11

An Experiment in Automatic Induction

R. J. Popplestone
Department of Machine Intelligence and Perception
University of Edinburgh

INTRODUCTION

The problem discussed in this paper, namely that of finding a function to satisfy a given argument-value table, is by no means new to computing science, or to mathematics. Thus, for example, the problem of fitting a curve to a set of points is a part of numerical analysis. However, I am concerned with finding a function over a non-metric space, and so my work is closer to that of Feldman *et al.* (1969) in what they call 'grammatical inference' or to the automaton-synthesizing programs described by Fogel, Owens and Walsh (1966).

The idea of putting together attributes with boolean connectives, which is part of the apparatus available to the induction engines described in this paper, is to be found in the psychological literature and referred to as 'concept formation', for instance, *see* Bruner, Goodnow and Austin (1956).

There have been some applications of learning devices. Perhaps the best known is Samuel's checkers program (Samuel 1967), but Murray and Elcock (1968) have a system for describing generalized board states in Go-Moku that employs a much richer language to describe the concepts learnt.

Relevant aspects of the problem are mentioned in McCarthy and Hayes (1969).

INDUCTION ENGINES

Induction is going from the particular to the general. Denote an induction engine by E. Here follows a definition of E that is sufficiently formal for my purpose.

Suppose there are two sets D and R, and let F be a set of functions from D to R, so that $f \in F$ can be regarded as a subset of $D \times R$. Then E is a

mapping which takes a finite subset s of f onto a function $E(s)$ from D to R so that the following conditions hold:

$IE1$ $\quad s \subset E(s)$ that is, $E(s)$ agrees with f on s

$IE2$ $\quad \exists s'(s' \in D \times R \;\&\; s' \subseteq f \;\&\; s' \subseteq E(s) \;\&\; s' \gg s)$

where $\gg$ means 'is significantly greater than'. What $IE2$ really says is that $E(s)$ must agree with f over a set significantly larger than s and by significant I mean that a human would prefer to have a function constructed by an induction engine than to construct and program one himself.

The notation may be informally summarized as follows:

F defines the family of functions over which the induction engine is to operate.

f is a function to be guessed.

s is a sample of f, in the form of an input–output table.

$E(s)$ is a function synthesized from s, – i.e., it is the induction engine's guess and must satisfy the stated conditions $IE1$ and $IE2$.

This definition is akin to that of the convergence of a sequence of functions in classical mathematics, but is in difficulties over defining an adequate metric. In some circumstances, e.g., for perceptrons with F being the set of linearly separable functions and Feldman's grammar learning programs (Feldman *et al.* 1969), a strong condition

$IE3$ $\quad \exists s' \; E(s') = f \;\&\; \mathit{finite}(s')$

can be imposed.

However, if we use a human as an induction engine, we would expect him to satisfy $IE1$ and $IE2$, but not to satisfy $IE3$. That is to say, there would be functions he could not guess definitions for.

For the further discussion of induction engines the concept of *language* is required. A language L is a pair (T, I) where T is a set of *sentences* and I is a function which takes $t \in T$ onto $I(t) \in F$. I is called the interpreter for the language.

Consider, for example, an Adaline (Widrow 1962). This is a simple linear threshold device. Suppose there are n inputs. Then T is the set of sequences of $n+1$ real numbers forming the weights $(w_1, w_2 \ldots w_{n+1})$. D, the set of possible inputs, is the Cartesian product $\prod_1^n B$ where B is the set $\{0, 1\}$. Thus a typical member of D would be $(d_1, d_2 \ldots d_n)$. R, the set of outputs, is B. Let $(w_1 \ldots w_{n+1})$ be a member of T, that is to say, a sentence, then

$$I(w_1 \ldots w_{n+1}) = \lambda d \textbf{ if } w_1 d_1 + w_2 d_2 + \ldots w_n d_n > w_{n+1} \textbf{ then } 1 \textbf{ else } 0$$

where $d_1 \ldots d_n$ are the components of d. An adaline training procedure is then an induction engine for adalines.

A concept of importance is the *power* of a language. This is simply the range of the interpreter. In particular, if I ranges over all computable functions then the language is said to have full power. Thus if T=all LISP expressions

and I=a LISP interpreter then we have a language with full power. On the other hand, an Adaline does not have full power.

