INDUCTION OF RELATIONAL PRODUCTIONS IN TOE PRESENCE OF BACKGROUND INFORMATION

Steven A. Vere

Department of Information Engineering
University of Illinois at Chicago Circle
Chicago, Illinois 60680

ABSTRACT

An operational theory is developed for the generalization of relational productions in the presence of background information. This provides a unified theoretical solution for a broad class of intelligence paradigms, including letter sequence completion, visual analogies, and the learning of operators from before-and-after situation pairs or from situation sequences. Relational productions are used as a formal model for operators. The generalization process is given access to a common body of background information. Background literals which have association chains to foreground literals may be incorporated into the generalization. This theory has been computer implemented in SN0B0L4, and examples illustrate its application in the four paradigms mentioned above.

1. INTRODUCTION

The long-range goal of this work is the development of an operational, general purpose mathematical theory of inductive learning whose implementation would be capable of emulating the behavior of empirical programs written for specialized tasks. This paper describes a generalization theory for the induction of relational productions in the presence of "background information", i.e., a separate body of facts potentially relevant to the generalization process. With only minor interface variations, a SN0B0L4 computer implementation of this theory exhibits behavior in four distinct intelligence paradigms:

1) Letter sequence completion (e.g., ABMCDM...);
2) Selection of best visual analogies in IQ test questions;
3) Learning operators ("rules") from before-and-after situation pairs and counterexample pairs;
4) Learning operators from observation of situation sequences.

Problems in the first paradigm have been solved repeatedly by programs specially written for this task. For example, Williams (1972) developed a program with behavior covering this and similar completion problems, and Waterman (1975) presents a short program for this task, expressed in condition/action productions. See also Egan and Greeno (1974). Evans (1968) developed a very capable program for visual analogies. Hayes-Roth (1976) has developed a program for learning "rules" from before-and-after examples. The empirical program of Hedrick (1976) solves problems in the letter sequence and before-and-after pairs paradigms. Soloway (1976) is developing a program for learning the rules of baseball from situation sequences. Nevertheless, it is difficult to find a fundamental theoretical principal common to all these programs. However, it will be seen that relational production induction does provide a common theoretical core for all four paradigms.

The relational production is a formal model for describing change in discrete relational systems (Vere 1976, 1977a). It is an amalgamation of two familiar concepts: string productions (Post 1943) and STRIPS operators (Fikes et al., 1972). Readers are cautioned not to confuse this model with the linearly ordered condition/action productions studied by Waterman and others. A major advantage of the relational production formulation over related AI "production" systems is that the applicability and effect of these productions can be described mathematically using analytic tools from theorem proving literature. This permits the formal proof of key results.

The relational production model will be informally reviewed in section 2. Section 3 develops the theory of relational production generalization, based on earlier work in concept induction, and discusses the role of background information. Section 4 discusses the application of relational production induction to each of the four paradigms together with an example run on the computer implementation.

2. REVIEW OF RELATIONAL PRODUCTIONS

A relational production is a relation revision mechanism, analogous to string productions. String productions specify possible revisions of a string of symbols; relational productions specify possible revisions of a conjunction of literals. In this paper, a literal will be taken to be an ordered list of terms, and a term will be a constant or a variable.

For the purposes of this paper, a relational production has the form a-*B, where a and B are conjunctions of literals. The substrate which a set of productions may revise is called a situation, which is also a conjunction of literals.

To illustrate the structure and application of relational productions, consider a "blocksworld" consisting of a number of cubic blocks of uniform size. Figure 2.1 shows schematically a blocksworld situation in which there are three blocks.

We will use three types of literals to describe this world:

1. (on .X .Y) means block X rests directly on top of block Y;
2. (ontable .X) means block X rests directly on the table;
3. (clear .X) means the top of block X is clear, i.e., no block rests on X.

The convention is that variables are identified by a period prefix. Otherwise, strings of letters and digits are constants.

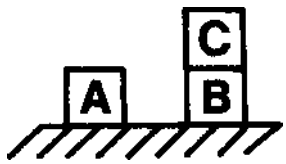


Figure 2.1 A Simple Blocksworld Situation

Using these literal types, a description of the situation of Figure 2.1 is:
(clear a)
(ontable a)
(on c b)
(ontable b)
(clear c)

The following relational production specifies the unstacking of a block:
(clear .X) (on .X .Y) ⟶ (clear .X) (ontable .X) (clear .Y)

In words, this specifies that if a block X is clear and rests on block Y, then the situation may change into one in which X rests on the tabic and Y is clear. We emphasize that a production is not the same as a predicate calculus implication, since the literals on the left of the production are deleted from the situation. This will be described precisely below.

