## Introduction

What does it mean to model behavior? First, suppose we have a discrete-time sequence of symbols that represent observations of a system under study — call this a *behavior sequence*. The symbols may be tokens drawn from a finite alphabet, or be real numbers, or constitute structured descriptors of some kind. They may represent actions in a story (or in real life), operations performed on a purchase requisition as it wends its way through an organization, or user input to a computer system. Typically the behavior sequence will be quite long — perhaps hundreds or thousands of symbols. A *causal model* of a behavior sequence is a representation of it which is useful for extrapolating into the future, or for interpolating parts that are unknown, or for identifying which symbols have high information content, or for producing a summary of the sequence. Often a model is a finite-state automaton, although it can assume other guises (for example, context models which store a set of subsequences of the behavior, or linear models which represent it as a set of weighting coefficients). Frequently the model can be viewed as a compact, approximate representation of the behavior, much smaller than would be needed to regenerate the sequence exactly.

To model behavior is to take a behavior sequence and build a causal model of it — an important kind of inductive inference. Certain kinds of sequence are very easy to model. For example, if the behavior is generated by a deterministic finite automaton, a causal state model can be constructed which is equivalent to the generator and hence predicts the sequence perfectly. Similarly, a sequence of real numbers which are values of a polynomial can be modeled by finite differences. However, if the behavior sequence is noisy or generated randomly from a non-deterministic automaton, the model itself must be non-deterministic†. Then the problem of causal modeling becomes more difficult. If an *optimal* model is sought (say one with a specified number of states that best fits the behavior in an entropy sense), the problem is very difficult indeed.

Why model behavior sequences? Knowledge — lots and lots of it — is the key to intelligent behavior. Many knowledge-based computer programs embody finite-state models which represent procedural knowledge, and use them to predict or interpret new observations. In most cases it would be of great value if the program could infer models automatically from observations of behavior. From the practical viewpoint, this could mean that the programmer or expert creating the system no longer has to try to recast his intuitive knowledge of procedures into formal, usually highly detailed, state machines. Instead he need only provide examples of the behavior being modeled. Moreover, the machine can continually update the model as it interacts with the environment, and thus (perhaps) improve performance as it gains experience. From a theoretical viewpoint, automatic modeling allows one to investigate the use of much more detailed models than were possible before. It is an open question whether this will improve performance (by providing a more specific context in which to interpret events) or worsen it (escalating the search problem by burying salient features within a morass of detail). No doubt it will be a bit of both. But experience in artificial intelligence tends to support the belief that performance increases as knowledge is added. Consequently models which are created automatically could be superior to their hand-crafted counterparts.

This paper takes a wide-ranging look at the problems involved in building causal models of discrete sequences of symbols, and surveys methods of modeling. Having established techniques for representing knowledge and reasoning with it in ways that can rival expert decision-making, the field of artificial intelligence is now turning attention to knowledge acquisition. Inductive inference — learning — is a key component of this enterprise; sequence modeling is a fundamental aspect of induction. Our intention in this paper is to review the problem of sequence modeling, take stock of the present state of the art, and assess what research directions hold most promise for the future.

The organization of the paper is as follows. The following section motivates the problem and provides a sharper context for thinking about sequence modeling by reviewing two areas of application of state-oriented sequential models in current AI systems. The following section discusses some dimensions along which modeling problems can vary. This gives an idea of the scope of sequence modeling and the range of situations to which it can apply. Perhaps the most important dimension is the structure of the alphabet of behavior

† So it must even for a deterministic sequence, if the model is constrained to have fewer states than the generator.

symbols (eg does it support a measure of similarity? — is it continuous?). Next, we survey methods which have been used for building models, broken down according to the kind of alphabet for which they are suitable. Some modeling methods rely on the user to help point out the structure of the sequence, and two such systems, in which the user intervenes in completely different ways, are reviewed. A key question for the future is the inference of goals and plans from observed behavior. Procedural representations are much less flexible than the goal-oriented approach which people seem to favor. However, inferring goals and plans is a very difficult problem, for the modeler must move up to a higher level by recognizing the intent behind acts. The final section discusses this issue and speculates on how such inferences might be accomplished.

## State-oriented models in AI systems

Many knowledge-based computer programs embody finite-state models which represent sequences of events and actions, although the formalisms adopted are generally *ad hoc* and not expressed in systematic terms. Such models are used to predict or interpret new observations. However, few current systems make any attempt to infer their models from observations. Instead, they are invariably created by the programmer explicitly for his program's use.

This strategy is in keeping with the last decade's emphasis on explicit representations of knowledge, constructed by the programmer. In other words, relevant knowledge is "programmed in" to a knowledge-based system. The approach has resulted in the very successful methodology of expert systems which has gained commercial respectability for AI. However, the acknowledged bottleneck of knowledge acquisition is forcibly turning attention to methods of automatic inference. The present paper is concerned with the inference of models which account for observed sequences of behavior.

