Stanford Artificial Intelligence Laboratory
Memo AIM-322

Computer Science Department
Report No. STAN-CS-79-716

January 1979

A FRAMEWORK FOR CONTROL IN PRODUCTION SYSTEMS

by

Michael Georgeff

Research sponsored by

Advanced Research Projects Agency

COMPUTER SCIENCE DEPARTMENT
Stanford University

A FRAMEWORK FOR CONTROL IN PRODUCTION SYSTEMS

by

Michael 'Georgeff *

Abstract

A formal model for representing control in production systems is defined. The formalism allows control to be directly specified independently of the conflict resolution scheme, 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 formalism 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 explanatory capacity. Most importantly, it is shown that these properties are not a function of the lack of control constraints, but of the type of information allowed to establish these constraints.

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.

Schemes for improving system efficiency and resolving nondeterminism are examined, and devices for representing such meta-level knowledge are described. In particular, the objectification of control information is shown to provide a better paradigm for problem solving and for talking about problem solving. It is also shown that the notion of control provides a basis for a theory of transformation of production systems, and that this provides a uniform and consistent approach to problems involving subgoal protection.

Present address:
School of Mathematical Sciences,
Flinders University,
Bedford Park,
South Australia, 5042,
Australia.

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 [Post 1943] have been among the most promising, and have been applied to a diverse collection of problems, including mass spectroscopy [Feigenbaum 197 1], medical diagnosis [Shortliffe 1975], electronic circuit design [McDermott 1977] and automated theory formation in mathematics [Lenat 1977). Most theorem provers can also be viewed as production systems (e.g. PROLOG [Warren 1977]).

Informally, a production system consists of a set of modules or procedures called production rules and a data base on which the production rules 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 an expert knowledge 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 [see Davis 1976-J

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 slot fiiiing specifications and 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 handcrafted markers [e.g. Moran 1973]. Other systems use more explicit means of control (e.g. NASL [McDermott 1977], annotated systems [Goldstein & Grimson 1977]), 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 uniform 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 (or 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. Rychener 1977], 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 [cf. Davis & Buchanan 1977]. 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 *lefthandside* (LHS), or *antecedent,* and the *righthandside* (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.*

Such an execution scheme is said to be *forward-chaining* or *data-driven.* However, in essence **a** sequence of productions simply defines a relation on states of the data base, and there is no reason why evaluation of this relation cannot proceed differently. Thus ***backward-chaining*** **or** *hypothesis-driven* schemes proceed from a final state to an initial state, and *bi-directional* schemes proceed from both ends towards the middle. Alternatively, partial evaluation may be made in one direction, and then full evaluation in the other direction. Schemes that in this way proceed from a final state (or set of states) to an initial state and then back to a final state are usually called *backward-unwinding* schemes.

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_2: $A \rightarrow aA$
p_3: $B \rightarrow bB$
p_4: $C \rightarrow cC$
p_5: $A \rightarrow a$

p_6: $B \rightarrow b$
p_7: $C \rightarrow c$

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 recogniser for these strings.

Now it is well known that the solutions to many problems of practical significance are best represented by sequences of actions rather than by sets of actions in which the order of application Is unimportant. The question arises as to how this can be achieved in a production system. One way to achieve such sequencing is for each production in the sequence to throw some special symbol into the data base that only invokes the next production in the sequence [e.g. Moran 1973]. However, this scheme has a number of significant drawbacks. Firstly, in order that the correct production is invoked, 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 eqivalently, two classes of productions. Nothing Is wrong with this, of course, except that we have changed the nature of the production system. Secondly, the philosophy of modular programming upon which production systems largely rest requires that the details of other independent modules be of no consequence to one another. However, 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 condtion 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. Most other schemes for Introducing control [e.g. Rychener 1977] suffer from similar failings,

We thus adopt an alternative approach whereby we specify control information explicitly and outside the object-level productions, 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. The important point to note is that control is not achieved *indirect/y* through the conflict resolution scheme (as, for example, in [Rychener 1977]), but is specified *independently* of it. 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

$$\{a^nb^nc^n : n \geq 1\}$$

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 [see Salomaa 1973).

On the other hand, it is important to stress that anything of which a controlled production system is capable so is some production system. One simply places the control information in the data base (see, for example, ACTS [Anderson 1976]). However, as pointed out above, there are serious disadvantages to the latter approach. It remains for us to demonstrate that such disadvantages are not a necessary consequence of Introducing control.

In the context of most of the literature on control In production systems, it is very important to note that here we are distinguishing between two types of control --- the one that is essential to the determination of the Intended solutions and the other which relates to system efficiency and the resolution of nondeterminism. The control language models the former of these --- that is, it acts in exactly the same way as control acts in the procedural languages (or more formally as in program schemes [Engelfriedt 7 974]), and has nothing to do with the selection mechanism involved in conflict resolution.

One should also note that a controlled production system is a very general abstract machine and encompasses all systems that are characterized by a sequence of transformations from one state to another. In particular, we are not intending any restrictions on the structure or size of the data base or on the complexity of the productions [cf. Lenat and McDermott 1977].

The following section may be skipped by those who have managed to unambiguously interpret the above description of controlled production systems. In any case, it should probably be skipped at first reading as it may induce certain unintended biases.

2.2. Formal Definitions.

A *production system* (PS) is a triple

$$\mathcal{P} = < \Sigma, D, h > \qquad (2.1.)$$

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 $< p, r >$, where p is a total predicate on D and r is a relation on D. For any production name a, we will say that a *denotes* $h(a)$. We will also say that the first co-ordinate of $h(a)$ is the *condition* denoted by a and that the second co-ordinate of $h(a)$ is th*e action* denoted by a. Elements of D will usually be referred to as *states* of the data base.

If for some a $\in \Sigma$ and x $\in D$ we have some p,r such that *h(a)* $=< p, r >$ and $p(x) = true$, then we will assume that there exists $y \in D$ such that $< x, y > \in r$. This is simply requiring that If the condition part of a production Is satisfied for some state of the data base, then this state of the data base will be in the domain of the action part of the production.

Although formally it is sufficient to consider only production names and their interpretation, in any concrete PS, each production name will correspond to a string of symbols called a *production.* Thus where no ambiguity can arise, we will often refer to a production name simply as a production.

Before we define the notion of execution for a PS, we will formally define 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

$$\mathcal{C} = < \Sigma, D, h, C > \qquad (2.2.)$$

where C is a control language over Σ and $< \Sigma, D, h >$ is a production system .

A CPS $< \Sigma, D, h, C >$ with $C = \Sigma^*$ is equivalent to the PS $< \Sigma, D, h >$. 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 $\mathfrak{C}$ be a CPS as defined in (2.2.). We first define a *state of execution* (or simply *state)* of $\mathfrak{C}$ to be a pair

$$s = < u, x > \qquad (2.3.)$$

where u is a prefix of some word in C and x is an element of D. Now let u and ua be prefixes for some word in C, where $u \in \Sigma^*$ and $a \in \Sigma$. Then we say a state $< ua, x_2 >$ is *directly computed from* a state $< u, x_1 >$, denoted $< u, x_1 > \Rightarrow_G < ua, x_2 >$, if and only if for some p, r, we have $p(x_1) = true$, $< x_1, x_2 > \in r$ and $h(a) = < p, r >$. If the above holds, we will also say that the state $< ua, x_2$ > results from *execution* of the production a in state $< u, x_1 >$.

