from LTM is coming from those parts of LTM associated in some way with STM. It will be convenient to treat this information as a special output coming from LTM that is a function of STM, so we will call it $LTM|_{STM}$ (See Figure 3). For the moment, we will do no more than note that equation (6) can now be interpreted as saying that the output can be derived from STM and $LTM|_{STM}$.

A similar argument can now be applied to equation (4). Again the contribution of LTM to the change in STM can only be via the restricted access, which we have called $LTM|_{STM}$. The combination of these special outputs from LTM with the input to the automaton is called **input***, so that equation (4) can now be rewritten with a suitable new function:-

$$\text{new STM} = f6(\text{input*}, \text{STM}) \qquad (7)$$

STM is a small memory being updated continually to represent the "here and now". The components of input* (called **events**) must be treated in an unstructured way because of the homogeneous encoding assumption. These two conditions compel the view that equation (7) describes STM as derived from the contents of a pushthru or FIFO store, shown in Figure 4, which is updated by pushing events into one end of the store, while the oldest events are pushed out at the other end. In other words, STM is updated from the information in a collection of **shift registers** into which events are shifted. That information is the set of events in the shift registers together with their inter-relationships. The longer the shift registers, the greater the possible span of STM. Call the set of positions in the shift registers the **span-set**. Each subset of the span-set is called a **context template** and it defines a distinct combination of the positions in the shift registers. The set of events in the shift register positions specified by a context template is called a **context**. The set of all context templates is the power set (set of all subsets) of the span-set. The corresponding set of contexts is the maximum information that can be extracted from the events at the shift register positions, since the homogeneous encoding assumption prohibits structural knowledge about the events. If the controller used all of the context templates derived from the power set of the span-set, most of them would be redundant, so we postulate that the controller's set of context templates will be a restricted subset of the power set and call this subset the **multiple context**. So STM is the set of contexts currently specified by the templates of the multiple context. The selection of a particular multiple context is *not* part of the general controller design; we discuss this in the next paragraph.

At the second stage of the robot's development (see section I.), the body designer must carefully select the multiple context, which allows the controller to make certain associations among

inputs and outputs. The multiple context specification is a robot-dependent part of the controller and we have just seen that the controller design can be isolated from particular multiple contexts, although the *notion* of multiple context is central. We expect that the choice of a particular multiple context will be important for the controller's optimum exploitation of a particular robot body. However, the multiple context design will be somewhat *ad hoc,* depending on the particular characteristics of the robot body, just as the robot body design cannot be entirely specified by general principles, but depends on the requirements of the robot and on the implementation technology. Research has not yet reached the stage of providing guidelines for the choice of a multiple context.

The arguments about access to LTM that were used to change equation (4) also apply to equation (5). Since LTM is a vast memory, it is impracticable to allow the updating of LTM in equation (5) to involve the scanning of large parts of LTM. Imagine an arbitrary operator that can act locally on one or several parts of LTM. If this operation is, say, a simple addition, marking or replacement of elements in LTM, then computation time will be limited and the updating in equation (5) will be possible in real time. Using the symbol ✚ for the local operator, equation (5) can be rewritten:-

$$\text{new LTM} = \text{LTM} \boldsymbol{\oplus} f_7(\text{input, STM}) \qquad (8)$$

But what is it that has to be operated on locally in LTM? The most general form of f_7, assuming homogeneous encoding, would allow any association between any component of the input vector and any context in the multiple context of STM. If each component of the input vector is assigned a different input channel, on which it arrives, then the most general function f_7 would be the Cartesian product of context templates and input channels. (The Cartesian product is the set of all pairs comprising one context template and one input channel.) No extra generality would be achieved by associating *sets* of input channels with context templates, because composite events can be defined as additional components arriving on extra channels. A (context template-input channel) pair will be called a **production template**. The (context-input event) pair specified by a production template at some moment in time will be called a **production**. The robot body designer will have to specify the production templates along with the multiple context, and again only a few combinations will be required as there will be considerable redundancy in having all contexts associated with all input events.[1]

If the operator ✚ introduced above is a simple addition, marking or replacement of elements in LTM, then these elements are productions and we must conclude that LTM is a store of productions. The productions will be independent of each other and LTM will grow incrementally as required by the incremental learning assumption. The chunks of information learned by the system will be productions.

The output equation (6) has already been partly interpreted. We argued above that the output can be derived from STM and from the special output from LTM, called $LTM|_{STM}$. We have not found any virtue in transferring the contents of STM to output, so we restrict our attention to $LTM|_{STM}$ and the way this special output from LTM is associated with STM. The contexts of STM are associated with those productions of LTM which have the same contexts. If nothing special is assumed about the contexts, or the inputs stored with those contexts as productions (the homogeneous encoding assumption again), then there is really no other way of interpreting equation (6) than as a process identifying those productions in LTM which have the same contexts as are currently in STM. So LTM is an associative memory, where STM provides the current associations. For each context, the set of input events stored with that context in productions in LTM will be called the **predictions** from that context. Equation (6) becomes:-

output = predictions from contexts in STM (9)

Why do we call them predictions? Each production records the occurrence of a certain event following the occurrence of a certain context. On the simple basis of "what has occurred in the past may occur again", we can expect the re-occurrence of a context to be followed by the re-occurrence of one of the events stored with that context in productions. Note that the input set and output set of our automaton must be the same, since productions store predicted inputs, which are then used as outputs. The controller remembers its experienced input events, and outputs predictions of future events based on this past experience.

We have shown that the four assumptions convert the general description of a state-output automaton into a production system of a definite form. However, the significance of the inputs to, and outputs from, this production system must be clarified, as promised at the beginning of this section.

B. Control Policy and World Model

Inputs from the robot body to the expert controller will provide information about what is going on in the robot body (Figure 3). This information will include stimuli that are the result of pre-processing signals from the robot sensors, actions that are sent, after pre-processing, to the effectors of the robot body, and various internal stimuli and actions, from and to the robot body, relevant to the expert controller (e.g. state of batteries, readings from balance gyros, temperature and stress indications, and thermal controls.)

Since the outputs in equation (9) are predictions of input events, and we have just seen that the input events are a variety of external and internal stimuli and actions, it follows that the outputs from the production system are also actions and stimuli. The production system predicts what actions to command and what stimuli to expect. In other words, it provides a **control policy** for the

expert controller and a **world model** to enable the system to anticipate how the world will react to its actions. This is enough to enable the system to make **plans** by a process of **forward chaining** similar to that used in expert systems [12]. Before describing how this can be done, we should close this section on justifying our proposed framework for a **multiple context learning system** (MCLS). What is to come is a collection of ways in which we have taken this framework, extended it, tested it and found it powerful. This next part is somewhat speculative and will be treated in less detail. In the next section we describe briefly how the framework has been developed and elaborated [24]. After that, examples are given of the operation of the MCLS and of its extensions.

III. EXTENSIONS TO THE MCLS FRAMEWORK

A. Plans and the There and Then

Most of our own thinking is not in the here and now. We do a lot of imagining. What should we do if such and such happens? Should we have done such and such when ... ? This should enable us to ... . Why did so and so do such and such? Rather than being in the here and now, such thinking is elsewhere in the past or future, the "there and then". Our expert controller may not need such a variety of modes of thought, but it will certainly have to plan paths and trajectories. It will have to consider the *effects* of planned actions, i.e. it will have to predict the stimuli that will follow an action in order to plan the next action, and so on. While planning, it may be necessary for the controller to be choosing actions in the here and now, as we do (while driving a car). The expert controller will need an STM that operates in the there and then, as well as the STM that works in the here and now.

Predictions of stimuli will be needed to model the expected reaction of the world to *planned* actions when a plan is being constructed. Predictions of actions will be needed to choose actions for the plan that would correspond to the robot's current control policy.

To introduce a second STM, we have only to change equation (3) to:-

$$\text{state} = (\text{STM-h\&n}, \text{STM-t\&t}, \text{LTM}) \qquad (10)$$

where STM-h&n is the STM acting in the here and now, while STM-t&t is an STM acting in the there and then. There will be two separate equations corresponding to equation (7), but equation (5) will need special consideration. To assume that there is no direct interaction between the two STM's and LTM is consistent with our use of special outputs from LTM (which extended the input vector to an input* vector) to cater for such interactions, as we did for the STM - LTM interaction. This would allow equation (5), and hence (8), to be divided into two separate equations:-

$$\text{new LTM} = \text{LTM} \blacksquare_{h\&n} f_7(\text{input}, \text{STM-h\&n}) \quad (11)$$

$$\text{new LTM} = \text{LTM} \oplus_{t\&t} f_7(\text{input, STM-t\&t}) \quad (12)$$

Almost certainly, the local operator for the there and then, $\oplus_{t\&t}$, would have to be limited in the changes it could make to LTM so that the system didn't build a false model of the world and its own behavior in it. In experiments to date, $\oplus_{t\&t}$ has been limited to the marking of productions in LTM so as to mark out planned paths through memory [25].

B. Forward and Backward Chaining

The homogeneous encoding assumption has been used to keep domain-specific knowledge away from the design of the expert controller. In other words, not knowing what the significance of the various inputs will be, the designer of the expert controller has to treat them all alike. This is, of course, a feature of expert system "shells" which are systems that people can use for a wide variety of different expert system problem areas [26]. Such systems have a "tabula rasa" memory for production rules and a general inference engine which operates on rules from any domain. The two most common, general purpose techniques used by an inference engine are **forward chaining** and **backward chaining** [12].

In forward chaining, the left-hand sides of rules are matched to current conditions, thus enabling the right-hand sides to "infer" new conditions which lead to other rules matching, and so on. In this left to right (i.e. forward) chaining, inference must be limited by constraints to prevent the system from generating an unmanageable number of inferences. For example the construction of plans discussed in the last section is a forward chaining process in which inferences (predicted actions and stimuli) are restricted to the formation of a single plan without branches.