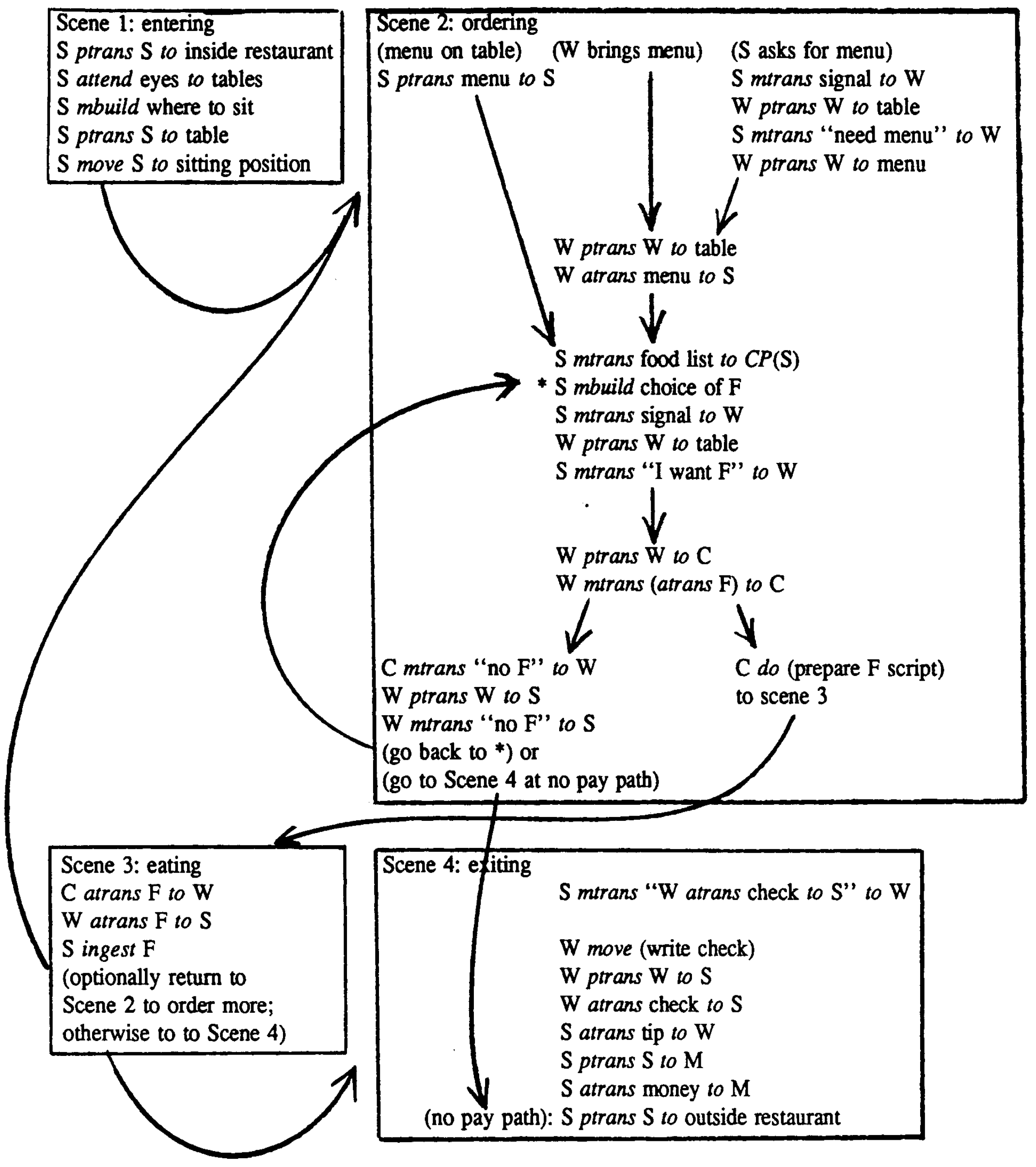
In order to provide a focus for thinking about sequence modeling, this section briefly reviews two applications of state-oriented sequential models in AI systems — scripts and office procedures — and gives examples of actual models which have been used. In neither of these domains have serious attempts been made to infer models in bottom-up fashion from examples of behavior, although it is clear that this is highly desirable, if only to ease the problem of knowledge acquisition.

*Scripts in natural language understanding.* The large body of research on natural language understanding which is based on the "conceptual dependency" theory of language has achieved impressive results†. Much of this success is due to the implantation of canned knowledge of stereotyped sequences of events into the systems to allow them to make sense of narratives which describe such events. This began as the notion of *scripts*: data structures that organize commonly shared knowledge about routine activities like eating at a restaurant, taking a bus, and so on (Cullingford, 1978). The most important part of a script is a state model which records the order in which things happen. This often takes the form of an expected "main-line" sequence, augmented by minority paths — shortcuts, loops, and the like. Scripts also specify roles that the actors play, props that they use, and entry and exit conditions.

Figure 1 shows a restaurant script (coffee shop track) taken from Schank & Abelson (1977). The actions of the script are described in terms of the underlying acts that take place, rather than as surface words. These acts are primitives of conceptual dependency theory. For example, *ptrans* means to transfer physical location; *attend* means to focus a sense organ on a stimulus; *mbuild* means to construct new mental information from old; *move* means to move a part of one's own body; *mtrans* means to transfer mental information; *atrans* means to transfer possession; and so on. A conceptual dependency analyzer would take each sentence of a story and map it into these terms before attempting to apply a script.

† For example, see Wilensky's (1983) and Dyer's (1983) work.

| | | | |
|---|---|---|---|
| Props: Tables<br>Menu<br>F – Food<br>Check<br>money | Roles: S – Customer<br>W – Waiter<br>C – Cook<br>M – Cashier<br>O – Owner | Preconditions: S is hungry<br>S has money | Results: S has less money<br>O has more money<br>S is not hungry<br>S is pleased (optional) |



Scene 1: entering
S *ptrans* S *to* inside restaurant
S *attend* eyes *to* tables
S *mbuild* where to sit
S *ptrans* S *to* table
S *move* S *to* sitting position

Scene 2: ordering
(menu on table) (W brings menu) (S asks for menu)
S *ptrans* menu *to* S
S *mtrans* signal *to* W
W *ptrans* W *to* table
S *mtrans* "need menu" *to* W
W *ptrans* W *to* menu
W *ptrans* W *to* table
W *atrans* menu *to* S
S *mtrans* food list *to* *CP*(S)
* S *mbuild* choice of F
S *mtrans* signal *to* W
W *ptrans* W *to* table
S *mtrans* "I want F" *to* W
W *ptrans* W *to* C
W *mtrans* (*atrans* F) *to* C
C *mtrans* "no F" *to* W
W *ptrans* W *to* S
W *mtrans* "no F" *to* S
(go back to *) or
(go to Scene 4 at no pay path)
C *do* (prepare F script)
to scene 3

Scene 3: eating
C *atrans* F *to* W
W *atrans* F *to* S
S *ingest* F
(optionally return to
Scene 2 to order more;
otherwise to to Scene 4)

Scene 4: exiting
S *mtrans* "W *atrans* check *to* S" *to* W
W *move* (write check)
W *ptrans* W *to* S
W *atrans* check *to* S
S *atrans* tip *to* W
S *ptrans* S *to* M
S *atrans* money *to* M
(no pay path): S *ptrans* S *to* outside restaurant

Figure 1 Restaurant script from Schank & Abelson (1977)

```
Main-line:  null          Event 1  Receive Preliminary
                                   1.1  verify Preliminary
                                   1.2  create Application
                                   1.3  select Interviewer from INTERVIEWERS
                                   1.4a send Final-application-forms
                                   1.4b send Interview-report to Interviewer
                          → waiting
            waiting       Event 2  Receive Final-application-forms
                                   2.1  verify Final-application-forms
                                   2.2  send Final-application-forms.check to MIT-OFFICES (Name="Cashier")
                          → complete
            complete      Event 3  January 20, year-applying-for
                                   3.1 ... specify actions, similarly to 1.1, 1.2, ... above
                          → reviewed
            reviewed      Event 4  February 20, year-applying-for
                                   4.1 ... specify actions
                          → admitted, done, reviewed (depending on actions)
            admitted      Event 5  Receive Reply
                                   5.1 ... specify actions
                          → done, accepted (depending on actions)
            accepted      Event 6  July 15, year-applying-for
                                   6.1 ... specify actions
                          → coming
            coming        Event 7  September 30, year-applying-for
                                   7.1 ... specify actions
                          → done
            reviewed      Event 8  April 15, year-applying-for
                                   8.1 ... specify actions
                          → done, admitted (depending on actions)

Variations:

 1. where Early-decision = "T"                        : variation for early-decision applicants
    add      reviewed     Event 9  November 30, year-applying-for - 1
                                   9.1 ... specify actions
                          → admitted, reviewed (depending on actions)
    replace  Event 3 by   Event 3  November 20, year-applying-for - 1

 2. where Foreign = "T"                               : variation for foreign applicants
    delete   Event 1, Step 1
    add      null         Event 10 Receive Preliminary
                                   10.1 ... specify actions
                          → wait-FSO-ok
             wait-FSO-ok  Event 11 Receive Preliminary-FSO-reply
                                   11.1 ... specify actions
                          → waiting
    replace  Event 3 by   Event 3  January 31, year-applying-for
```