Let $\Rightarrow_{\mathfrak{C}}{}^*$ denote the reflexive transitive closure of the relation $\Rightarrow_{\mathfrak{C}}$. Then the *reiation computed by* $\mathfrak{C}$ is defined to be the set

$$R(\mathfrak{C}) = \{< x, y > : < \lambda, x > \Rightarrow_{\mathfrak{C}}{}^* < w, y > \textit{ for some } w \textit{ in } C\}$$

Informally, a CPS $\mathfrak{C}$ 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 $< \lambda, x >$. Suppose that execution has arrived at some state $< u, y >$. Now a production a can be considered for evaluation if ua is a prefix of some word in C. Let us call the set of all such productions the active *production set.* Suppose there is no active production a. 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 a is (nondeterministically) selected from this set, and execution continued from a (not necessarily unique) state $< ua, z >$, where $< y, z >$ is an element of the action denoted by a. The final state of the data base is obtained on successful termination of execution.

The above execution scheme is a forward-chaining scheme, Other methods for determlning the relation computed by a CPS are not difficult to construct.

We could now go on to prove properties of this formal system. For example, it is not difficult to show that the power of deterministic CPSs is equal to the power of non-deterministic CPSs. However, such is not the intent of this paper and we will leave formal considerations here.

3. The Control Language

Having defined what we mean by control, the question arises as to how we are going to realize such control, and how different realizations will affect the behavlour and properties of the system as a whole,

3.1. Specification of Control languages.

Perhaps the most obvious way to specify the control language is syntactically, where the alphabet for such a specification is taken to be the set of production names.

For example, consider the problem of cascading two amplifiers [McDermott 1977]. We could represent this problem using productions of the form

$$\begin{array}{lll} t_0: & \textit{cascade} - \textit{amp} & \rightarrow & (\textit{collector}\ ?x)(\textit{emitter}\ ?y)(\textit{couple}\ ?x\ ?y) \\ t_1: & \textit{collector} & \rightarrow & \ldots \\ t_2: & \textit{emitter} -- & \rightarrow & \ldots \\ t_3: & \textit{couple} & \rightarrow & \ldots \end{array}$$

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. To achieve the same effect using a production system, it is necessary either to make tasks t_1 and t_2 preconditions for task t_3, or to include the control information in the data base (e.g. using SUCCESSOR indicators [McDermott 1977]). The former approach is simply inappropriate, and the latter can suffer from the disadvantages to which we have earlier referred. But the approach given above Is not much better --- it allows more general control constructs, but -many of the desirable features of production systems are lost. For example, if we add to the production set another method for making collectors then unless we explicitly augment the control language this new method will never be used for cascading amplifiers.

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 their is no need to be so restrictive. As the interpretation is a component of the formal model (see eqn 2.1), it can also be used in specifying the control language, that is, the control language can be specified by its *semantlc 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 $p_1p_2p_3$ *and* $p_2p_1p_3$ 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. Such a specification clearly retains the additivity of the system. In fact, there is not one desirable property of production systems that is lost when control is specified in this way. The system remains highly modular, is potentially extensible, and retains its explanatory capacity [cf. Davis 1975]. Of course, the dependence of these properties on the means of production reference is well known [e.g. Davis 1977a] --- the only difference is that here we are using it to effect control constraints rather than to improve system performance.

Because the semantic specification of control information (or *content-directed invocation* as it is usually called) makes possible any amount of information transfer between sender and receiver, it is potentially a very powerful control mechanism. it allows private channels of communication to be established between productions, in effect, through a mutual exchange of information among the productions themselves. Most importantly, this information is not constrained to passing through the object-level data base as it is in most other approaches [e.g. Rychener 1977, McDermott 1977].

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 implementing an interrupt where the action b is to be taken if the variable X obtains a value less than 10^{-10}. We could achieve this by introducing a production

$$int: \quad \mathbf{true} \rightarrow \quad if\, X < \; 1.0E - 10 \; then\; b\; else\; no - op$$

and inserting the production name 'int' between each pair of symbols in each control word. That is, we just simulate the usual hardware interrupt. However, it is more natural, and requires less knowledge of the implementation, to express the desired behaviour of the interrupt semantically

if the last production used assigns a *value to the variable X, then invoke int before proceeding further*

Not only have we been able to effect the interrupt by such a specification, but we have also achieved a more efficient implementation of the interrupt --- *at the object level* --- than through the syntactic approach [cf. Petterson 1976]. Of course, we could do the same syntactically, but only at considerable expense on the part of the programmer --- in any case, one would be reluctant to call the resulting syntactic equivalent an **interrupt.** Other examples of semantically based control information are not difficult to find (e.g. robot plans).

One problem with implementing such a scheme is that it 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 computationaily 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 [e.g. Davis 1977b].

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, such as for the above interrupt, 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.

3.2. Devices for Specifying the Control Language,

We have so far said nothing about 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 are inappropriate, however, if the control language is to be specified semantically, that is, on the basis of the content of the productions. We could program some scheme in some procedural language, but we then limit the additivity and fiexibiiity 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 [McDermott 1977] --- 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. it is also worth noting that much expert knowledge involving control will be piecemeal --- thus, as in the example of the interrupt, we will often have an accumulation of control constraints --- the original control language specification, modified by subsequent specifications, and these perhaps further modified. In a sense, the control language is approached by a sequence of approximations. It is just this kind of knowledge that is best manipulated by production systems,

The distinction between the control-level CPS and the object-level CPS is an important one. The advantages of maintaining a conceptual distinction are hopefully obvious. But advantages also exist at the implementation level. The active production set is usually much smaller than the entire production set and thus considerable gains in efficiency are achieved if the non-active productions are masked before the conditions on object-level productions are evaluated. As the active production set is determined by the control language, this requires that control-level productions be executed before object-level productions. In fact, most systems that include some form of control-level productions (e.g. NASL) indeed do just this. (in this respect the architecture of such a system is quite different to TEIRESIAS [Davis 1977a] where meta-level productions must be evaluated after the conditions on object level productions are evaluated. The difference arises because the meta-level productions of TEIRESIAS are used solely for conflict resolution which, quite clearly, must take place after the conflict set has been determined).

Formally, the data base of the control-level CPS should include sequences of productions together with their interpretations, so that the set of final states of the data base would constitute the control language. In such a scheme, any dependence of control on object-level conditions would have to be handled by the introduction of appropriate object-level productions. However, it is most likely that one would want to interleave execution of the control-level CPS and the object-level CPS, and this may be more easily and efficiently achieved if such restrictions on the elements of the control-level data base where relaxed.