Of course, a production may not be applicable in a given situation. For example, an unstacking operation cannot occur if no block rests on another. Let o represent a situation and θ a conventional substitution of the kind used in deductive theorem proving, and having the form $\{t_1/v_1, \ldots, t_r/v_r\}$, where each t^ is a term and each v^ is a variable. Then production α→β is applicable to situation a iff aθCa. In this relation, the conjunctions a and o are regarded as sets of literals. Thus the unstacking production above is applicable to the situation of Figure 2.1 since

$$\begin{bmatrix}\text{(clear .X)}\\ \text{(on .X .Y)}\end{bmatrix}\{\text{c/.X, b/.Y}\} = \begin{bmatrix}\text{(clear c)}\\ \text{(on c b)}\end{bmatrix} \subseteq \begin{bmatrix}\text{(clear a)}\\ \text{(ontable a)}\\ \text{(on c b)}\\ \text{(ontable b)}\\ \text{(clear c)}\end{bmatrix}$$

To specify the new situation after a production has revised a situation, we will use additional set operators:

$$\sigma' = \sigma \cap \overline{\alpha\theta} \cup \beta\theta$$

Here ∪ and ∩ denote set union and intersection, respectively, and the overbar denotes set complement, θ is any substitution which satisfies the applicability relation. In the absence of parentheses, intersection takes precedence over union in expressions.

Continuing the example above, if the unstacking production is applied to the situation of Figure 2.1, the new situation is determined as follows:

$$\sigma' = \begin{bmatrix}\text{(clear a)}\\ \text{(ontable a)}\\ \text{(on c b)}\\ \text{(ontable b)}\\ \text{(clear c)}\end{bmatrix} \cap \overline{\begin{bmatrix}\text{(clear .X)}\\ \text{(on .X .Y)}\end{bmatrix}\{\text{c/.X, b/.Y}\}}$$

$$\cup \begin{bmatrix}\text{(clear .X)}\\ \text{(ontable .X)}\\ \text{(clear .Y)}\end{bmatrix}\{\text{c/.X, b/.Y}\}$$

$$= \begin{bmatrix}\text{(clear a)}\\ \text{(ontable a)}\\ \text{(ontable b)}\\ \text{(clear c)}\\ \text{(ontable c)}\\ \text{(clear b)}\end{bmatrix}$$

Thus the production serves to unstack a block. Similar productions can be formulated to transfer a block from one stack to another and to stack blocks. By continuing to apply productions, a situation can be revised repeatedly. The notation $\sigma \stackrel{p}{\Rightarrow} \sigma'$ will denote that production p transforms situation o into situation a'.

Beyond artificial intelligence applications, relational productions are also capable of modelling the semantics of computer programs, including list processing and structured data (Vere 1976).

The Context Form for Relational Productions

The antecedent a and consequent B of a production may contain common literals. Let γ = α ∩β. This set of common literals y will be called the context of the production. Consideration of ttie applicability relation a6Ca and next situation relation σ' = σ∩ᾱθ̄ ∪ βθ shows that adding or deleting literals in the context affects applicability of a production, but not the transformation caused by that production. The context form of a production

$$[\alpha \cap \beta]\ \alpha \cap \bar{\beta} \rightarrow \bar{\alpha} \cap \beta$$

For example, the context form of the blocksworld unstacking production
(clear .X) (on .X .Y) ⟶ (clear .X) (ontable .X) (clear .Y)
is [(clear .X)] (on .X .Y) ⟶ (ontable .X) (clear .Y)

3. GENERALIZATION OF RELATIONAL PRODUCTIONS

This section develops an analytic, operational theory for the generalization of relational productions. First, "generalization" is defined for productions; second, properties of generalizations are reviewed; third, a method for generalizing

productions is developed based on an earlier method for generalizing simple conjunctions of literals; fourth, a method is presented for augmenting the context of productions with relevant background information using the concept of association chains; fifth, the concept of deterministic productions is introduced and defined in terms of association chains.

3.1 Definition and Theorem for Generalization of Relational Productions

Production p_1 is a generalization of production p_2, denoted $p_1 \leq p_2$, iff for all situations σ and σ', $(\sigma \overset{p_2}{\Rightarrow} \sigma')$ implies $(\sigma \overset{p_1}{\Rightarrow} \sigma')$. In other words, if p_1 is a generalization of p_2, the p_1 can cause any transformation caused by p_2, and possibly others as well.