AN INDUCTION PROGRAM

In this section I will describe a program to realize an induction engine. The domain D is a set of *boards*, where a board is an $n \times n$ array with entries taken from a set of symbols, $\{.\} \cup A$, where A is a finite set. n is fixed but arbitrary. "." is called the null symbol. Thus for tic-tac-toe $A = \{X, 0\}$. The range R is the set of truth values which we shall take to be (0, 1).

The language L consists of sentences of the predicate calculus formed by the rules *I0–I6* below, and interpreted by a special purpose interpreter. Since the domain is finite, the system is decidable. The sentence formation rules are as follows:

I0 There is a set P of *primitive* sentences contained in T. P consists of sentences such as $occ(1, 2, 'X')$ where occ is a predicate meaning 'is occupied by', so that the sentence $occ(1, 2, 'X')$ means 'square 1, 2 is occupied by an X'.

I1 $t \in T \Rightarrow t \,\&\, t' \in T$

I2 $t \in T \Rightarrow t \vee t' \in T$

where either (i) $t' \in T$ or (ii) for some predicate p, t' is $p(c_1 \ldots c_n)$ where the c_i are constants one of which occurs in t.

I3 $t \in T \Rightarrow \neg t \in T$

I4 $\mathscr{S}(c) \in T \Rightarrow \exists x_\alpha (x_\alpha \in C(c) \,\&\, \mathscr{S}(x_\alpha)) \in T$

I5 $\mathscr{S}(c) \in T \Rightarrow \forall x_\alpha (x_\alpha \in C(c) \,\&\, \mathscr{S}(x_\alpha)) \in T$

In *I4* and *I5* $\mathscr{S}(c)$ means a sentence containing the constant c, and $C(c)$ means a set containing c. Thus if c is a number then $C(c)$ would be the set of numbers $\{1, 2, \ldots, n\}$ n being the size of the board. *I4* and *I5* provide the means of generalizing from a statement about individuals to a statement about a class containing those individuals. x_α is a symbol not occurring in $\mathscr{S}(c)$.

Finally

I6 $\mathscr{S}(c) \in T \Rightarrow \mathscr{S}(f(c_1 \ldots c_n)) \in T$

where f is drawn from a set of standard functions. Thus *I6* would convert $occ(1, 3, 'X')$ into $occ(1, 1+2, 'X')$.

There are two possible interpretations of *I4* and *I5*. These differ in whether abstraction is of a constant or an instance of a constant. Thus in one interpretation $occ(1, 1, 'X')$ would only give rise to $\exists x_\alpha\ occ(x_\alpha, x_\alpha, 'X')$ by abstraction on 1 using *I4*, whereas in the other, one would also get $\exists x_\alpha\ occ(x_\alpha, 1, 'X')$ and $\exists x_\alpha\ occ(1, x_\alpha, 'X')$. In existing versions of the program, the first interpretation is used.

Two programs have been written which form sentences according to *I0* to *I6*. They are referred to as the Mark-1 and Mark-2 programs.

THE MARK-1 PROGRAM

This is an induction engine which uses rules $I0$, $I1(i)$, $I2(i)$, $I3$, $I4$ and $I5$ as follows. A set of current base sentences (CBS) is maintained. This is initially the primitives of $I0$. Let s be a 'sample' of argument–value pairs (as section 1.) An algorithm called ORDISCRIM is entered which produces, using the propositional rules $I1(i)$, $I2(i)$, and $I3$, a sentence t such that $s \subseteq I(t)$ where I is the interpreter. Thus ORDISCRIM itself satisfies $IE1$ (or more exactly ORDISCRIM $\circ$ I where $\circ$ means function product).

It will also in some measure satisfy $IE2$, because it has a built-in preference for simple explanations, and so will not specify that particular primitives need to be true (or false) if they are not needed for $IE1$. Thus, from Occam's Razor, we would expect the sentence formed by ORDISCRIM to work for some wider class than s.

Notice that ORDISCRIM performs a task equivalent to that of designing a digital logic circuit from AND, OR, and NOT gates which produces an output from a set of inputs, where the input–output relationship is tabulated. The inputs correspond to the primitive sentences of $I0$, and the input–output table to the set s.

The sentence resulting from the application of the propositional rules is then decomposed into all its subsentences, and the generalizing rules $I4$ and $I5$ are applied to these, and a new CBS is made from the union of the result of this application of $I4$ and $I5$ and the primitives. ORDISCRIM is then applied to the new CBS and the process repeated until no new result is produced by ORDISCRIM.