Backward chaining depends on the setting of goals which are matched to the right-hand sides of rules, thus causing the conditions in the left-hand sides of rules to become sub-goals. The sub-goals are treated in a similar way so a right to left (i.e. backward) chaining results. Here again, the process needs to be constrained or unmanageable numbers of sub-goals will be generated. There is no point in constructing plans which do not lead to goals, so the forward chaining process of constructing plans by a MCLS must be directed by a backward chaining process from goals. The backward chaining process that we have used is called **leakback** and it marks productions in LTM with their expectation of leading the system to goals. The forward chaining planning process uses these expectations to choose between alternative inferred actions so as to construct a single nonbranching plan.

C. Expectations and World Graphs

Outputs are selected from the combined predictions of the automaton, choosing only one of any mutually exclusive group of predictions. The selection procedure varies among implementations but should be simply based on the majority evidence given by the predictions, as indicated in Figure 3.

Both actions and stimuli are selected, both becoming inputs to the automaton, while actions are sent to the body for execution. The stimuli are used for planning, as set out in section III.*A*.

To allow the leakback (backward chaining) process to generate expectations for the productions in LTM, we have to throw a probabilistic network over the stored productions without weakening the computational structure of the MCLS. This is explained in detail in the rest of this section. The network will never cause the system to choose an action or stimulus that the underlying MCLS didn't predict. The MCLS remains the computational system, while the network creates a motivational system. Together they constitute a motivated learning system called PURR-PUSS. (Early versions of PURR-PUSS had a non-probabilistic network [20,27].)

The set of production templates which control the storage of productions in LTM divide LTM into separate collections of productions. Each collection includes all productions having contexts and predicted events as specified by one of the production templates. (Since some of the production templates are chosen to operate in combination, we also recognize **clusters** of production templates.) The productions associated with each template (or cluster of templates) are stored together and share a probabilistic net or **world graph** [28]. Within a world graph, all productions from the same template (or cluster) and having the same context are stored together at a **context node** (Figure 5). When the context of a context node matches a context in STM and is followed in time by another context node of the same world graph matching a context in STM, then we consider a transition to have occurred from the first node, or "from-node", to the second, or "to-node". A forward transition arc is stored at the from-node, pointing to the to-node. A backward arc will also be stored at the to-node pointing to the from-node. (Note that the forward arc is associated with a particular production of the from-node, while the backward arc is merely from node to node.) It is these arcs which turn the net into a directed graph, justifying the name world graph. There are no arcs between nodes in different world graphs.

Transition probabilities are stored with each of the forward transition arcs to estimate the conditional probability that, given a production of the from-node, there will follow an STM match with the to-node. The word "follow" should not be read as "follow immediately", but rather as "follow before any other node of the same world graph matches its context with STM". The probabilities are estimated by a stochastic process. The transition probabilities allow a backward chaining calculation of the expectation that a goal node can be reached from a given context node. These calculations are carried out along the lines of Howard's policy iteration method with discounting [2,29]. The expectation is held at the given context node and is regularly updated. Although the calculation of expectations of nodes within a world graph is sound and stable, the expectations cannot be used to do more than bias the choice of actions (or stimuli) of the expert controller because each expectation is based on the narrow view of a single world graph. Our solution to this difficulty is to arrange that the controller plans robot paths by forward-chaining with

the whole of STM-t&t. In this way the plans take into account the competing predictions from all the world graphs. Backward chaining still guides the planned path towards the nearest and strongest goal nodes.

D. Goals

We have experimented with several different types of goal, both positive (reward) and negative (punishment or pain). The positive goals have been reward given directly by a teacher (by pushing a "reward button"), and **novelty**. The negative goals have been used less, but they include in-built punishment and a transient punishment by teacher called **disapproval**. Novelty and disapproval have been found particularly useful for teaching, so these will be discussed briefly.

Every new context node that is stored in a world graph becomes a novelty goal until that context reappears in STM. The great advantage of teaching with novelty is that the system is constantly wanting to repeat what is new but having done so it is ready to learn the next lot of new things. Disapproval is a teacher-given punishment which marks the nodes having their contexts in STM at the time of the punishment. If the contexts appear again in STM without disapproval being applied, these negative goal markers are removed. Disapproval enables the teacher to remove conflicting contexts from interacting behaviors without having to know anything about the contexts that are being removed. The negative goals it produces leakback negative expectation, biassing the system's decisions away from the goals, just as positive, novelty goals leakback positive expectation, biassing the system's decisions toward novelty.

IV. EXAMPLES

A. Universal Turing Machine

MacDonald and Andreae [30] have shown that the basic framework of an MCLS is readily taught to emulate the behavior of a Universal Turing Machine (UTM), complete with finite state controller and indefinitely extendable tape. After being taught the initial tape contents, how to behave like the UTM, and how to add to its tape, the system will carry on computing on its own until it eventually runs out of memory. Full details are given [2,30], including how the MCLS is taught to count modulo-n, using the case of n=4, the teaching of the initial tape of the UTM, the teaching of the quintuples of the UTM, and a short teaching session giving the system the ability to add on tape. When the system is shown the final link between the various parts that it has been taught, it needs no further help to continue with the computations of the UTM.

This example is a formal proof of the computational power of the MCLS. However, it also illustrates features that prove useful in less formal interaction with the system and its extensions [31,32]. The ability to count is frequently needed for tasks that are taught. The way a number of

subtasks can be integrated into an overall task is an important feature of the system. We expect to use more often in the future the parallel streams of input events, demonstrated in the UTM example, as we gain experience in designing multiple contexts.[2]

Since the MCLS is a collection of fixed length predictors, from what does it derive its computational power? "Auxiliary" actions provide the power. These are extra actions not needed for the particular task at hand, which can be used for symbolic purposes, effectively representing state information. Without auxiliary actions a fixed length predictor is unable to distinguish simple sequences such as *ba...a* and *ca...a* because once the context contains only *a*'s the system has forgotten whether a *b* or a *c* occurred before the *a*'s. However, once auxiliary actions are provided, symbolic information can represent state information and can be put in contexts, making predictions depend on previous states. After a *b* at the start the system might be taught to say "One B", then continue saying "One B" until the sequence ended, thus enabling it to remember that a *b* came first. With slight elaboration the system need not continue repeating "One B", as the speech actions for this can remain in a context indefinitely so long as no other speech actions are performed [20]. More specific and complex work using auxiliary actions is described in [2,25,30]. The Turing Machine simulation is achieved by combining a simulation of the tape controller---which uses auxiliary actions to remember the controller state---and a simulation of an extendable tape---using auxiliary actions to remember the tape position. The tape is extendable as much as desired for the lifetime of the system (upward of 70 years if necessary!). In addition unbounded *external* memory can be used for the tape in the environment [2,27]. Speech is particularly useful as auxiliary actions since speech can be performed without noticeable physical effect on the robot's world, so it is free to be used symbolically. Other actions---perhaps subliminal or internal to the robot's body---might also be used as auxiliary actions.

B. Controlling a Real One-Dimensional Robot Arm

This section describes a simple experiment in which an MCLS controls a real robot arm [2,34,35]. The arm has a single link with one joint axis; it is a bar on a shaft (Figure 6). The MCLS is also a simple one, having only two contexts. The task for the system is to put its arm into a light beam at a particular, fixed position. The task must be accomplished despite the weight on the arm. Larger weights cause the arm to sag more, so that different motor commands are required for lifting a weighted arm.

The task is taught by showing the robot the goal of putting the arm into the light beam. To do this the teacher simply leads (i.e. lifts) the robot arm into the beam. This teaches a sequence of motor commands for moving the arm, but the sequence is not the correct one for reaching the beam, since gravity causes the arm to sag when the robot moves it. So the teacher also shows the robot several motor commands by leading it to move up and down a few times. This gives the

controller some useful "domain knowledge". The (extended) MCLS is then able to select a sequence that will reach the goal; it learns to control the arm under gravity.

The two production templates of the arm-robot MCLS are:

current arm angle, current arm velocity	==>	*arm action*	*(ANGLE template)*
latest arm action, current arm velocity	==>	*arm action*	*(ACTION template)*

for example

10 degrees, Stopped	==>	-10	*(ANGLE)*
+30, Up	==>	-10	*(ACTION)*

The ANGLE productions have a system similar to leakback for selecting actions on a sub-optimal path to the goal.

Figure 7 shows a record of one teaching session with the robot. The figure shows the angle of the arm over a period of time. The shaded lines show the arm movements when the teacher is leading it, while the solid lines show the arm movements under the learning system's control. The arm automatically relaxes after a few seconds if the learning system does not select a motor command to be performed, thus allowing the teacher to lead the arm when the learning system does not know what to do. The figure shows the arm being led to 100 degrees from 0 degrees. Then the teacher holds the arm, and drops it, allowing it to fall to 0 degrees. The teacher then leads the arm to the goal (70 degrees, where the light beam is), pushes it up over 100 degrees and drops it again. The system begins to perform actions itself, lifting to 10 degrees. This is a new sensed position---the arm sagged under gravity---so the learning system does not know what to do; there are no productions in LTM that match the STM contexts. The arm relaxes and the teacher helps it up to 30 degrees. The teacher continues to help the robot until finally at the end of the figure it is able to reach the goal all by itself. Complete details are given in [2,34,35].

This experiment shows that an MCLS can indeed control a real time, real world---albeit simple---robot arm. It can be shown a simple goal and a selection of motor commands, then reach that goal along new paths of motor commands that counteract the effect of gravity on the arm's load. There is considerable noise in the system's sensory information and it handles real world stimuli and actions without having to cut them up into arbitrary time slots. A simulated vacuuming robot demonstration is reported in [36]. Here a robot learns to vacuum a very simple room, and is still able to clean the room when a piece of furniture has been moved. It does this by learning sequences to vacuum parts of the room, but learning more flexible goals for moving from one part of the room to another. The goals learned enable it to cope with the shifted furniture.