Figure 2 Procedure admit-freshman-class from Kunin (1982)

| accepted | accepted, but refused | rejected | waitlisted, then accepted | waitlisted, then rejected |
|---|---|---|---|---|
| see 1 | see 1 | see 1 | see 1 | see 1 |
| do 1.1 | do 1.1 | do 1.1 | do 1.1 | do 1.1 |
| do 1.2 | do 1.2 | do 1.2 | do 1.2 | do 1.2 |
| do 1.3 | do 1.3 | do 1.3 | do 1.3 | do 1.3 |
| do 1.4 | do 1.4 | do 1.4 | do 1.4 | do 1.4 |
| see 2 | see 2 | see 2 | see 2 | see 2 |
| do 2.1 | do 2.1 | do 2.1 | do 2.1 | do 2.1 |
| do 2.2 | do 2.2 | do 2.2 | do 2.2 | do 2.2 |
| see 3 | see 3 | see 3 | see 3 | see 3 |
| do 3.1 ... | do 3.1 ... | do 3.1 ... | do 3.1 ... | do 3.1 ... |
| see 4 | see 4 | see 4 | see 4 | see 4 |
| do 4.1 | do 4.1 | do 4.1 | do 4.1 | do 4.1 |
| do 4.2 | do 4.2 | do 4.2 | do 4.2 | do 4.2 |
| see admit | see admit | see reject | see waitlist | see waitlist |
| do 4.3 | do 4.3 | do 4.3 | do 4.3 | do 4.3 |
| do 4.4 | do 4.4 | do 4.4 | do 4.4 | do 4.4 |
| see 5(ok) | see 5(no) | done | see 8 | see 8 |
| do 5.1 | do 5.3 | | do 8.1 | do 8.1 |
| do 5.2 | do 5.4 | | see admit | see reject |
| do 5.5 | done | | do 8.2 | do 8.2 |
| see 6 | | | do 8.3 | do 8.3 |
| do 6.1 ... | | | see 5(ok) | done |
| see 7 | | | do 5.1 | |
| do 7.1 ... | | | do 5.2 | |
| done | | | do 5.5 | |
| | | | see 6 | |
| | | | do 6.1 ... | |
| | | | see 7 | |
| | | | do 7.1 ... | |
| | | | done | |

Table 1 Example traces through the procedure of Figure 2

4.2 group applications into {admit, reject, waitlist}

which creates tokens *see admit*, *see reject*, and *see waitlist* in the traces. The crux of the applicant's letter accepting or refusing an offer made to him is described by the tokens *see 5(ok)*, *see 5(no)*. The lines prefaced by "do" are all actions of the procedure in Figure 2.

Some simplifications have been made to Kunin's representation. The most notable are:

- Kunin allows the specification of constraints, like "Event 2 precedes Event 4". Violation of constraints constitutes an exception, and it is possible to specify actions to take under such circumstances (like "send late-letter").

- Kunin permits parallelism between actions within a state, and includes a syntax for expressing it. This does not seem to cause a problem for inductive inference, for parallelism could be inferred from a reasonable number of example cases.

- Actions in Kunin's scheme can be complex and involve testing and branching. Again, this would appear as an elaboration of the state-machine generated by sequence modeling.

Capturing behavior sequences does not seem to present such difficulty in this application as with the natural language scripts above. The input to the modeling process would be traces such as those of Table 1. These could be captured from manual operation of the admissions procedure before it had been formalized. Of course, it would be unreasonable to expect such "clean" traces as those shown. The extent of discrepancies from the formalized procedure is a question that can only be answered experimentally.

*Discussion.* It would be attractive to be able to form structural models such as those illustrated in Figures 1 and 2 from samples of observed behavior, and methods for doing so are surveyed in the next section. However, it is important to realize that modeling a raw sequence of observed events occurs at a fairly low epistemological level (Klir, 1976). The desirability of going higher, into levels involving goals and intentionality, has been noted separately in the literature for the example areas reviewed above.

Schank and his co-workers discovered that representing knowledge at the level of primitive events led to proliferation of scripts. To avoid this, a representation of the *goals* that occurred in a stereotyped situation was needed, together with various *plans* that could realize each goal (Schank & Abelson, 1977). Then a script constitutes many goals in sequence, and the actor chooses a suitable plan for each, resulting in a much richer set of possibilities at the event level than the kind of behavior-sequence model shown in Figure 1.

In the office systems area Suchman has been foremost in dispelling the myth that real work consists of blindly executing pre-specified procedures (Suchman 1983, 1985). She argues that procedures are at most a *post hoc* rationalization of what actually happens, which really has the character of problem-solving in terms roughly equivalent to goals and plans. Much office work is concerned with deciding how to handle exceptions to procedures — so much that Suchman considers exceptions to be normal and the procedural model an illusion. This argues strongly against the kind of representation depicted in Figure 2.

The idea of modeling action sequences has been rejected in favor of a goal-based approach in other areas too. For example, programming by example is another endeavor which involves modeling behavior sequences. At its simplest, it just involves recording a sequence of commands issued by the user, so that the sequence can be played back later to re-execute the task. The sequence constitutes a program, and is constructed using the normal user interface of the system. Generally, however, much more is required than the availability of "start-remembering", "stop-remembering", and "doit" actions. Control constructs such as iteration and conditional execution must be inferred from user actions or stated explicitly by the user. Examples of programming by example can be found in Andreae (1984), Gaines (1976a), Halbert (1981, 1984), Witten (1981a).

But some programming-by-example work advocates a goal-based approach instead of working bottom-up from action sequences. In his study of editing by example, Nix (1983, 1984) concluded that the sequence of editing actions that occurs is the wrong level to approach the problem. Command traces are noisy (because the user makes errors) and idiosyncratic (because there are many ways to perform a given editing task). Consequently Nix based his work on first analyzing the difference between the original file and the edited version, and then synthesizing a suitable transformation from scratch. This is quite an unusual case because the goal is quite explicit — to achieve the given transformation — and can be determined independently of the example actions used to realize it. In most applications of programming by example, determining the user's goal is a major problem.

While sequence modeling may be useful in many applications, the views sketched above all indicate that its power is likely to increase when it is combined with domain knowledge to provide a higher-level representation of the intentionality which underlies the overt behavior. We return to this question in the final section of the paper.

