

A FRAMEWORK FOR CONTROL IN PRODUCTION SYSTEMS

M.P. Georgeff
School of Mathematical Sciences
Flinders University
Bedford Park, S.A., 5042.
Australia

A formal scheme for representing control in production systems is defined. The scheme allows control to be directly specified independently of conflict resolution and thus allows the issues of control and nondeterminism to be treated separately. Unlike previous approaches, it allows control to be examined within a uniform and consistent framework. It is shown that the scheme provides a basis for implementing control constructs which, unlike existing schemes, retain all the properties desired of a knowledge based system---modularity, flexibility, extensibility and adaptive capacity. Within the formalism it is also possible to provide a meaningful notion of the power of control constructs. This enables the types of control required in production systems to be examined and the capacity of various schemes to meet these requirements to be determined.

1. INTRODUCTION

Over the years a range of different mechanisms have been proposed for representing and using knowledge about general and possibly ill-defined problem domains. Of these, production systems [12] have been among the most promising, and have been applied to a diverse collection of problems, including mass spectroscopy [4], medical diagnosis [15], electronic circuit design [10] and automated theory formation in mathematics [8]. Most theorem provers can also be viewed as production systems (e.g. PROLOG [16]).

Informally, a production system consists of a set of modules or procedures called productions and a data base on which these productions operate. Now one of the most fundamental and significant characteristics of production systems is the lack of explicit control information---that is, production invocation can only be achieved indirectly through the data base. The primary effect of this indirect means of production invocation is to produce a system which is strongly modular, flexible and adaptive, and thus well-suited as a knowledge based system. However, it is also perhaps the most significant factor in complicating the programming of production systems, in making the behaviour flow more difficult to analyse, and in increasing the difficulty of an adequate formalization [1].

Now there are two reasons why we would like to have control. The first, and to which we alluded above, is that the solutions to many problems are most naturally represented by sequences of

actions rather than by sets of actions in which the order of application is unimportant. We tend to use plans or strategies, even when we are manipulating declarative knowledge or facts about the world (e.g. consider the wide use made of procedural attachment in most knowledge based systems). Secondly, any large production system that does not somehow constrain the number of productions that are active at any one time becomes so inefficient as to be unworkable. The problem is: how do we achieve such control without sacrificing the *raison d'etre* of production systems.

Many production systems use some form of control information. Usually, this remains hidden in the productions or in the data base in the form of special flags and other hand-crafted markers [11]. Other systems use more explicit means of control (e.g. NASL [10], annotated systems [6]), but the control structures are limited and are often difficult to access or modify. But most importantly, no system provides a uniform framework in which control issues can be addressed, nor a direct means of implementing control constraints---in every case the control schemes are essentially ad hoc.

In this paper we approach the problem from the other direction. That is, we formally characterize the notion of control applied to production systems, and then consider how best to implement such a scheme. The scheme we propose, which we will call a *controlled production system*, simply consists of a production system together with a control device called a

control language. This provides us with a uniform framework for representing control in production systems, and allows us to examine in a very general way how the various approaches to realizing (implementing) control affect the properties and behaviour of the system as a whole.

In fact, it first appears that the notion of control in production systems is quite obvious, and hardly needs formalizing. However, if we examine the control schemes of current production systems---apart from their ad hoc nature---we see control being used in quite different ways. For example, in most schemes (e.g. PROLOG) control is *intended* to simply enhance the efficiency of the system---given enough time, the system without the control component would find the same solutions as the system with control. In other schemes (e.g. [13]), and *in fact* in most of the aforementioned schemes (e.g. PROLOG), control can be used in the same manner as in procedural languages---that is, the solutions obtained depend critically on the order in which the productions are invoked. Such confusion leads to ad hoc systems the behaviour of which it is very difficult to predict and the solutions to which it is very difficult to validate. That is, it leads to a programming methodology quite the opposite of that favoured for procedural languages. Further, domain specific control knowledge is difficult to realize as it becomes confused with efficiency issues [3]. Formalizing the notion of control avoids these difficulties, and allows system efficiency to be treated as a separate issue.

