International Institute for Applied Systems Analysis
IIASA www.iiasa.ac.at

Bureaucracies, Bureaucrats and Information Technology

Lee, R.M.

IIASA Working Paper

WP-83-027

February 1983

BUREAUCRACIES, BUREAUCRATS AND INFORMATION TECHNOLOGY

Ronald M. Lee

February 198
WP-83-27

INTERNATIONAL INSTITUTE FOR APPLIED SYSTEMS ANALYSIS
2361 Laxenburg, Austria

CONTENTS

BUREAUCRACIES, BUREAUCRATS AND INFORMATION TECHNOLOGY

Ronald M. Lee

A. INTRODUCTION

Bureaucracy. The term is laden with negative connotations. One thinks of large, rigidified organizations with baroque, ritualized procedures incapable of adapting to changing needs and conditions in the environment. In mentioning the term bureaucracy one usually also speaks of its means of perpetuation: the professional bureaucrat. These are usually cast as unimaginative, plodding individuals socialized into the rule system of the bureaucracy to the point where the rules themselves, and not the purposes behind the rules, become the reason and guides of their employ. In recent years, another force has appeared which threatens to spread the phenomenon of bureaucracy even further, namely, the implementation of these bureaucratic rules and procedures in the form of computer-based administrative systems.

The purpose of this paper is to review in somewhat more depth the nature and interaction of these three forces: the bureaucratic organization itself; the bureaucrats that populate such organizations; and the special impact of information technology on their operation.

B. BUREAUCRATIC ORGANIZATIONS

The term "bureaucracy," as both a popular and scientific term, has come to have a variety of often overlapping definitions. The definition used here is due to Weber (1956/1978). To Weber, the process of bureaucratization is a shift from organizational management based on the interests and personalities of specific individuals, to one based on explicit ***rules*** and ***procedures***. These rules and procedures are identified with ***roles*** in the organization rather than individual people. Bureaucratic organizations thus take on an impersonal, mechanical character. To Weber, this is a positive development leading to greater effectiveness and efficiency:

> Bureaucracy develops the more perfectly, the more it is "dehumanized," the more completely it succeeds in eliminating from official business love, hatred, and all purely personal, irrational, and emotional elements which escape calculation (Weber 1956/1978:975).

Bureaucracies are sometimes characterized as having a 'mechanistic,' form of administration based on fixed rules and procedures as opposed to 'organic' organizations which rely more on individual discretion (Burns and Stalker 1961).

Bureaucracies in this sense are becoming of increasing importance in both planned and free market economies though the roles are somewhat different.

In a planned economy, the rationalization of management is central to the ideology. However, to Marx, bureaucracy was a major evil to be abolished:

> Bureaucracy becomes an autonomous and oppressive force which is felt by the majority of the people as a mysterious and distant entity — as something which, although regulating their lives, is beyond their control and comprehension, a sort of divinity in the face of which one feels helpless and bewildered (quoted in Abrahamsson1977:38).

Here the term 'bureaucracy' is used in a slightly different sense from Weber, denoting government bureaucracies in particular. The relevance for Marx was that these are an important concentration of social power.

In market economies, bureaucracy seems to be regarded more as a concession to inadequacies in market mechanisms. Here we need to distinguish bureaucracy from hierarchy. Williamson (1973) discusses 'markets vs hierarchies' as a problem of economic organization. In certain cases resources are allocated via market mechanisms, in other cases they are allocated within an organizational hierarcy, which may be under either public or private control. Hierarchies become bureaucracies (in the sense used here) when their administration becomes rationalized, embodied in explicit rules. In the case of hierarchical organizations in the private sector, this rationalization process tends to evolve gradually, as the organization discovers regularity in its environment.

Governmental hierarchies, by contrast, are typically created by legislation and so become bureaucracies from the outset. Downs (1967:32,34) cites a number of factors for the creation of governmental hierarcies. One is the case of consumer goods with large 'external' costs or benefits. An external cost or benefit is one not reflected in the good's free market price—for instance, the smog created by automobile exhaust, or non-biodegradable detergents which pollute rivers. The point is that market mechanisms do not take these external costs into account in selecting an equilibrium consumption level. To compensate for these inadequacies, a bureaucracy is often created.