C. Nursery Rhymes

At an early stage of the development of the PURR-PUSS system, when each context template was taken only from consecutive events of the same shift register, we wondered how long such templates might have to be. Now that we use templates that can draw events from any positions in the span-set (section II.*A*), the same question is still relevant: How long do the shift registers need to be? A strong answer came from a book of 71 nursery songs [37]!

A melody comprises a sequence of notes characterized by pitch, note-length and stress, the note-length and stress prescribing the rhythm of the melody. Nursery tunes are so similar to each other that a child who identifies one of them must be distinguishing one sequence of notes and its stress pattern from all the other possible sequences and stress patterns.

In this experiment an early version [20] of PURR-PUSS (section III.*C*) was given two templates to hold pitch and note-length, and stress and stress-length events, respectively [38]. Stress and non-stress are coded as "<" and "=". Stress-lengths are coded as "2" for the duration of a quarter note, "1" for an eighth note, and ":" for a sixteenth note. In Figure 8, the notes and note-lengths are shown in musical notation, while the stress and stress-lengths are given by the rows of codes below the staves. One other code "<>" starts the stress/stress-length sequence for each tune. This follows the name of the tune which is learned by the productions of a third "speech" template.

A rough inspection of the nursery rhymes indicated that context lengths of at least 7 events would be needed. After 15 hours of interaction and 39 nursery tunes learned, we ran out of computer memory (100kB), but had confirmed that the 7 event context lengths were about right. The PURR-PUSS system predicted future plans, which were in this case to sing some sequences of notes with appropriate duration and stress. The excerpt illustrated in Figure 8 shows a planned song in which there is a switch from the tune of Love Little Pussy to the tune of Ride A Cock Horse. This was the only example found and if you play the notes, you will find the error to be acceptable even to the adult ear. For shorter lengths of context, there would be a great deal of confusion between tunes. George Miller's [39] magic number seven rears its head again!

More details of this experiment are available in Andreae [38].

It is worth mentioning that in this early version of PURR-PUSS, it was established that the system could learn a wide variety of tasks for different environments without confusion. In fact, the nursery rhyme task used the same templates as were concurrently being used for other tasks discussed in Andreae [20].

D. Teaching What Can't Be Shown

Space prevents us from giving many examples and from giving more than one example in detail (section IV.*E*). Any interactive teaching session is a long business and it is not possible to convey to a reader the delights and frustrations of teaching a system which has its own plans for action.

Interactions have to be sanitized to bring them down to a communicable length. All the little detours and distractions that accompany an impromptu and spontaneous teaching session have to be removed by repeating the procedure with carefully thought out and tailored sequences. In chapter 7 of Andreae [20] an attempt is made to give some idea of the nature of a more spontaneous session. The description given of MacDonald's one-dimensional robot arm in section IV.*B* is also unsanitized. As an indication of the difference between a spontaneous and a sanitized version, we can mention a two-dimensional rectangular grid environment involving a stylized mother's finger, baby's hand and eye, and three beads. The environment was designed so that teacher acted only through the mother's finger and could not demonstrate to PURR-PUSS (acting as the baby) how to move one of the beads out of a corner of the grid. The object of the exercise was to prove that teacher could teach something to PURR-PUSS without being able to do it for her or force her to do it. In the first session, it took two hours to get the baby to move a bead out of the corner and at the end of that the teacher had no idea how it had been done. By repeating the process and analyzing it, the interaction was reduced to a brief, 35 step, sanitized sequence [40].

E. Numbers in the Head

The Turing Machine example demonstrated the computational power of the MCLS and PURR-PUSS. There is no point in giving examples which do no more than illustrate that power. We should, of course, like to show the system learning language, but to date we have not had the time or resources to simulate an environment in which the system could even begin to learn to use significant constructions in a natural language. In one series of experiments we have been exploring the problems of intentionality and emotions [25]. In the experiment to be described briefly here, we were concerned with the system's ability to manipulate concepts "in its head". Regrettably, the reader will have to be referred elsewhere for full details and computer programs [41].

Human thinking is a strange mixture of reaction to the environment and internal meanderings. While we drive a car or dig a garden, our minds may be partially occupied with other things, such as planning a party or imagining talking to someone not present. The internal meanderings do not need to be irrelevant to the task at hand. While driving the car, we may be deciding whether to turn left or right at the intersection which is just appearing in the distance. By imagining the possible effects of our gardening, we may well change direction and start some new bit of landscaping.

A common example of this mixed thinking is the use of numbers in the head for controlling tasks in the environment. For example, we may be manipulating numbers in our heads in order to share out equal quantities of some item amongst several people, or for sticking stamps on a letter to satisfy the postal regulations.

In the following experiment (which was later repeated with the full PURR-PUSS system), an MCLS (i) learns to manipulate 3-digit numbers in the environment, (ii) manipulates the numbers "in

its head" when the environment is taken away, and (iii) uses the internal manipulation of numbers to count "money" from a "box" in the environment.

Counting in the Displays and Buttons Environment

The MCLS is taught to count in an environment with three single-digit displays and three associated buttons (Figure 9). Each button causes its display to increment the displayed digit, modulo 4. The MCLS has five arm-movement actions ("push-units", etc.), four eye-movement actions ("look-units" etc.), and speech actions to say WORDS. The stimuli from the environment to the MCLS are either the hearing of WORDS said by the teacher or by itself, or seeing stimuli ("see-", "see-0", etc.). The units digit is displayed on the right in Figure 9, the tens digit in the middle and the hundreds digit on the left.

In order to produce a sanitized teaching sequence (c.f. section IV.*D*), we carefully follow the action schedule shown in Figure 10. The production templates given to the MCLS are shown in Figure 11. We can see how everything comes together by following the first 20 teaching steps. The digits displayed at the start are three 0's.

Step 1. Action = **raise-arm**	Teacher raises the robot's arm. No stimulus is received back from the environment after this action.
Step 2. Stimulus = **hear-REPEAT**	Teacher says "REPEAT" and is heard by MCLS.
Step 3. Action = **look-hundreds**	Teacher wobbles the hundreds display and a reflex eye movement causes the MCLS to look at that display, with the result that it sees the "0".
Stimulus = **see-0**	Two of the production templates (2 and 7) have now been satisfied by the events, so the following productions are stored in LTM: [2: raise-arm hear-REPEAT ==> look-hundreds] [7: look-hundreds ==> see-0]
Step 4. Stimulus = **hear-NO**	Teacher says "NO" and is heard by MCLS. [6: hear-REPEAT look-hundreds see-0 ==> say-NO] is stored. Notice that although the "NO" was heard, it is stored as though it had been said by the MCLS. This arrangement, called "mimic-speech" in [20], enables the MCLS to learn to speak from hearing others speak to it.

Step 5. Stimulus = **hear-HUNDREDS**	Teacher says "HUNDREDS" and is heard by MCLS. [4: look-hundreds hear-NO ==> say-HUNDREDS] is stored.

The next six steps are similar and will be abbreviated:

Step 6. Action = **look-tens,**	[3: hear-NO hear-HUNDREDS ==> look-tens]
Stimulus = **see-0**	[7: look-tens ==> see-0]
Step 7. Stimulus = **hear-NO**	[6: hear-HUNDREDS look-tens see-0 ==> say-NO]
Step 8. Stimulus = **hear-TENS**	Teacher says "TENS" and is heard by MCLS. [4: look-tens hear-NO ==> say-TENS]
Step 9. Action = **look-units,**	[3: hear-NO hear-TENS ==> look-units]
Stimulus = **see-0**	[7: look-units ==> see-0]
Step 10. Stimulus = **hear-NO**	[6: hear-TENS look-units see-0 ==> say-NO]
Step 11. Stimulus = **hear-UNITS**	Teacher says "UNITS" and is heard by MCLS. [4: look-units hear-NO ==> say-UNITS]
Step 12. Action = **shut-eyes**	Reflex shuts robot's eyes when teacher puts his hands over them. [3: hear-NO hear-UNITS ==> shut-eyes]
Stimulus = **see-**	[7: shut-eyes ==> see-]
Step 13. Action = **raise-arm**	Teacher raises the robot's arm. No stimulus. [1: hear-UNITS shut-eyes ==> raise-arm]
Step 14. Stimulus = **hear-ONES**	[5: see- raise-arm ==> say-ONES] Notice from the schedule that "ONES" initializes a subtask in which the MCLS counts a one.
Step 15. Action = **look-units**	[2: raise-arm hear-ONES ==> look-units]
Stimulus = **see-0**	
Step 16. Action = **push-units**	Teacher leads robot's arm to push units button. [1: hear-ONES look-units ==> push-units]
Stimulus = **see-1**	Units display has been incremented by button push. [7: look-units ==> see-1] [8: see-0 push-units ==> see-1]

Step 17. Stimulus = **hear-REPEAT**	[5: see-1 push-units ==> say-REPEAT]
Step 18. Action = **look-hundreds**	[2: push-units hear-REPEAT ==> look-hundreds]
Stimulus = **see-0**	This stimulus was expected since it was predicted immediately after the action by production 7 stored on step 3. However, the prediction of stimuli doesn't become important until the environment isn't there to provide them. The simple MCLS we are using here doesn't have an STM-t&t to do planning with predicted stimuli.

At this point, the events "hear-REPEAT" and "look-hundreds" match the context of production 6 stored on step 4, so the predicted action "say-NO" from that production is passed to the MCLS to act on.