The control language of the control level CPS will also need to be specified, and one could envisage a hierarchy of CPSs, each determining the control language of the one below it. We do not see much advantage in such a wealth of control language CPSs, although in Section 4 we will suggest the desirability for a further CPS of different purpose.

3.3, Types of Control Language.

In designing control devices of the sort mentioned In the previous section, it is important to know how the complexity of the control language affects the power and usefulness of the system.

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 greatly improve constructibiiity.

For example, consider the problem of adding two positive integers. This problem is considered representative of a class of problems unsuited to production system architecture (see [Davis 1975); the example is by [Waterman 1974]). However, as a controlled production system, the solution is very simple. We let the productions be as follows

$$\begin{array}{lll}
a: & true & \rightarrow read(n, m) \\
b: & true & \rightarrow count \leftarrow 0; nn \leftarrow n \\
c: & count \neq m & \rightarrow count \leftarrow succ(count); nn \leftarrow succ(nn) \\
d: & count = m & \rightarrow write(nn)
\end{array}$$

and let the control language be the regular expression

abc^*d

Note that for readability we have allowed composite actions on the RHS of productions. Whereas for a production system this is against the spirit of things", for a controlled production system composite actions are just syntactic sugar. Looked at another way, each production of a controlled production system is decomposable into (sequences of) more prirnitive productions. With standard production systems this cannot be done without explicitly placing control Information in the data base, thus changing the character of the entire problem space.

Regular control languages are also powerful enough to define partitioned production systems (also called *procedures* [Moran 1973], *packets* or *multiple production memories* [Lenat and McDermott 1977]). 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. If, for each state, the set of productions labelling the outgoing arcs is disjoint from the set of productions labeliing the outgoing arcs of the other states in the network, then the resulting CPS is equivalent to a partitioned PS. On the other hand, CPSs with regular control languages offer more flexibility than partitioned PSs because there is no need to keep these sets disjoint.

Most other types of sequencing used in production systems or knowledge based systems are also readily described by regular control languages. For example, the slot filling operations in frame-like representations (e.g.to-fill, when-filled) are simple successor and predecessor specifications.

Context-free control languages allow of more interesting constructions. For example, the control language $\{a^kb^k : k \geq 1\}$ allows constructs of the form

do S_1 until B; do S_2 the same number of times

without the explicit use of counters.

Context-free control languages may also be used to define recursive programs (In fact, the above construction may be realized by a call to a procedure that executes a on entry, then calls itself, then on exit executes b). Informally, 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 subroutine or 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.

Let us consider again the problem of constructing a cascade of amplifiers. The solution proposed in Section 3.1 was not realty 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

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

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 realty 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 of the object level system is better transferred to the control level).

Some existing systems use similar devices for controlling production invocation. For example, the SUCCESSOR and SUBTASK relations in NASL can be used to explicitly represent control information in the same way that the rewrite rules do in the above context-free control grammar. Most of the other control-level productions of NASL (e.g. the CONSTRAINT relation) are also realizable as context-free rewrite rules.

It is also interesting that specification of the control language by means of a phrase structure grammar is simply a special case of specification by a control level production system. That is, the classical means of language specification have directly resulted in a uniform formalism for specifying object-level and control-level information. On the other hand, 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 [Davis 1977a] 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. 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.

The similarity of context-free CPSs to augmented transition networks [Woods 1970] should not have escaped the reader. 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 over the productions attached to each arc of the network.

From this point of view it is interesting that in natural language applications the natural 'language input to the ATN ***acts purely as a control word*** --- that is, the natural language sentences simply constrain *the* ***generation*** *of semantic structures.* In one sense, the very thing that production systems throw away Is precisely that which provides the mapping from surface strings to semantic structures. In fact, the success of ATNs in many different domains demonstrates quite convincingly the significance of control information, even when this control is specified syntactically.

The control language of a CPS need not be restricted to being context-free, and other language types may prove useful, For example, macro languages [Fischer 1968] allow one to represent procedures with parameters. Interested masochists should refer to Engelfriet [1974].

3.4. Bridging the Gap.

It should be clear that the standard functional and procedural languages are formally a restricted class of controlled production system. In the case where the control language is regular, a controlled production system is simply a representation of a flowchart for a (nondeterministic) sequential algorithm. Context-free control languages give us recursive procedures (see Section 3.3), and macro languages procedures with parameters. (For a more formal treatment see [Engelfriedt1974]). This is a nice property of controlled production systems. We see at the one extreme, where the control language places no constraints on production invocation, we have the declarative languages, and at the other extreme, where the control language constraints give determinacy, we have the sequential procedural languages. (in fact, the similarity between controlled deduction and procedural programs is well known).

in previous sections we discussed the usefulness of semantically based invocation and separate specification of control for systems which in general were non-deterministic. What are the implications of these types of control when applied at the deterministic extreme of the spectrum?

In the first instance, it means that such systems, even though deterministic, will retain most of the desirable properties of nondeterministic CPSs, and in particular remain extensible. That is, the addition, deletion, or replacement of a functional unit need not require modification of other functional units to provide for the change. Thus, for example, a condition or test can be appended to the main stream algorithm --- without modification to the original text --- after an error in execution is observed. Similarly, unlike most programming languages, deletion of a functional unit may injure but does not necessarily kill a deterministic CPS.

Further, because no restrictions are placed on the distribution of control, *separate chunks of knowledge can be separately specified, independently of how they are used.* In contrast, the standard procedural languages require that control always lie with the sender, never with a (possible) recipient. Thus to test for special conditions one *has* to place some command in the main stream program text, and this very easily obscures the purpose of the algorithm --- consider a program that begins with a long sequence of conditionals, some of these embedded within one another, and ail of which are to test for rarely encountered conditions.

We thus see that the notion of a controlled production system, and consideration of the various means of realizing the control device, provides a uniform programming methodology. The techniques and schemes that provide powerful, constructible and flexible knowledge based systems are exactly those that provide the properties desired of deterministic programming languages [cf. Winograd 1979]. In the other direction, the formalization of control aiiows us to apply the methods of program verification and program synthesis that are used in the procedural language domain to knowledge based systems.

4. Non-determinism

At any stage during the execution of a non-deterministic CPS the conflict set may contain more than one production. Production invocation must therefore be handled using some form of backtracking scheme or parallel processing scheme. Such schemes are characterized by the existence of multiple environments representing the state of the computation either at previous choice points (backtracking schemes) or for the current (or suspended) processes (parallel processing schemes). For each such environment, we will say that an applicable production is *open* if it has not yet been invoked (tried) in that environment, Any conflict set containing an open production is also said to be *open;* otherwise it is closed. Usually, there is no need to retain an environment with a closed conflict set, unless one wants to check for recurrence of states of the data base.

Non-determinism of course presents no problem if we have an infinite amount of time with which to play around. As this is rarely the case, we need to consider means by which the amount of computation required to find a solution can be reduced. There are essentially two means of achieving this, one being to to avoid multiple evaluation of equivalent production sequences and the other being to order the execution of productions so that one has the highest probability of successful termination with the least amount of effort. In the next section we consider the first of these approaches.