Another case where a free market mechanism does not operate well is with so-called 'collective goods.' These are goods with indivisible benefits; once the good exists, everyone benefits whether or not they have paid their share. An example is national defense. In a free market, each person is motivated to avoid paying his/her part; since everyone makes this assumption, the collective good is not acquired. Again, to avoid this pathology of the market system, control of such goods is given over to a bureaucracy.

A somewhat related situation arises in certain industries such as oil production or telephone services where economies of scale or patent controls create strong monopolistic tendencies. In order to protect the consumer from unfair pricing, two options have been employed, both bureaucratic. One, is to nationalize the entire industry into a governmental agency. Examples are PEMEX, Mexico's national oil company and the various PTT's in European countries. The other alternative, effectively only slightly different, is to create a governmental regulatory agency to control the monopoly's behavior, e.g., the FTC and FCC in the U.S.

The rationalization of organizations, in itself, would seem to be inherently positive and equitable. Indeed, this is the implicit goal behind most of management science and operational research.

However, there seems to be an undesirable side effect that accounts for much of the negative connotations we attach to the term bureaucracy, namely, that highly rationalized organizations apparently become inflexible and unresponsive to changes in the environment. Weber comments:

> Once fully established, bureaucracy is among those social structure which are hardest to destroy. Bureaucracy is *the* means of transforming social action into rationally organized action... the ruled, for their part, cannot dispense with or replace the bureaucratic apparatus once it exists, for it rests upon expert training, a functional specialization of work, and an attitude set on habitual virtuosity in the mastery of single yet methodically integrated functions...
>
> Such an apparatus makes "revolution," in the sense of forceful creation of entirely new formations of authority, more and more impossible—technically, because of its control over the modern means of communication (telegraph, etc.), and also because of its increasingly rationalized inner structure (Weber 1956/1978:987-989).

One aspect — at least in market economies — for the unresponsiveness of bureaucracies is that they typically have achieved a monopolistic or protected position where they are not forced to change by competitive pressures. Nonetheless, newly elected politicians and corporate presidents often recognize and attempt to relieve the problem, though typically with little success.

Jay Galbraith (1973, 1977) offers a useful framework for analyzing the problem. A currently popular theory of organizations is the information processing view, due principally to Simon (e.g., Simon 1955, March and Simon 1958). The key concern is how the organization copes with the *complexity* of its environment, given the bounded rationality (cognitive limitations) of its managers. Galbraith extends the information processing view of organizations, to a 'contingency theory' approach. He regards the complexity of the organizations task as only one dimension of its information processing difficulties.

Another dimension is added to the organizational design problem, what Galbraith calls *uncertainty*. This refers to the degree of unpredictability of the tasks performed in the organization:

> Uncertainty is defined as the difference between the amount of information required to perform the task and the amount of information already possessed by the organization (1973:5).

The importance of this relates to the organization's ability to plan or pre-program its activities:

> The greater the task uncertainty, the greater the amount of information that must be processed among decision makers during task execution in order to achieve a given level of performance (1973:4).

Galbraith classifies the nature of the organization's overall cognitive task (as well as any of its subtasks) on a two dimensional framework of complexity and uncertainty. This may be viewed as a matrix (Figure 1) characterizing the different types of cognitive tasks which organizations face. In situations of high complexity but low uncertainty, the organization is able to plan and routinize its activities. These are the conditions

when bureaucracy is most effective. In situations of low complexity and high uncertainty, by contrast, the organization is constantly being surprised by changes in the environment. Here, the most effective form of administration seems to be one that relies heavily on the discretion of its employees. Burns and Stalker (1961) use the terms 'mechanical' and 'organic' to describe these contrasting forms of administration.