## Dimensions of the problem

There are a number of dimensions to the modeling problem which strongly affect the ease with which causal models can be inferred to account for particular behavior sequences, and the techniques which can be used. First we list the dimensions and then discuss them individually.

- What is the purpose of modeling?
- Is the model to be exact or approximate?
- What kind of symbols are used to express the behavior?
- Is modeling done once only when the complete behavior sequence is available, or incrementally as new symbols appear?
- How large is the problem?

*What is the purpose of modeling?* As noted in the Introduction, there are several reasons for modeling behavior sequences:

- to extrapolate behavior into the future
- to interpolate parts of it that are unknown
- to pinpoint parts with high information content (or "surprise" value)
- to provide a compact representation which summarizes what has happened.

Predicting the future is appropriate in prescriptive applications which determine, or suggest, what to do next. An excellent test of the predictive power of a model is to use it for data compression, for the reduction achieved is a quantitative measure of success at guessing. For this reason we make occasional reference to data compression applications throughout the paper.

Interpolating missing information allows a partially-specified behavior sequence to be fleshed out, perhaps constituting an "explanation" of it. In story understanding, for example, certain key events are mentioned explicitly and the reader is expected to infer the intervening actions. In order to interpolate missing parts of a behavior sequence, the modeler must identify the sequence of states the model passes through from the partial information available to it. This is a very powerful way of using prior knowledge or past experience to fill in missing detail. However, I know of no research on inductive inference of models from partially-specified behavior sequences.

Models are useful for predicting expected or "default" information based on extrapolating history. Often it is *departures* from the default which are most interesting and convey most information. Modeling allows machines to share our surprise at an unexpected turn of events.

In these three situations the internal representation of the model is irrelevant to the user: only its success in prediction or identification is of interest. The fourth reason above creates a different requirement, that the entire model be presented or "explained" to the user as a way of summarizing the behavior sequences it represents. Then it *is* necessary to consider the internal representation. This issue becomes of the greatest importance when considering whether context-based or highly redundant state-based models will suffice, or whether attempts should be made to minimize state models by coalescing equivalent states. For example, in data compression applications only predictive power is of importance, and huge models can be constructed which considerably reduce the bandwidth required for transmission (Cleary & Witten, 1984; Cormack & Horspool, 1985). Huge models may not be tolerable in expert system applications which place emphasis on the ability to explain decisions and summarize for the user the knowledge structures which support them.

*Is the model to be exact or approximate?* If the sequence being modeled has been generated by a deterministic finite-state system, the classical technique of Nerode equivalence (Nerode, 1958) will find an exact causal model of it. However, this method breaks down completely when noise is present (Gaines, 1976b) — the universe becomes incredibly complex and models of it nonsensical if determinism is assumed in the face of even the slightest trace of acausal behavior. Or consider a sequence of real numbers that are values of a polynomial at equally-spaced ordinates. This can be modeled very simply by finite differences. For high-order polynomials, however, the method is unstable if any noise (eg rounding error) is present.

Consequently a robust modeler is forced to abandon the deterministic approach, since if nothing else, occasional errors in the input will introduce acausality. There is no real alternative to approximate models.

*What kind of symbols are used to express the behavior?* The alphabet of symbols, or "descriptors", from which elements of the behavior sequence are drawn, may have structure of its own. Following Michalski (1983), we distinguish three types of alphabet:

- categorical
- metric
- structured.

Categorical descriptors are independent atomic symbols or names, with no more structure than the set. They are used in traditional automata theory, often with rather small alphabets.

Linear system theory uses descriptors which take on values in a metric space. In practice this is invariably the real line or a Euclidean space. Although an infinite alphabet is involved, modeling is simplified because there are richer possibilities for combining symbols. Arithmetic operators can be used for such purposes as averaging several different symbols into a representative value, reflecting relative importances with numeric weights, basing predictions on successive differences, etc. Moreover, there exists a generally accepted measure of approximation, the least-mean-square criterion. Unfortunately the assumption of linearity (with Gaussian noise) on which essentially all existing techniques depend rules out these models for most knowledge-based applications, and so they are described here only briefly.

The third kind of descriptor takes on values from a "generalization hierarchy". Knowledge-based systems frequently make use of such structures, which are typically sets partially ordered by a relation called "generalization", or more complex structures of classes under the inclusion relation and instances or members of the classes (Brachman, 1983). Such partial orderings induce a relation between descriptors based on whether they have a common generalization, and a partial function from related pairs to their most specific common generalization. Descriptors may be parametrized, giving a very large set of potential symbols. Another kind of structured descriptor is a vector whose components represent different aspects of behavior. The events in the script of Figure 1 are expressed as primitive actions (*ptrans*, *attend*, etc) with parameters (such as "S", "inside restaurant", "table", "menu") which are themselves taken from a generalization hierarchy. There are clearly limitless possibilities for structured descriptors.

*Is modeling once-off or incremental?* In some modeling situations it is sufficient to wait until the behavior sequence has been completed, and then to create an *a posteriori* model of it which will never be changed again. More often, a model is sought which is updated incrementally as the behavior sequence unfolds. Of course it is possible to remodel from scratch whenever an event is encountered which is inconsistent with the current model, thereby using the *a posteriori* technique to maintain an evolving model; but this requires complete storage of past history, and moreover implies a heavy time penalty in remodeling.

One option with incremental modeling is to let the model itself stand as the only memory of the behavior sequence which is retained, so that when an inconsistency is discovered it must be rectified by an incremental change to the current model. A second is either to store past history separately, or to augment the model with enough information to recreate the behavior exactly. A third is to avoid inconsistency altogether by insisting that enough evidence is available when building the model that ''mistakes'' do not occur. Finally, one can use a control mechanism such as Prolog's backtracking to hide the details of how the behavior sequence is remembered.

*How large is the problem?* The size of modeling problems can vary enormously. For data compression, behavior sequences with up to 0.5 Msymbols have been used, with models occupying several Mbytes (Cleary & Witten, 1984). In other work (eg Gaines 1976a) on problems of a different character, useful results are obtained with models having only a handful of states. Since non-deterministic modeling is not a well-understood problem, it is not surprising that most research involves relatively small models.

## Methods of attack

This section reviews methods which have been proposed and used for modeling behavior sequences. The intention is to take stock of the state of the art, not to provide a tutorial survey — for this reason the reader who is unfamiliar with the workings of individual methods will have to consult the references for detailed descriptions. It is convenient to break the review down into subsections according to the character of the alphabet — categorical, metric, and structured. A final subsection looks at how the user can intervene to increase the power and reliability of modeling.

*Categorical descriptors.* A number of methods have been investigated for non-deterministic finite-state modeling of behavior sequences with categorical descriptors.