4.1. Equivalent Production Sequences.

in order to throw more light on the nature of non-determinism in CPS's, we will approach this problem from a somewhat unusual, and at first appearance rather clumsy, perspective.

For a CPS $\mathfrak{C} = <\Sigma, D, h, C>$, we will call the control word corresponding to a sequence of productions that computes a state y in D from a state x in D an associate *word* for $<x,y>$. Thus an associate word, and similarly an associate subword, determines a relation on D. The associate language *of* $\mathfrak{C}$ is then defined to be the set of words $w \in C$ such that w is an associate word for some element of the relation computed by $\mathfrak{C}$. Intuitively, an associate word for $<x,y>$ is simply a successful execution trace for $<x,y>$. The recognition problem for a pair $<x,y>$ in $D \times D$ is thus to determine whether there exists an associate word for $<x,y>$. This can be established by evaluating the relation determined by each word in the associate language, either sequentially or In parallel, until one is found that contains $<x,y>$.

Now an important property of associate words is that they provide a reasonably direct measure of computational effort, so that we may be able to achieve significant gains in efficiency if we **can** either reduce the number of associate words that require evaluation or somehow avoid total evaluation of each associate word. The first of these can be achieved if we **can** establish equivalences between associate words, where equivalence between words is to mean that they determine the same relation on the data base.

For example, in many problems each state of the data base consists of **a** set of elements which we will call *state elements.* These could be symbols, variables and their bindings, representations of subproblems, etc. Let us call two sequences of productions *disjoint* if the set of state elements modified by the one sequence is disjoint from the set of state elements accessed (observed or modified) by the other. Clearly, the relation computed by two disjoint production sequences does not depend on the order in which the sequences **are** executed. On this basis we can define equivalence classes of associate words, and in the recognition problem thus need only evaluate one representative member from each class. The derivations of a context-free grammar, for example, form equivalence classes for which we **can** take leftmost derivations as the normal form. Intuitively, where we can view a sequence of productions as representing the solution to a particular problem, then disjointness of production sequences corresponds to problem independence (or what is often called linearity"of the problem space). In such cases the above equivalences can be used to achieve considerable savings in computational effort, as is done, for example, in the standard problem reduction methods involving AND/OR tree search [Nilsson1971].

Similar equivalences on associate words and similar savings in computational effort can be made in other problem domains, as, for example, when productions or production sequences commute, (in fact the type of problem described above is simply a special case of commutivity which allows the Church-Rosser property to be satisfied). The essential point is that by establishing such equivalences we are saved the necessity of evaluating every associate word. Unfortunately, in many interesting problem domains it is difficult to determine these equivalences on **associate** words, and we often have no choice but to exhaustively try all alternatives.

In many cases it is also possible to avoid total evaluation of all associate words, in the sense that each production need not separately be executed for each associate word in which it appears. As each associate subword uniquely determines a relation on the data base, multiple evaluation of identical (matching) subwords is often unnecessary. For example, if two associate words have matching prefixes, then the relation determined by this prefix need only be evaluated once, provided that other constraints (such as storage requirements) are satisfied. The resulting reduction in computational effort can be very large, especially in problem domains which are characterized by disjoint production sequences. As mentioned **above,** problems which can **be represented** by **AND/OR trees** fall into this class, and the classical AND/OR search methods exploit subword matching to some degree. However, the standard methods still re-evaluate matching subwords if they occur on different OR branches, and this can be grossly inefficient. In the area of language processing, a number of schemes exist which make better use of matching subwords [Eariey 1967, Kaplan 1973], but such do not seem to have been used in other problem domains.

4.2. Ordering the Alternatives,

The second means of improving the efficiency of production systems is by ordering the open productions or associate words so that the more promising ones are tried first. This problem has received considerable attention, and a large number of schemes have been proposed [see Davis 1975). In most of these schemes the ordering of productions is specified for the current conflict set only and cannot be subsequently updated or altered. However, greater control over the efficiency of the system can be obtained by allowing the open productions in earlier conflict sets to be (dynamically) re-ordered.

The question then arises as to whether one can do better than an optimum dynamic ordering and re-ordering of the productions occurring in the current and predecessor conflict sets. In a standard backtrack scheme we usually backtrack to the most recent choice point i.e. to the closest open conflict set. Clearly, it may be that none of the open productions in this conflict set are very promising, and we may wish to return to an earlier choice point. We could effect such a return by closing all subsequent conflict sets. However, a more flexible scheme would expand the highest priority production, irrespective of which conflict set it appeared in. Thus we could backtrack to the conflict set containing the highest priority production while retaining ail the conflict sets so far generated, Such best-first schemes are quite well known in AND/OR tree search [see Niisson 1971] but have rarely been Incorporated in production systems.

We can adopt a different perspective and view the choice as being between associate words rather than productions. This choice may simply be based on such properties as length of associate word, or may at the other extreme be dynamically determined on the basis of success indicators in the data base, For a pure production system, however, we are faced with two problems, one being that the associate language is usually difficult to ascertain and the other being that it is difficult to ascribe any intuitive meaning to the associate words. One consequence Is that providing heuristic guidance on the choice of associate words is extremely difficult. The situation is different for a controlled production system. The associate language is a subset of the control language, so that any ordering of control words provides an ordering for associate words. Secondly, to the extent that the control language has an intuitive basis, so, to a greater or lesser degree, would this ordering reflect intuitively meaningful knowledge of the problem domain.

Perhaps at this stage we should try and provide some of that intuitive support. A control language can be viewed as a class of strategies or plans for using our knowledge about the problem domain. Now we could attempt to obtain a solution to a given problem by trying first one strategy, and then others, until one proved to be successful --- that is, by trying one control word, then another, etc. If we have no knowledge of the likely outcomes of the various competing strategies, the order in which we attempt the strategies will depend on such properties as strategy depth (control word length), strategy similarities (subword matching), etc., and perhaps various implementation characteristics. The usual backtrack and parallel processing schemes are just two of the numerous possibilities. On the other hand, if we have knowledge about the problem domain that enables us to order the strategies on the basis of likelihood of success, then this information can be used to dynamically select the most promising strategy.

What we are suggesting, then, is that if a problem is well modelled by a controlled production system, then we should be looking at choosing between strategies or lines of reasoning" (as, for example, in NASL) rather than between individual productions (as in TEIRESIAS). Implementation-wise the two approaches amount to the same thing, but the difference in conceptual viewpoints can lead to quite different problem solving methodologies. In particular, we can free ourselves of the backtracking mentality --- backtracking to the most recent choice point is seen as a general but very weak meta-strategy. It takes no account of information that may have been derived since that choice point, and does not allow for dynamically changing lines of reasoning. More powerful schemes are suggested by the alternative paradigm. For example, the presence of two alternative strategies at a particular stage of execution may suggest a third and completely different strategy [see McDermott 1977], or the lack of success of a certain strategy may suggest a corrective strategy with which we can continue [again, see McDermott 1977].

