# PROGRAMMING BY EXAMPLE FOR THE CASUAL USER: A CASE STUDY

I.H. Witten

*University of Calgary*

## ABSTRACT

One way of giving casual users access to some of the power of a computer, without the need to learn formal programming methods, is to allow complex tasks to be defined extensively by example, rather than intensively by a procedural specification. This paper studies the extent to which iterative computations on an electronic calculator can be inferred interactively from an initial part of the sequence of key-presses, using techniques of non-deterministic structural identification of behaviour sequences. The aim is to construct an interactive device which is partially self-programming.

Despite the fact that the device has no prior knowledge of the syntax or semantics of the dialogue, it is remarkable successful. Several short, repetitive calculator problems have been analysed. Even in a mixed sequence, over 75% of the dialogue elements were predicted, with an error rate between 0.5% and 1.5%. However, casual users have not yet been exposed to the scheme.

## RÉSUMÉ

Il est possible de permettre aux usagers occasionnels de faire appel à une partie de la puissance de traitement d'un ordinateur sans avoir à étudier les méthodes de programmation standard si l'on définit les tâches complexes de manière "extensive", au moyen d'un exemple, plutôt que de façon "intensive", à l'aide d'une procédure de traitement. La présente communication évalue dans quelle mesure on peut obtenir des calculs itératifs par voie interactive, sur une calculatrice électronique, à partir d'une portion initiale de la séquence des pressions sur les boutons en faisant appel à des méthodes d'identification structurale non déterministe des séquences de comportement. Il s'agit de créer un organe interactif qui soit partiellement autoprogrammable.

En dépit du fait que cet organe ne soit préalablement doté d'aucune connaissance syntaxique ou sémantique, il donne d'excellents résultats: plusieurs courts problèmes de calcul répétitifs ont été analysés avec succès. Même dans le cas d'une séquence mixte, plus de 75% des éléments de dialogue ont été prédits, avec un taux d'erreur variant entre 0.5% et 1.5%. Toutefois, les utilisateurs occasionnels n'ont pas encore mis ces méthodes à l'essai.

## Introduction

The casual user, who can be characterized as one who uses computers relatively infrequently as an information processing tool for his other professional or recreational activities, has been the subject of some controversy in the literature. Anderson (1980), discussing the role of programming in the home of the future, bases his work on the premise that programming will be a widespread activity, and declares that

> "this must be so for effective use of resources."

Cuff (1980) argues against the view that programming ability will in the future be as common in civilized society as numeracy or literacy is today. For him, casual users may or may not have some programming ability; but even if they have, such skills

> "are unlikely to be sharply honed through working with the system."

It may be that such skills need not be "sharply honed", at least not in any formal sense. Anderson sees programming in the future

> "as gardening is today — a subject about which many people have knowledge at different levels, where knowledge resources are many and varied, and accessed in different ways."

Perhaps the view one takes depends upon how far ahead one is looking. At any rate, it seems clear that lack of programming ability is likely to remain a stumbling-block to casual use of computers for a long time to come.

One way of giving casual users access to some of the power of a computer without the need to learn formal programming methods, is to allow complex tasks to be defined extensively by example, rather than intensively by a procedural specification. This paper studies the extent to which iterative computations on an electronic calculator can be inferred interactively from an initial part of the sequence of key-presses, using simple techniques of non-deterministic structural identification of behaviour sequences. The aim is to construct an interactive device which is partially self-programming. Even this rather modest goal presents some interesting problems of man-machine engineering. If it is successful, the technique may have application to a far wider range of tasks involving simple command languages — interactive editing, operating system control languages like the Unix "shell", and the like — where the user occasionally has to choose between writing a program for a short repetitive sequence of operations and executing them manually.

## Related work

Extensive rather than intensive problem specification was studied by Zloof (1977) in the context of data-base retrieval. The Query-by-Example system, which is intended for a user with no programming and little mathematical experience, allows him to specify the information to be retrieved by presenting an example of the kind of item which should be included. Although the system can be criticized — one quickly gets bogged down in built-in functions, condition boxes, and metalinguistic symbols like single and double underlining — it illustrates that useful interactive systems can be built which allow users to define complex tasks extensively instead of intensively. One of its great advantages is that it frees the user from thinking of his retrieval problem in an artificially sequential form: instead he can specify the links, conditions, and constraints as they occur to him. In contrast, the present paper explores how strictly sequential information may be inferred by a machine from an initial subsequence.