*Enumeration and evaluation.* One can enumerate all automata which could have generated a given behavior sequence, beginning with the simplest (generally the smallest). This has been developed into a well-defined methodology by Gaines (1976a, 1977), who defines the subset of *admissible* models, given a measure of poorness-of-fit of a model to the behavior sequence and a criterion of model simplicity. The admissible subset contains only models which cannot be improved in simplicity without increasing poorness-of-fit, and cannot be improved in goodness-of-fit without increasing complexity. If a *probabilistic* model is desired the problem may appear to be insoluble, for the space of possible models is continuous rather than discrete; but this is not so, for an optimal assignment of probabilities to the branches of a non-deterministic model may be made by running the behavior sequence through the model and making frequency counts (Gaines, 1976a). As in other domains (eg coding, see Cover, 1973), enumeration is completely impractical for large problems, but provides a useful standard by which to compare other, heuristic, methods. In the case of non-deterministic modeling of behavior sequences, enumeration has super-exponential complexity in the size of the model†.

† If determinism is assumed the problem is much easier. For example, the finite-state control for Turing machines has been inferred using a data-driven, simplest-first search with backtracking; see Biermann (1972).

*Reduction and evaluation.* An alternative to building and evaluating successively more elaborate models is to start with a very large one and reduce it. The initial model could be a direct representation of the sequence itself (that is, an $n$-state model of an $n$-element behavior sequence); if this is too large then a fixed-context method with long context could be used (see below). Reduction can be done by heuristic methods (eg Evans, 1971), or by evaluating each reduced model according to a goodness-of-fit yardstick (Witten, 1981b). The most straightforward way to determine which states should be merged is to evaluate the models which result from merging each pair of states and select the pair with the best evaluation. This gives polynomial rather than exponential complexity in the number of states, but sacrifices the optimality which enumeration guarantees.

*Fixed-context methods.* Fixed-context methods of building a causal model of a behavior sequence characterize the sequence by the set of $k$-tuples of symbols which occur in it, for some given "context" length $k$. Different techniques have been developed by a number of researchers. Biermann & Feldman's (1972) method is suitable for approximate modeling of deterministic sequences, but breaks down in the face of non-determinism. Andreae (1977) has pioneered methods of fixed-context modeling which literally store all $k$-tuples in a hash table (for rapid access). The same information can be recoded as a state-transition model and reduced to a near-minimum number of states; this can be accomplished without sacrificing the possibility of incremental modifications to the model when new behavior is encountered (Witten, 1979).

*Variable-length matching.* Fixed-context methods effectively use $k$-tuples for prediction by matching the first $k-1$ symbols of each tuple in memory against the last $k-1$ symbols seen. The final $k$th symbol of those that match are taken to be predictions of the next symbol. However, if no tuple in memory matches the last $k-1$ symbols seen, there is no prediction. The method of variable-length modeling weakens the match criterion under these circumstances by reverting from $k$-tuples to $(k-1)$-tuples, and then (if necessary) to $(k-2)$-tuples, and so on until a match is found. This makes it much more likely that predictions are actually made, especially in the early stages of modeling when little behavior has been seen. Limited-context methods with variable-length modeling can exhibit remarkable predictive power in data compression applications, which provide an excellent way of evaluating predictions quantitatively (Cleary & Witten, 1984).

*Dynamic Markov modeling.* A method of incrementally updating a state model directly has recently been developed for data compression (Cormack & Horspool, 1985). This maintains a non-deterministic model which can generate all sequences, with the observed frequency count recorded on each state transition. A state is "cloned" — that is to say copied — when

• it leads non-deterministically to two or more successor states
and • it is entered from two or more predecessor states.

In other words, when the model enters that state, some contextual information is lost which may help resolve the ambiguity of successor; and cloning the state gives the opportunity to recreate that context. One clone is selected for one predecessor state, the other for the remaining predecessors, and the frequency counts associated with the original state are split between the two new versions in an appropriate ratio. To start, the model is chosen as the trivial one-state one, and the cloning process causes it to grow. Experiments have shown that it is beneficial to clone states as soon as practicable (ie as soon as they have been entered and exited a couple of times from different places). This makes the model grow very quickly. However, no notion of model contraction has been investigated: the model expands continually until it reaches a maximum upper limit imposed by system resources. This technique has been evaluated on data compression problems and has much the same predictive power as the variable-length matching method above.

*Metric descriptors.* Behavior sequences whose symbols are real numbers are common in signal processing applications. The method of *linear prediction*, due to Wiener (1947) and resurrected by Atal & Hanauer (1971) for speech processing, is by far the dominant technique for modeling real-valued signals, and can easily be extended to the prediction of vectors as well (Ha & Witten, 1980). Linear prediction has proven advantages over other methods of signal analysis and low data-rate speech storage and synthesis. However, it seems too tightly-defined and simple to deserve the label "inductive inference", and the underlying assumption of linearity is inappropriate for most knowledge-based applications. A brief description is included here for completeness.

A linear predictor is a fixed-context model, with context length typically between 10 and 15 for speech applications, so that the most recent $k$ symbols seen are used to predict the next one. The prediction is a weighted sum of the context values. The weights are derived by minimizing the squared distance between the predicted values and the observed next values, over the past few (between 30 and 250 for speech) samples. Minimization is easily accomplished by partially differentiating with respect to each weight, and the result is a matrix equation involving correlation coefficients of the recent observed behavior. The matrix has different properties depending on whether the "recent observed behavior" is terminated abruptly or tapered gradually with a windowing function, and various methods of efficient solution of the matrix equation have been proposed.

*Structured descriptors.* Structured descriptors lie between categorical and metric ones. Work with the former generally uses a rather small ($\leq$256 symbols) alphabet of symbols. The latter invariably involves the infinite alphabet of real (or rational) numbers, although of course in practice they are quantized and discrepancies between the model and its implementation are treated as noise. Structured descriptors usually form a large, and in some cases (when descriptors are parametrized) infinite, set. Although they occur naturally in sequential modeling situations like those reviewed earlier, surprisingly few methods are known for modeling with structured descriptors.

*Vector descriptors.* Klir and his colleagues (Klir 1975, 1976; Uyttenhove 1980) consider behavior sequences in which each symbol is a vector whose components represent different aspects which are sampled simultaneously. In this work the components are categorical descriptors and have no further structure. The limited-context modeling method is used, but the context does not necessarily have the same length for each component. The problem is to find that pattern of context lengths for the components which has most predictive power; rather extensive searches are required for this.

*Distinguishing symbol types by confirmation level.* In an investigation of programming a simple electronic calculator by example, Witten (1981a) noted a difference between the way numbers and operators are used in calculation sequences. When numbers constitute "inputs" to the calculation they are inherently unpredictable, which contrasts with the case of operators or numbers used as "constants". Consequently the system was constructed to be more conservative about predicting numbers than operators. 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). This *ad hoc* policy of altering the level of confirmation required depending on the type of symbol is a simple but effective way of dealing with prior knowledge about the character of descriptors.