Relational Production Generalization Theorem (Vere 1977b). If $p_1 = ([\gamma_1]\alpha_1 \rightarrow \beta_1)$ and $p_2 = ([\gamma_2]\alpha_2 \rightarrow \beta_2)$ are two relational productions, and $\forall\theta\ \alpha_1\theta \cap \beta_1\theta = \emptyset$ and $\alpha_2\theta \cap \beta_2\theta = \emptyset$, then $p_1 \leq p_2$ iff there exists a substitution θ_a such that $\gamma_1\theta_a \subseteq \gamma_2$, $\alpha_1\theta_a = \alpha_2$, and and $\beta_1\theta_a = \beta_2$.

3.2 Properties of Generalizations

In this section we briefly review some of the properties of generalizations.

Strict Generalizations. If production p_1 is an alphabetic variant of production p_2, then it follows immediately from the generalization definition for productions that $p_1 \leq p_2$. If $p_1 \leq p_2$ and p_1 is not an alphabetic variant of p_2, then p_1 is a strict generalization of p_2, denoted $p_1 < p_2$.

Common Generalizations. If $p_1 \leq p_2$ and $p_1 \leq p_3$ it is natural to say that p_1 is a common generalization of p_2 and p_3. It is also useful to talk of the common generalization of more than two productions.

Maximal Common Generalizations. Loosely speaking, a maximal common generalization is one which contains all the common features of the productions of which it is a generalization. More precisely, a production p_1 is a maximal common generalization of productions p_2 and p_3 iff:

1. $p_1 \leq p_2$ and $p_1 \leq p_3$ (i.e., it is a common generalization.)
2. no p_1' exists such that $p_1 < p_1'$, $p_1' \leq p_2$, and $p_1' \leq p_3$ (i.e., it is maximal.)

As an example, consider the four productions

p_1 = {[(window open)] (at mky a1) ⟶ (at mky a2)}
p_2 = {[(window open) (at box a3)] (at mky a7) ⟶ (at mky a8)}
p_3 = {[(window open)] (at mky .X) ⟶ (at mky .Y)}
p_4 = {[ ] (at mky .U) → (at mky .V)}

$p_4 \leq p_1$ and $p_4 \leq p_2$. However, p_4 is not a maximal common generalization of p_1 and p_2 because p_3 is also a common generalization and $p_4 < p_3$. In fact, p_3 is a maximal common generalization (mcg).

3.3 Computing Maximal Common Generalizations of Productions

In previous work on the generalization of simple conjunctions (Vere 1975), $A \leq B$ iff $A\theta \subseteq B$, and a procedure is presented for computing the mcg of two conjunctions. As shown below, in relational production generalization a production may be regarded as an ordered list of three conjunctions: the context γ, the antecedent α, and the consequent β. Given two productions p_1 and p_2, their mcg's are obtained by computing the mcg's of each of their three components. However, generalization of the antecedent and consequent are special cases. To see why, consider three productions $p_1 = \{[\gamma_1]\alpha_1 \rightarrow \beta_1\}$, $p_2 = \{[\gamma_2]\alpha_2 \rightarrow \beta_2\}$, and $p_3 = \{[\gamma_3]\alpha_3 \rightarrow \beta_3\}$. It follows immediately from the theorem and definitions of sections 3.1 and 3.2 that if p_3 is an mcg of p_1 and p_2, then $\gamma_3\theta_a \subseteq \gamma_1$, $\gamma_3\theta_b \subseteq \gamma_2$, $\alpha_3\theta_a = \alpha_1$, $\alpha_3\theta_b = \alpha_2$, $\beta_3\theta_a = \beta_1$, and $\beta_3\theta_b = \beta_2$ for some substitutions θ_a and θ_b. Thus the contexts γ_1 and γ_2 need not have the same number of literals, but α_1 and α_2, and β_1 and β_2 respectively must have the same number. Consequently α_3 must not only be an mcg of α_1 and α_2, but it must also have exactly the same number of literals. The same is true for β_3.

For example, consider the two productions

p_1 = {[ ] (at mky a1) ⟶ (at mky a2)} and
p_2 = {[ ] (at mky b1) (at box b1) ⟶ (at mky b2) (at box b2)}

The mcg of the antecedents is (at mky .X), and the mcg of the consequents is (at mky .Y). However, for both antecedent and consequent these generalizations have fewer literals than the antecedent and consequent of p_2. Hence no mcg of p_1 and p_2 exists.

