delim $$

PROGRAMMING BY EXAMPLE REVISITED

by John G. Cleary
Man-Machine Systems Laboratory
University of Calgary.

Efforts to construct an artificial intelligence have relied on ever more complex and carefully prepared programs. While useful in themselves, these programs are unlikely to be useful in situations where ephemeral and low value knowledge must be acquired. For example a person (or robot) working in a normal domestic environment knows a lot about which cupboards have sticky doors and where the marmalade is kept. It seems unlikely that it will ever be economic to program such knowledge whether this be via a language or a discourse with an expert system. It is my thesis, then, that any flexible robot system working in the real world must contain a component of control intermediate between hard wired 'reflex' responses and complex intellectual reasoning. Such an intermediate system must be adaptive, be able to carry out complex patterned responses and be fast in operation. It need not, however, carry out complex forward planning or be capable of introspection (in the sense that expert systems are able to explain their actions). In this talk I will examine a system that acquires knowledge by constructing a model of its input behaviour and uses this to select its actions. It can be viewed either as an automatic adaptive system or as an instance of 'programming by example'. Other workers have attempted to do this, by constructing compact models in some appropriate programming language:e.g. finite state automata [Bierman, 1972], [Bierman and Feldman, 1972]; LISP [Bierman and Krishnaswamy, 1976]; finite non-deterministic automata [Gaines,1976], [Gaines,1977], [Witten,1980]; high level languages [Bauer, 1979], [Halbert, 1981]. These efforts, however, suffer from the flaw that for some inputs their computing time is super-exponential in the number of inputs seen. This makes them totally impractical in any system which is continuously receiving inputs over a long period of time. The system I will examine comprises one or more simple independent models. Because of their simplicity and because no attempt is made to construct models which are minimal, the time taken to store new information and to make predictions is constant and independent of the amount of information stored [Cleary, 1980]. This leads to a very integrated and responsive environment. All actions by the programmer are immediately incorporated into the program model. The actions are also acted upon so that their consequences are immediately apparent. However, the amount of memory used could grow linearly with time. [Witten, 1977] introduces a modelling system related to the one here which does not continually grow and which can be updated incrementally. It remains to be shown that the very simple models used are capable of generating any interestingly complex behaviour. In the rest of this talk I will use the problem of executing a subroutine to illustrate the potential of such systems. The example will also illustrate some of the techniques which have been developed for combining multiple models, [Cleary, 1980], [Andreae and Cleary, 1976], [Andreae, 1977], [Witten,1981]. It has also been shown in [Cleary, 1980] and in [Andreae,1977] that such systems can simulate any Turing machine when supplied with a suitable external memory. Fig. 1 shows the general layout of the modeller. Following the flow of information through the system it first receives a number of inputs from the external world. These are then used to update the current contexts of a number of Markov models. Note, that each Markov model may use different inputs to form its current context, and that they may be attempting to predict different inputs. A simple robot which can hear and move an arm might have two models; one, say, in which the last three sounds it heard are used to predict the next word to be spoken, and another in which the last three sounds and the last three arm movements are used to predict the next arm movement. When the inputs are received each such context and its associated prediction (usually an action) are added to the Markov model. (No counts or statistics are maintained — they are not necessary.) When the context recurs later it will be retrieved along with all the predictions which have been stored with it. After the contexts have been stored they are updated by shifting in the new inputs. These new contexts are then matched against the model and all the associated predictions are retrieved. These independent predictions from the individual Markov models are then combined into a single composite prediction. (A general theory of how to do this has been developed in [Cleary, 1980]). The final step is to present this composite prediction to a device I have called the 'choice oracle'. This uses whatever information it sees fit to choose the next action. There are many possibilities for such a device. One might be to choose from amongst the predicted actions if reward is expected and to choose some other