2. CONTROLLED PRODUCTION SYSTEMS

2.1 An Informal Description

Informally, a production system (PS) consists of a set of modules or procedures called *production rules* and a *data base* or *working memory* to which these rules are applied. Each production rule is an expression or string of symbols which consists of two parts called the *lefthand-side* (LHS), or *antecedent*, and the *righthand-side* (RHS), or *consequent*. These respectively denote a *condition* which is to be satisfied before the production can be applied or invoked, and an *action* which specifies the result of application of the production to the data base.

In the simplest execution scheme, the conditions in each production are evaluated for the current state of the data base, one of the satisfied productions is selected, and the action specified by that production then executed. The procedure is then repeated for this new state of the data base. Execution terminates either when there

are no satisfied productions or when some desired state of the data base is achieved. In general execution is nondeterministic, as at any stage during execution more than one production may be applicable. The set of productions applicable at each stage of execution is known as the *conflict set*, and the selection procedure is usually called *conflict resolution*.

For example, consider the following production system where states of the data base are words over the alphabet $\{S, A, B, C, a, b, c\}$ and where the productions (which are to be interpreted in the normal rewriting sense) include

$p_1: S \rightarrow ABC$	$p_5: A \rightarrow a$
$p_2: A \rightarrow aA$	$p_6: B \rightarrow b$
$p_3: B \rightarrow bB$	$p_7: C \rightarrow c$
$p_4: C \rightarrow cC$	

For an initial state S of the data base, the set of possible final states is the set of strings $\{a^l b^m c^n: l \geq 1, m \geq 1, \text{ and } n \geq 1\}$. Depending on how the system is implemented, it can be used either as a generator of these strings or as a recognizer for these strings.

Suppose that we now wish to introduce explicit control constraints into such a system. One way to achieve this is to allow productions to throw special symbols into the data base to constrain invocation of other productions (e.g. [11]). However, this scheme has numerous drawbacks. Firstly, in order that the desired constraint is effected, it is necessary to invest the productions that respond to these special symbols with a higher priority of invocation than all other productions that might also be satisfied by the current state of the data base. We thus end up with two classes of symbols in the data base, or equivalently, two classes of productions. Nothing is wrong with this, of course, except that we have changed the nature of the production system. Secondly, it is clear that the above scheme will not work unless the special symbols are known to be unique to the invoking and the invoked productions, and such uniqueness can only be established by reference to the condition part of all other productions. Other problems are also present in the above scheme. The system loses its potential extensibility as these special symbols essentially invoke productions by name rather than by content. Further, augmentation and modification of control information is extremely difficult.

Other control schemes have been proposed which avoid some of these problems (e.g. [13]), but control is still achieved indirectly through the conflict resolution scheme. One consequence of this indirection is that code is also indirect and cumbersome, as, for example, that

needed for keeping loop productions dominant during iteration [13]. What is worse, control information can only be expressed having a full knowledge of the conflict resolution scheme, and this is not always possible nor desirable - consider, for example, systems that involve a dynamic conflict resolution scheme e.g. TEIRESIAS [2]. Furthermore, in order that control constraints are not eventually overridden, production sequencing determined during conflict resolution must be irrevocable, at least for control elements, and this brings into question the function of the conflict resolution scheme as a means of handling non-determinism.

We thus adopt an alternative approach whereby we specify control information explicitly and independently of the conflict resolution scheme. To do this we will simply require that any constraints on production invocation be specified by means of a language over the production set. We will call such a language a *control language*. A production sequence and the relation it defines on the data base is then only allowed if this production sequence is included in the control language. Thus at each stage of execution, the control language restricts the set of productions that may be considered for invocation and only a subset of the total production set is *active*. The only productions that can enter the conflict set are those that both have their condition satisfied by the current state of the data base and are contained in the active production set. We will call a production system together with a control language a *controlled production system* (CPS).