In summary, to find the mcg of two productions p_1 and p_2, compute the mcg's of their respective components. However, if α_3 or β_3 has fewer literals than the corresponding components of p_1 or p_2, no mcg exists.

The method for two productions is the basis for generalizing n productions, where $n > 2$. In Vere (1977b) an iterative, bottom-up approach is discussed. If a conjunctive generalization can be assumed, a "linear" method is faster: given productions $p_1, p_2, \ldots, p_n$ compute $mcg(p_n, mcg(p_{n-1}, (\ldots\ mcg(p_2, p_1)\ \ldots))$.

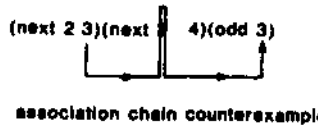
3.4 Background Information

In the concept learning paradigm of earlier work, instances of a concept are each described by a conjunction of literals and these conjunctions are generalized to form an abstraction of the concept. In many cases, both in concept induction and production induction, there is a body of information potentially pertinent to the generalization and yet not specifically associated with any particular instance. For simplicity, the following discussion treats only concept induction examples.

Consider the problem of learning poker hands from examples. The two "full house" hands 4 S, 2 H, 4 D, 4 C, 2 C and 10 H, 10 D, J S, J C, J H could be described by the following conjunctions of literals: (card 4 S)(card 2 H)(card 4 D)(card 4 C) (card 2 C) and (card 10 H)(card 10 D)(card J S) (card J C)(card J H). From these we could obtain the generalization: (card .X1 .Y1)(card .X1 .Y2) (card .X2 .Y3)(card .X2 .Y4)(card .X2 .Y5). No information is required beyond that present in the descriptions of the instances. However, to learn the concept of "straight" from the following two hands: 3 H, 4 D, 5 H, 6 D, 7 C and 10 H, J S, Q D, K D, A C requires additional background information on the relative ranking of the cards, e.g., that Queen is just higher than Jack and so on. We use the term background information to refer to a body of facts potentially relevant to a class of generalization problems, but not actually part of the description of any particular instance. It then seems natural to refer to the descriptions of the instances as the foreground information. Here background information might take the form: (next 2 3)(next 3 4)...(next K A). The problem is to mechanically augment the literals of each instance in the foreground with "relevant" background literals before generalizing. For example, we would want to add the background literals (next 3 4)(next 4 5)(next 5 6)(next 6 7) to the description of the first "straight" hand shown above. For the second "straight" hand, a different subset of background literals are relevant if we are to obtain the generalization: (card .XA .YA) (card .XB .YB)(card .XC .YC)(card .XD .YD) (card .XE .YE)(next .XA .XB)(next .XB .XC) (next .XC .XD) (next .XD .XE). To describe how relevant background literals are selected requires the concepts of associations and association chains.

Two literals L_1 and L_2 have an association $A_{i,j}(L_1,L_2)$ iff the ith term of L_1 is identical to the jth term of L_2. An association chain is a sequence of associations: $A_{i_1,i_2}(L_1,L_2)\ A_{i_3,i_4}(L_2,L_3)\ \ldots\ A_{i_n,i_{n+1}}(L_{n-1},L_n)$ where for even r: $i_r \neq i_{r+1}$. For example, if L_1 = (next 2 3), L_2 = (next 3 4), L_3 = (next 4 5), and L_4 = (odd 3) then $A_{3,2}(L_1,L_2)\ A_{3,2}(L_2,L_3)$ satisfies the definition of an association chain, but $A_{3,2}(L_1,L_2)\ A_{2,2}(L_2,L_4)$ does not. These two cases are illustrated in Figure 3.1, which shows that an association chain can be viewed as a threading together of literals such that the thread enters through one term and leaves through another.

association chain example



association chain counterexample

Figure 3.1 Example and Counterexample of Association Chains

We are interested in forming association chains in which the first and last literals in the chain are foreground literals and the rest are backbround literals. Background literals in such chains will be deemed relevant to the foreground instance to which they are linked. For production induction, these background literals will be added to the context of the production, and generalization will proceed as described earlier in this section.

Let γ_f denote the original production context, and γ_b denote the relevant background literals. Then the complete context is $\gamma = \gamma_b \cup \gamma_f$. The examples of section 4 will illustrate this context augmentation.