4.3. Meta-level Knowledge.

Any system that treats a controlled production system as an object to be manipulated or reasoned about is called a *meta-level* system relative to that CPS, and the knowledge invested in that system is called *meta-level* knowledge.

Knowledge of the equivalence classes of production sequences is of such type. On the other hand, knowledge about the ordering of open productions or associate words, and similarly about the direction of evaluation (forward-chaining, backward-chaining, etc.), constitutes a somewhat different type of meta-level knowledge. Unlike the first type of meta-level knowledge, the latter is relative not to the abstract CPS but rather to the implementation of the CPS on a sequential machine. It is an important distinction to make, but having made it, it is sufficient for our purposes to treat the two types of meta-knowledge as the same.

Now in all the above cases, the meta-level knowledge functions solely as a means for improving efficiency. Given enough time, the same solutions would be achieved by a system without meta-level knowledge as with meta-level knowledge. *In contrast to the control-level component of the system, solutions* are *independent of the meta-level component,* although the order in which they are produced is so dependent.

The difference between control-level knowledge and meta-level knowledge is critical, both on intuitive and formal grounds. Intuitively, control knowledge specifies a sequence of actions to take, perhaps dependent on conditions maintaining in the object-level world. For example, it may be that we first move our hand to a block and then grasp it [Rychener 1977], or that we construct a collector and emitter before coupling them [McDermott 1977]. On the other hand, meta-level knowledge is about the utility of such plans and which are the most appropriate in a given situtation. If lost in the jungle, you may decide to opt for a plan that takes you back to your starting point or, alternatively, for one that tries to correct for your error. You reason about plans or strategies, deciding which is the best one in the circumstances.

On a formal level the distinction is even more important. The relation computed by a controlled production system is completely defined by the production set and the control language. Meta-level knowledge cannot alter that relation.

Unfortunately, the distinction between control-level knowledge and meta-level knowledge is simply never made. For example, in one of the nicer studies on invocation in production systems [Davis 1977a], meta-level knowledge is viewed as “information about which chunk of knowledge to invoke next when more than one chunk may be applicable” --- a definition which is equally applicable to control-level knowledge. The problem arises from the nature of production system architecture. When a conflict set contains more than one production, some conflict resolution scheme needs to be employed to select one of these productions for evaluation. Now if the conflict resolution scheme allows for all (non-equivalent) production sequences to be tried (under some backtracking or parallel processing scheme), then only system performance is affected, On the other hand, if the conflict resolution scheme is such that some alternative (and non-equivalent) production sequences are never tried, then it acts as a constraint on production invocation that affects solution in the same way as does a control language. Because there are no other means available, conflict resolution schemes are usually used in both ways --- partly to gain efficiency and partly to effect control [e.g. Rychener 1977]. What a solution then represents is anybody’s guess.

There is another way in which one may wish to use meta-level knowledge --- that is, as a means for determining, rather than just guiding, solution. Some of the corrective strategies of McDermott [1977] may be seen in this way, as also can some of the schemes used in PLANNER [Hewitt 1972](e.g THNOT construct) and other production systems (e.g. resource limited reasoning [Winograd 1978]). Hbwever, the “solutions,, obtained by using such meta-levelknowledge would not be solutions of the object-level CPS, and from a formal viewpoint this is highly undesirable, A particular case in point is PROLOG [Warren 1976], where so-called control information is primarily meant to enhance efficiency, and in this sense functions as a rather limited form of meta-level knowledge. However, certain useages of control information also throw away proofs that would otherwise be obtainable (e.g. the slash symbol), and thus act as constraints on solution in the manner of a control language. This confusion of control-level and meta-level knowledge can lead to serious error. Further, some control information (e.g. the slash symbol) is used to obtain solutions not otherwise obtainable (in a sense like THNOT of PLANNER), and this aggravates the problem. Similar difficulties are found in other systems *(e.g. if you’ve been trying to determine investment timescale for more than 5 cycles, give and try something else* [Davis 1977a]).

Of course one way around this difficulty is to change the controlled production system so that the knowledge contained In those meta-level rules that constrain solution is embedded either in new object-level productions or in a new control language. A less drastic course is to define the object that the "solutions" are solutions to. Perhaps one of the nicest ways of doing this is to represent the meta-level knowledge in a meta-level CPS, so that the “solutions,, are solutions of the meta-level CPS (in effect, this allows the interpreter to be dynamically modified [Lenat & McDermott 1977], see also [Weyhrauch 1979)). As soon as we do this, we allow that control-level knowledge be represented at the meta level rather than at the object level. The question then arises as to why we bothered to objectify the control component at all, This can be answered in two ways, If one likes, the control component can be treated as meta-level knowledge of a certain kind which does not depend on previous states of the data base or on previous attempts at solution. Its importance, and the reason we distinguish it from other types of meta-level knowledge, is that it is particularly common and natural in problem solving, and Is lntultlvely and practically distinct from other forms of knowledge. But a more fundamental reason for having a formal model that includes a control component is that the underlying interpreter --- which operates at the topmost level --- can then order production execution. If, in contrast, this interpreter is modelled by a (pure) production system without control, then there is no means of ordering execution --- one can *mention* orderings on productions, but the order in which they are actually evaluated will depend on the order in which the interpreter chooses to evaluate the top-level productions. Consequently, a necessary component of such schemes is the appendum "We have chosen to evaluate expressions in such and such an order,,.

4.4. Representation of Meta-level Knowledge.

The objects of the meta-level system are the-productions, control words and the data base of the object-level controlled production system, and can be specified either syntactically or semantically, Most of the schemes used in present production systems limit meta-level knowledge to ordering productions in the current conflict set, and most specify this ordering syntactically (i.e. in terms of the names of the productions). As one would expect, the additivity of the system quickly deteriorates. Additivity can be preserved, however, if the ordering on productions appearing in each conflict set is specified, not syntactically, but semantically i.e. in terms of the interpretation of the productions. Furthermore, semantic specification is richer than syntactic specification, and more complex forms of meta-level knowledge are more easily represented. Such schemes have been successfully used in both the MYCIN system [Shortliffe 1975] and the TEIRESIAS system [Davis 1977a]. Both these systems specify the ordering of productions in terms of their properties rather than their names. Systems like HEARSAY II [Ermann & Lesser 1975] and STRIPS [Fikes & Nilsson 1971] also use semantically based ordering schemes, but the amount of accessible semantic information is limited.

Control words appear as objects of meta-level knowledge in NASL [McDermott 1977], where CHOICE procedures are based on task and plan analysis. Recent work with ATNs has also seen the introduction of such meta-level knowledge [e.g. Finen & Hadden 1977], although this is represented syntactically rather than, as in NASL, in terms of the interpretation or function of the control words.