Automatic inference of programs from examples is another domain in which the problem is specified extensively. Given a set of input-output correspondences, the goal is to construct automatically a program that implements them. Amarel (1971), during the course of a long investigation of this topic, distinguished derivation problems from formulation problems. In the former, one is given parts of a solution and asked to complete it by using given rules for construction. A classic example is automatic theorem proving in elementary logic, for which several successful systems have been built (eg Wang, 1960 for propositional calculus; Robinson, 1965 for predicate calculus). "Formation" problems are more complex: construction rules do not exist and one must proceed to the goal by hypothesizing solutions and testing them against the given input-output correspondences. Little real progress seems to have been made in automatic inference of programs for these problems.

When a detailed trace of an example

```
t=a(0) s=0 i=1 i<t-1 a(i+1)<a(i) s=i x=a(i) a(i)=a(i+1) a(i+1)=x i=i+1 i<t-1 a(i+1)>a(i)
i=i+1 i<t-1 a(i+1)<a(i) s=i x=a(i) a(i)=a(i+1) a(i+1)=x i=i+1 i<t-1 a(i+1)<a(i) s=i x=a(i)
a(i)=a(i+1) a(i+1)=x i=i+1 i=t-1 s>0 s=0 i=1 i<t-1 a(i+1)>a(i) i=i+1 i<t-1 a(i+1)>a(i) i=i+1
i<t-1 a(i+1)<a(i) s=i x=a(i) a(i)=a(i+1) a(i+1)=x i=i+1 i<t-1 a(i+1)>a(i) i=i+1 i=t-1 s>0
s=0 i=1 i<t-1 a(i+1)>a(i) i=i+1 i<t-1 a(i+1)<a(i) s=i x=a(i) a(i)=a(i+1) a(i+1)=x i=i+1
i<t-1 a(i+1)>a(i) i=i+1 i<t-1 a(i+1)>a(i) i=i+1 i=t-1 s>0 s=0 i=1 i<t-1 a(i+1)>a(i) i=i+1
i<t-1 a(i+1)>a(i) i=i+1 i<t-1 a(i+1)>a(i) i=i+1 i<t-1 a(i+1)>a(i) i=i+1 i=t-1 s=0 end
```

Figure 1. Trace of a sorting process (after Gaines, 1976b)

execution of a program is available, however, program inference can be easy. Biermann (1972) discusses the inference of Turing machines from traces of sample computations. For example, the trace of a sort of the three-element tape "baa" into "aab" can be expressed as

```
b bR a bL b bL # -R b aR b bR a bL b bL a
aR b aR b bR
```

where single a's and b's represent the current symbol read from the tape, and aL means "write an a in the current place and move the scanning head to the Left". The "#" input signifies that the end-marker has been read; nothing should be written over it so a null writing action "-" is specified. Given that the model should be deterministic, with the single a's, b's, and #'s as inputs and the composite symbols as outputs, Biermann showed that a data-driven, simplest-first search with recursive back-tracking finds it very quickly.

While interesting, this method is not of any great practical significance because it requires the trace to be error-free. Consider the example of a sort program considered by Gaines (1976b) and reproduced as Figure 1. It is unlikely that such a trace would be entered without error, and even if it were, it is not obvious that it is any easier for a casual user to generate than a structural description — a program — for the sorting process.

## Non-deterministic modelling of behaviour sequences

Recently-developed techniques of non-deterministic modelling seem to offer a way out. Whilst non-deterministic input will cause havoc in a deterministic modeller (Gaines, 1976a), sensible results can be obtained if the modeller does not assume determinism in the first place. There is a trade-off between model simplicity and goodness of fit to the behaviour sequence. Structurally simple models cannot provide a good fit to non-deterministic behaviours, while if an exact fit is sought the model becomes unreasonably complex. Models which cannot be improved in simplicity without sacrificing goodness-of-fit, and cannot be improved in goodness-of-fit without sacrificing simplicity, are called admissible, and in general there is a set of such models for a given behaviour, each with a different complexity.