In practice, a limit must be placed on the length of the association chains which are considered, lest γ_b become too large. This prevents the computation times from becoming excessive, at the cost of ignoring information potentially relevant to the generalization process. None of the examples of this paper require association chains of more than four literals.

3.5 Deterministic Productions

A relational production whose context has been augmented with background information, $[\gamma_b,\gamma_f]\alpha \rightarrow \beta$, is defined to be deterministic iff all variables in β are either:

1) also present in α or γ_f, or
2) are linked by association chains through γ_b back into α or γ_f.

For example, the production

[ ] (at robot .X) ⟶ (at robot .Y)

is nondeterministic, since the variable .Y of the consequent does not appear in the antecedent or context. Given a particular present situation to which this production is applicable, .Y is a free variable, and so the new situation obtained by applying the production is not uniquely determined. However, the following production is deterministic:

[(next .N9 .N10), null](string .N13 .N11 .N12 .N9) ⟶ (string .N11 .N12 .N9 .N10)

Here γ_b = (next .N9 .N10) and γ_f = null. The variables .N11, .N12, and .N9 all appear in the antecedent. The variable .N10 participates in an association chain through the background context into the antecedent via .N9.

In summary, this definition approximately captures the intuitive idea that if a "deterministic" production is applied to a situation, the new situation should be uniquely determined.

4. APPLICATION OF PRODUCTION INDUCTION WITH BACKGROUND INFORMATION TO FOUR INTELLIGENCE PARADIGMS

This section illustrates the application of production induction with background information to four intelligence paradigms: letter sequence completion, visual analogies, operator induction from before-and-after pairs, and operator induction from a situation sequence. In each paradigm an

example is presented which was run on the computer implementation of the theory previously described. This program is designated Thoth-pb, one of a series of induction programs which have been developed. (In Egyptian mythology, Thoth is the god of knowledge and learning.) Thoth-pb consists of about 1,600 executable SN0B0L4 statements.

4.1 Application to the Letter Sequences Paradigm

In the letter sequences paradigm, a sequence of letters is presented, such as ABMCDMEFMG, and the problem is to predict the next letter in the sequence. For this paradigm, the background information is the ordering of the letters of the alphabet: (next A B)(next B C) ... (next Y Z). In common with Waterman (1975), we assume the sequence has a fixed "period" i, i.e., any letter in the sequence is a function of the previous i symbols. This assumption is not valid for sequences such as ABBCCCDDDD.... Suppose we assume the period is 2 (actually it is 3). Wc could then construct the following set of productions:

```
[ ]  (string A B) ⟶ (string M C)
[ ]  (string M C) ⟶ (string D M)
[ ]  (string D M) ⟶ (string E F)
```

Augmenting the context with background literals using a maximum chain length of four gives:

```
[(next A B)
 (next B C), null]  (string A B) ⟶ (string M C)

[(next D E)
 (next E F), null]  (string D M) ⟶ (string E F)

[(next G H), null]  (string M G) ⟶ (string H M)
```

Generalizing these productions gives:

[(next .NI .N2),null](string .N5 .N6)+(string .N3 .N4)

which is nondeterministic, as previously defined. Thus the period cannot be 2. Thoth-pb initially assumes the period is 1, constructing the productions:

```
[ ]  (string A) ⟶ (string B)
[ ]  (string M) ⟶ (string C)
          etc.
```

It then augments these productions with background literals and generalizes. If the result is not deterministic, it increases i by one and tries again, and so on, until either a deterministic generalization is found, or the period i becomes so large that two productions cannot be constructed. (Thoth-pb requires at least two productions to generalize.) For this example, for i = 3, the program computed the following deterministic generalization:

```
[(next .N1 .N2)
 (next .N4 .N1), null](string .N3 .N4 M) ⟶
 [(next .N3 .N4)]        (string .N1 .N2 M)
```

The total execution time was 11 seconds. To predict the next letter, we can match the left side of the production to the right end of ABMCDMEFMG using the substitution 8 = {E/.N3,F/.N4,G/.N1 ,H/.N2} The next letter is thus H, since .N2 follows .NI in the production. Thoth-pb only computes the generalization, without actually using it to predict the next letter; this is a simple clerical task. Thoth-pb has successfully solved all 15 problems in Simon and Kotovsky (1963).

4.2 Application to Visual Analogies

It seems significant that the same production induction process can be used in a problem class which, to the present time, has been regarded as distinct: the familiar visual analogy problems studied by Evans (1968). One such problem is shown in Figure 4.1. Using the analytic concepts of this paper, it is possible to give a formal representation and solution for such problems.