Step 19. MCLS Action = **say-NO.**	Likewise, the next four actions are predicted.
Stimulus = **hear-NO**	
Step 20. MCLS Action = **say-HUNDREDS**	predicted by production 4 stored on step 5.
Stimulus = **hear-HUNDREDS**	

Another 245 steps are needed to teach the MCLS to increment units, tens and hundreds, and to teach it to pass the carry from units to tens and from tens to hundreds, since this is not done for it by the displays and push buttons. At four places (steps 77, 140, 185, and 215) teacher has to "shout-down" the MCLS to make it change subtask. If teacher says a WORD at the same time as the MCLS, teacher gets priority. This is called shouting down or crashing in. Details are given in Andreae [41].

Counting in the Head

At step 245 the MCLS can carry out any of the sequences through the action schedule of Figure 10, except the branch through "take-from-box". Also the displayed digits have been returned to three 0's. Teacher now removes the displays and the buttons to check that the MCLS can continue counting without them. This is made possible by three of the production templates (5, 6 and 8) shown in Figure 11, which accept predicted visual stimuli when the environment doesn't produce any. (The possibility of noisy stimuli is another good reason for having templates which allow strongly predicted stimuli to take preference over stimuli from the environment.) The

following few steps, taken immediately after the displays and buttons were removed, illustrate the behavior of the system.

Step 266. Stimulus = **hear-REPEAT** This is teacher shouting down the MCLS.

Step 267. MCLS action = **look-hundreds**
Predicted stimulus = "**see-0**"

Step 268. MCLS Action = **say-NO.**
Stimulus = **hear-NO** The MCLS can still hear itself speak.

Step 269. MCLS Action = **say-HUNDREDS**
Stimulus = **hear-HUNDREDS**

...

Now teacher introduces the box which randomly dispenses dollar notes. The notes have values of $1, $10 and $100 or the box may dispense nothing ($-). To get the MCLS to count these notes, teacher only has to show the MCLS how to respond to each output from the box. By chance, the box dispenses nothing the first time teacher leads the robot arm to trigger the box dispenser.

Step 276. MCLS action **shut-eyes**
Predicted stimulus = **see-**

Step 277. Action = **take-from-box** Teacher leads robot's arm to box.
Stimulus = **see-$-** Stimuli "see-$-", "see-$1", "see-$10", and "see-$100" from the box indicate the amount dispensed in dollars. Since the box happens to have dispensed nothing, teacher shows the appropriate subtask of repeating the numbers already in the MCLS's "head". (Productions are stored from templates 1, 7 and 8 here.)

Step 278. Stimulus = **hear-REPEAT** [5: see-0 take-from-box ==> say-REPEAT]

Step 279. MCLS action = **look-hundreds**
Predicted stimulus = **see-0**

Step 280. MCLS Action = **say-NO.**
Stimulus = **hear-NO**

...

The next time round the box dispenses a $10 note.

Step 288. MCLS action **shut-eyes**
Predicted stimulus = **see-**

Step 289. MCLS action = **take-from-box**	Teacher is not needed this time.
Stimulus = **see-$10**	Productions stored are not shown.
Step 290. Stimulus = **hear-TENS**	Teacher gives subtask.
Step 291. Action = **look-tens**	Teacher helps since context is new.
Predicted stimulus = **see-0**	MCLS takes over.

Step 292. MCLS action = **push-tens**
Predicted stimulus = **see-1**

...

As soon as the box has dispensed a $1 and a $100 note and teacher has told the MCLS what subtasks these correspond to, the MCLS will be able to count anything that the box dispenses and will keep a correct running total.

We have shown how the MCLS can be taught to count with help from the environment and then without help from the environment and, finally, to count the output from a box in the environment using its internalized counting ability.[3]

V. CONCLUSION

The design of an expert controller for a robot body, as presented above, was grounded in the belief that sophisticated mobile robots will need sophisticated expert controllers. Dismissing the possibility that expert controllers for such robots could be pre-programmed for all eventualities, we followed the consequences of four reasonable assumptions (section I.*A*) for the design of an expert controller which started with no specific knowledge, either about the particular robot body to which it would be attached, or about tasks and environments that body would encounter. In this, we were influenced by the common understanding [42] that the human neocortex is the site of learning and that it is, as its name implies, a relatively recent development of the human brain, while aeons of evolution have gone into the development of the specialized faculties of the human body, such as vision, hearing, manipulation and mobility. The multiple context, which determines what specific associations our expert controller can make between different inputs, is seen as corresponding to the association areas of the neocortex. Elsewhere [19-21], we have indicated how readily the expert controller could be implemented in a neural net technology without reliance on the arbitrariness of many connectionist systems [43].

Nevertheless, while similarities with the human design may be useful and encouraging, the expert controller has to stand on its own merits. The experiments illustrated in this paper, together with others referenced, confirm that the system has the necessary computational power and that it can learn in a wide variety of situations. As more elaborate robot bodies and environments can be simulated and real, advanced robots become available for its testing, we expect further significant extensions to the design of the controller and, in particular, the emergence of a methodology for selecting multiple contexts.

REFERENCES

[1] Ambler, A.P., Popplestone, R.J. and Kempf, K.G. An experiment with the offline programming of robots. *Proc. 12th Int.Symp. on Industrial Robots; 6th Int.Conf. on Industrial Robot Technology,* Paris, France, 491-504, June, 1982.

[2] MacDonald, B.A. Designing Teachable Robots. PhD thesis, University of Canterbury, Christchurch, New Zealand, 1984.

[3] Summers, P.D. and Grossman, D.D. XPROBE: an experimental system for programming robots by example. *Int J Robotics Research, vol 3,* no 1, 25-39, Spring, 1984.

[4] Bonner, S. and Shin, K.G. A comparative study of robot languages. *Computer, vol 15,* no 12, 82-96, 1982.

[5] Lozano-Perez, T. Robot programming. *IEEE Proceedings, vol 71*, no 7, 821-41, 1983.

[6] Brooks,R.A. Autonomous mobile robots. in *AI in the 80's and beyond.* edited by E.L.Grimson and R.S.Patil, MIT Press, 343-363, 1987.

[7] Burks,B.L., deSaussure,G.,Weisbin,C.R.,Jones,J.P. and Hamel,W.R. Autonomous navigation, exploration, and recognition using the Hermies-IIB robot. *IEEE Expert, vol.2*, no.4, 18-27, 1987.

[8] Crowley,J.L. Coordination of action and perception in a surveillance robot. *IEEE Expert, vol.2*, no.4, 32-43, 1987.

[9] Goto,Y. and Stenz,A. Mobile robot navigation: the CMU system. *IEEE Expert, vol.2*, no.4, 44-54, 1987.

[10] McTamaney,L.S. Mobile robots: real-time intelligent control. *IEEE Expert, vol.2*, no.4, 55-68, 1987.

[11] Minsky, M.L. Adaptive control: from feedback to debugging. In Selfridge, O.G., Rissland, E.L. and Arbib, M.A. *Adaptive Control of Ill-defined Systems,* 115-125, Plenum, 1984.

[12] Barr, A. and Feigenbaum, E.A. *The Handbook of Artificial Intelligence.* vol.2, Pitman, 1982.

[13] Michalski, R.S., Carbonell, J.G. and Mitchell, T.M. *Machine Learning*. vol.2, p.3, Morgan Kaufmann, 1986.

[14] Witten, I.H. and MacDonald, B.A. Using concept learning for knowledge acquisition. Presented at *AAAI Workshop on knowledge acquisition for knowledge-based systems,* Banff, October, 1987. (to be published in *Int.J.Man-Machine Studies*).

[15] Witten, I.H. and MacDonald, B.A. Concept learning: a practical tool for knowledge acquisition? *Proc. 7th Intl Workshop on Expert Systems and Their Applications,* Avignon, France, 1535-55, May, 1987.

[16] Hasegawa, T. An interactive system for modeling and monitoring a manipulation environment. *IEEE Trans.SMC, vol SMC-12,* no 3, 250-8, 1982.

[17] MacDonald, B.A. and Witten, I.H. Programming computer controlled systems by non-experts. *Proc.IEEE Conf.on Systems, Man and Cybernetics,* Alexandria, VA, October, 432-7, 1987.

[18] Polanyi, M. The Logic of Tacit Inference. *J.Roy.Inst. of Philosophy, vol 41*, no 155, 1-18, 1966.

[19] MacDonald, B.A. Connecting to the past. To be published in Collected papers of IEEE conf. on Neural Information Processing Systems, 1987, Denver, Co, November, American Institute of Physics.

[20] Andreae, J.H. *Thinking with the Teachable Machine*. Academic Press, 1977.

[21] Dowd,R.B. A digital simulation of mew-brain. Report no. UC-DSE/10, ref.[24], 25-46, 1977.

[22] Newell, A. and Simon, H.A. *Human Problem Solving*. p.804, Prentice-Hall, 1972

[23] Minsky, M.L. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1966.

[24] Andreae, J.H. *Man-Machine Studies*. Progress Reports nos. UC-DSE/1-29 to the Defence Scientific Establishment. Dept of Electrical and Electronic Engineering, University of Canterbury, Christchurch, New Zealand. (These reports are also available from the N.T.I.S., 5285 Port Royal Rd, Springfield, Virginia 22161.) 1972-1987.

[25] Andreae, J.H. Purposeful Computer Systems. *Proc.Int.Conf. on Future Advances in Computing*, Christchurch, N.Z. Published by Dept of Computer Science, University of Calgary, Alberta 2TN 1N4, Canada,1-26, 1986.

[26] Hayes-Roth, F., Waterman, D.A. and Lenat, D.B. *Building Expert Systems*. Addison-Wesley, 1983.

[27] Andreae, J.H. and Cleary, J.G. A New Mechanism for a Brain. *Int. J. Man-Machine Studies, vol 8,* 89-119, 1976.

[28] Lieblich, I. & Arbib, M.A. Multiple Representation of Space Underlying Behavior. *Behavioral and Brain Sciences, vol 5,* 627-659, 1982.