For example, consider the above production system together with a control language defined by the regular expression

$$p_1(p_2p_3p_4)^*p_5p_6p_7$$

Then the set of final states of the data base given the initial state S is the set of strings $\{a^n b^n c^n : n \geq 1\}$. Note that this controlled production system is nondeterministic and that after executing production p_4 the conflict set will contain two productions (namely, p_2 and p_5).

Thus it is seen that a controlled production system differs from the more usual production systems only in that it has an explicit and independently specified control structure. This control structure acts as a *constraint* on production invocation---it effectively reduces the possible interactions between productions. In one sense, the control structure provides private channels of communication between productions. This allows the power of a production system to be increased without increasing the complexity of the productions. For example, if we allow productions that

check for a symbol not being in the current state of the data base, then it can be shown that context-free rewriting systems with regular control languages generate all the recursively enumerable languages [14].

2.2 Formal Definitions

A *production system* (PS) is a triple

$$P = \langle \Sigma, D, h \rangle \quad (2.1)$$

where Σ is an alphabet called the set of *production names*, D is a set called the *data base* and h , called the *interpretation* of Σ , assigns to each element of Σ a pair $\langle q, r \rangle$, where q is a total predicate on D and r is a relation on D . For any production name p , we will say that the first co-ordinate of $h(p)$ is the *condition* denoted by p and that the second co-ordinate of $h(p)$ is the *action* denoted by p . Elements of D will usually be referred to as *states* of the data base. For each production p , we will also require that all states of the data base which satisfy the condition of p be in the domain of the action of p .

We now introduce the notion of a controlled production system. We take as our control device a language over Σ . Formally a *control language* over Σ is any subset of Σ^* , where Σ^* is the free monoid over Σ with identity λ .

We define a *controlled production system* (CPS) to be a quadruple

$$C = \langle \Sigma, D, h, C \rangle \quad (2.2)$$

where C is a control language over Σ and $\langle \Sigma, D, h \rangle$ is a production system. A CPS $\langle \Sigma, D, h, C \rangle$ with $C = \Sigma^*$ is equivalent to the PS $\langle \Sigma, D, h \rangle$. We can therefore consider a PS to be a special case of a CPS.

We are now in a position to define the execution of a CPS. Let C be a CPS as defined in (2.2). We first define a *state of execution* (or simply *state*) of C to be a pair $s = \langle u, x \rangle$ where u is a prefix of some word in C and x is an element of D . Now let u and up be prefixes for some word in C , where $u \in \Sigma^*$, $p \in \Sigma$ and $h(p) = \langle q, r \rangle$. Then we say a state $\langle up, x_2 \rangle$ is *directly computed* from a state $\langle u, x_1 \rangle$, denoted $\langle u, x_1 \rangle \Rightarrow_c \langle up, x_2 \rangle$, if and only if we have $q(x_1) = \text{true}$ and $\langle x_1, x_2 \rangle \in r$. If the above holds, we will also say that the state $\langle up, x_2 \rangle$ results from *execution* of production p in state $\langle u, x_1 \rangle$.

Let $\Rightarrow_c^*$ denote the reflexive transitive closure of the relation $\Rightarrow_c$. Then the *relation computed by C* is defined to be the set $R(C) = \{ \langle x, y \rangle : \langle \lambda, x \rangle \Rightarrow_c^* \langle w, y \rangle \text{ for some } w \text{ in } C \}$

Informally, a CPS C may be (nondeterministically) executed for some initial state x of the data

base D in the following manner. At each moment of time, execution is at some state. Initially this state is $\langle \lambda, x \rangle$. Suppose that execution has arrived at some state $\langle u, y \rangle$. Now a production p can be considered for evaluation if up is a prefix of some word in C. Let us call the set of all such productions the *active production set*. Suppose this set is empty. Then execution terminates successfully if u is an element of C and terminates unsuccessfully if u is not in C. Otherwise, execution may either terminate successfully (if u is in C) or continue by evaluating the condition of each active production with respect to the current state of the data base. All of those productions which are satisfied form what is known as the *conflict set*. If the conflict set is empty, then execution terminates unsuccessfully. Otherwise, a production p is (nondeterministically) selected from this set, and execution continued from a (not necessarily unique) state $\langle up, z \rangle$, where $\langle y, z \rangle$ is an element of the action denoted by p. The final state of the data base is obtained on successful termination of execution.