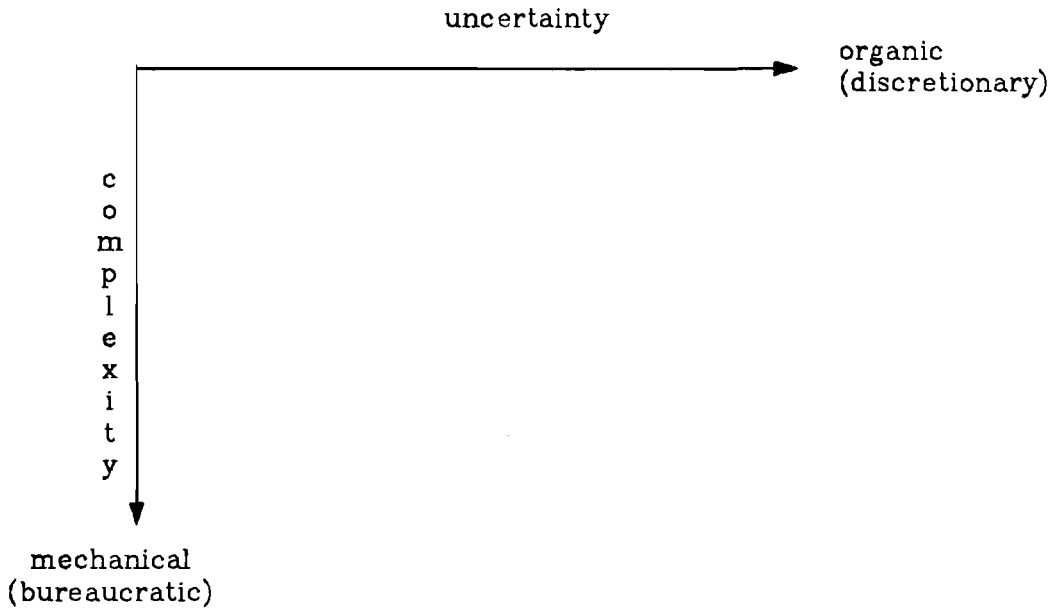

Figure 1.

The problem, of course, is what form of administration is appropriate when the environmental demands are both highly complex and highly uncertain.

As observed, rationalization is the typical response to complexity. An apparent difficulty with rationalization, however, is that when a once stable environment becomes more uncertain, the organization seems to have difficulties de-rationalizing, that is, removing rules and procedures and relying more on individual discretion in order to become more adaptive. One factor is likely to be that it has reached a level of internal complexity that cannot be maintained in a less rationalized type of organization.

The desired response would be to move quickly to another highly rationalized configuration. However the complex of bureaucratic procedures represents a large scale intellectual effort of many people over time. Bureaucracies are not built in a day. The time required to construct a new configuration may be too long compared to the rate of environmental change.

Implicit here is the observation that the rationalization of administration and organizational adaptability *seem* to be conflicting principles. In the next sections we examine possible reasons why.

C. THE BUREAUCRATIC PERSONALITY

Seldom are bureaucracies discussed without considering the role played by the people who staff them. Weber for instance remarks:

> the professional bureaucrat is chained to his activity in his entire economic and ideological existence. In the great majority of cases he is only a small cog in a ceaselessly moving mechanism which prescribes to him an essentially fixed routine of march (Weber 1956/1978:988).

A bureaucrat, unlike many other vocations, is heavily socialized and hence psychologically dependent on his/her active role in the organization. Bureaucracies such as have been described generally only arise in large organizations and then usually only after a fairly long period of adjustment and stabilization. Thus the activities of a bureaucrat are not only explicitly prescribed, but their full extent and interplay with other parts of the organization is also complex and difficult to learn. The bureaucrat therefore becomes an expert in his/her role in the *particular* organization. This is for instance quite different from professionals or trade workers whose specialities are generally transferable to other organizations.

A bureaucrat's training is thus peculiar to his/her organization. This makes it unsurprising that these people cling tenaciously to their positions, building defenses and guarding informational resources to make their positions more secure.

This, I think, is one of the primary reasons why bureaucracies are so persistent. Indeed, they survive even national revolutions. For instance, speaking about the post-revolutionary period in Russia, Lenin complained:

> [During the revolutionary upheavals, the bureaucrats from the Tsaristic time had been shaken up and placed in new posts. But they did not remain there. They tried to regain their old positions.] The Tsarist bureaucrats began to enter the Soviet institutions and practice their bureaucratic methods, they began to assume the coloring of communists and, for greater success in their careers, to procure membership cards of the Russian Communist Party. And so, having been thrown out of the door, they fly in through the window! (Lenin, *Selected Works*, Vol VIII:353, quoted in Abrahamsson 1977:41-42).