[29] Howard, R.A. *Dynamic Programming and Markov Processes.* MIT Press, 1966.

[30] MacDonald, B.A. and Andreae, J.H. The Competence of a Multiple Context Learning System. *Int. J. General Systems. vol 7,* 123-137, 1981.

[31] Andreae, J.H. and Andreae, P.M. Machine learning with a multiple context. Proc.9th IEEE Int.Conf.on Cybernetics and Society, Denver, October, 734-9, 1979.

[32] Andreae, P.M. and Andreae, J.H. A teachable machine in the real world. *Int.J.of Man-Machine Studies, vol.10:* 301-12, 1978.

[33] Andreae, J.H. Bicameral Mind. Report no. UC-DSE/16, ref. [24], 39-73, 1980.

[34] MacDonald, B.A. A robot learns by being led through movements. Report no. UC-DSE/19, ref. [24], 7-73, 1981

[35] MacDonald, B.A. Leading teaches robot any movement task. Report no. UC-DSE/20, ref. [24], 5-68, 1982.

[36] MacDonald,B.A. Improved robot design. *Trans.IPENZ Elec/Mech/Chem section, vol 14,* no 1/EMCh, 33-48, 1987.

[37] *My First Sing-A-Song Book.* London: Paul Hamlyn, 1966.

[38] Andreae, J.H. A Dual Model of the Brain. Report no. UC-DSE/10, ref.[24], 58-84, 1977.

[39] Miller, G.A. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. *Psychological Review, vol.63,* 81-97, 1956.

[40] Andreae, J.H. A Robot Mentality. Report no. UC-DSE/20, ref.[24], 69-128, 1982.

[41] Andreae, J.H. Numbers in the Head. Report no. UC-DSE/24, ref.[24] 4-28, 1984.

[42] Eccles, J.C. The cerebral neocortex. A theory of its operation. Chapter 1 in *Cerebral Cortex.* Vol.2 Functional Properties of Cortical Cells. Edited by E.G. Jones and A. Peters, Plenum, 1-36, 1984.

[43] Rummelhart,D.E. and McClelland,J.L. *Parallel distributed processing.* Volumes 1 and 2, MIT Press, 1986.

Figure Captions.

Manuscript received ....

Authors:

John H. Andreae
Dept of Electrical and Electronic Engineering
University of Canterbury
Christchurch
New Zealand

Bruce A. MacDonald
Knowledge Sciences Institute
Department of Computer Science
The University of Calgary
Calgary, Alta.
Canada

FOOTNOTES

This paper was written during a visit of J.H.A. to the University of Calgary. Financial support has been given by the Natural Sciences and Engineering Research Council of Canada, the New Zealand Ministry of Defence, the University of Calgary and the University of Canterbury. We appreciate Ian Witten's constructive comments on drafts of this paper.

---

[1]The careful reader will have noticed that the controller no longer satisfies our homogeneous encoding assumption, once the production templates have been chosen. However, the framework of the controller---our first stage from section I---still satisfies it in that no particular production templates are chosen, so all inputs and outputs are still treated in the same way.

[2]A short BASIC program is listed in Andreae [33] to implement the UTM-simulating MCLS and to allow the user to teach actions to the system.

[3]A BASIC program for carrying out the whole process is listed in Andreae [41].

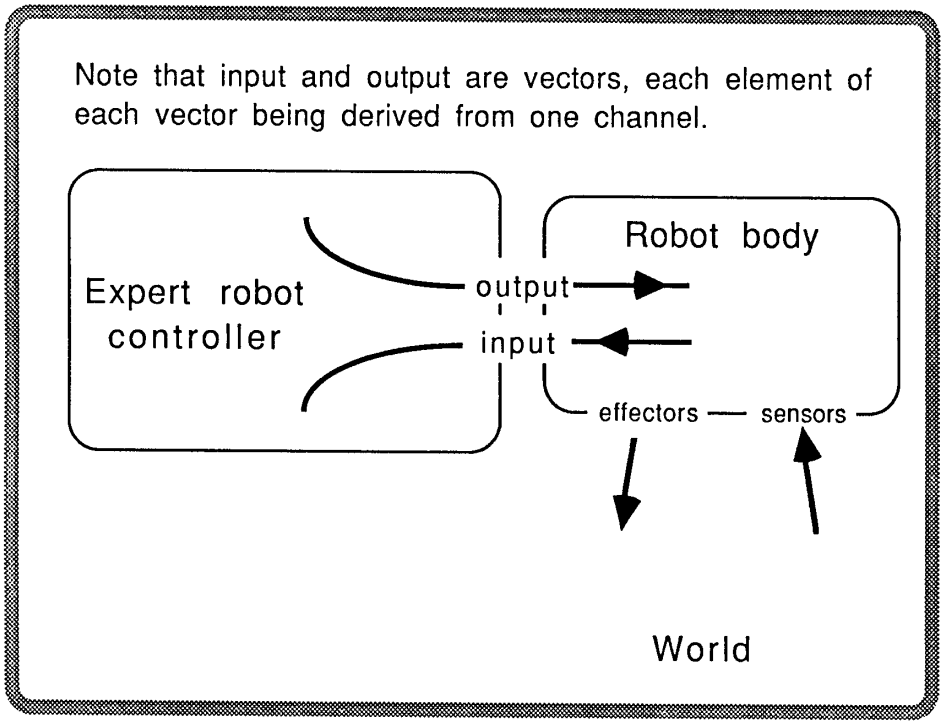

Figure 1 Expert robot controller.

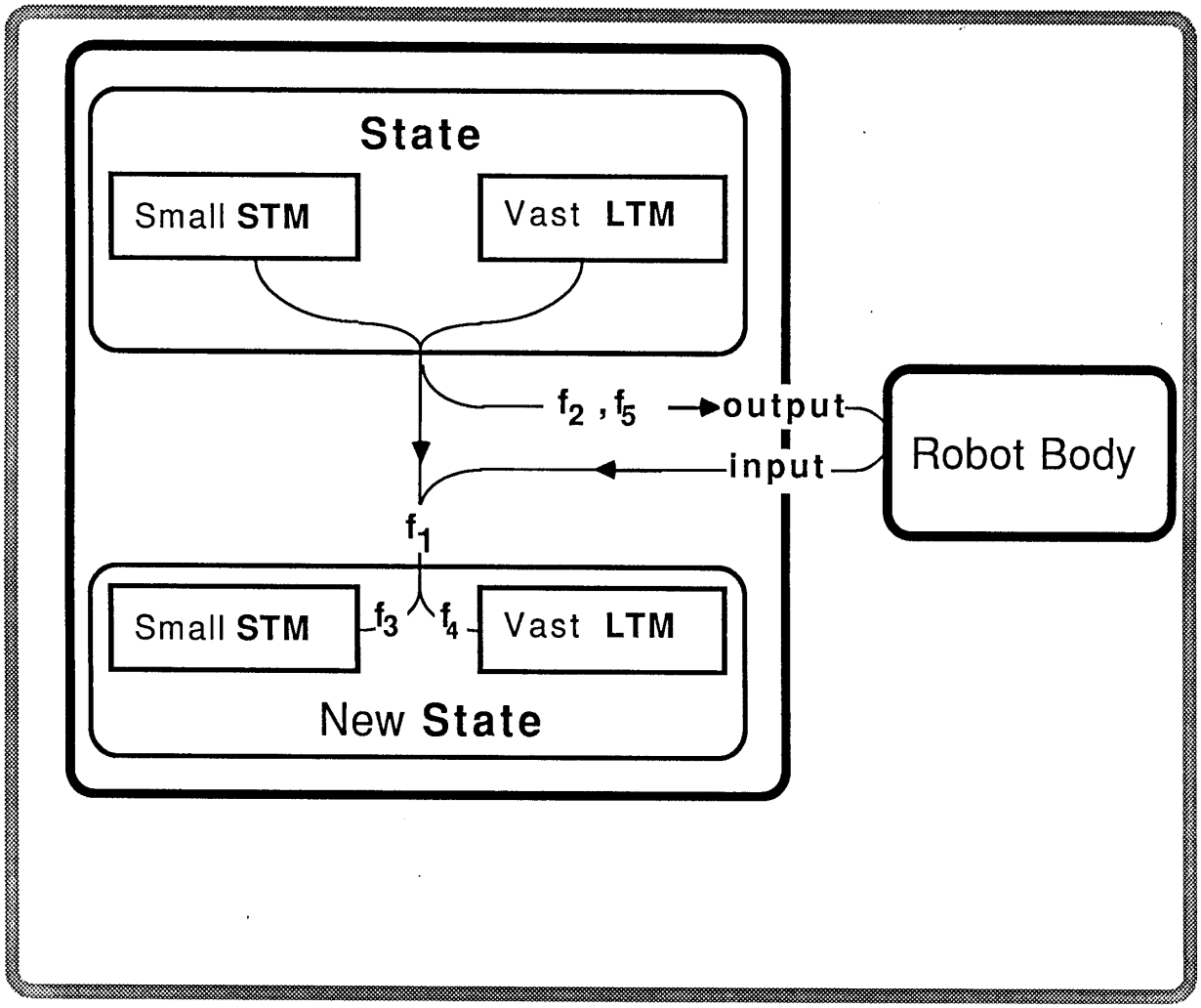

Figure 2. Diagram to Aid the Understanding of Eqns (1)-(6)