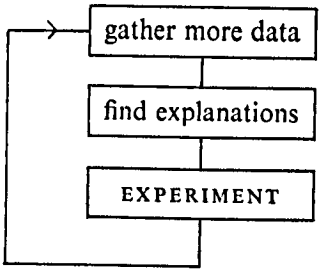
The above description is of the action of the engine when it is working on the first sample set. When working on a sequence of examples, the CBS for the final explanation of one is carried over to the next. This means that it is possible for the engine to find a sentence s for which $I(s) = f$ via a sequence of examples graded in difficulty which could not be found if the engine were presented with the last example straight off.

An annotated run of the mark-1 program is to be found in Appendix 1. In this example, the board is 3×3 and f, the function to be guessed, is the property of there being three Xs in a line, either row, column, or diagonal.

The limitations of the Mark-1 program were fairly obvious. Firstly, the series of examples by which it 'learned' a complex function had to be carefully chosen. Thus in the example quoted above, it was necessary to get an adequate understanding of a horizontal line before trying to express a vertical line. Also, since the generalization processes, $I4$ and $I5$, worked on the output of ORDISCRIM there was no possibility of redress if ORDISCRIM seized on features of particular boards which did not generalize correctly. Thus it was not possible to get correct generalizations if the board contained other non-null symbols than X. Finally, there is a severe restriction in expressive power through the inability to introduce predicates other than *occ*. To overcome some of these limitations, Mark-2 was written.

THE MARK-2 PROGRAM

The program has the flow-diagram shown in the figure.



In the 'gather more data' phase, the program reads-in some more argument–value pairs to add to the sample. In contrast to the Mark-1 program where all data is entered in this way, relatively little is entered thus.

In the 'find explanations' box, the Graph Traverser (Doran and Michie 1966, Doran 1968) is used to search for an explanation. This is done as follows.

Nodes are sentences in T. $I1$ to $I5$ are used as rules to generate sentences, and thus form the basis of the 'develop' function. The primitives are used as starting nodes. For any node t, the goodness of t is measured by

(1) how well $I(t)$ agrees with f on s

(2) the complexity of t.

In the present program, selection of t' for rules $I1(i)$ and $I2(i)$ is done by using the best k nodes to date, for some fixed k. This is not optimal, and better methods can be devised by considering the algorithm ORDISCRIM described above.

The Graph Traverser runs until certain limits either of space or of time are exceeded. The routine EXPERIMENT is then entered.

The EXPERIMENT routine consists in examining the nodelist, as follows. If the best two nodes t_α and t_β are not both explanations, the routine exits. Otherwise the sentence $t = t_\alpha \,\&\, \neg t_\beta$ is formed, and a Beth tree theorem prover (Popplestone 1967) is entered to find a board position for which t is true. If there is none (and since the domain is finite this is decidable) then $t' = \neg t_\alpha \,\&\, t_\beta$ is formed, and again a board position for which t' is true is searched for. If, again, there is none, then $t_\alpha \equiv t_\beta$ and the sentence with the higher score (the 'worse' sentence) is discarded, and the process repeated with the new nodelist. Otherwise, if a position satisfying t or t' has been found, it is printed out, and the user is asked what the value of f is for it. All nodes are then re-evaluated with the new sample, and EXPERIMENT is re-entered.

The EXPERIMENT routine is important because it enables the machine to decide between hypotheses in the manner of a scientist thinking of a crucial experiment.

Acknowledgement

This work was made possible by facilities supplied through a research grant from the Science Research Council.

REFERENCES

Bruner, J.S., Goodnow, J.J., & Austin, G.A. (1956) *A Study of Thinking*. New York: John Wiley & Sons.

Doran, J.E. & Michie, D. (1966) Experiments with the Graph Traverser program. *Proc. R. Soc.* (A), **294**, 235–59.

Doran, J.E. (1968) New developments of the Graph Traverser. *Machine Intelligence 2*, pp. 119–35 (eds Dale, E. & Michie, D.). Edinburgh: Oliver and Boyd.

Feldman, J., Gips, J., Horning, J.J., & Reder, S. (1969) Grammatical complexity and inference. *Technical Report No. CS* 125, Stanford Artificial Intelligence Project, Stanford, California.

Fogel, L.J., Owens, A.J. & Walsh, M.J. (1966) *Artificial Intelligence through simulated evolution*. New York: John Wiley & Sons.

McCarthy, J. & Hayes, P. (1969) Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence 4*, pp. 463–502 (eds Meltzer, B. & Michie, D.). Edinburgh: Edinburgh University Press.