*Constraint-limited generalization.* Andreae (1984) describes a system for acquiring procedures from examples which makes use of an explicit generalization hierarchy for descriptors. It works as follows. The problem is to meld two example traces of execution of a procedure into one state-transition representation which encompasses them both. If two descriptors are identical they are assumed to emanate from the same state — a degenerate case of fixed-context modeling with a context length of zero. In particular, each trace begins with a *start* descriptor which forces the initial states of the two sequences to be merged, and ends with *stop* which merges final states. Thus a non-deterministic state model is built from the traces. However it is usually little use for prediction, since descriptors are rarely identical (apart from *start* and *stop*). The second stage examines states which are different but whose predecessor states are the same, or whose sucessor states are the same, and attempts to unify the descriptors associated with each — fixed-context modeling with a context length of one. If they can be unified, then the states are coalesced. Unification is done through the generalization hierarchy, and succeeds if the two descriptors have a common generalization. Finally, a third stage examines states which are different but whose predecessors and successors are the same, and again attempts to unify the descriptors but this time with a more liberal unification procedure. This unification uses the same generalization hierarchy as before, but involves synthesizing functions which unify parameters of the descriptors. Synthesis is accomplished by searching a function space, possibly utilizing numbers which have been "mentioned" in nearby descriptors as constants in the function. This is a very expensive process in terms of search time. However, it is only undertaken when there is strong evidence (identical predecessor and successor states) that the descriptors match†.

Using limited-context methods on sequences with structured descriptors, there are three dimensions in which different points in a behavior sequence can be tested for similarity. The first is the *spatial* dimension — the extent to which the descriptors match in the generalization hierarchy. The second is the *sequential* dimension — whether the preceding (or even succeeding) context is the same. The third is the dimension of *degree of confirmation* — how often the sequences have occurred before. If the points match sufficiently well on these counts, then they can be merged in the state diagram. Andreae's constraint-limited generalization stresses the spatial dimension, merging states immediately if they produce the same symbol. It makes limited use of the sequential dimension, using a context-1 match in either forward or backward directions to permit a looser spatial match, or in both forward and backward directions to permit an even looser match. Witten's system made simple use of the degree of confirmation dimension to distinguish between "constants" and "variables". It is easy to model sequences based on any mix of the three dimensions: what is lacking is a general theory of the trade-offs involved.

## *Symbiotic modeling: involving the user.*

Sequential modeling can be made easier and more reliable by allowing the user to intervene in the modeling process. Two rather different projects have shown how useful this can be in the sequential and spatial domains.

*Increasing the power of limited-context modeling.* Limited-context modelers discard all information in a behavior sequence except the set of *k*-tuples which occur in it. This restricts their power to the "non-counting events", a strict subset of finite-state events (McNaughton & Papert, 1971). However, it is rather obvious that if appropriate extra information is made available to the modeler, the full power of finite-state events can be regained. In particular, supplying it with state markers which reveal the states passed through by a (non-deterministic) finite-state model gives the game away and makes the modeler's task trivial (Witten, 1981b). Moreover, state markers need only be supplied initially; once they have been assimilated the modeler is capable of regenerating them automatically and thereby keeping itself synchronized with the behavior sequence. It would be unusual to have available markers which precisely represent states. However, even partial information is useful, and since a limited-context modeler can learn non-counting events itself, only information pertinent to the "counting" component need be included. These ideas have been developed within the PURR-PUSS learning system of Andreae (1977). This achieves its power through a simple, uniform storage structure for fixed-length pieces of behavior, combined with a method of operator intervention which enables counting events to be learned. To allow the user to provide state-feedback, Andreae has introduced the idea of "multiple contexts", each context being an input stream, so that state information can be provided separately from the behavior sequence itself. This makes it easier for the identification procedure to take over the user's role and supply the state markers itself when it has learned how to do so.

*Data descriptions: user-controlled generalization.* In his study of programming by example for the Star office information system, Halbert (1984) concluded that it was simply not feasible in this domain to generalize user actions automatically. The problem is that when an icon is selected on the screen, the user's intention is concealed from the system. It can be guessed by inductive inference, particularly if the example is repeated several times in different ways; but the system can never be sure — especially in a rich problem domain where there are likely to be a number of reasonable alternatives. Consequently Halbert resolved to let the user choose how descriptors are generalized by presenting a "data description" property sheet for him to fill in. For example, a document can be described by *name* (in which case a name pattern may be specified as a regular expression), by *position* (eg first-in-folder or last-in-folder), by *prompt* (in which case the user is prompted every time the item is required), or as a special type *any* for use in set iteration. Other kinds of descriptor offer more

† In fact, the procedure does not require both predecessor and successor states to be identical, but is applied to parallel chains of states which begin and end in the same state and, if successful, merges corresponding members of the chains. Consequently the merging is only done if both predecessor and successor states end up being identical.

sophisticated options. This is not pure programming by example, because the user must interrupt his example to clarify the generalization mode. However, having introduced the idea of user intervention to specify generalization, Halbert found he could make more extensive use of it. For example, set iteration is specified by selecting the *any* generalization mode. As one generates the behavior sequence which constitutes the example that defines a procedure, a written representation of the procedure appears step by step on the screen. The scope of iteration is indicated by selecting the appropriate part of the written procedure (with a mouse). Conditionals are added in a somewhat similar way. Already-recorded programs can be corrected or altered by editing them using the written representation. The result is a powerful and easy-to-use casual-user programming system, but one which is somewhat removed from the behavior-sequence modeling paradigm of programming by example.

## Inference of goals and plans

The next step in sequential modeling is to allow the modeler to move to a higher level by recognizing the intent behind the acts, and infer the teleological structure of the input. This is indeed an ambitious aim, which needless to say has not yet been achieved! But the time is ripe to speculate on what may be involved in the inference of purpose. And although it seems next to impossible to divine the more grandiose of human aspirations from mere observations of behavior, there are a host of modest goals which present an easier target. Obviously we must assume that domain knowledge is available about the preconditions and consequences of individual acts which are represented by descriptors in the behavior sequence, and probably also about the overall "top-level" goals of actors. Moreover, it is clear that the task will be much easier if the user is allowed to intervene in the modeling process to "explain" aspects of the behavior sequence — the benefits of symbiotic modeling become even more striking at this level.

*What are goals and plans?* A goal is an intention, the object of one's effort. Some goals — satisfaction of hunger, reproduction, self-esteem, etc — are top-level ones that stem from the intrinsic nature of humanity. Others, such as changing location, or gaining control of some object, are desired changes in state of the world. These are subordinate goals which are executed as necessary parts of a higher-level plan because they are needed as preconditions by other parts of the plan (Schank & Abelson, 1977). Even at the lower levels, it is not easy to specify unambiguously what the goals are — we are now dealing with intentionality rather than the "objective reality" of behavior sequences. The nature of goals is inextricably intertwined with their realization in terms of plans.