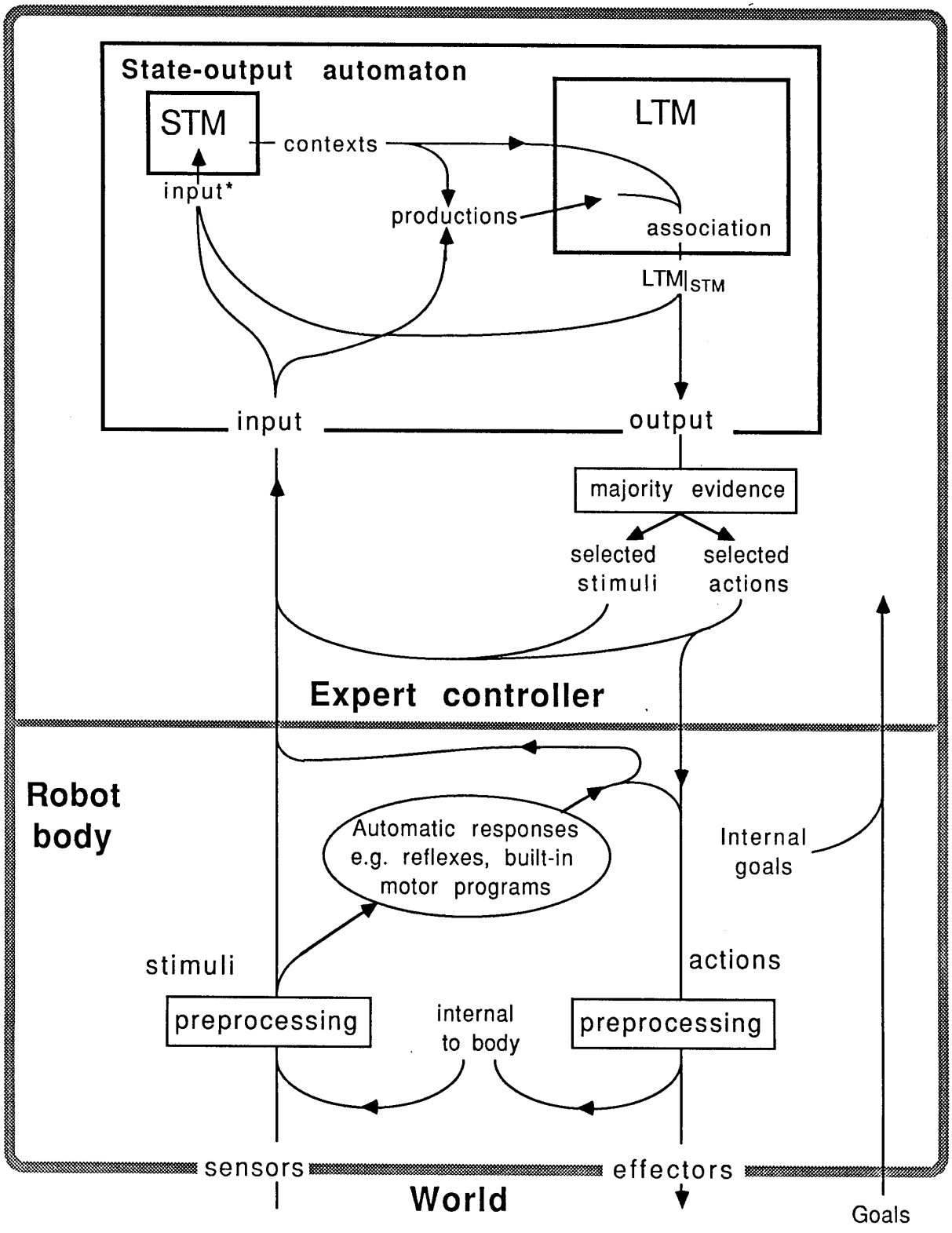

Figure 3 Expert controller, robot body, and world

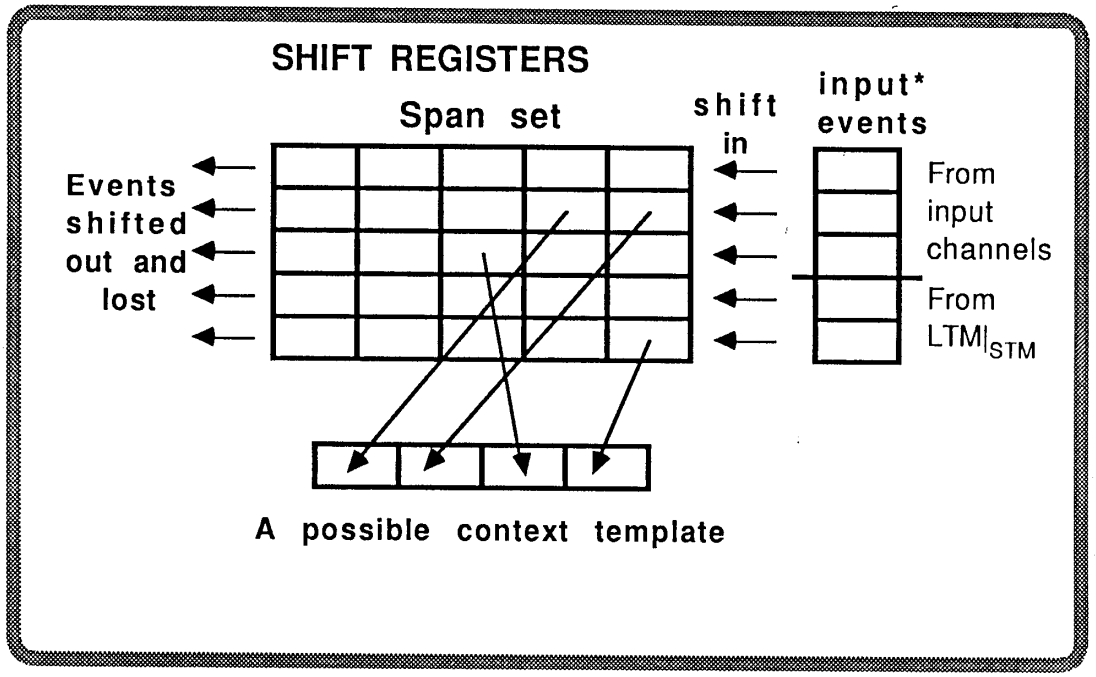

Figure 4. Shift Registers for updating STM.

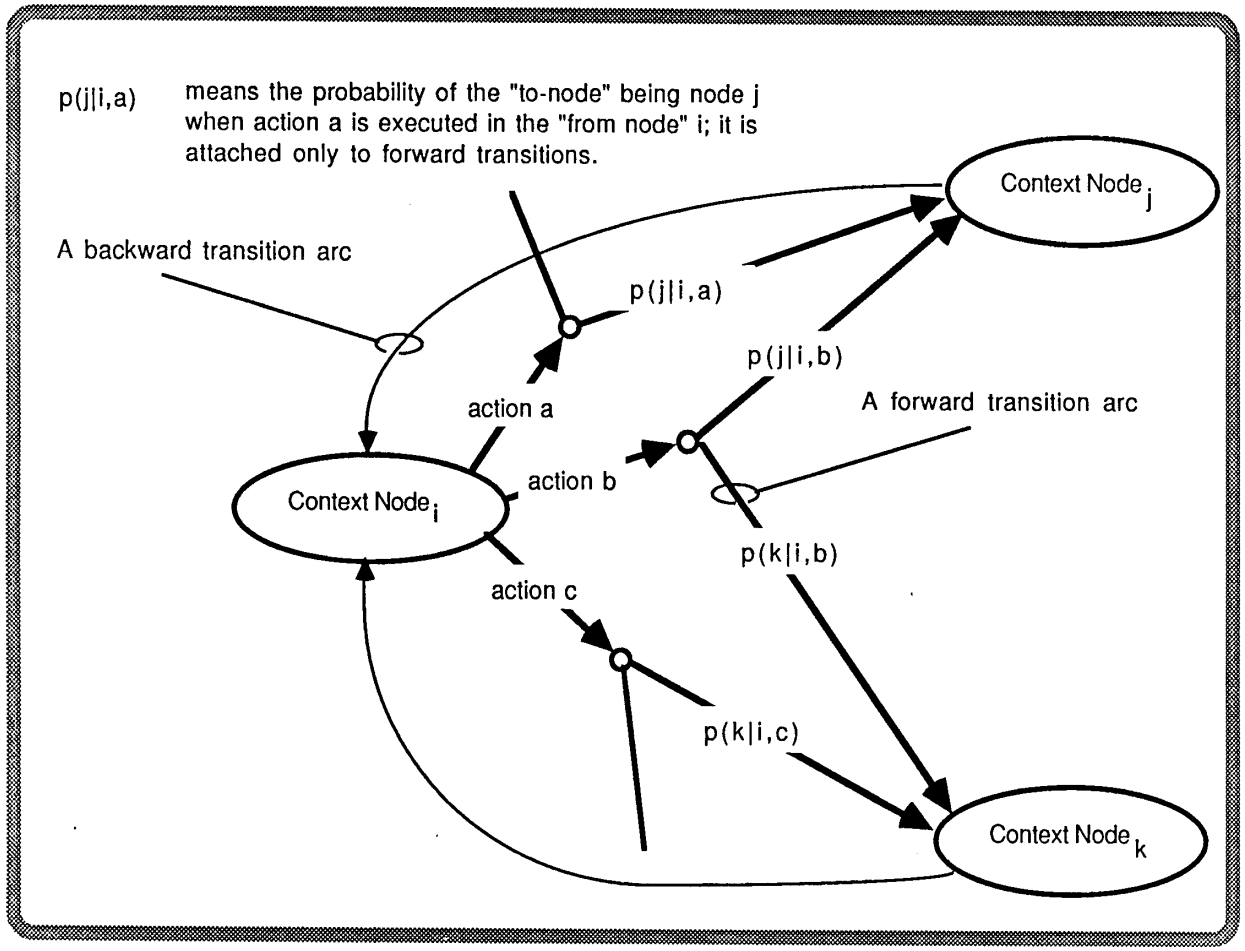

Figure 5 World graph formed by productions with common contexts, and forward and backward transition arcs.