These remarks relate to the complexity and specialization of the bureaucrat's training. But the socialization process of the bureaucrat is not merely cognitive, it is also epistemic. The bureaucrat does not merely understand and obey the organizations rules and procedures, (s)he also comes to ***believe*** in them with an almost patriotic or religious faith. This leads to a concept of 'organizational myth.' Michael (1977) notes that as regards the social/economic world, there are no scientific

truths. Yet we need some coherent set of beliefs in order to plan and act. We need to have 'both feet planted firmly in mid-air.' An important aspect of a successful organization is to provide a certain philosophy or set of 'myths' which provide social unity and focus.

Deal and Kennedy (1982) propose a similar concept in what they call 'corporate culture,' which they regard as a major factor in the success of such giants as Westinghouse, General Electric, IBM, and 'Japan, Inc.' The concept of corporate culture is an enthusiastic one. It has something of the flavor of a large scale football rally, complete with mottos such as 'progress is our most important product' (General Electric), 'better things for better living through chemistry' (Du Pont), and so on. The image is on of growth, innovation, aggressive and spirited competition. Peters (1980) arrives at a similar view in his remarks about building 'organizational character.'

Bureaucracies, by contrast, have typically reached a stage where further growth and innovation are limited. The emphasis is rather on stability, correctness, and control. Bureaucratic commandments are intoned, "Thou shalt not ...." Aspiration and inspiration are tempered by the guilt of transgression. This results in what Thompson (1961) calls the 'bureaupathic reaction' where,

> strict control from above encourages employees to 'go by the book,' to avoid innovations and chances of errors which put black marks on the record. It encourages the accumulation of records to prove compliance... It encourages decision by precedent, and unwillingness to exercise initiative or take a chance. It encourages employees to wait for orders and do only what they are told (Hampton 1978:365).

However, egocentric drives are not always so much suppressed as diverted. Trotsky, for instance, observed:

> Bureaucracy owns neither shares nor state bonds. It is recruited, replenished, and renewed as an administrative hierarchy, independently of property relationships. The individual bureaucrat cannot transfer the right to exploit the state apparatus to his heirs. Bureaucracy enjoys its privileges in the form of power abuse (Trotsky, *The Revolution Betrayed*, pp. 179-180; quoted in Abrahamsson, 1977:46).

This last observation is interesting in light of the remarks by Weber earlier. Weber views the process of bureaucratization as tending towards operations based on impartial rules and procedures rather than personalities and personal motives.

But, as the Trotsky quote suggests, one of the pathologies of mature bureaucracies is practically the reverse. Because bureaucrats become so wedded to their roles, they not only depend on them psychologically but also may tend to re-interpret them to satisfy their own personal ends. It is perhaps in reaction to the inflexibility of the bureaucratic structure that certain aged bureaucracies tend to develop a side market of graft and favoritism.

D. INFORMATION TECHNOLOGY IN BUREAUCRACIES

Bureaucrats are no longer the only active force in bureaucracies. Whereas a bureaucrat is trained and socialized to follow prescribed procedures, a computer can likewise be programmed to follow many of these same procedures.

Indeed, the computerization of a bureaucratic process is the ultimate form of organizational rationalization. The computer is the archetype of Weber's dictum to eliminate "love, hatred and all purely personal, irrational and emotional elements" from the organization's procedures.

Yet while computers presumably help remove the undesirable caprice of bureaucrats themselves, they nonetheless have become symbols of pathological bureaucratic rigidity. We are all acquainted with the agonies of trying to rectify a computer based billing error, etc.

But is this really because the computerization of such process actually makes them less adaptive, or is it rather that computers provide a convenient scapegoat for organizational incompetence? Systems analysts will often argue that the latter is the case. While this may be partially true, it is also true that computerization, at least in its most prevalent forms, does add to inflexibility. This stems from two interrelated problems.