With each goal are stored a number of plans which represent different ways of achieving it. A plan is a sequence of subordinate goals and, at the lower levels, basic actions, which will (probably) achieve the higher-level goal. The realization of goals through hierarchical decomposition into plans containing lower-level goals is somewhat analogous to the decomposition of a program into a hierarchical structure of subprograms, with the following important differences:

- for each goal (ie subprogram name) there exist several alternative plans (ie subprogram definitions) for realizing that goal;
- the order in which goals are executed may not necessarily be completely determined in advance (although it may be constrained);
- goals include an explicit representation of themselves from which it can be tested if they are already satisfied, or if a plan which purports to realize that goal has succeeded;
- plans include an explicit representation of the preconditions which must be met if they are to work, the postconditions which they are designed to produce, and any side effects they may generate — ie they satisfy the "Strips assumption" that all changes they induce in the state of the world are listed in the description of the plan.

What is meant by "goals" in our example domains of natural-language understanding and office information systems? For the restaurant script of Figure 1, the following goals can be identified†:

```
satisfy hunger
    go to restaurant
    get seated
    order
        get menu
        choose
        communicate choice
    get food
    eat
    pay
    leave
```

*Satisfy hunger* is a top-level goal which is intrinsic to the actor; the others are subgoals. The *order* subgoal is further elaborated into second-level subgoals, through choice of a plan. There are alternative plans for ordering in other kinds of restaurants. The *get menu* and *communicate choice* subgoals will themselves have a few associated plans. The other first-level subgoals can also be achieved by several alternatives (eg *go to restaurant* can be accomplished by walking, hitch-hiking, riding a horse). Achieving a goal is accomplished by selecting and executing one of the plans stored with it. For the admit-freshman-class procedure of Figure 2, part of the goal tree looks like this:

```
process a student
    get complete application
    make decision
    communicate and confirm decision
```

This is perhaps a better-defined example than the previous one, for it is a less familiar and natural activity.

*Why use goals and plans?* It was noted earlier that the move to higher levels of intentionality has been advocated apparently independently in each example domain. For example, Schank & Abelson (1977) argued that the old notion of scripts as the medium for the representation of sequential knowledge should be replaced by a more flexible representation involving plans and goals. In essence, this allows questions like "why did you do that" to be answered by reference to the goal stack along the lines of "because I was trying to do this". Contrast this with procedure-based scripts, where the answer might be "because that's what's usually done after ...".

In an office information system, the advantage of a goal-based over a procedure-based representation is seen to be that more unstructured tasks can be handled by establishing user goals and attempting to achieve them (Croft & Lefkowitz, 1984). The goal-oriented perspective does not commit one to a fixed sequence of actions. Any particular goal can be realized in a number of different ways (by different plans), including, perhaps, recourse to the user. If a plan fails to achieve the intended goal, this can be detected and alternative plans tried.

In both domains, systems have been developed which reason about goals and plans, and can even contemplate contradictions between goals and isolate inconsistencies between plans (Barber, 1983; Wilensky, 1983).

† This is by no means the *only* way of assigning goals to Figure 1; we noted above that goals depart from objective reality and, in a sense, exist only in the mind of the planner.

*Can goals and plans be inferred from observed behavior?* It seems an enormous leap from the behavior-sequence modeling techniques reviewed earlier to representations in terms of goals and plans. Suppose a script like that of Figure 1 has been constructed from observation of behavior. It will likely be considerably more complex, however; for all sorts of other events — relevant or irrelevant — will have occurred as part of the restaurant experience. Moreover, we must assume the existence of a large number of other scripts for other events as well. These scripts constitute the "input" to the goal/plan inference process. The extent to which goals and plans can be inferred in bottom-up fashion from event-based scripts is an open question. In particular, it seems necessary that the system is supplied with advance knowledge of top-level goals, and that some top-down reasoning is incorporated to make use of this knowledge.

The first step towards generating a higher-level representation of a finite-state model of behavior from a causal model is to detect "subprocedures" which occur elsewhere. For example, in Figure 1, Scene 1 — entering and choosing a seat — occurs in a number of other circumstances (lectures, theaters, ...). The first part of Scene 2 — gaining control of an object from one whose role it is to serve — is also not confined to restaurants. So also for the pay-and-tip-for-service part of Scene 4. Detecting common parts of existing scripts is clearly a very difficult search problem, and one that will require clever schemes for indexing and storage. It is, however, possible in principle†. Perhaps the higher-order statistics of behavior sequences will help in segmenting scripts into subprocedures.

A subprocedure which occurs in several different places is a candidate plan. To identify a plan, its preconditions and postconditions must be determined. These can only come from an analysis of the causal chain of acts involved in this plan, and in those that precede and follow it. Clearly the structure of the descriptors will be essential here. The restaurant script of Figure 1 uses as descriptors conceptualizations built around a small number of primitive acts. Associated with each act are pre- and post-conditions, and perhaps other deductions (for example, before actor $S$ causes himself to go from $A$ to $B$ he must be at $A$, afterwards he will be at $B$, and a likely intention is that he wanted to be at $B$). Thus a complete behavior sequence will be a causal chain with the descriptors interlinked through preconditions and postconditions. Since several instances of the plan have been observed, there is some basis for separating commonly-occurring conditions from fortuitous ones.

Associating postconditions with plans allows goals to be postulated, and plans indexed under them. A goal constitutes a set of plans with the same postconditions. It is critical to the plan-goal methodology that several alternatives exist for satisfying each goal (otherwise it is tantamount to fixed procedures).

While this is clearly highly speculative and leaves many details open to question, it does at least indicate that there are possible lines of attack on the very difficult problem of inferring goals and plans from observation of behavior.

## Conclusions

This paper has taken a wide-ranging look at the problems involved in creating models of sequences of symbols. Under certain tightly-defined circumstances — discrete sequences from a deterministic finite-state automaton, or real-valued signals from a linear system with Gaussian noise — the modeling process is supported by well-developed theory and optimal procedures are completely understood. When the descriptors are categorical but sequences are generated non-deterministically, one must resort to weaker methods — enumeration, or limited-context modeling — although they are still amenable to theoretical analysis and comparison. When the alphabet of descriptors from which elements of the behavior are drawn has its own structure, the problem becomes more complex. "Knowledge" can be added to the modeler in the form of more structured alphabets (typically through generalization hierarchies, or multidimensional, parametrized,

† For example, the enumeration and evaluation paradigm for behavior sequence modeling has been extended to recursive transition networks, and thus can identify common subprograms (Witten, 1981b). No infinite search is involved because the stack depth is obviously bounded for finite behavior sequences. However, the enumeration procedure is exceedingly time-consuming.

descriptors). However, the number of dimensions in which similarity can be measured increases, and one has to resort to heuristic methods, losing the benefits of any insight into the process from a formal perspective. One of the challenges for modeling systems of the future is to find ways of dealing with more highly-structured situations while still retaining the ability to analyze what is going on from a theoretical point of view.