The most obvious way to implement such a scheme is simply to augment the recognize-act cycle of classical production systems with a stage that determines, on the basis of the control language, the set of active productions (or equivalently masks off the non-active productions). However, it is not clear how the control language should be specified.

3. SYSTEM BEHAVIOUR

It is first necessary to consider how the control language and its means of specification will affect the behaviour of the system as a whole. In particular, we need to consider how the imposition of a control language will affect the additivity and adaptive behaviour of the system.

3.1 Additivity and Adaptive Behaviour

Extensibility and the capacity for self modification are important characteristics of intelligent systems, and any control scheme should allow of such behaviour. Consider for example the problem of learning to add two positive integers given only the successor function. We will need the following two actions: DEP(d) which places d in the data base and PUT(p,q,r,c) which (i) adds the production with name p, LHS q and RHS r to the production set and (ii) adds the symbol string c to the control language. GENSYM is a parameterless procedure which generates a new symbol (production name), and s is the successor function.

The initial production set for solving this problem is as follows

```
p1: (ready) → READ; DEP(count 0)
p2: (count ?x1)(m ≠ ?x1)(n ?x2) → DEP(count (s ?x1));
    DEP(n (s ?x2))
p3: (count ?x1)(m ?x1)(n ?x2) → WRITE(?x2)
p4: (n ?x1) → DEP(n0 ?x1)
p5: (count ?x1)(n ?x2)(n0 ?x3) → ?x4 + GENSYM;
    PUT(?x4, ((m ?x1)(n0 ?x3)), (WRITE ?x2), p1 p4 ?x4)
(The notation obeys typical AI language conventions i.e. variables prefixed by ? have values bound to them and when bound must match that value, and variables prefixed by ≠ are taken to match anything other than the value bound).
With the control language p1 p2 * p3 the system performs addition without learning anything, with the control language p1 p4 p2 * p3 p5 we have essentially the example in [17], and with the control language p1 p4 (p2 p5) * p3 the system learns intermediate results as well.
```

The important point to note is that adaptive behaviour and additivity - in the sense of being able to modify the production set - is not precluded by the introduction of a control component. Further, a control language need not, and rarely will, specify all and only those production sequences that represent solution paths, but will simply place (greater or lesser) constraints on production invocation. Thus the synergic properties of the system will also be retained.

3.2 Semantic Specifications

In the above example we achieved additivity at the expense of having to explicitly augment the control language. Now in many applications such explicit modification of the control language will be difficult.

For example, consider the problem of cascading two amplifiers [10]. We could represent this problem using productions of the form

```
t0: (make cascade-amp) → (make collector)(make emitter)(make couple)
```

```
t1: (make collector) → ...
t2: (make emitter) → ...
t3: (make couple) → ...
```

together with the control constraint given by the regular expression $(t_1 t_2 + t_2 t_1) t_3$. In English, this simply says that to cascade two amplifiers we first construct a common collector and a common emitter (in any order) and then couple them. Now if we add to the production set a new method t4 for making collectors this new method will never be considered unless we explicitly substitute $(t_1 + t_4)$ for each occurrence of t_1 in the above control sequences. Furthermore, we should do the same thing for every control sequence in which t_1 appears, not just those appearing above.