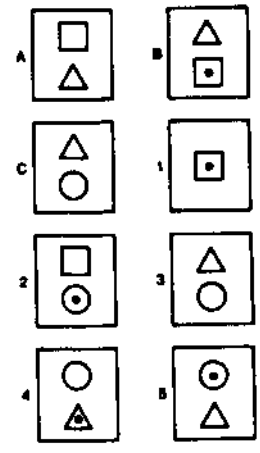


Figure 4.1 A Visual Analogy Problem

To begin, each scene must be translated into a conjunction of literals. This translation is a significant problem in itself, but not an induction problem. For example, scene B is translated into the conjunction: (ISA P3 TRIANGLE) (ISA P5 DOT)(ISA P4 SQUARE)(ABOVE P3 P4)(IN P5 P4). The end result of this translation, here performed manually, is a set of eight conjunctions s_A, sB, SC, S], ..., S5. From these Thoth-pb constructs the following six relational productions: SA * sB and $s_C \rightarrow s_i \quad 1 \le i \le 5$. The maximal common generalizations are computed for each of the following five pairs of productions: $g_i = mcg(s_A \rightarrow s_B,\ s_C \rightarrow s_i)\ 1 \le i \le 5$. Scene i_D is the "best" answer iff *g1* < gjh for all i # ib,. This example does not require background information, although more complex analogies may require background information for selection of the best answer.

For g1 and g3, Thoth-pb reported that no generalization exists; g_2, g4 and g5 are shown below.

```
                              (ISA .N1 DOT)
(ISA .N7 .N9)                 (ISA .N2 .N5)
(ISA .N6 .N8)        ⟶       (ISA .N3 .N4)
(ABOVE .N6 .N7)               (ABOVE .N3 .N2)
                              (IN .N1 .N2)

                              (ISA .N1 DOT)
(ISA .N7 .N4)                 (ISA .N2 .N5)
(ISA .N6 .N5)        ⟶       (ISA .N3 .N4)
(ABOVE .N6 .N7)               (ABOVE .N3 .N2)
                              (IN .N1 .N2)
```

```
                              (ISA .N1 DOT)
(ISA .N8 .N5)                 (ISA .N4 .N6)
(ISA .N7 .N6)      ⟶          (ISA .N3 .N5)
(ABOVE .N7 .N8)               (ABOVE .N3 .N4)
                              (IN .N1 .N2)
```

Applying the generalization theorem of section 3 the program observes that g2 < g4 and g5 < *g4.* Consequently scene 4 is chosen as best answer. Total execution time was 11 seconds.

4.3 Application to the Before-and-After Pairs Paradigm

In the before-and-after pairs paradigm, pairs of situations are presented which describe a system both before and after an action has occurred. More than one action may be exemplified. Optionally, "nonpairs" may also be presented, in which the transformation between the two situations was not caused by one of the actions.

Each before-and-after pair (o~b,o"$_a$) may be viewed as a single relational production $\sigma_b \rightarrow \sigma_a$. The new situation equation from section 2, $\sigma' = \sigma \cap \alpha\bar{0} \cup \beta\theta$, shows that this production exactly accomplishes the transformation from the first situation to the second. Consequently, a set of such pairs may be treated as a set of productions, which may then be augmented with background information and generalized as in section 3. Generalized productions which are "inconsistent" (Vcre 1975) with counterexample productions arc automatically discarded.

As an example, consider Figure 4.2 in which the arrows indicate legal and illegal white pawn moves in chess. The background information for this problem consists of the relative positions of the squares on a chess board, here labelled si through s64, plus information on the squares which form