Another advantage of non-deterministic modelling is that it can automatically separate (unpredictable) inputs to the system from its (predictable) outputs. Thus the user need not specify which symbols are predictable and which are not, nor remember the necessary metalinguistic conventions for such specification. For example, the conditional statements in Figure 1 are inputs, and this fact is inferred by non-deterministic modellers (Gaines, 1976b; Witten, 1979).

A recent paper (Witten, 1981) identifies three primary techniques which have been used to form non-deterministic models of behaviour sequences, together with some variants of each method. The present application calls for incremental modelling, where new elements of the incoming sequence are integrated into the existing model as they are seen. Most non-deterministic modelling methods require the complete input sequence to be stored, and re-model it from scratch when new information shows the current model to be inadequate. The only exceptions are "limited-context" modelling methods, which assume that the structure of the sequence can be characterized by the set of overlapping k-tuples of symbols that occur in the sequence, for some limited "context" length k.

Witten (1979) has investigated how k-tuples can be recorded economically by massaging them into the form of an automaton model. First the "ingenuous" model of k-tuples is constructed, with one state for each tuple and transitions which reflect the succession of tuples in the behaviour sequence. Only the last element of a tuple is

recorded as the output of the state. The model is then subjected to a reduction process which coalesces states in a way that does not destroy information about the k-tuples which occurred in the behaviour. For example, in an extreme case when modelling a random sequence whose elements are drawn from an alphabet of q symbols, $q^{**}k$ different k-tuples will occur, but this will become a mere q states (for a Moore model) after reduction. Thus although an increased value of k normally provides a larger, more accurate, model; it does not necessarily do so if structure does not exist in the behaviour sequence — as in our example.

With this method, which we call length-k modelling, each new element of the behaviour sequence can be incorporated into an already reduced model to form an updated version of it. This crucial advantage offsets the inherent weakness of limited context modellers, namely that they can only cope with "non-counting" events (McNaughton and Papert, 1971). In fact, this weakness has not proven to be a disadvantage in the present study.

Programming a calculator by example

People who use interactive computers regularly know that there are many situations in which it is difficult to decide whether to do a minor, but repetitive, task by hand or to write a program to accomplish it. Interactive editing provides many good examples. It is often necessary to change each occurrence of a particular token in a file to another token, and many of those who have used small computers for years will recall their joy when first encountering an editor with a "global change" facility. However, some tasks are not quite so simple: hence the notion of regular-expression-searching, together with a special character which designates the matched expression, was introduced. More flexibility is provided by making the editor programmable, either from within via a macro facility or from without via editing command files. As a consequence, using the editor has become a more highly skilled task.

Simple, repetitive arithmetic operations are a second problem domain which often presents a quandary as to whether a task should be done by hand or by program. For example, one may wish to plot y = x exp(1-x) for a dozen or so values of x: should one do it on a hand calculator or write a BASIC program? The first is easier and more certain; it will not take more than 10 minutes. The second may be quicker, but could involve a session with the manual to refresh one's memory with the vagaries of BASIC syntax. This seems to be an ideal domain to investigate the application of non-deterministic modelling techniques to automatic program formation.

Consider the possibility of an invisible non-deterministic modeller "looking over the shoulder" of the user; and prepared to perform actions automatically for him if it has sufficient confidence that it knows what to do. The user must pay a price for this service, for the modeller cannot help but be wrong occasionally. Extra keys must be provided to enable him to accept or reject the entry.

Sample dialogues. Figures 3, 4, and 5 show some results obtained from a simulation of length-k modelling applied to a calculator. The device chosen was the Casio fx-20, an infix machine whose relevant keys are summarized in Figure 2. These illustrations were produced under somewhat idealized conditions — we will discuss later the detrimental effects of operator error and negative transfer of learning between tasks. However, they are remarkably successful.

Figures 3 and 4 are each divided into two, showing the keys pressed by the operator and those suggested by the model. Time proceeds from left to right and top to bottom, and for conciseness the symbols are run on to the same line wherever possible. Figure 3 evaluates x exp(1-x) for a range of values of x, and the task has been learned by halfway

```
0  4  8
1  5  9          numeric keys
2  6  .
3  7

+  *          infix operators
-  /

+/-      (negate)
exp           postfix operators
log
cos

=        (evaluate)
mc       (clear memory)
mr       (retrieve operand from memory)
m+=      (evaluate and add to memory)
```