The problem clearly lies in the means used for specifying the control language. Let us briefly consider what's going on. Most languages are specified syntactically because of a desire to describe the language solely in terms of its alphabet without regard to any semantic interpretation of the language. But in the present case there is no need to be so restrictive. As the interpretation is a component of the formal model (see eqn 2.2), it can also be used in specifying the control language, that is, the control language can be specified by its *semantic content* rather than its syntactic form. Thus in the above example, we could have specified the control constraint semantically:

if p_1 is a production that makes a collector (or whose LHS mentions 'collector')

and p_2 is a production that makes an emitter and p_3 is a production that couples them, then $p_1 p_2 p_3$ and $p_2 p_1 p_3$ are in the control language.

p_1, p_2 , and p_3 are variables ranging over the set of productions, and the specification says, in effect, that a production that makes a common collector and a production that makes a common emitter should be invoked before a production that couples the two components.

Semantic specification thus avoids the problem of explicit modification of the control language, and additivity is better realized. Most other properties desired of knowledge based systems are also retained - the system remains highly modular and flexible, and retains its explanatory capacity. Of course, the dependence of these properties on the means of production reference is well known [2] - the only difference is that here we are using semantic reference to effect control constraints rather than to improve system performance.

Semantic control information is also the type of information most likely to be possessed by an expert interacting with such a knowledge based system. For example, consider a world consisting of some blocks and a (large) table. Now we know that in an optimum solution we will never put one block on top of another block that has to be subsequently cleared, which we may express as follows:

if the action part of a production puts a block x on a block y then it cannot be followed (no matter how much later) by a production whose condition part requires y to be clear.

We can then interpret this directly as a semantic specification of the control language. (Interestingly, such a simple control constraint considerably reduces the search space as, for example, production sequences of the form

... (PUTON x y) ... (PUTON y z) ...

are excluded from consideration.)

One problem with implementing such a scheme is that in the general case it is very difficult to extract the required semantic information from a body of code. The problem is not so severe in (controlled) production systems because the code usually consists of a number of relatively small, independent chunks of knowledge, and these often represent computationally primitive operations. More important, however, is that in a CPS the control information is specified in such a way that semantic criteria can be used as a means of invocation---how the semantic information is obtained can be treated as a separate issue [3].

The other problem is one of system efficiency. Even in cases where the object level system is made more efficient, the time spent in evaluating semantic criteria can easily offset these gains. However, if the production set is fixed then it is not necessary that the semantic criteria be evaluated at execution time---in many cases it will be more efficient to transform the semantic specification into a syntactic specification at compile time. In these cases, semantically based invocation can be expected to provide both a powerful and relatively efficient means of control.

4. CONTROL LANGUAGE DEVICES

Any desired control constraint can be specified by a control language. In turn, however, we require some device for specifying control languages. Now we are in a better position to construct such a device if we know what class of languages the device is required to generate. Thus in this section we examine some typical AI control constructs and see how they are placed within the standard hierarchy of languages [14].

4.1 Control Language Types

One of the simplest classes of languages is the type 3 or regular languages. As control languages they are surprisingly powerful---thus, as mentioned in Section 2.1, they can increase the power of context-free rewriting systems to that of a Turing machine. As the solutions to many problems are often conceptualized as a sequence of actions, regular control languages also improve constructibility.

Both direct sequencing and iterative control can be specified by means of regular control languages (e.g. see the example of Section 3.1). This class of languages is also powerful enough to realize partitioned production systems (procedures [11], packets or multiple production

memories [9]). If we consider the control language to be generated by a transition network, then each state in that network defines a set of active productions, these being the productions that label the outgoing arcs. Each transition either loops and thus leaves control in the same state (i.e. with the same set of active productions) or transfers control to another state (i.e. to another set of active productions). Thus as long as we continue to loop on a given state we effectively operate on a subset of the entire production memory.