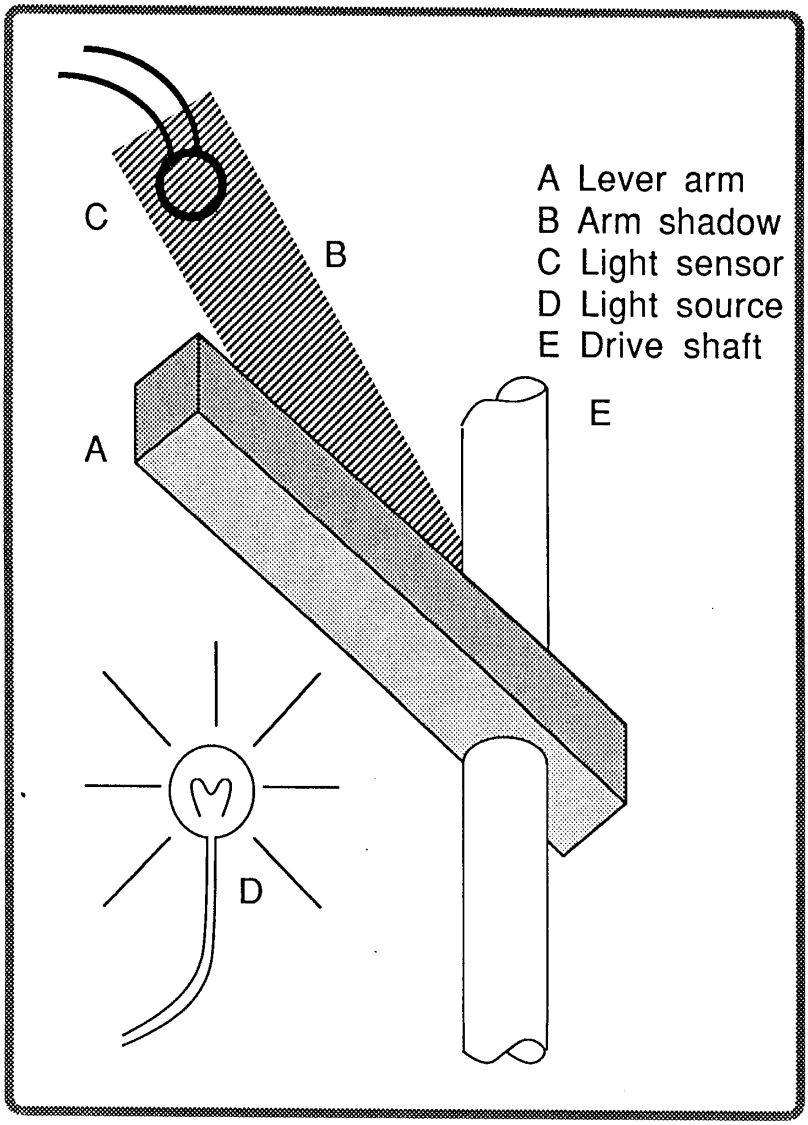

Figure 6 The Arm Robot

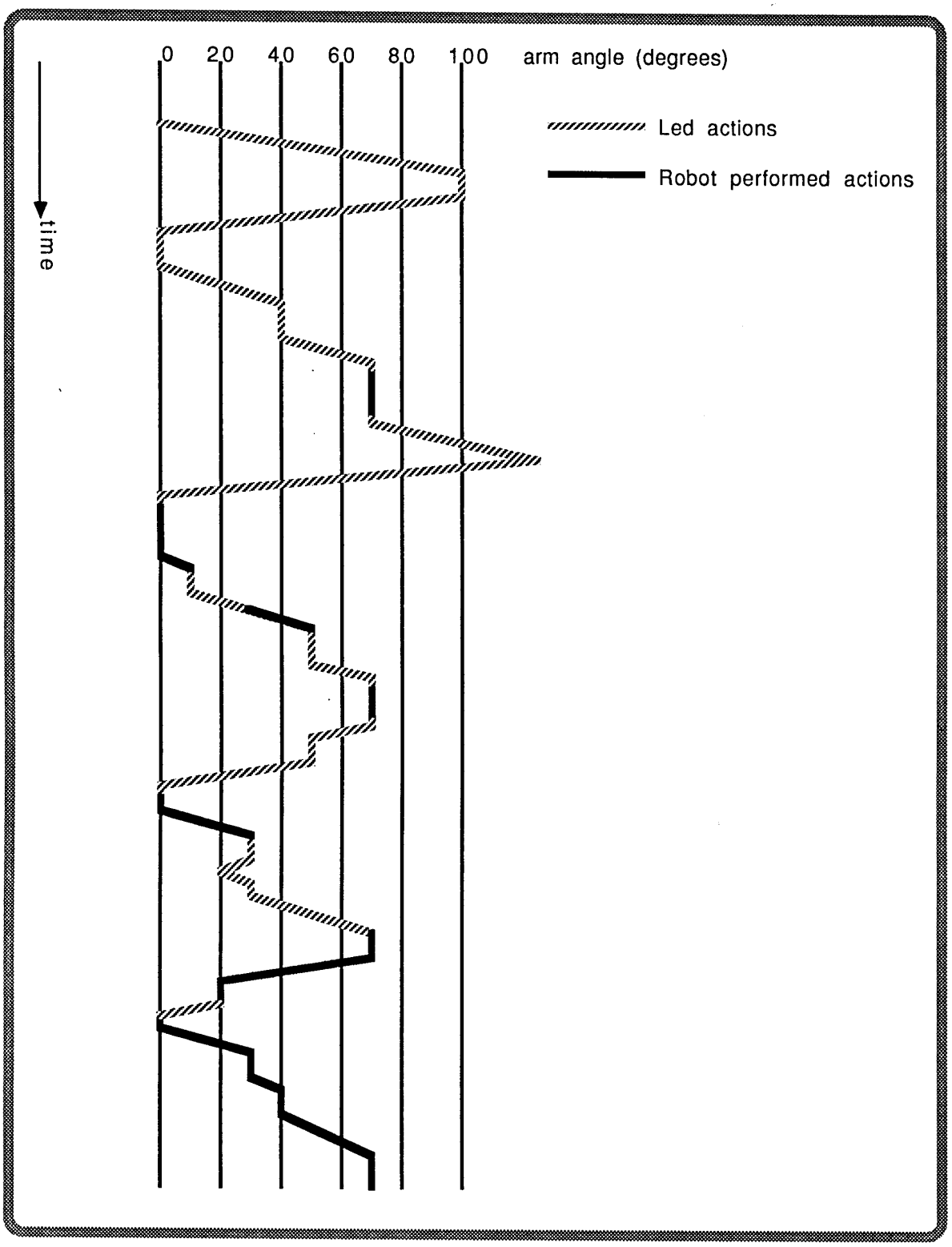

Figure 7 The arm robot performs the task of raising its arm into a light beam.