Another facet of sequence modeling is the need to communicate with the user. The modeler can often benefit from the user's knowledge, and we have examined systems which permit this kind of interaction. Efforts need to be made to increase the bandwidth of communication for maximum effect. Moreover, as sequence modeling is used for knowledge acquisition, the need arises for better communication in the other direction. Modelers must become able to describe the models they have inferred, in different ways and at different levels of detail. For example, a rough sketch of the dominant path through the state representation of the model may suffice, with the ability to describe minority paths and elaborate on specific areas as directed. The emphasis on "explanation" in expert systems research encourages the view that all programs should be able to articulate their own workings.

Finally, a key question for the future is the inference of goals and plans. Can the modeler move from an operational summary of behavior sequences to higher epistemological levels by recognizing the intent behind the acts, without human intervention? We have tried to indicate that while this is an open question at the moment, it is not beyond the realm of possibility.

## Acknowledgements

I am grateful to Roy Masrani for helping me towards an understanding of scripts, goals, plans, and their interrelationships; and to Dan Freedman, Saul Greenberg, and Roy Masrani for their very useful comments on early versions of this paper. This research is supported by the Natural Sciences and Engineering Research Council of Canada.

## References

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

Andreae, P.M. (1984) "Constraint limited generalization: acquiring procedures from examples" *Proc 1984 Conf of the American Association of Artificial Intelligence*, Austin, Texas, August.

Atal, B.S. and Hanauer, S.L. (1971) "Speech analysis and synthesis by linear prediction of the acoustic wave" *J Acoustical Society of America, 50*, 637-655, August.

Barber, G.R. (1983) "Supporting organizational problem solving with a workstation" *ACM Trans Office Information Systems, 1* (1) 45-67, January.

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

Biermann, A.W. and Feldman, J.A. (1972) "On the synthesis of finite-state machines from samples of their behaviour" *IEEE Trans Computers, C-21* (6) 592-597, June.

Bower, G.H. and Black, J.B. (1979) "Scripts in memory for text" *Cognitive psychology, 11*, 177-220.

Brachman, R.J. (1983) "What IS-A is and isn't: an analysis of taxonomic links in semantic networks" *IEEE Computer, 16* (10) 30-36, October.

Cleary, J.G. and Witten, I.H. (1984) "Data compression using adaptive coding and partial string matching" *IEEE Trans Communications, COM-32* (4) 396-402, April.

Cormack, G.V. and Horspool, R.N. (1985) "Data compression using dynamic Markov modelling" Research Report, Department of Computer Science, University of Waterloo, April.

Cover, T.M. (1973) "Enumerative source encoding" *IEEE Trans Information Theory, IT-19* (1) 73-77, January.

Croft, W.B. and Lefkowitz, L.S. (1984) "Task support in an office system" *ACM Trans Office Information Systems, 2* (3) 197-212, July.

Cullingford, R.E. (1978) "Script application: computer understanding of newspaper stories" PhD thesis, Research Report 116, Yale University.

Dyer, M.G. (1983) *In-depth understanding.* MIT Press, Cambridge, Massachusetts.

Ellis, C. and Nutt, G. (1980) "Office information systems and computer science" *Computing Surveys, 12* (1) 27-60, March.

Evans, T.G. (1971) "Grammatical inference techniques in pattern analysis" in *Software Engineering*, edited by J.Tou, pp 183-202. Academic Press, New York.

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

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

Gaines, B.R. (1977) "System identification, approximation and complexity" *Int J General Systems, 3,* 145-174.

Gibbs, R.W. and Tenney, Y.J. (1980) "The concept of scripts in understanding stories" *J Psycholinguistic Research, 9* (3) 275-284.

Ha, Y.K. and Witten, I.H. (1980) "The stability of linear predictive descriptions of curves used in handwriting" *Electronics Letters, 16* (24) 909-911, November.

Halbert, D.C. (1981) "An example of programming by example" Technical Report, Xerox Office Products Division, Palo Alto, California.

Halbert, D.C. (1984) "Programming by example" Technical Report, Xerox Office Products Division, Palo Alto, California, December.

Klir, G.J. (1975) "On the representation of activity arrays" *Int J General Systems, 2* (3) 149-168.

Klir, G.J. (1976) "Identification of generative structure in empirical data" *Int J General Systems, 3* (2) 89-104.

Kunin, J.S. (1982) "Analysis and specification of office procedures" PhD Thesis, Department of Electrical Engineering and Computer Science, MIT, February.

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

Michalski, R.S. (1983) "A theory and methodology of inductive learning" in *Machine learning,* edited by R.S. Michalski, pp 83-134. Tioga Press.

Nerode, A. (1958) "Linear automaton transformations" *Proc American Mathematical Society, 9,* 541-544, August.

Nix, R. (1983) "Editing by example" PhD dissertation, Computer Science Department, Yale University, New Haven, Connecticut.

Nix, R. (1984) "Editing by example" *Proc 11th ACM Symposium on Principles of Programming Languages,* 186-195, Salt Lake City, Utah, January.

Schank, R.C. and Abelson, R. (1977) *Scripts, plans, goals and understanding.* Lawrence Erlbaum Associates.

Suchman, L.A. (1983) "Office procedure as practical action: models of work and system design" *ACM Trans on Office Information Systems, 1* (4) 320-328, October.

Suchman, L.A. (1985) "Plans and situated actions: the problem of human-machine communication" PhD thesis, Issued by Xerox PARC, Palo Alto, CA.

Uyttenhove, H.J.J. (1980) "On the two-dimensionality in the behavioral system identification problem, Part 1" *Int J Man-Machine Studies, 12,* 325-340.

Wiener, N. (1947) *Extrapolation, interpolation and smoothing of stationary time series.* MIT Press, Cambridge, Massachusetts.

Wilensky, R. (1983) *Planning and understanding: a computational approach to human reasoning.* Addison-Wesley.

Witten, I.H. (1979) "Approximate, non-deterministic modelling of behaviour sequences" *Int J General Systems, 5,* 1-12, January.

Witten, I.H. (1981a) "Programming by example for the casual user: a case study" *Proc Canadian Man-Computer Communication Conference,* 105-113, Waterloo, Ontario, June.

Witten, I.H. (1981b) "Some recent results in non-deterministic modelling of behaviour sequences" *Proc Society for General Systems Research Annual Conference,* 265-274, Toronto, Ontario, January.

Zisman, M.M. (1977) "Representation, specification, and automation of office procedures" PhD Dissertation, Wharton School, University of Pennsylvania.