Murray, A.M. & Elcock, E.W. (1968) Automatic description and recognition of board patterns in Go-Moku. *Machine Intelligence 2*, pp. 75–88 (eds Michie, D. & Dale, E.). Edinburgh: Oliver and Boyd.

Popplestone, R.J. (1967) Beth Tree methods in automatic theorem proving. *Machine Intelligence 1*, pp. 119–35 (eds Collins, N.L. & Michie, D.). Edinburgh: Oliver and Boyd.

Samuel, A.L. (1967) Some studies in machine learning using the game of checkers – 2 – recent progress. *IBM Journal Research and Dev.*, **11**, 601–17.

Widrow, B. (1962) Generalization and information storage in networks of adaline neurons. *Self-Organizing Systems* – 1962, pp. 435–61 (eds Yovitz, Jacobi & Goldstein). Washington D.C.: Spartan Books.

APPENDIX A ABBREVIATED AND ANNOTATED OUTPUT

The program is here learning to recognize a win at tic–tac–toe.

** 'TYPE BOARD'	The experimenter types in first position
: X . .	
: . . .	
: . . .	
** 'I SAY WIN' : WRONG	The machine guesses, on no evidence, that this is a winning position and the experimenter tells it that it is wrong.
** [ORF]	This is the machine's next explanation which is simply that win(s) is false for every state s.

```
** 'TYPE BOARD'
: X X X
: . . .
: . . .
```

This time the machine is presented with a winning position and it guesses that it is not a winning position (because its current explanation is that all positions are not winning positions).

```
** 'I SAY NOT WIN': WRONG
```

It is again corrected by the experimenter.

```
** [ ORF [ ANDF [ EXISTS 9400000A [ 1 2 3 ]
ALL 8400000A [ 1 2 3 ] LAMBDA SS OCC SS
9400000A 8400000A [ QUOTE . X ] ] ] ]
```

Its next explanation is that for a win there must be an *i* such that for all *j*, square *ij* is occupied by an X, that is to say, one row of the board must be occupied by Xs.

```
** 'TYPE BOARD'
: . . .
: X X X
: . . O
** 'I SAY WIN': RIGHT
```

The next example confirms this hypothesis.

```
** 'TYPE BOARD'
: X . .
: X . .
: X . .
** 'I SAY NOT WIN': WRONG
```

But this one does not.

```
** [ ORF [ ANDF [ ALL 4500000A [ 1 2 3 ]
EXISTS 3500000A [ 1 2 3 ] LAMBDA SS OCC
SS 4500000A 3500000A [ QUOTE . X ] ] ]
[ ANDF [ ALL 5500000A [ 1 2 3 ] EXISTS
2500000A [ 1 2 3 ] LAMBDA SS OCC SS
2500000A 5500000A [QUOTE . X ] ] ] ]
```

The next explanation offered by the machine is ingenious but wrong, namely that either there must be an X in every row or there must be an X in every column.

```
** 'TYPE BOARD'
: O X O
: . X .
: . X O
** 'I SAY WIN': RIGHT
** 'TYPE BOARD'
: X . .
: . X .
: . . X
** 'I SAY WIN': RIGHT
```

```
** 'TYPE BOARD'
: X . .
: . . .
: . . X
** 'I SAY NOT WIN': RIGHT

** 'TYPE BOARD'
: X . .
: X . .
: . X .
** 'I SAY WIN': WRONG

** [ ORF [ ANDF [ EXISTS 4600000A [ 1 2 3 ]
ALL 0600000A [ 1 2 3 ] LAMBDA SS OCC SS
4600000A 0600000A [ QUOTE . X ] ] ]
[ ANDF [ EXISTS 3600000A [ 1 2 3 ] ALL
1600000A [ 1 2 3 ] LAMBDA SS OCC SS
1600000A 3600000A [ QUOTE . X ] ] ] [ ANDF
[ LAMBDA SS OCC SS 2 2 [ QUOTE . X ] ] ] ]

** 'TYPE BOARD'
: . . .
: . X .
: . . X
** 'I SAY WIN': WRONG

** [ ORF [ ANDF [ ALL 2700000A [ 1 2 3 ]
LAMBDA SS OCC SS 2700000A 2700000A
[ QUOTE . X ] ] ] [ ANDF [ EXISTS 8700000A
[ 1 2 3 ] ALL 4700000A [ 1 2 3 ] LAMBDA SS
OCC SS 8700000A 4700000A [ QUOTE . X ] ] ]
[ ANDF [ EXISTS 7800000A [ 1 2 3 ] ALL
5700000A [ 1 2 3 ] LAMBDA SS OCC SS
5700000A 7700000A [ QUOTE . X ] ] ] ]

** 'TYPE BOARD'
: . . X
: . X .
: X . .
** 'I SAY NOT WIN': WRONG
```

The machine is only disillusioned here,

and now offers the explanation that there must either exist a row of xs or a column of xs or that square 2 2 must be occupied with an X

which is immediately faulted

Finally, an explanation is produced which says that there is a win if there are xs in the leading diagonal or there is a row or column occupied by xs.