Figure 2. Relevant keys on the calculator which was used for the examples

| Operator presses | Calculator indicates |
|---|---|
| .1 mc m+= +/- + 1 = exp x mr = | [answer] |
| .2 mc m+= | +/- + |
| 1 | = exp x mr = [answer] |
| .3 | mc m+= +/- + 1 = exp x mr = [answer] |
| .4 | mc m+= +/- + 1 = exp x mr = [answer] |
| ... | ... |

[from here on the device behaves as though it had been explicitly programmed for the calculation]

Figure 3. Evaluation of x exp(1-x)

| Operator presses | Calculator indicates |
|---|---|
| 2 log m+= | |
| .0078 log / mr / 8 = + 1 = | [answer] |
| .0156 log / | mr / |
| 8 | = + |
| 1 | = [answer] |
| .025 | log / mr / 8 = + 1 = [answer] |
| .03125 | log / mr / 8 = + 1 = [answer] |
| ... | ... |

[from here on the device behaves as though it had been explicitly programmed for the calculation]

Figure 4. Evaluation of 1 + (log x)/(8 log 2)

45 cos x 2 x .9 mc m+= .9 x x = + 1 - mr = log x 10 = [answer: -2.69858]
100 / 4000 x 180 mc <DEL>
= cos x 2 x .9 mc m+= .9 x x = + 1 - mr = log x 10 = <DEL>
+ 20 + 2.69858 + <DEL>
= [answer]
500 / 4000 x 180 = cos x 2 x .9 mc m+= 0.9 x x = + 1 - mr = log x 10 + 20 + 2.69858 = [answer]
1000 / 4000 x 180 = cos x 2 x .9 mc m+= 0.9 x x = + 1 - mr = log x 10 + 20 + 2.69858 = [answer]
1500 / 4000 x 180 = cos x 2 x .9 mc m+= 0.9 x x = + 1 - mr = log x 10 + 20 + 2.69858 = [answer]
2000 / 4000 x 180 = cos x 2 x .9 mc m+= 0.9 x x = + 1 - mr = log x 10 + 20 + 2.69858 = [answer]
2500 / 4000 x 180 = cos x 2 x .9 mc m+= 0.9 x x = + 1 - mr = log x 10 + 20 + 2.69858 = [answer]
3000 / 4000 x 180 = cos x 2 x .9 mc m+= 0.9 x x = + 1 - mr = log x 10 + 20 + 2.69858 = [answer]
3500 / 4000 x 180 = cos x 2 x .9 mc m+= 0.9 x x = + 1 - mr = log x 10 + 20 + 2.69858 = [answer]
4000 / 4000 x 180 = cos x 2 x .9 mc m+= 0.9 x x = + 1 - mr = log x 10 + 20 + 2.69858 = [answer]

Figure 5. A more complicated calculation (suggestions from the model are underlined)

through the second iteration. From this point onwards the calculator behaves as a special-purpose device tailored for the problem: it executes all necessary instructions, pausing only for input. The evaluation of
1 + (log x)/(8 log 2) in Figure 4 shows similar behaviour.

The rather more complicated calculation of

```
20 + 10 log[1 + a**2 - 2a cos(180x/4000)]
   - 10 log[1 + a**2 - 2a cos 45]
```

(for a=0.9) is depicted in Figure 5. For conciseness, the interaction is shown in a different form from that of the previous figures. Suggestions from the model are underlined. There is no special significance in the placing of the line breaks. Since the calculator possesses only one "memory" location, it was expedient to compute the last sub-expression first and jot down the result. Some interference occurred between this initial task and the main one: three suggestions had to be rejected by the operator (the <DEL> key rejects the previous suggestion). On the positive side, note that the result of the preliminary calculation, 2.69858, had to be keyed only twice before the system picked up the fact that it could be predicted. However, only towards the end of the interaction does the device become fully programmed, for the penultimate "+" in most of the lines has to be inserted by the user.

Method. These illustrations use the length-k modelling technique outlined above, with k set to 4. However, the technique was tailored somewhat to the problem at hand. It is clear from a cursory analysis of calculator sequences that numbers and operators should be treated rather differently, for a typical sequence comprises different numbers embedded in a fixed template of operators. This rule is not universal, because fixed constants appear in the stream as well as variable input data. Notice how the constants 1 in Figure 3, 8 and 1 in Figure 4, and 4000, 180, 2, .9, 1, 10, 20, and 2.69858 in Figure 5 are all quickly picked out as predictable by the system.