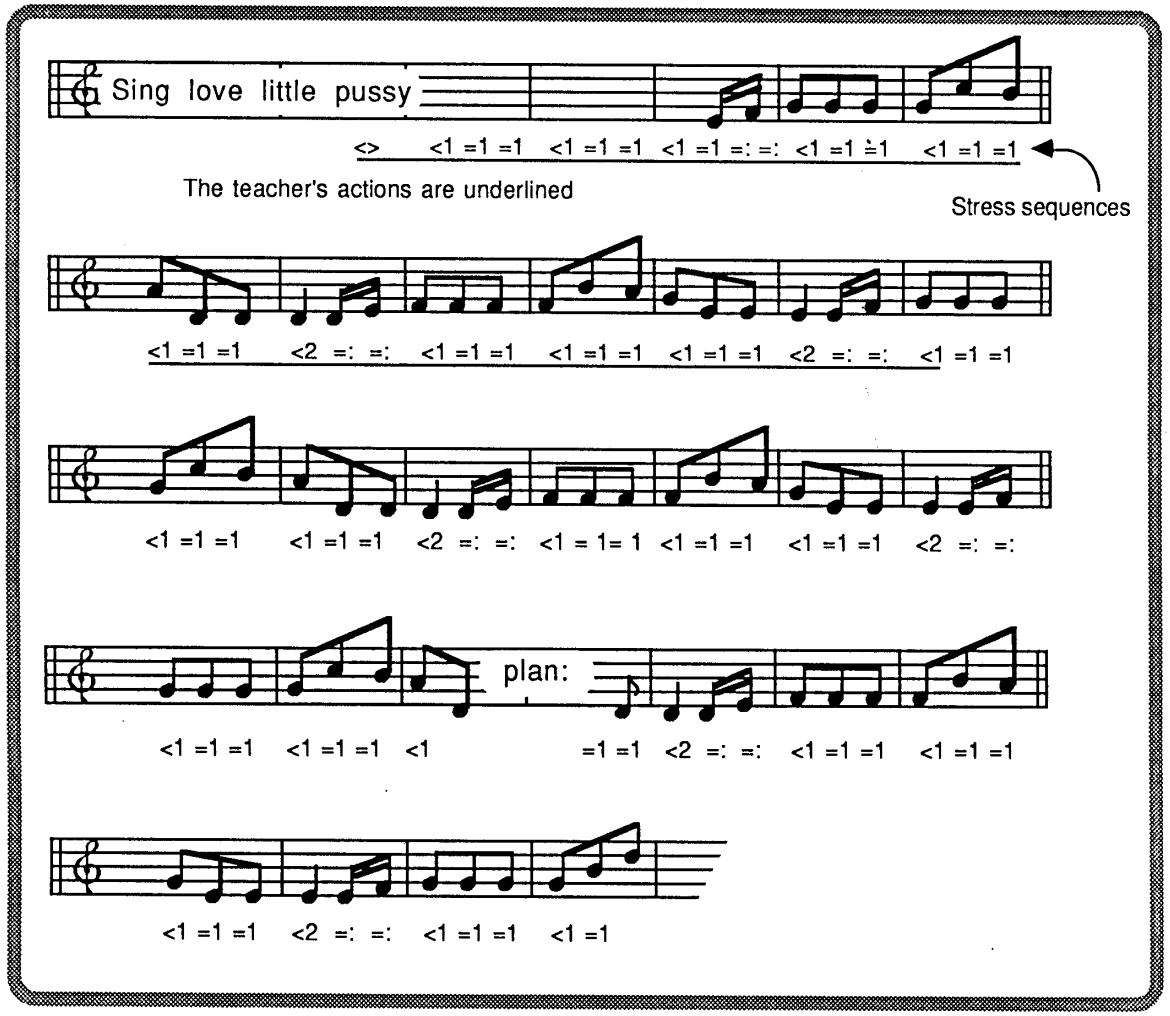

Figure 8 Teaching nursery rhymes

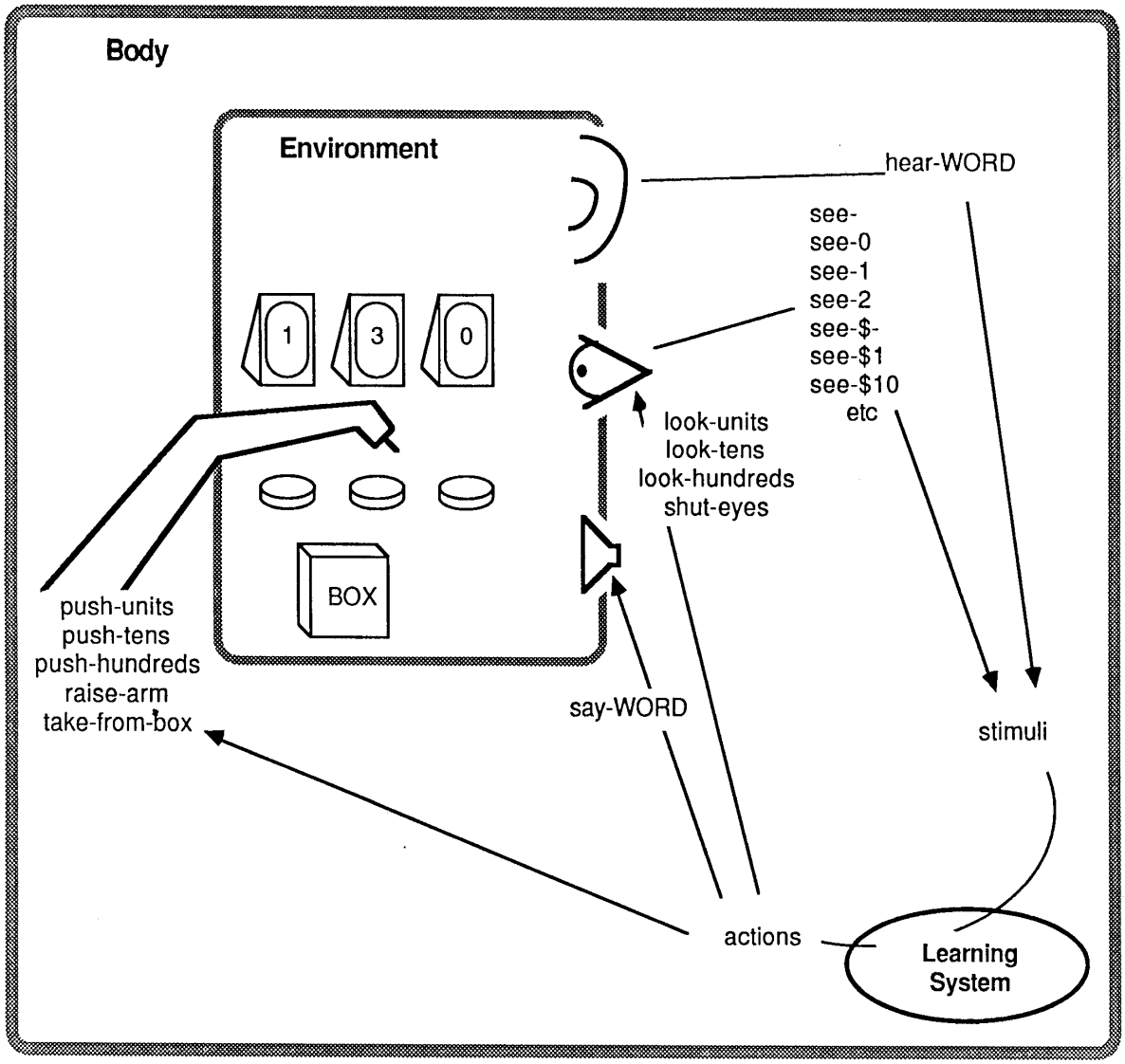

Figure 9 Numbers in the Head. Displays and Buttons

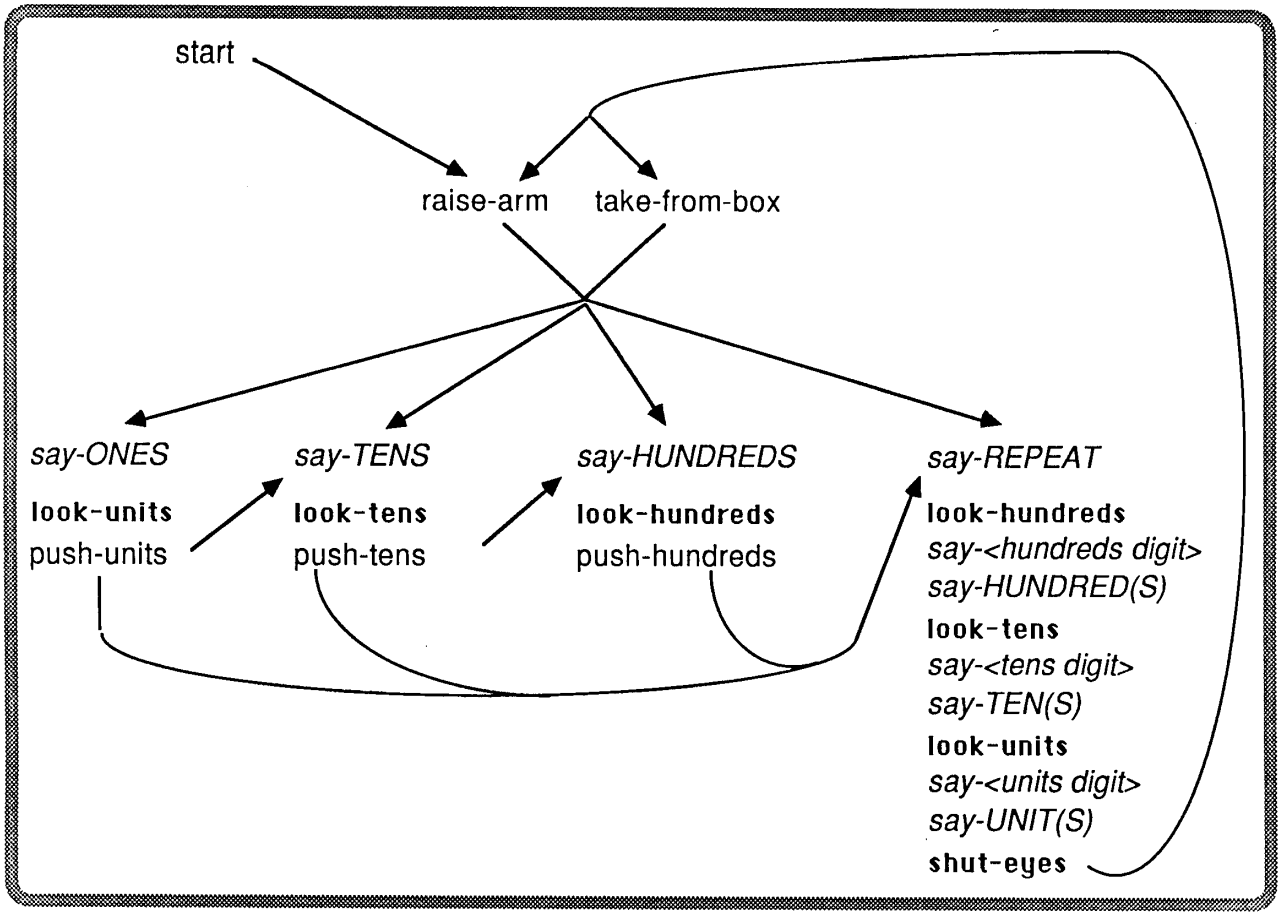

Figure 10 Action schedule for Numbers in the Head

[1: $speech_{n-2}$	$eye\text{-}movement_{n-1}$	==>	$arm\text{-}movement_n$]
[2: $arm\text{-}movement_{n-2}$	$speech_{n-1}$	==>	$eye\text{-}movement_n$]
[3: $speech_{n-2}$	$speech_{n-1}$	==>	$eye\text{-}movement_n$]
[4: $eye\text{-}movement_{n-2}$	$speech_{n-1}$	==>	$speech_n$]
[5: $see\text{-}stimulus/pred_{n-1}$	$arm\text{-}movement_n$	==>	$speech_n$]
[6: $speech_{n-2}$	$eye\text{-}movement_{n-1}$ $see\text{-}stimulus/pred_{n-1}$	==>	$speech_n$]
[7: $eye\text{-}movement_{n/n-1}$		==>	$see\text{-}stimulus_n$]
[8: $see\text{-}stimulus/pred_{n-1}$	$arm\text{-}movement_n$	==>	$see\text{-}stimulus_n$]

Note. n is the action-stimulus step number when the production is stored.
stimulus/pred means predicted stimulus is used if there is no stimulus.
n/n-1 subscript means use $n\text{-}1^{th}$ if there isn't an n^{th} event.

Figure 11. Production Templates for Numbers in the Head.