The first is one of organizational responsibility: The people that use the computer programs are very seldom the ones that write them. Thus the people that are close to the problem and able to recognize needed modifications as they arise, must request the assistance of a programmer, who typically resides in a different (data processing) department. This problem has been widely recognized and is oft cited as a motivation for localized (microprocessor) computing and associated high level languages that the functional departments themselves can control; see e.g. Fick (1980). However, this is likely to be only a partial solution, applicable only to those procedures that are modular and separable to individual departments. The problem still would remain as to the management of procedures that pervade large segments of the organization, especially where these are complex and interdependent.

E. DIFFICULTIES IN SOFTWARE ADAPTATION

A second source of inflexibility arising from computerization arises, from the character of the computer languages used to describe these procedures (Lee 1980a).

Anyone who has written even small programs will know that it is much easier to incorporate a given feature in the program logic in its original writing rather than try to add this feature afterwards. This difficulty rises exponentially with the complexity of the original program or system. (By 'system,' is meant a collection of programs and data files with interdependent functions.) Indeed, the cost and effort of modifying such systems often exceeds that of their original development. For instance, Wulf (1977) refers to:

> the extreme difficulty encountered in attempting to modify an existing program. Even though we frequently believe that we know what we will want a piece of software to do and will be able to specify it precisely, it seems to be invariably true that after we have it we know better and would like to change it. Examination of the history of almost every major software system shows that so long as it is used it is being modified! Evolution stops only when the system is dead. The cost of such evolution is almost never measured, but, in at least one case, it exceeded the original development cost by a factor of 100.

Altering existing computer systems is not only expensive, it is also risky. DeMillo et al. (1979) noted:

> Every programmer knows that altering a line or sometimes even a bit can utterly destroy a program or mutilate it in ways we do not understand and cannot predict...

Indeed, beyond expense and risk, there seems to be an eventual limit to the number of modifications these systems can undergo. Winograd (1979:392) remarks

> Using current programming techniques, systems often reach a point at which the accretion of changes makes their structure so baroque and opaque that further changes are impossible, and the performance of the system is irreversibly degraded.

To summarize, the basic problem with current application systems is that they are 'brittle;' i.e., they cannot easily be reformed to adapt to changing circumstances. This brittleness has profoundly disturbing consequences as more and more organizations, ranging from small and medium size companies to immense governmental agencies, convert their information processing to computer software. The immediate gains of increased efficiency, speed of processing, rapid access to centralized data files, etc., are clear (or the investment would not be justified).

However, there may be a long term, possibly devastating hidden cost as the organization finds its ability to adapt and respond to new environmental conditions hampered by its inability to modify its information systems accordingly.

F. THE PROBLEM WITH PROGRAMS: PROCEDURAL LANGUAGES VS. PRODUCTION SYSTEMS

Statements in a programming language are in the form of *commands* to the machine —i.e., add this, move this data from here to there, print this on the terminal, etc.

A computer program is thus a *sequence* of such statements, e.g.,

```
10  LET X = 2
20  LET Y = 3
30  LET Z = X + Y
40  PRINT Z
```

Here, the statements have been numbered for identification purposes. Importantly, the ordering of the statements in this program indicates the sequence in which the commands are to be performed by the machine.

This otherwise linear sequence of execution can be modified by what are called 'control statements'. Consider, for instance, the program:

```
10  LET X = 0
20  ADD 1 TO X
30  PRINT X
40  IF X = 100 GO TO 60
50  GO TO 20
60  STOP
```

When executed, this program prints the numbers from 1 to 100. Here, statements 40 and 50 are control statements. In statement 40, if X has reached 100, program control jumps to statement 60 where it stops. Otherwise, statement 50 directs the program control back to statement 20 where X is again incremented, printed, etc.

Thus, the execution sequence in such computer programs normally follows the top to bottom ordering of the statements, except when superceded by the effects of control statements. Computer languages of this type are called *procedural*. These are basically the only type used in commercial practice, and include all the well known languages for data processing and scientific applications — e.g., COBOL, FORTRAN, PL/I, BASIC, ALGOL, etc.

In these cases, the 'knowledge' embodied in the computer program is expressed as the specific steps for doing it. A key thing to recognize is that this procedurality makes the statements of the program interdependent. Generally (though not always) changing the order of any two statements makes a serious change to the program's operation. While it may not be patently obvious from the two tiny examples above, it is this inter-dependence that makes computer programs so difficult to modify.