In order to prevent differences in data values from rendering the length-k sequences inoperative, two length-k models were formed side by side. One used the raw behaviour sequence as observed, and the other mapped all numbers into the same token <NUM>. Predictions from the latter model were only used when the former one failed to yield a prediction. Furthermore, the system was constructed to be more conservative about predicting a number than an operator. No prediction was made unless it would have been correct the previous n times it occurred, and n was set differently for operators (n=1) and numbers (n=2).

For example, consider the state of the model after the second line of Figure 3 has been generated, that is, after

```
.1 mc m+= +/- + 1 = exp x mr =
.2 mc m+= +/- +
```

has been seen. With k=4, the tuples which have accumulated are

```
 .1  mc m+= +/-   (2)    <NUM> mc m+= +/-   (3)
 mc m+= +/-  +    (2)     m+= +/-  + <NUM>  (1)
m+= +/-  +   1    (1)     +/-  + <NUM> =    (1)
+/-  +   1   =    (1)      + <NUM> =  exp   (1)
 +   1   =  exp  (1)    <NUM> =  exp  x    (1)
 1   =  exp  x    (1)      x   mr  = <NUM>  (1)
 =  exp  x   mr   (1)      mr  = <NUM> mc   (1)
exp  x   mr  =    (1)      = <NUM> mc m+=   (1)
 x   mr  =   .2   (1)
 mr  =   .2  mc   (1)
 =   .2  mc m+=   (1)
 .2  mc m+= +/-   (1)
```

The numbers in parentheses count how many times each tuple has been encountered. Hence, although the current context "m+= +/- +" predicts a "1", it has only been confirmed once and this falls short of the threshold for predicting numbers (namely 2). Once the "1" has been entered, the new context "+/- + 1" predicts an "=", again with a single previous confirmation. This is used because the threshold for predicting operators is only 1.

If an erroneous suggestion is made, the modeller is very cautious about making a prediction from that tuple in the future. However, as time passes and its belief in the model is continually confirmed, it will eventually venture a suggestion. This is the reason why the penultimate "+"s on most of the lines in Figure 5 are only suggested at the end of the interaction. Both of the tuples

```
log x 10 =
log x 10 +
```

have occurred in the interaction, but a long run of the second is enough to cause it to be used eventually.

Possible enhancements. There are several fairly obvious modifications which could be

made to the algorithm to improve its performance. Firstly, it does not notice multiple occurrences of the same input data. For example, if x exp(1-x) were evaluated on a calculator without a memory element, x would have to be entered twice. The modelling procedure we have outlined is incapable of spotting this redundancy. It would clearly be easy to incorporate a special-purpose "demon" which noticed such repetition. On the whole, however, we prefer more generally-applicable techniques. (It may, for example, be possible to solve this problem using a recursive model; see Witten, 1981.) Furthermore, calculator manufacturers very sensibly try to design their machines to minimize such redundancy!

Secondly, the procedure is entirely lexical and does not recognize the properties of numbers. For example, in the sequence 0.1, 0.2, 0.3, ... (which occurs in Figure 3), it is evident what probably comes next; however, the system as presently implemented does not spot the pattern. Again, a special-purpose "demon" could easily be introduced.

Finally, calculator interactions have a definite syntax which is not exploited at all. For example, the system will learn nonsense sequences like "1 1 + + 1 1 + +" and regurgitate them just as readily as it will syntactically meaningful sequences. (The sequence "x x" which occurs in Figure 5 may seem anomalous too: in fact this is Casio's way of squaring a number.) This, we feel, is an advantage rather than a disadvantage of the scheme. It is obedient, uncritical, and — above all — general.

Effects of noise. Now let us examine the effects of free variation in the behaviour sequence. As noted above, non-deterministic modelling methods do not produce absurd models in the presence of noise as do deterministic ones.

One source of variation in calculator control sequences is the discovery of an easier way to do the task. For example, half-way through Figure 5 one may decide to enter 1.81 directly instead of ".9 x x = + 1", for it is the same quantity. However, this will surely not be a temptation in the interactive system we propose, for no penalty is associated with the latter sequence once it has been assimilated.