random action if reward is not expected. The whole system then looks like a reward seeking homeostat. At the other extreme the oracle might be a human programmer who chooses the next action according to his own principles. The system then functions more like a programming by example system — [Witten, 1981] and [Witten, 1982] give examples of such systems. [Andreae, 1977] gives an example of a 'teachable' system lying between these two extremes. After an action is chosen this is transmitted to the external world and the resultant inputs are used to start the whole cycle again. Note that the chosen action will be an input on the next cycle. An important part of any programming language is the ability to write a fragment of a program and then have it used many times without it having to be reprogrammed each time. A crucial feature of such shared code is that after it has been executed the program should be controlled by the situation which held before the subroutine was called. A subroutine can be visualised as a black box with an unknown and arbitrarily complex interior. There are many paths into the box but after passing through each splits again and goes its own way, independent of what happened inside the box. Also, if there are p paths using the subroutine and q different sequences within it then the amount of programming needed should be proportional to $p + q$ and not $p * q$. The example to follow possess both these properties of a subroutine. The actual model we will use is described in Fig. 2. There are two Markov models (model-1 and model-2) each seeing and predicting different parts of the inputs. The inputs are classified into four classes; ACTIONs that move a robot (LEFT, RIGHT, FAST, SLOW), patterns that it 'sees' (danger, moved, wall, stuck) and two types of special 'echo' actions, # actions and * actions (*home, #turn). The # and * actions have no effect on the environment, their only purpose is to be inputs and act as place keepers for relevant information. They may be viewed as comments which remind the system of what it is doing. (The term echo was used in [Andreae,1977], where the idea was first introduced, in analogy to spoken words of which one hears an echo.) Model-2 is a Markov model of order 2 and uses only # actions in its context and seeks to predict only * actions. Model-1 is a Markov model of order 3 and uses all four classes of inputs in its context. It seeks to predict ACTIONs, # actions and * actions. However, * actions are treated specially. Rather than attempt to predict the exact * action it only stores * to indicate that some * action has occurred. This special treatment is also reflected in the procedure for combining the predictions of the two models. Then the prediction of model-2 is used, only if model-1 predicts an *. That is, model-1 predicts that some * action will occur and model-2 is used to select which one. If model-1 does not predict an * then its prediction is used as the combined prediction and that from model-2 is ignored. The choice oracle that is used for this example has two modes. In programmer mode a human programmer is allowed to select any action she wishes or to acquiesce with the current prediction, in which case one of the actions in the combined prediction is selected. In execution mode one of the predicted actions is selected and the programmer is not involved at all. Before embarking on the actual example some points about the predictions extracted from the individual Markov models should be noted. First, if no context can be found stored in the memory which equals the current context then it is shortened by one input and a search is made for any recorded contexts which are equal over the reduced length. If necessary this is repeated until the length is zero whereupon all possible allowed actions are predicted. Fig. 3 shows the problem to be programmed. If a robot sees danger it is to turn and flee quickly. If it sees a wall it is to turn and return slowly. The turning is to be done by a subroutine which, if it gets stuck when turning left, turns right instead. Fig. 4 shows the contexts and predictions stored when this is programmed. This is done by two passes through the problem in 'program' mode: once to program the fleeing and turning left; the other to program the wall sequence and the turning right. Fig. 5 then shows how this programming is used in 'execute' mode for one of the combinations which had not been explicitly programmed earlier (a wall sequence with a turn left). The figure shows the contexts and associated predictions for each step. (Note that predictions are made and new contexts are stored in both modes. They have been omitted from the diagrams to preserve clarity.) The type of simple modelling system presented above is of interest for a number of reasons. Seen as a programing by example system, it is very closely integrated. Because it can update its models incrementally in real time functions such as input/output, programming, compilation and execution are subsumed into a single mechanism. Interactive languages such as LISP or BASIC gain much of their immediacy and usefulness by being interpretive and not requiring a separate compilation step when altering the source program. By making execution integral with the process of program entry (some of) the consequencs of new programming become immediately apparent. Seen as an adaptive controller, the system has the advantage of being fast and being able to encode any control strategy. Times to update the model do not grow with memory size and so it can operate continuously in real time. Seen as a paradigm for understanding natural control systems, it has the

advantage of having a very simple underlying storage mechanism. Also, the ability to supply an arbitrary choice oracle allows for a wide range of possible adaptive strategies.

ANDREAE, J.H. 1977 Thinking with the Teachable Machine. Academic Press.

ANDREAE, J.H. and CLEARY, J.G. 1976 A New Mechanism for a Brain. Int. J. Man-Machine Studies 8(1):89-119.

BAUER, M.A. 1979 Programming by examples. Artificial Intelligence 12:1-21.

BIERMAN, A.W. 1972 On the Inference of Turing Machines from Sample Computations. Artificial Intelligence 3(3):181-198.

BIERMAN, A.W. and FELDMAN, J.A. 1972 On the Synthesis of Finite-State Machines from Samples of their Behavior. IEEE Transactions on Computers C-21, June: 592-597.

BIERMAN, A.W. and KRISHNASWAMY, R. 1976 Constructing programs from example computations. IEEE transactions on Software Engineering SE-2:141-153.

CLEARY, J.G. 1980 An Associative and Impressible Computer. PhD thesis, University of Canterbury, Christchurch, New Zealand.

GAINES, B.R. 1976 Behaviour/structure transformations under uncertainty. Int. J. Man-Machine Studies 8:337-365.

GAINES, B.