

MANIPULATION EXTRAPOLATION A SYSTEM FOR CONTROLLING TRAINABLE ROBOTS

Ronald W. Colman
Computer Science Department
California State University
Fullerton, California 92634

Program synthesis from exemplary data is employed as a context for the design of trainable robots. A method for inferring programs from sample execution traces is illustrated. The method provides a possible model of a way humans are trained-by-example. The convergence characteristics and the conditions sufficient to ensure them are briefly discussed.

1. INTRODUCTION

When training another human, we frequently demonstrate, by example, how the first few of a sequence of similar problems are solved. The trainee induces a program from such a demonstration. For example, a program to insert shims in an assembly might be as follows:

- 1) Test fit and exit if within tolerance.
- 2) Select largest shim not overcompensating.
- 3) Insert selected shim. Go to step (1).

An industrial trainee would likely learn this by repeatedly

- a) observing the initial state of an assembly.
- b) observing a sequence of actions by a trainer which modify this state. And perhaps,
- c) hearing comments such as, "Always use the largest shim possible."

This paper proposes a model for such an interface between a human trainer and a computer. The most general context, program synthesis, is employed. The input for the i -th problem is assumed to be as follows:

- a) an initial state, s_1 , of data in the memory of a computer.
- b) an execution trace, t_1 , of a hypothetical program to be induced. This is the sequence of instructions executed by the program in transforming s_1 to a final state, from which control commands are deleted. And perhaps,
- c) identification by the trainer of those variables of significance at decision points. This reduces combinatorial explosion and is not required by the example to follow.

A training sequence, as follows, is assumed:

$$T_N = [(s_1, t_1), (s_2, t_2), \dots, (s_N, t_N)]$$

2. MANIPULATION EXTRAPOLATION

The process to be presented will employ two subsystems, both well-defined in the literature elsewhere. Grammatical inference, the problem of inducing a grammar from a sample of sentences, is solved by a variety of published methods for regular languages [4]. One by the author has convergence characteristics especially suited to this application [1]. The calculation of discriminant functions for numerical vectors has also been extensively studied [2]. An algorithm for the linear functions sufficient for the following is found in [5]. More general polynomial functions may use linear programming as demonstrated by the author [1].

2.1 The Process

In the following, G is a formal grammar induced to generate the set of execution traces. Alternative productions in this grammar (productions with identical left parts) are selected by discriminant functions defined over the set of partially transformed states occurring at the associated point in the calculation.

The process is specified in detail by specifying the inference procedures to be used for G and the discriminant functions. This is done in (1) where convergence is established.

A micro-processor controlled manipulator is envisioned, communicating interactively with a large computer where its program would be induced as follows:

- 0) INITIALIZE: Set grammar, G, to generate the empty language, $T_0 = \text{nil}$ and $i = 0$.
- 1) INCREASE SAMPLE: Set $i = i+1$ and $T_i = T_{i-1} \cup \{(s_i, t_i)\}$
- 2) TEST G: If t_i is in the language generated by G, go to step 4.
- 3) INFER GRAMMAR: Reform G to generate at least t_j $j=1,2,\dots,i$ and go to 5.
- 4) TEST STATE DISCRIMINATORS: If discriminant functions involved in processing s_i select the correct path to generate t_i , go to 1.
- 5) COMPUTE DISCRIMINANT FUNCTIONS: Reform discriminant functions for each set of productions altered in step 3 or found to be improperly selected in processing s_i . Go to 1.

3. ILLUSTRATIVE EXAMPLE

The computation of N-factorial will provide a simple illustration. Let the data state consist of two elements, (x_1, x_2) , with x_1 initialized at $i-1$ for the i -th problem (the computation of $(i-1)$ -factorial) and let x_2 be initialized at 1. Associated with each such starting state will be the shortest sequence of instructions possible to transform x_2 into the factorial of the initial value in x_1 . The training sequence for the first three problems is as follows:

$$T_3 = [((0,1), \text{nil}), ((1,1), \text{nil}), ((2,1), \text{load-x1-store-x2})]$$

Each t_i will be treated as a 'sentence' over the 'alphabet' of commands shown in figure 1.

CHARACTER	INTERPRETATION
1. load x1	(load x1 into accumulator)
2. load x2	(load x2 into accumulator)
3. store x1	(deposit accumulator in x1)
4. store x2	(deposit accumulator in x2)
5. decr x1	(decrement contents of x1)
6. decr x2	(decrement contents of x2)
7. mul x1	(multiply accumulator by x1)
8. mul x2	(multiply accumulator by x2)

Figure 1: EIGHT 'CHARACTERS' IN ALPHABET OF G

A grammar generating this language, $\{\text{nil}, \text{load-x1-store-x2}\}$, is shown in figure 2. The language is more concisely represented using the numbers of the commands in figure 1, $\{\text{nil}, 14\}$, instead of the commands above.

```

<start> ::= <1>
<1> ::= <end>
<1> ::= <2>
<2> ::= load-x1-store-x2 <end>