If the simplifying discovery is made very early on in the interaction, however, it may cause some incorrect predictions. For example, Figure 6 shows the evaluation of x exp(1-x) by two different methods. One evaluates 1-x using the calculator and the other evaluates it mentally. Three suggestions had to be rejected in this dialogue. The last error was rather unlucky, being caused by the coincidence of the .8 in ".8 exp(.2)" and ".8 exp(1-.8)". This illustrates how difficult it is to predict the behaviour of the modeller in advance: it is only of use in truly interactive applications.

All three errors are caused by incorrect predictions across the boundaries between individual "calculations", that is to say, between separate evaluations of the function x exp(1-x). If the modeller were informed of the points in the sequence where an "answer" is obtained, this could be used to delimit the individual calculations and the erroneous suggestions would be avoided. However, since the calculator produces a display after every entry it is not possible to discern these points automatically: a separate "delimiter" key would be required.

These errors are examples of negative transfer of learning from one task to another, for the two ways of doing the calculation are really two separate tasks from the point of view of the non-deterministic modeller. For a larger example, the calculations of Figures 3, 4, and 5 were concatenated together and run through the modeller with the same parameters as before. The results were much worse than for the problems considered separately. There were 484 good predictions and 9 bad (incorrect) ones out of a total of 643 lexical items. Note that less than 2% of predictions are incorrect.

This figure can be reduced even further by changing the parameters of the modeller. For example, using a length-5, -6, or -7 model instead of the length-4 one increases the number of correct suggestions slightly to 510, and decreases the number of incorrect ones to 3 (0.6%). A single error occurs at the transition between each problem; one between the calculation shown in Figure 3 and that of Figure 4, one between those of Figures 4 and 5, and one between the two parts of Figure 5 (which as noted above are really separate tasks). No suggestions were made for the remaining 130 lexical items. Over the three problems treated individually (as shown in Figures 3-5), a total of 119 items are not predicted, and thus the penalty paid for combining the problems into one behaviour sequence is quite small.

| Operator presses | Calculator indicates |
|---|---|
| .1 mc m+=+/- + 1 = exp x mr = | [answer] |
| .8 exp x 0.2 = | [answer] |
| .3 mc m+= | +/- + |
| 1 | = exp x mr = [answer] |
| .4 | exp |
| <DEL> mc | m+= +/- + 1 = exp x mr = [answer] |
| .5 exp | x |
| .5 | = [answer] |
| .4 | mc |
| <DEL> exp | x |
| .6 | = [answer] |
| .7 mc | m+= +/- + 1 = exp x mr = [answer] |
| .8 | exp |
| <DEL> mc | m+= +/- + 1 = exp x mr = [answer] |
| .1 exp | x |
| .9 | = [answer] |
| 1.1 mc | m+= +/- + 1 = exp x mr = [answer] |
| -.2 exp x 1.2 | = [answer] |
| 1.3 mc | m+= +/- + 1 = exp x mr = [answer] |
| 1.4 mc | m+= +/- + 1 = exp x mr = [answer] |
| -.5 exp x 1.5 | = [answer] |
| -1 exp x 2 | = [answer] |
| 2.5 mc | m+= +/- + 1 = exp x mr = [answer] |
| 3 mc | m+= +/- + 1 = exp x mr = [answer] |
| -3 exp x 4 | = [answer] |

Figure 6. Evaluation of x exp(1-x) using two different methods

## The user interface

As was noted above, there are severe problems in engineering such an interface so that it can help the user in an unobtrusive way. Nothing is more infuriating than an assistant who keeps giving wrong hints!

When the system we have described makes mistakes, it makes them in a way that seems to the user to be erratic and capricious; thus breaking one of the cardinal rules of interactive programming — that the user should see the system as predictable and reliable (Gaines and Facey, 1975). Thus we have aimed for a very small number of errors, and a substantial amount of help from the modeller — even over a short interactive sequence.

It is possible to moderate the operation of the non-deterministic modeller in two different ways, and these could be provided to the user in the form of

(i) an under-confident/over-confident control
(ii) a complex-model/simple-model control.

The first is provided by altering the number of times a tuple is seen before the modeller ventures a suggestion; while for the second, the parameter k of the model is adjusted. Whether such controls should be provided is an open question at present. It is interesting to note that they are a form of "phatic" communication (Jacobson, 1960), which is a mode that heretofore has played little part in man-machine dialogue (Witten, 1980). A different communication modality, such as a limited word recognizer, could be an ideal medium for these controls.