As in the case of specifying control-level knowledge, there are considerable advantages in using a controlled production system to represent meta-level knowledge [see also Davis 1977a]. Such an approach has the practical advantage that the same inter- - preter can be used for both the object-level CPS and the meta-level CPS. More important, however, is that it allows of a uniform formalism. In particular, this means that we can recursively provide higher and higher levels of meta-knowledge. Secondly, it allows an adequate formalism of the type of meta-knowledge that is used in determining, rather than just' guiding, solution. That is, the solutions can be seen as solutions of the meta-level CPS, rather than as some intuitive object somehow arising from the execution of the object-level CPS.

4.5. Subgoal Dependencies and Subgoal Protection,

As we mentioned in Section 4.1, considerable gains In efficiency can be achieved if the production system is disjoint (or decomposable). Not only can we readily establish equivalences on production sequences but we can treat each subtask independently of other subtasks and thus further reduce the amount of computation required. However, many interesting problems do not satisfy this property. One approach to such problems is simply to assume disjointness, and then to interleave or patch the solution where violations of subgoal preconditions occur [e.g. Sacerdoti 1975, Dawson & Siklossy 1977, Rieger & London 1977]. However, all these schemes are essentially ad hoc, and it is difficult either to establish the validity of the solutions obtained or to ascertain which kind of problems are suited to such an approach.

These difficulties arise primarily because there is no adequate notion of control in production systems. When we do have a model of control it becomes apparent that what we are really trying to do is to transform the given production system into an equivalent disjoint production system with control constraints, It is not too difficult to provide a formalism defining such equivalences, and it is then relatively straightforward to construct schemes for realizing such transformations in a uniform and consistent way.

We can indicate what we have in mind using the classical 3-block problem of Sussman [1973]. Consider a simple blocks world environment consisting of three blocks, A, B and C, and a table. In the initial state, blocks A and B are on the table and block C is on block A. The goal is to achieve a new configuration of blocks where block A is on block B which in turn is on block C. The only action that can be applied to the blocks is PUTON(x,y), which places block x on y. PUTON(x,y) is not applicable unless x has a clear top and either y is the table or y is a block with a clear top. The problem is to develop a sequence of actions that will achieve the goal state.

Let us represent the goal state by the statement AND(ON(A,B),ON(B,C)). Then using a recursive backward-unwinding scheme the subgoals ON(A,B) and ON(B,C) would be set up. Now suppose the system tries to achieve ON(A,B) first. This can be done by PUTON(A,B), but requires that the subgoals CLEARTOP(A) and CLEARTOP(B) be set up. The second of these is achieved immediately, and the first by doing PUTON(C,TABLE). Having thus achieved ON(A,B), the system will attempt to achieve ON(B,C). But in order to achieve this goal, B will have to be cleared, thus undoing the subgoal it achieved first.

On the other hand, if the system tries to achieve ON(B,C) first, it will end up even further away from the goal state than it was initially.

Now consider what happens if we use a CPS to represent the blocks world. The problem is, of course, to find an (optimal) control language sentence having the specified initial and goal states. So it would appear that we really have not got very far by our reformulation. However, we do have some information about the control language, which is to the effect that **the** post-conditions resulting from the application of an operator should not violate the pre-conditions required of a subsequent operator. Now this is **no** more than a control constraint, and can be semantically specified as follows

if the action part of an (instantiation of **a)** *production contains PUTON(x,y) where y $\neq$ TABLE, then it cannot be immediately followed by an (instantiation of* **a)** *production whose condition part contains CLEARTOP(y)*

Further, we know that In an optimum solution we will never place one block on top of another block that has to be subsequently cleared, This means that we can drop the "immediately" from the above constraint on the control language. Given, then, this partial specification of the control language, let us run through the actions of a backward-unwinding recognizer.

As before, the initial goal will cause the subgoals ON(A,B) and ON(B,C) to be set up. Now in order to satisfy the above condition on the control language, the production that **achieves** ON(A,B) must follow the production that achieves ON(B,C). This means that prior to achieving ON(A,B) we must achieve not only CLEARTOP(A) and CLEARTOP(B), but also ON(B,C). Of these subgoals, the condition on the control language, plus matching with the initial state, requires that ON(B,C) be achieved after CLEARTOP(A). The new set of subgoals (to be achieved before ON(B,C)) is then CLEARTOP(A), CLEARTOP(B) and CLEARTOP(C). The latter two are satisfied Immediately, and CLEARTOP(A) is directly achieved by PUTON(C,TABLE).(PUTON(C,B) **cannot** be used **as** it would violate the conditions on the control language).

The conditions placed on the control language have thus resulted in achieving **an** optimum solution without backtracking (i.e. deterministically). While determinism is not guaranteed in all such blocks world problems, the degree of non-determinacy is considerably reduced. On the other hand, a PS in which all production sequences are tried will be non-deterministic, and one which assumes independence of subproblems, and thus uses the equivalences on production sequences to reduce the degree of non-determinacy, will fail to achieve **a** solution. This is because the subproblems are not independent and the problem reduction operators do not have the Church-Rosser property --- that is, the solution is not independent of the order of their evaluation, In particular, the "depth-first" orderings on production invocation that are realized in a recursive scheme do not represent all possible orderings and in this case fail to contain the solution.

Most other approaches also have difficulty with the above problem, essentially on the same grounds. STRIPS [Fikes and Nilsson1971], ABSTRIPS [Sacerdoti 1974] and HACKER [Sussman1973] only manage non-optimal solutions, whereas most methods that do find optimal solutions (e.g. Manna & Waldinger [1974], INTERPLAN [Tate,1974]) use extensive backtracking. A number of more powerful systems have been more successful. Sacerdoti [1975] describes a system that finds an optimal solution deterministically, using a collection of "critics" to perform the same role as the control language condition does in the above CPS. Dawson & Siklossy [1977) obtain a solution using a preprocessing scheme, and Rieger & London [1977] use "guardian clusters" to protect against subgoal annihilation, Again, both of these schemes can be viewed as somewhat restricted means for achieving control language constraints.

The common characteristic of most of these approaches is that subgoal dependence is viewed as a problem rather than an asset. However, as should be clear from the above, such dependencies can be used to as much advantage as the independencies in other problem domains, To paraphrase Rieger [1977], a controlled production system gives us the best of both worlds: modularity in problem solving knowledge, yet harmony in the synthesis of large plans.

5. Some Issues

In this section we will touch on some of the issues raised in earlier sections.

5.1. Syntax and Semantics,

There are two occasions on which we need to refer to productions: one when specifying the control language, and the other when representing meta-level knowledge. Now each production has a name and an interpretation (Defn. 2.1.), and either may be used to reference that production. We will call reference to a production or production sequence a *syntactic* specification if it is by name alone, and a *semantic* specification if it depends on the interpretation of the production(s).