```

Figure 2: PRODUCTIONS OF GRAMMAR INDUCED FROM T_3

The fourth and fifth problems yield the training sequence:

$$T_5 = T_3 \cup [((3,1), 145184), ((4,1), 1451845184)]$$

Each of these new sentences is found to be absent from the language generated by the grammar induced from the preceding training sequence and a new grammar is induced. The resulting grammar induced from T_5 is shown in figure 3. This grammar will be seen equivalent to the flowchart in figure 4. It generates a solution for each of the infinite set of problems, N-factorial for $N=0,1,2,\dots$. Branches in the flowchart correspond to alternative productions in the grammar. These must now be associated with discriminant functions to transform this grammar into a program.

```

<start> ::= <1>
<1> ::= <end>
<1> ::= <2>
<2> ::= 1 4 <3>
<3> ::= <end>
<3> ::= <4>
<4> ::= 5 1 8 4 <3>

```

Figure 3: PRODUCTIONS OF G INFERRED FROM T_5

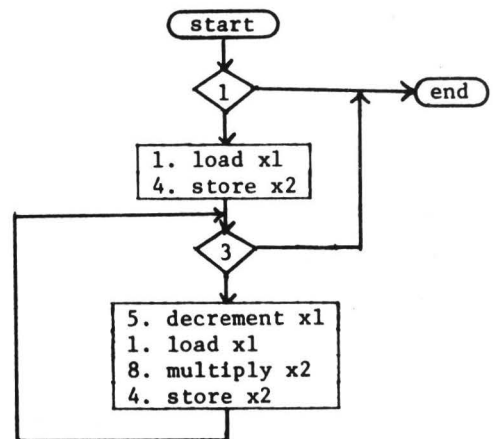


Figure 4: FLOWCHART EQUIVALENT TO GRAMMAR

Figure 5 shows the data states associated with each of the branch points in the flowchart. Each of these sets of states is dichotomously partitioned into mutually exclusive subsets associated with a particular alternative production in the grammar. A linear discriminant function, $F(x_1, x_2) = A \cdot x_1 + B \cdot x_2 + C$, is adequate for both branches. For branch 1, $(A, B, C) = (3, -2, -2)$ will cause $F(x_1, x_2)$ to be negative for states associated with rule, $\langle 1 \rangle ::= \langle \text{end} \rangle$, and positive for states associated with its alternative, $\langle 1 \rangle ::= \langle 2 \rangle$. For branch point 3, $(A, B, C) = (2, 0, -5)$ will similarly select the correct 'sentence' for each problem. An algorithm exists, guaranteed to find such linear separators if they exist [5].

ALTERNATIVE PRODUCTIONS	ASSOCIATED STATES
Branch Point 1	
$\langle 1 \rangle ::= \langle \text{end} \rangle$	(0,1), (1,1)
$\langle 1 \rangle ::= \langle 2 \rangle$	(2,1), (3,1), ..., (i,1), ...
Branch Point 3	
$\langle 3 \rangle ::= \langle \text{end} \rangle$	(2,2), (2,6), ..., (2,i!), ...
$\langle 3 \rangle ::= \langle 4 \rangle$	(3,3), (3,12), (4,4), (4,20), ...

Figure 5: DATA STATES ASSOCIATED WITH ALTERNATIVE PRODUCTIONS (GRAMMAR) AND DECISION POINTS (FLOWCHART)

3.1 Interaction with Trainer

Trainees often ask questions. Some grammatical inference methods use interrogation to determine sentences to be excluded from the language generated by the grammar. This is not appropriate to the application represented here and the grammar inference process used in [1] avoids a need for this by assuming that the training sequence contains all sentences shorter than the longest sentence present. The trainer would, however, likely have insights about which variables in the state vector are irrelevant at decision points. Interrogation of the trainer for this purpose, while not yet implemented, would greatly reduce the complexity of the discriminant function calculations.

4. CONVERGENCE

While flowcharts are equivalent to regular grammars, execution traces, representing a subset of all possible paths through the flowchart, are often a proper subset, selected by predicates at decision points, and are not regular languages. The regular grammar induced

from such execution traces will, in general, generate a language larger than the solutions to all problems under consideration. A convergence criteria "approach from below," captures the notion appropriate here [1]. A general discussion of such convergence is in [3].

Convergence of the grammar inference process is necessary but not sufficient. Separability of states at decision points in the induced grammar must be assured. The assumption that the training sequence could have been generated by a program with separable decision points is not sufficient since the induced grammar may be structurally dissimilar to the hypothetical generator.

A constructive grammar inference process devised by the author has been shown to preserve polynomial separability under the assumption of the existence of such a hypothetical program (1). Polynomial separability is reasonably general but excludes some frequently used tests such as "branch if variable x is even."

REFERENCES

- [1] Colman, R. W. "Inductive Extrapolation." Ph.D. Dissertation, Info. and Computer Science Dept., University of California at Irvine, January, 1977.
- [2] Duda, R. O. and P. Hart. Pattern Classification and Scene Analysis. J. Wiley and Sons, New York, 1973.
- [3] Feldman, J. A. "Some Decidability Results on Grammatical Inference and Complexity." *Information and Control* 20 (1972) pp. 244-262.
- [4] Fu, King-Sun and T. L. Booth. "Grammatical Inference: Introduction and Survey," IEEE Transactions on Systems, Man and Cybernetics. January, 1975, pp. 95-111.
- [5] Nilsson, N. J. Learning Machines: Foundations of Trainable Pattern Classifying Systems. McGraw-Hill, 1965.