At this stage the time-sharing system failed.

APPENDIX B A RUN OF THE MARK 2 PROGRAM

```
** TYPE BOARD
: X X X
: . . .
: . . .
```
The human types in a board state.

```
** IS THIS AN INSTANCE: YES
```
And says it is an instance of the concept.

```
** DO YOU WANT TO TELL ME MORE: YES
```
The concept (property) to be guessed is 'there is a row or a column or a diagonal of xs'.

```
** TYPE BOARD
: X . .
: . . .
: . . .
```
Then he types in another.

```
** IS THIS AN INSTANCE: NO
```
And says it is not an instance.

```
** DO YOU WANT TO TELL ME MORE: NO
```
The program then starts work

```
** [ 21.19 16 SEPT 1969 ]
```

```
** INDDEVEL
t1: OCC ( 1, 3, QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 16
```
Hypothesis t1 – square 1, 3 is occupied by an X.

```
** INDDEVEL
t2: OCC ( 1, 2, QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 16
```
Hypothesis t2 – square 1, 2 is occupied by an X.

```
** INDDEVEL
t3: EXISTS A55000 ( 1 2 3 ) OCC ( A55000,
2, QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 17
```
Hypothesis t3 – there is an X in column 2.

```
** MAXNJOBS
```
Space bound exceeded, exit from Graph Traverser.

```
** CHECK JOBLIST
t1: OCC ( 1, 3, QUOTE ( X ) ) 16
t2: OCC ( 1, 2, QUOTE ( X ) ) 16
t3: EXISTS A55000 ( 1 2 3 ) OCC ( A55000,
  2, QUOTE ( X ) ) 17
t4: ALL A55000 ( 1 2 3 ) OCC ( 1, A55000,
  QUOTE ( X ) ) 17
t5: EXISTS A45000 ( 1 2 3 ) OCC ( A45000,
  3, QUOTE ( X ) ) 17
```
Print best 5 hypotheses.

```
** EXPERIME
t1: OCC ( 1, 3, QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 16
t2: OCC ( 1, 2, QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 16
. . X
. . .
. . .
```

Experiment. The machine produces a state for which t1 & ¬t2 is true.

** IS THIS AN INSTANCE: NO

The human says it is not an instance of the concept, thus contradicting t1.

```
** EXPERIME
t2: OCC ( 1, 2, QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 16
t4: ALL A45000 ( 1 2 3 ) OCC ( 1, A45000,
  QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 17
. X .
. . .
. . .
```

t4 says:
'Row 1 is full of xs'.

Machine produces state of board for which t2 & ¬t4 is true.

** IS THIS AN INSTANCE: NO

Human says 'not an instance' so contradicting t2 and also t3.

```
** EXPERIME
t4: ALL A55000 ( 1 2 3 ) OCC ( 1, A55000,
  QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 17
t5: ALL A45000 ( 1 2 3 ) OCC ( 1, A45000,
  QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 17
```

Experiment cannot find a state for which t4 & ¬t5.

or for which t5 & ¬t4 and so concludes t5≡t4

```
** EXPERIME
t4: ALL A55000 ( 1 2 3 ) OCC ( 1, A55000,
  QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 17
t6: ALL A65000 ( 1 2 3 ) EXISTS A55000
  ( 1 2 3 ) OCC ( A55000, A65000,
  QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 18
. X X
X . .
. . .
```

t6 says 'there is an X in every column' – so this board is t6 & ¬t4

```
** IS THIS AN INSTANCE: NO
** DO YOU WANT TO TELL ME MORE: NO
```