Now the syntactic specification of productions or production sequences is well understood. The problem that concerns us is how to specify the semantic content of productions, or in more general terms, of procedures, modules or knowledge sources. Of course, the semantic content of a production is simply its interpretation, but this is not usually explicitly represented. Now as a production is, in general, going to be represented by a string of symbols, one means of obtaining semantic information is by analysis of this symbol string. . That is, if we wish to know whether the action of a production P assigns the variable X the value 2, we check the RHS code of P to see whether it contains, say, '$X \leftarrow$ 2'. More rigidly specified, this information may be represented as

$$substring(RHS(P), 'X \leftarrow 2') \ .$$

Similar devices are used in TEIRESIAS (e.g. the MENTIONS predicate).

One problem with this approach is that it is in direct opposition to the currently prevalent view that only the effect of a production or module should be visible to the external world. -Consequently, should the RHS of the production be recoded to look something like

$$Z \leftarrow 2; \ X \leftarrow Z$$

the above rule for specifying the semantic content of the production would fail.

A better approach may be to provide some language for describing the semantic domain together with an Inference device for determining the semantics associated with each production. For example,the above semantic information could be expressed in a predicate calculus like form:

$$action(P, assign(X, 2)))$$

and **the** inference device have rules of the form:

$$\forall pzxy\ .\ RHS(p,z) \wedge contains(z, x \leftarrow y) \supset action(p, assign(x,y))$$

Of course, the inference rules would need augmenting if recoding as above was allowed, but to the external world the workings of the production would remain hidden. Perhaps the most sophisticated approach along these lines is the use of rule models in TEIRESIAS [Davis 1977b].

One problem with this approach is that the inference rules become exceedingly complex when the productions are allowed to take more complex forms. This may be an argument in favour of keeping productions relatively simple [cf. Lenat & McDermott 1977]. In fact, much of the power of a controlled production system may well lie in the extent to which the semantics of productions are machine recoverable from their syntactic form.

Another alternative is to include with each production a specification (in the language of the semantic domain) of its effect or intent. In the simplest case, this may be a list of external descriptors or a list of pre-conditions and post-conditions (e.g. the add and delete lists of STRIPS [Fikes & Nilsson 1971]). More complex descriptions would require some inference device, but the inference rules would be independent of the actual coding of the productions (for arguments against this approach, see [Davis 1977a]). Such a system would also enhance the explanation facilities of the system (cf. the "rationale " commentary in annotated production systems (Goldstein & Grimson 1977]).

Of course, the difficult task is to define the language of the semantic domain and to construct the inference device. Again, there is considerable advantage to using a controlled production system to represent the inference device (see Section 3.2). In the above example, the above implication simply becomes a production (see also [Davis 1977a]). Such an inference device could then be directly Included In the control level CPS, although if meta-level information is also semantically based it may be desirable to keep the inference system separate.

5.2. Distribution of Control.

Formally, the control information in a controlled production system is not available to the object level productions --- in a sense, it takes place above them. Thus object level productions cannot of themselves send control to other productions (on either syntactic or semantic grounds), nor can they take control from other productions (on either syntactic or semantic grounds), However, in an implementation of a controlled production system it may be desirable to specify some control level information along with each production. We will call such units of object-level and control-level knowledge *modules* **or** *knowledge sources.*

In order to retain the flexibility and extensibility of the system, there should be few restrictions on the forms of control allowed to modules. Thus a module may specify a successor module or set of modules and thus "send" control, or It may specify a predecessor module or set of modules and thus "take" control, In fact, there is no reason why a module could not specify an nth successor or predecessor, although the usefulness of such schemes would appear limited.

Similarly, both syntactic and semantic forms of control should be allowed. Semantic specification allows modules to send tasks to other modules that satisfy given properties, or to restrict the modules from which it will take or receive tasks, Thus invocation need not be solely dependent on the state of the data base returned by the preceding module --- as it is in the usual production system --- but can be specified in terms of any of the properties of either the sending module or of the receiving modules. Further, specifications can go both ways, so that a given module m will only be invoked if m is contained in the successor set of the current module and the current module is contained in the predecessor set of m. If these specifications are semantically based, then there is the potential for any degree of information transfer between the caller and the respondent. Contract nets [Davis & Smith 1978) provide one means of implementing such a scheme.

6.3. Additivity and Learning,

As for production systems, controlled production systems are potentially extensible and able to accept new information additively without major re-organization. The reasons for this are twofold. Firstly, and most importantly, invocation criteria can be based on the semantic content of productions rather than on name, This, together with the non-deterministic nature of invocation, renders the system quite resilient to changes in the production set. Secondly, invocation does not have to reside with the sender, so that there is no need to modify existing productions to provide for the invocation of new productions.

The relative ease of augmenting and modifying the productions in a controlled production rystem augers well for their ability to aquire new knowledge, either through instruction or experience. For a controlled production system there are three areas in which such learning can take place, viz.

(i) object level knowledge

(ii) control level knowledge

(iii) meta leve! knowledge

Knowledge acquistion in the latter two areas is of particular importance as, apart from the possible effect on the solutions obtained, It is here that significant improvements In efficiency can be achieved. Interestingly, the more we know about the control language the closer we get to a deterministic algorithm, and consequently the less we need know at the meta-level.

There is a considerable body of literature on learning in production systems [e.g. Davis 1976, SIGART 1977], although in all cases this is limited solely to the modification of object level knowledge. Although we will not address the problem here, it is not difficult to see how many of these schemes could be extended to handle the other levels as well.

Another area of learning concerns production reference. If the system is reasonably stable, it is clear that we can achieve considerable gains in efficiency by transforming semantically specified invocation information into syntactically specified information during execution. Thus as the system discovers that certain productions or modules satisfy the required semantic properties, it could replace the semantic specification with a syntactic specification, perhaps retaining the former in the background in case these productions are modified or deleted. From a cognitive point of view, we could consider this to correspond to a transfer of invocation criteria from the conceptual (or formal operations) domain to a subconscious (stimulus-response) domain. Memo functions are somewhat analagous. The advantage of such a mechanism is that it achieves the best of both the semantic and syntactic worlds: the power, flexibility and robustness of the semantic specification is retained while at least approaching the efficiency of the syntactic specification.

4. Conclusions

A formal model for representing control In production systems has been defined, providing a uniform framework in which control issues can be addressed. Most importantly, the formalism allows control to be directly specified independently of the conflict resolution scheme, and thus allows the issues of control and nondeterminism to be treated separately.

Most control constructs were shown to be easily specified within this formalism --- in particular, multiple production memories [Lenat & McDermott 1 977], subtask and successor relations [e.g. McDermott 1977], slot filling advice, augmented transition networks [Woods 1970], and the standard control constructs of procedural languages.

It was also shown that the formalism provides a basis for implementing control constructs which retain all the properties desired of a knowledge based system --- modularity, flexibility, extensibility and explanatory capacity. The introduction of a separate control component was shown to provide additional channels of communication between knowledge sources, and to remove most of the difficulties associated with the single, public channel available to production systems without control. Most importantly, it was seen that all of the properties desired of a knowledge based system are not a function of the lack of control constraints, but of the type of information allowed to establish these constraints.