As a result of an interesting blend of computer science and formal linguistics, an alternative approach has emerged over the last decade or so. This approach is based on so-called 'production systems' (PS's) which permit the knowledge of the program to be expressed in a form that is independent of its execution sequence.

The concept of production systems was first proposed by the linguist Post in 1943 to aid in the formal specification of natural language grammars. The basic idea is extremely simple. A single production is a rule of the form:

IF <pattern> THEN <action>

or, in the more usual notation,

<pattern> → <action>

A production *system* consists of a 'data base' and a collection of such production rules. (This is a database in a fairly restricted sense, not to be confused with those maintained by database management systems.)

The pattern in each rule is some condition to be matched by the database and the action is typically some modification to the database. In the 'purest' form of a production system, the rules are arranged in a linear order. Starting from the beginning the patterns are compared to the database until a successful match is found. The corresponding action is then performed and the process is then repeated, starting once again from the beginning comparing the patterns to the database.

Consider for instance the following example for recognizing a certain type of English declarative sentence.

1	THE	→	DET	8	N	→	NP
2	ON	→	PREP	9	ADJ NP	→	NP
3	HUNGRY	→	ADJ	10	DET NP	→	NP
4	BIT	→	VT	11	PREP NP	→	PP
5	DOG	→	N	12	VT NP	→	VP
6	CAT	→	N	13	VP PP	→	VP
7	NECK	→	N	14	NP VP	→	S

The production rules on the left represent a 'lexicon' indicating the grammatical categories of various words. The rules on the right indicate the grammar proper. When the terminal symbol "S" is reached, the sentence is accepted as grammatical. Thus, suppose we have the following sentence:

"The hungry dog bit the cat on the neck."

This is analyzed as follows:

DET ADJ N VT DET N PREP DET N	Rules 1-7
DET ADJ NP VT DET NP PREP DET NP	3 x rule 8
DET NP VT DET NP PREP DET NP	1 x rule 9
NP VT NP PREP NP	3 x rule 10
NP VT NP PP	1 x rule 11
NP VP PP	1 x rule 13
S	1 x rule 14

The initial application of production systems in computer science were in the area of compiler theory, i.e., in specifying the syntax and interpretation of programming languages (as opposed to natural languages). Subsequently, it has been recognized that PS's have a potential much broader range of usefulness. For instance, one classic application was the Logical Theorist of Newell, Shaw and Simon (1963). Beginning with the initial axioms and rules of inference of Russell and Whitehead's *Principa Mathematica*, the Logical Theorist successfully proved all the theorems of this massive text. Indeed, in several cases it found original proofs, simpler than the original.

Another famous example of the use of production systems was Shortliffe's MYCIN system (1976). The purpose of MYCIN is to perform medical diagnosis. In this case, the database is the patient's symptoms, as revealed by various laboratory tests, etc. The production rules are thus the sort of medical deductions a doctor might make based on these symptoms. Within the area of Artificial Intelligence (AI) numerous other applications of production systems have been explored.

Davis and King (1975), an excellent survey article on production systems, comment on the types of applications where PS's are best suited:

> [These are] where the emphasis of a task is on recognition of large numbers of distinct states, PS's provide an advantage. In a procedurally-oriented approach, it is both difficult to organize and troublesome to update the repeated checking of large numbers of state variables and the corresponding transfers of control....
>
> [PS's are] characterized by the principle that 'any rule can fire at any time,' which emphasizes the fact that at any point in the computation, any rule could possibly be the next to be selected, depending only on the state of the database at the end of the current cycle. Compare this to the normal situation in a procedurally oriented language, where such a principal is manifestly untrue: it is simply not the case that, depending on the contents of the database, any procedure in the entire program could potentially be the next to be invoked.
>
> PS's therefore appear to be useful where it is important to detect and deal with a large number of independent states, in a system which requires a broad scope of attention and the capability of reacting to small changes.