## Conclusions

This paper has shown how systematic processing of a "history list" of previous interactions can be used to predict future entries. Some time ago, artificial intelligence researchers discovered the potency of a history list of subgoals attempted during the solution of a problem. (Recall the "Why did you pick up the green pyramid?" type of question in Winograd's (1972) famous dialogue.) Explicit reference to the history list is permitted in some recent command language interpreters (eg Joy, 1979). However, we propose here implicit use of the history list for providing assistance

to the casual user.

Despite the fact that the modeller has no prior knowledge of the syntax or semantics of the dialogue, it is remarkably successful. Three short repetitive problems for an electronic calculator were examined. Although the problems are quite different, they were concatenated into a single input sequence. Even with the default settings for the modeller (which were intended for isolated problems), over 75% of the sequence elements were predicted; and less than 1.5% of incorrect predictions were made. There is always a temptation to adjust the parameters of the modeller in retrospect to give good performance. When this was done, nearly 80% correct predictions were achieved with an error rate of under 0.5%. However, the reaction of casual users to the scheme has not been obtained: clearly this is a next step.

Like much research, this project raises many more questions than it answers. For example,

-- should the complex/simple-model parameter for the raw behaviour sequence be different from that of the model where numeric tokens are mapped into <NUM>?
-- should the under/over-confident parameter be different for the two models?
-- if the modeller made several models with different values of k, could it choose between them on an adaptive basis to improve its predictions?

These are system-theoretic questions, which we hope to resolve using arguments based on the distribution of real-world data for calculator interactions.

Finally, it is crucial that if such a modeller is employed, it works instantly. It would be extremely irritating to have suggestions for a token made after it had actually been typed, due to non-realtime response! The technique is suited to powerful, personal computers; to the home or office environment; to the microcomputer age, where casual users must employ sophisticated information-processing tools.

Acknowledgements

It is a great pleasure to acknowledge the influence and encouragement of Brian Gaines, John Cleary, and especially John Andreae. The techniques described here (but not the applications) have their origin in Andreae (1977).

References

Amarel, S. (1971) Representations and models in problems of program formation, in Machine Intelligence 6, edited by B.Meltzer and D.Michie, 411-466. Edinburgh Univ Press.

Andreae, J.H. (1977) Thinking with the teachable machine. Academic Press, London.

Anderson, D.B. (1980) Programming in the home of the future, Int J Man-Machine Studies 12, 341-365.

Biermann, A.W. (1972) On the inference of Turing machines from sample computations, Artificial Intelligence 3, 181-198.

Cuff, R.N. (1980) On casual users, Int J Man-Machine Studies 12, 163-187.

Gaines, B.R. and Facey, P.V. (1975) Some experience in interactive system development and application, Proc IEEE 63, 894-911.

Gaines, B.R. (1976a) On a danger in the assumption of causality, IEEE Trans Systems, Man, and Cybernetics SMC-6, 56-59.

Gaines, B.R. (1976b) Behaviour/structure transformations under uncertainty, Int J Man-Machine Studies 8, 337-365.

Jacobson, R. (1960) Linguistics and poetics, in Style in language, edited by T.A.Sebeok, 350-377. Wiley, New York.

Joy, W. (1979) An introduction to the C shell, Computer Science Division Report, Univ of California, Berkeley.

McNaughton, R. and Papert, S. (1971) Counter-free automata. MIT Press, Boston, Mass..

Robinson, J.A. (1965) A machine-oriented logic based on the resolution principle, J Association for Computing Machinery 12, 23-41.

Wang, H. (1960) Toward mechanical mathematics, IBM J Research and Development 4, 2-22.

Winograd, T. (1972) Understanding natural language. Academic Press.

Witten, I.H. (1979) Non-deterministic modelling of behaviour sequences, Int J General Systems 5, 1-12.

Witten, I.H. (1980) Semiotics in the real world, International Congress on Applied Systems Research and Cybernetics, Acapulco, December.

Witten, I.H. (1981) Some recent results on non-deterministic modelling of behaviour sequences, Proc Society for General Systems Research Conference, Toronto, 265-274.

Zloof, M.M. (1977) Query-by-example, IBM Systems Journal 16, 324-343.