Context-free control languages can be used to specify recursive procedures or goal-subgoal control constructs where the subgoals are to be achieved in a specified order. This is readily seen as follows. Let the control language be generated by a context-free grammar. We can interpret each non-terminal symbol appearing in the grammar as the name of a procedure in the CPS, and each terminal symbol simply as the name of a production in the CPS. Now (top-down) left-right generation of control language sentences produces possible execution sequences of the CPS, where each expansion of a non-terminal symbol is interpreted as a call to the procedure having that name. In AI terms, each expansion of a non-terminal is equivalent to the reduction of a goal to a sequence of subgoals. Such control constraints correspond to the SUCCESSOR and SUBTASK relations in NASL [10] and to the fall-back and held-result-usage control constructs described in [13].

Let us consider again the problem of constructing a cascade of amplifiers. The solution proposed in Section 3.2 was not really sufficient, as firstly it did not restrict the scope of the control constraint to the case of cascading amplifiers, and secondly because it required the subtasks (i.e. the making of the common collector, etc.) to be primitive. However, we can overcome both these difficulties by letting the production names stand as non-terminals in a context-free grammar which contains the rules

$$t_0 \rightarrow t_1 t_2 t_3 \quad t_0 \rightarrow t_2 t_1 t_3$$

The result is that in trying to achieve t_0 , we have to achieve t_1 and t_2 before t_3 , where this time t_1 , t_2 and t_3 may themselves be expanded into further (sequences of) subtasks. (In fact, we have really changed the entire nature of the object level production system, as it now need contain only primitive productions (i.e. productions with no subtasks). For our purposes this is not important, but it does indicate that much domain specific knowledge, in our terms at least, occurs at the control level).

One particularly interesting aspect of context-free CPSs is that they encompass augmented

transition networks (ATNs) [18]. An ATN is simply a controlled production system where the context-free control language is specified not by a phrase structure grammar but by a recursive transition net. The conditions and actions on each arc of the transition net are simply the conditions and actions denoted by the LHS and RHS of the production corresponding to that arc. Thus an input word to an ATN simply acts as a control word (strictly a homomorphism of a control word) over the productions attached to each arc of the network.

There are also a number of control constructs which cannot be specified by context-free control languages. For example, the control language of the blocks world example (Section 3.2) is not a context-free language. Similarly, direct-result-usage control constructs [13] need language generators of the form merge (t_1, t_2), where t_1 and t_2 act as non-terminals in a phrase structure grammar and merge (t_1, t_2) is to be interpreted as providing all merges of the words generated by t_1 with those generated by t_2 . Again, such languages are not context-free.

Of course, we are not suggesting that we should use phrase structure grammars for specifying control languages - what we are saying is that we need to make sure our control device, whatever it is, has the power to describe the types of control that we need. For example, if we are to allow context-free control languages - and the above considerations suggest that we should - then the type of productions used in TEIRESIAS [2] for specifying production orderings at the meta-level are not going to be powerful enough at the control level as they can only describe regular languages.

4.2 Control Device Structure

We have so far said nothing about the structure of the device to be used in specifying the control language. If the specification is to be syntactic, then any of the standard methods --- explicit enumeration, property specification, finite state automata, phrase structure grammars, augmented transition networks, etc. --- could be used. These devices could also be used where the control language is to be specified semantically, but would need to be augmented with some inference device. Alternatively, the system could be provided with a number of control primitives, such as those used in NASL [10] or described in [13]. Of course, for a CPS these primitives would be implemented through a separate control device, rather than indirectly through the conflict resolution scheme.

However, all the above approaches limit, to a greater or lesser degree, the additivity and

flexibility of the system to the object level. A potentially more powerful approach is thus to specify the control language using a second level or *control level* CPS. This reflects very closely what McDermott had in mind when designing NASL - that the order of steps within and between subplans be itself rule governed. One advantage of this approach is that the programmer has available a language and framework in which he can easily define his own invocation criteria. Furthermore, the representation of knowledge at both the object level and control level is uniform, and additivity and flexibility is preserved at both levels.

The control language of the control level CPS would need to be specified, and one could envisage a hierarchy of CPSs, each determining the control language of the one below it. There is probably not much advantage in such a wealth of control language CPSs. Of course, this is not to suggest that there should be no further levels of knowledge - for example, it might be desirable to handle non-determinism using a meta-level CPS [1,3,5].