Within the formalism it was also possible to provide a meaningful notion of the power of control constructs. This enabled the types of control required in production systems to be examined and the capacity of various schemes to meet these requirements to be determined.

Schemes for improving system efficiency and resolving nondeterminism were examined, and devices for representing such meta-level knowledge were described. In particular, the objectification of control information was shown to provide a better paradigm for problem solving and for taiking about problem solving --- we have plans or strategies for solving problems, and we have meta-plans or meta-strategies for reasoning about them. It was also shown how the notion of control provides a basis for a theory of transformation of production systems, and how this provides a uniform and consistent approach to problems involving subgoal protection.

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 [1977], it is precisely the restrictions on interactions in controlled production systems that makes them so useful.

Acknowledgements

I would like to acknowledge the value of my discussions with Jorge Phillips, Nils Niisson and Wolfgang Polak.

References

Anderson, J. (1976) *Language, Memory* and *Thought* L. Erlbaum Assoc., N.Y.

Davis, R. (1976) "An Overview of Production Systems", Report STAN-CS-76-624, Stanford AI Lab.

Davis, R. (1977a) "Generalized Procedure Calling and Content-Directed Invocation" SIGPLAN/SIGART Newsletter (August), pp. 45--54.

Davis, R.(1977b) "Interactive Transfer of Expertise: Acquisition of New Inference Rules", *Proc. IJCAI 5,* pp. 321--328.

Davis, R.& Buchanan, B.G. (1977) "Meta-Level Knowledge: *Overview* and Applications", *Proc. IJCAI 5,* pp. 920--927.

Davis, R. & Smith, R.G. (1978) " Distributed Problem Solving: the Contract Net Approach", TR-CS-688, Stanford Unl. Comp. Sci. Dept.

Dawson, C.& Siklossy, L. (1977) "The Role of Preprocessing in Problem Solving Methods", *Proc. IJCAI 5,* pp. 465--471.

Earley, J.C. (1967) "Generating a Recognizer for a BNF Grammar" Computing Center Paper, Carnegie-Mellon Uni.

Engelfriedt, J. (1974) *Simple Program Schemes and Forma/ Languages,* Lecture Notes in Comp. Sci. Series, ed. G. Goos & J. Hartmanis, No. 20, Springer Verlag, N.Y.

Ermann, L.D.& Lesser, V.R. (1975) "A Multi-level Organization for Problem Solving using many diverse cooperating Sources of Knowledge", *Proc. IJCAI 4,* pp. 483--490.

Feigenbaum, E.A., Buchanan, B.G.& Lederberg, J, (1971) "On Generality and Problem Solving --- a case study involving the DENDRAL program", STAN-AIM-1 31 Stanford Uni., **Ca.**

Fikes, R.E.& Nilsson, N.J. (1971) "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", *A/ Journal, 2,* pp. 189--208.

Finin, T. & **Hadden,** G. (1977) "Augmenting ATNs", *Proc. IJCAI 5,* pp. 193.

Fischer, M.J. (1968) "Grammars with Macro-like Productions", IEEE 9th Annual Symposium on Switching & Automata Theory, pp. 131--142.

Goldstein, I.P. & **Grimson,** E, (1977) "Annotated Production Systems: A Model for Skili Acquisition" *Proc. IJCAI 5,* pp. 311--317.

Hayes, P.J. (1977) "In Defense of Logic", *Proc. IJCAI 5,* pp. 559--565.

Hewitt, C. (1972) "Description and Theoretical Analysis (using Schemata) of PLANNER: a Language for Proving Theorems and Manipulating Models in a Robot", TR-268, MIT AI Lab.

Kaplan, R.M. (1973) "A General Syntactic Processor" in Natural *Language Processing,* Courant Comp. Sci. Series, ed. R. Rustin, Algorithmics Press, N.Y.

Lenat, D.B. (1077) "Automated Theory Formation in Mathematics", *Proc. IJCAI* **5, pp. 833--842.**

Lenat, D.B., & McDermott, J. (1077) "Less Than General Production System Architectures", *Proc. IJCAI 5,* pp. 28--932.

McDermott, D. (1077) "Flexibility and Efficiency in a Computer Program for Designing Circuits',, AI-TR-402, MIT AI Lab.

Manna, Z. & Waldinger, R. (1074) "Knowledge and Reasoning in Program Synthesis,', Tech. Note 08, AI Center, SRI, Menlo Park, **Ca.**

Moran, T.P. (1073) "The Symbolic Imagery Hypothesis: A Production System Model',, Comp. Sci. Dept., Carnegie-Mellon University, (December).

Nilsson, N.J. *(1071) Problem Solving Methods in Artificial Intelligence,* McGraw-Hill, N.Y.

Pettersen, O. (1075) "Procedural **Events as Software Interrupts,, Report** STAN-CS-75-501, Stanford Uni.

Post, E. (1943) " Formal Reductions of the General Combinatorial Problem',, *Am Jnl Math,* **65,** pp. 197--268. --

Rieger, C. & London, P. (1077) "Subgoal Protection and Unravelling during Plan Synthesis", *Proc. IJCAI 5,* pp. 487--493.

Rychener, M.D. (1977) "Control Requirements for the Design of Production System Architectures" SIGPLAN/SIGART Newsletter (August), **pp. 37--44.**

Sacerdoti, E.D. (1074) "Planning in **a Hierarchy of Abstraction Spaces,,,** *AI Journal,* **5,2,** pp. 115--1 35.

Sacerdoti, E.D. (1975) "The Nonlinear Nature of **Plans,,,** *Proc. IJCAI 4,*

Salomaa, A. (1073) *Forma/* Languages, Academic Press, N.Y.

Shortliffe, E.H., Davis, R., Buchanan, B.G., Axline, S.G., Green, C.C. & Cohen, B. N. (1975) "Computer-Based Consultations in Clinical Therapeutics --- explanation and rule acquisition capabilities of the MYCIN system,', *Computers and* Biomedical *Research.*

SIGART Newsletter No, *63* (1977) *Proceedings of the Workshop on Pattern Directed Inference Systems*

Sussman, G.J. (1073) "A Computational Model of Skill Acquisition", AI-TR-207, MIT AI Lab.

Tate, A, (1074) "INTERPLAN: A Plan Generation System which can deal with Interactions between Goals,,, Mem. MIP-R-109, Machine Int. Research Unit, Edinburgh Uni.

Warren, D.H.D. (1077) "Implementing Prolog --- Compiling Predicate Logic Programs" AI-Report, Edinburgh Uni.

Weyhrauch, R.W. (1070) "Prolegomena to a Theory of Mechanized Formal Reasoning", *to* **appear in** *AI Journal.*

Winograd, T. (1078) “Extended Inference Modes in Reasoning by Computer Systems,, to appear in the Proceedings of the Conference on Inductive Logic, Oxford Uni.

Winograd, T. (1070) “Beyond Programming Languages,, to appear.

Woods, A.T. (1070) “Transition Network Grammars for Natural Language Analysis,,, ***Comm. ACM,*** **13,10.**