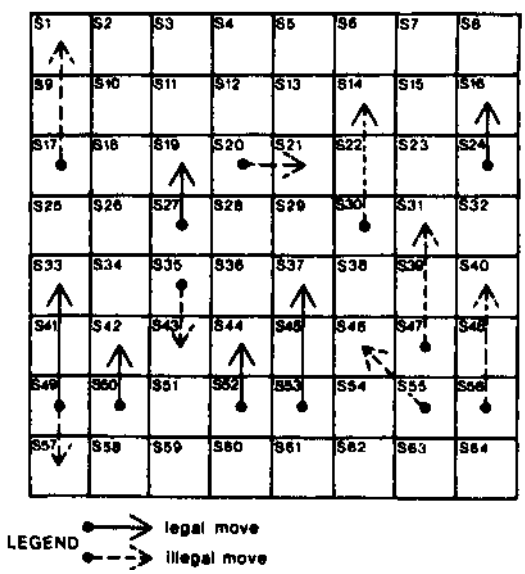


Figure 4.2 Examples and Counterexamples of the White Pawn's Move in Chess

the bottom row for white and for black. This information totals 128 literals. Literals such as (E SI S2), i.e., east of SI is S2, specify the horizontal position of the squares. Literals such as (S SI S9), i.e., south of S1 is S9, specify the vertical position of the squares. Literals such as (WBR S57) and (BBR SI), i.e., S57 is in the white bottom row and S1 is in the black bottom row, specify the bottom rows for white and black. The position of the pawn before and after a move is foreground information. For example, the move from square S49 to S33 is represented by the situation pair: (WP S49), (WP S33). Here the "WP" literals indicate the position of a white pawn. Nonpairs are represented in the same way. This problem consists of seven pairs and seven "nonpairs". Thoth-pb computed the following generalizations in 100 seconds:

```
[(S .N5 .N6), null](WP .N6) ⟶ (WP .N5)

⎡(WBR .N3)                ⎤
⎢(S .N4 .N2)              ⎥
⎢(S .N1 .N4), null        ⎥(WP .N2) ⟶ (WP .N1)
⎣(S .N2 .N3)              ⎦
```

The first production describes the standard "one square north" move by a white pawn. The second production describes a "two squares north" move, which is possible only when the pawn is in its initial position, located one square above the white bottom row.

If Thoth-pb is given only the seven legal move pairs, without counterexamples, it overgeneralizcs, obtaining the following conjunctive generalizations:

```
[(S .N3 .N2), null](WP .N2) ⟶ (WP .N1)
[(S .N4 .N6), null](WP .N5) ⟶ (WP .N4)
```

Both productions cover all seven before-and-after pairs, but they are nondeterministic, as defined in section 3.5. By requesting Thoth-pb to ignore nondeterministic generalizations, we can obtain (in 84 seconds) the two correct, disjunctive generalizations seen above without providing any counterexamples.

4.4 Application to the Situation Sequence Paradigm

The situation sequence paradigm is closely related to the before-and-after pairs paradigm. A sequence of situations is presented, describing successive "snapshots" of a discrete system undergoing change. The problem is to find a minimal set of productions, one of which is capable of causing the transformation between any pair of adjacent situations in the sequence.

Figure 4.3 shows a sequence of six situations in the familiar Tower of Hanoi puzzle. Each scene will be represented using "on" and "clear" literals, as in section 2. The elements involved are the three discs A, B, C and the three pegs PI, P2, P3. For example, the second scene is represented by the following conjunctions: (clear C) (on C PI)(clear A)(on A B)(on B P2)(clear P3). The background information for this problem consists of literals giving the relative sizes of the elements: (L B A)(L C A)(L C B)(L P1 A)(L P1 B)

(L P1 C)(L P2 A)(L P2 B)(L P2 C)(L P3 A)(L P3 B) (L P3 C). Here (L X Y) may be interpreted as "X is larger than Y". For example, the literal (L P2 A) may be interpreted as "the base of P2 is larger than A".

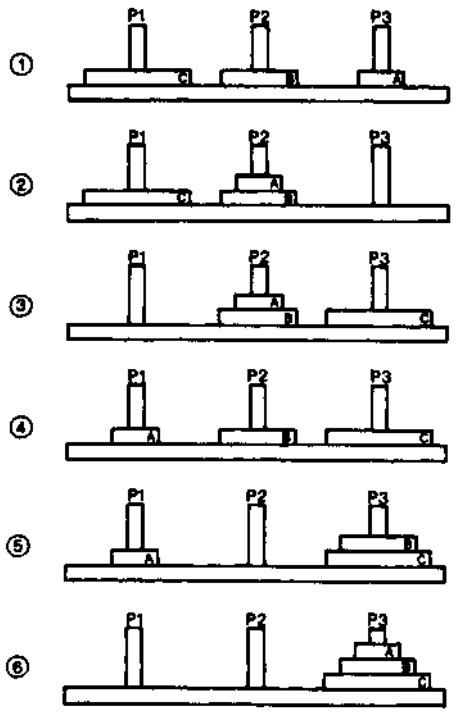


Figure 4.3 A Tower of Hanoi Situation Sequence

From the six conjunctions describing the scenes, Thoth-pb constructs five productions from the adjacent pairs of descriptions, and then augments the context of each production with relevant background information. For example, from the first two descriptions is constructed the production:

$$\begin{bmatrix}(\text{L B A}) & (\text{clear C})\\ (\text{L P3 B}) & (\text{on C P1})\\ (\text{L P3 A}), & (\text{clear A})\\ & (\text{on B P2})\end{bmatrix} \begin{matrix}(\text{clear B})\\ (\text{on A P3})\end{matrix} \longrightarrow \begin{matrix}(\text{on A B})\\ (\text{clear P3})\end{matrix}$$