5. CONCLUSIONS

The importance of control information should not be underemphasized. In placing constraints on production invocation, control reduces the interaction between knowledge units. The more control constraints we impose, the fewer patterns of interaction have to be explored, and the smaller and less complex the search space. To paraphrase Hayes [7] it is precisely the resulting restrictions on interactions that makes control so useful.

In this paper we have presented a uniform framework in which control issues can be addressed, and which provides a basis for implementing such control. We have attempted to show that the introduction of a explicit control device need not result in the loss of any of the properties desired of a knowledge based system, and in particular that additivity and adaptive behaviour can be retained. Given that the power of production systems is equal to that of controlled production systems, the question remains as to whether it is advantageous to introduce control at all. At an intuitive level it would appear that the answer is yes - it is not difficult to find problem areas where expert knowledge is expressed in terms of sequences of actions. Further, if we look at the formal language domain, regulated (controlled) rewriting systems are usually simpler to construct than equivalent (uncontrolled) rewriting systems. Of course, until a system is actually implemented and applied to typical AI problems, the question

must still remain open.

REFERENCES

- [1] Davis, R. & King, J.J. "An Overview of Production Systems", in Mach. Int. 8: Machine Representations of Knowledge, ed. Elcock & Michie, Wiley, N.Y., 1977.
- [2] Davis, R. "Generalized Procedure Calling and Content-Directed Invocation" SIGPLAN/SIGART Newsletter, August, 1977, pp. 45-54.
- [3] Davis, R. & Buchanan, B.G. "Meta-Level Knowledge: Overview and Applications", Proc. IJCAI-77, 1977, pp. 920-927.
- [4] Feigenbaum, E.A., Buchanan, B.G. & Lederberg, J. "On Generality and Problem Solving---a case study involving the DENDRAL program", STAN-AIM-131. Stanford Uni., Ca., 1971.
- [5] Georgeff, M.P. "A Framework for Control in Production Systems" Stanford AI-Memo, Stanford Uni., Ca., 1979.
- [6] Goldstein, I.P. & Grimson, E. "Annotated Production Systems: A Model for Skill Acquisition" Proc. IJCAI-77, 1977, pp. 311-317.
- [7] Hayes, P.J. "In Defense of Logic", Proc. IJCAI-77, 1977, pp. 559-565.
- [8] Lenat, D.B. "Automated Theory Formation in Mathematics", Proc. IJCAI-77, 1977, pp. 833-842.
- [9] Lenat, D.B., & McDermott, J. "Less Than General Production System Architectures". Proc. IJCAI-77, 1977, pp. 928-932.
- [10] McDermott, D. "Flexibility and Efficiency in a Computer Program for Designing Circuits", AI-TR-402, MIT AI Lab, 1977.
- [11] Moran, T.P. "The Symbolic Imagery Hypothesis: A Production System Model", Comp. Sci. Dept., Carnegie-Mellon University, December, 1973.
- [12] Post, E. "Formal Reductions of the General Combinatorial Problem", Am Jnl Math, 65, 1943, pp. 197-268.
- [13] Rychener, M.D. "Control Requirements for the Design of Production System Architectures" SIGPLAN/SIGART Newsletter, 1977, pp. 37-44.
- [14] Salomaa, A. Formal Languages, Academic Press, N.Y., 1977.
- [15] Shortliffe, E.H. MYCIN: Computer-Based Consultations in Medical Consultations, American Elsevier, 1976.
- [16] Warren, D.H.D. "Implementing Prolog---Compiling Predicate Logic Programs" AI-Report, Edinburgh Uni., 1977.
- [17] Waterman, D.A. "Adaptive Production Systems" Proc. IJCAI-75, 1975, pp. 296-303.
- [18] Woods, A.T. "Transition Network Grammars for Natural Language Analysis", Comm ACM, 13:10, 1970, pp. 591-606.