Experiment is finished, and no more boards are given manually. So start Graph Traverser.

```
** [ 21.23 16 SEPT 1969 ]
** INDDEVEL
t4: ALL A55000 ( 1 2 3 ) OCC ( 1, A55000,
  QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 17
** INDDEVEL
t7: EXISTS A75000 ( 1 2 3 ) ALL A55000
  ( 1 2 3 ) OCC ( A75000, A55000,
  QUOTE ( X ) )
TRUECOUN 1 TARGCOUN 1INDVALOF 18
** MAXNJOBS
** CHECK JOBLIST
ALL A55000 ( 1 2 3 ) OCC ( 1, A55000,
  QUOTE ( X ) ) 17
EXISTS A75000 ( 1 2 3 ) ALL A55000 ( 1 2 3 )
  OCC ( A75000, A55000, QUOTE ( X ) ) 18
NOT (( ALL A55000 ( 1 2 3 ) OCC ( 1,
  A55ØØØ, QUOTE ( X ) ) ) 19
NOT ( EXISTS A75000 ( 1 2 3 ) ALL A55000
  ( 1 2 3 ) OCC ( A75000, A55000,
  QUOTE ( X ) ) ) 20
OCC ( 1, 1, QUOTE ( X ) ) 21
** EXPERIME
t4: ALL A55000 ( 1 2 3 ) OCC ( 1, A55000,
  QUOTE ( X ) )
  TRUECOUN 1TARGCOUN 1INDVALOF 17
t7: EXISTS A75000 ( 1 2 3 ) ALL A55000
  ( 1 2 3 ) OCC ( A75000, A55000,
  QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 18
. . . t7 & ¬t4
X X X
. . .
** IS THIS AN INSTANCE: YES
** DO YOU WANT TO TELL ME MORE: NO
** [ 21.25 16 SEPT 1969 ]
```

t7 says 'there is a row of xs'

```
** INDDEVEL
EXISTS A75000 ( 1 2 3 ) ALL A55000 ( 1 2 3 )
  OCC ( A75000, A55000, QUOTE ( X ) )
TRUECOUN 2TARGCOUN 2INDVALOF 18

** INDDEVEL
NOT ( EXISTS A75000 ( 1 2 3 ) ALL A55000
  ( 1 2 3 ) OCC ( A75000, A55000,
  QUOTE ( X ) ) )
TRUECOUN 4TARGCOUN 0INDVALOF 20

** MAXNJOBS

** CHECK JOBLIST
EXISTS A75000 ( 1 2 3 ) ALL A55000 ( 1 2 3 )
  OCC ( A75000, A55000, QUOTE ( X ) ) 18
NOT ( EXISTS A75000 ( 1 2 3 ) ALL A55000
  ( 1 2 3 ) OCC ( A75000, A55000,
  QUOTE ( X ) ) ) 20
OCC ( 2, 2, QUOTE ( X ) ) 21 . 0
NOT ( NOT ( EXISTS A75000 ( 1 2 3 ) ALL
  A55000 ( 1 2 3 ) OCC ( A75000, A55000,
  QUOTE ( X ) ) ) ) 22
ALL A65000 ( 1 2 3 ) EXISTS A55000 ( 1 2 3 )
  OCC ( A55000, A65000, QUOTE ( X ) )
  24 . 67

** DO YOU WANT TO TELL ME MORE: YES

** TYPE BOARD

: X . .
: X . .
: X . .

** IS THIS AN INSTANCE: YES

** DO YOU WANT TO TELL ME MORE: NO

** [ 21.28 16 SEPT 1969 ]

** INDDEVEL
OCC ( 1, 1, QUOTE ( X ) )
TRUECOUN 3TARGCOUN 2INDVALOF 24 . 33

** INDDEVEL
OCC ( 2, 2, QUOTE ( X ) )
TRUECOUN 1TARGCOUN 1INDVALOF 24 . 33

** MAXNJOBS
```

Next the idea of a column is introduced, but the capability of the program of analysing disjunctive concepts is limited.

```
** CHECK JOBLIST
OCC(1, 1, QUOTE ( X ) ) 24 . 33
OCC ( 2, 2, QUOTE ( X ) ) 24 . 33
EXISTS A75000 ( 1 2 3 ) ALL A55000 ( 1 2 3 )
   OCC ( A75000, A55000, QUOTE ( X ) )
   24 . 67

NOT ( OCC ( 1, 1, QUOTE ( X ) ) ) 27 . ø
NOT ( EXISTS A75000 ( 1 2 3 ) ALL A55000
   ( 1 2 3 ) OCC ( A75000, A55000,
   QUOTE ( X ) ) ) 28 . 0

** DO YOU WANT TO TELL ME MORE: NO
```

The program now revisits earlier simple-minded proposals which were formerly not even aired.