The following conjunctive generalization of the five productions was obtained in 36 seconds:

$$\begin{bmatrix} & (\text{clear .N2})\\ (\text{L .N3 .N2}) & (\text{clear .N8})\\ (\text{L .N1 .N2}), & (\text{on .N6 .N7})\\ & (\text{on .N4 .N5})\end{bmatrix} \begin{matrix}(\text{clear .N1})\\ (\text{on .N2 .N3})\end{matrix} \longrightarrow \begin{matrix}(\text{on .N2 .N1})\\ (\text{clear .N3})\end{matrix}$$

This generalization corresponds closely to our intuitive idea of the Tower of Hanoi operator, except for the last three literals of the foreground context, which reflect an idiosyncrasy of this sequence: in every case when a move occurred, there happened to be an .N8 clear, an .N6 on an .N7 and an .N4 on an .N5. Note that none of these variables have associations to the production antecedent or consequent. Such "noise" literals could be screened out by checking this property. Alternatively, we could show an additional sequence involving just one disc to break the idiosyncrasy.

5. CONCLUSION

Relational production induction has been shown to be a common process in four distinct, intelligence paradigms, which suggests that it may be a basic principal of inductive inference. The examples of section 4 all require only modest computation times, and are far from being upper bounds on the difficulty of problems solvable by these methods. In induction, as in problem solving and computation in general, the style of representation of information influences the form and difficulty of solutions. This paper has assumed error free data: "noisy" data presents a topic for future work. The background information concept could be generalized by allowing the literals to be dynamically inferred, instead of requiring them to be stored explicitly.

REFERENCES

Egan, D.E. and Greeno, J.G. 1974 "Theory of Rule Induction." in Knowledge and Cognition, L.W. Gregg (Ed.). Wiley, New York, pp. 43-103.

Evans, T.G. 1968 "A Program for the Solution of Geometric-Analogy Intelligence Test Questions." in Semantic Information Processing, M.L. Minsky (Ed.). MIT Press, Cambridge, pp. 271-353.

Hikes, R.E. et al. 1972 "Learning and Executing Generalized Robot Plans." Artificial Intelligence, Vol. 3, pp. 251-28B.

Hayes-Roth, F. and McDermott, J. 1976 "Knowledge Acquisition from Structural Descriptions." Department of Computer Science, Carnegie-Mellon University.

Hedrick, C.L. 1976 "Learning Production Systems from Examples." Artificial Intelligencc, Vol. 7, pp. 21-49.

Post, E.L. 1943 "Formal Reductions of the General Combinatorial Decision Problem." American Journal of Math., Vol. 65, pp. 197-268.

Simon, H.A. and Kotovsky, K. 1963 "Human Acquisition of Concepts for Sequential Patterns." Psychological Review, Vol. 70, pp. 534-546.

Soloway, E.M. and Riseman, E.M. 1976 "Mechanizing the Common-Sense Inference of Rules with Direct Behavior." Proc. AISB Summer Conference, University of Edinburgh, pp. 307-321.

Vere, S.A. 1975 "Induction of Concepts in the Predicate Calculus." Proc. Fourth Intl. Joint Conf. on Artificial Intelligence, Tbilisi, USSR, pp. 281-287.

Vere, S.A. 1977a "Relational Production Systems." Artificial Intelligence, Vol. 8, pp. 47-68.

Vere, S.A. 1976 "Composition of Relational Productions for Plans and Programs." Dept. of Information Engineering, University of Illinois at Chicago Circle.

Vere, S.A. 1977b "Inductive Learning of Relational Productions." Proceedings of the Workshop on Pattern-Directed Inference, Hawaii.

Waterman, D.A. 1975 "Adaptive Production Systems." Proc. Fourth Intl. Joint Conf. on Artificial Intelligence, Tbilisi, USSR, pp. 296-303.

Williams, D.S. 1972 "Computer Program Organization Induced from Problem Examples." Representation and Meaning, H.A. Simon and L. Siklossy (Eds.), Prentice-Hall, pp. 143-205.