With regard to the ease of modification of PS's, they continue (p.20):

> We can regard the *modularity* of a program as the degree of separation of its functional units into isolatable pieces. A program is *highly modular* if any functional unit can be changed (added, deleted, or replaced) with no unanticipated change to other functional units. Thus program modularity is inversely related to the strength of coupling between its functional units.
>
> The modularity of programs written as pure production systems arises from the important fact that the next rule to be invoked

> is determined solely by the contents of the database, and no rule is ever called directly. Thus the addition (or deletion) of a rule does not require the modification of any other rule to provide for (delete) a call to it. We might demonstrate this by repeatedly removing rules from a PS: many systems will continue to display some sort of 'reasonable' behavior, up to a point. By contrast, adding a procedure to an ALGOL-like program requires modification of other parts of the code to insure that it is invoked, while removing an arbitrary procedure from such a program will generally cripple it...
>
> Thus where the ALGOL programmer carefully chooses the order of procedure calls to create a selected sequence of environments, in a production system it is the environment which chooses the next rule for execution. And since a rule can only be chosen if its criteria of relevance have been met, the choice will continue to be a plausible one, and system behavior remain 'reasonable,' even as rules are successively deleted.

As described so far, pattern matching proceeds from the beginning of the rule set each time until a match is found, in which case that corresponding action is taken and the process is repeated. However, in the notion of a 'pure' PS, each rule supposedly has an equal chance of firing — i.e., its position in the rule set should not affect its chances of firing. This only causes difficulty when the patterns of more than one rule match the database, in which case a choice must be made which action to take. A variety of approaches have been used to resolve such rule contention, for instance:

rule order	—	use the first matching rule.
data order	—	data elements are assigned priority: pick the rule whose match gives the highest priority.
generality order	—	use the most specific rule
recency order	—	use the most recently executed rule.

Recall that each rule is matched against the entire database and that two simultaneously activated rules may have matches on completely separate parts of the database. Clearly, rule contention is only problematic when the firing of one rule would disable the database match of the other candidate rule(s).

Thus, in the 'pure' form of a PS, *all* of the rules should be tested against the database on each cycle, the subset of matching rules selected, and a choice made (by some criterion) which of those should be allowed to fire. However, as the database and/or number of rules gets large, the system degrades for lack of efficiency.

In face of this, a number of production system implementations have allowed some degree of control structure to creep back in. Thus, various strategies or 'heuristics' have been employed to increase the likelihood that, for certain contexts, the applicable rules will be found quickly and

that the entire rule set need not be examined without danger of ignoring an applicable rule.

Thus, a number of PS implementations exhibit a greater or lesser degree of 'partial procedurality' as production systems augmented with a control structure mechanism. The design of such control structures, providing efficient search without nullifying the advantages of flexibility offered by the basic PS orientation, has become a matter of intense interest and debate within computer science (see, e.g., Winograd 1975; Kowalski 1979b).

G. THE PROBLEM WITH DATA: DATA FILES VS PREDICATE CALCULUS

Most application software used in organization centers around the processing of large amounts of data (as opposed to, for instance, optimization routines which are much more computation intensive on relatively small amounts of data). Hence, inflexibilities introduced by the way data is organized in data files and databases are equally (if not more) important than those introduced in the design of procedural programs. At any rate, as will be seen shortly, the problems are highly inter-related.

A note on terminology. In the last section, the term database was used to designate the data repository of a production system. In this section, the term database will be used more in the sense associated with database management (DM). Somewhat later we return to compare the two views at which point they will be distinguished as PS databases and DM databases.

For the moment, however, we focus on a *general* view of data maintained in data processing applications, whether this data is accessed through a database management system or not. The term 'data file' is therefore used to indicate a conventional data processing file or a logical segment of a database (e.g., the tuples of a single relation in a relational database; the instances of a single record type in a CODASYL database). The term 'database' will then be used to refer to a collection of such data files with inter-related subject matter (e.g., sales file, inventory file, back-order file), whether or not the access to these is coordinated by a DBMS.

Data files are usually organized as a rectangular table with labeled columns called 'fields.' For instance, a file on employees might have fields for the employee's name, address, age, salary, etc.

EMPLOYEE FILE

Name	Address	Age	Salary
Adams	5 Pine Street	30	20,000
Peters	101 Broadway	45	18,000
Smith	3 Park Place	37	24,000

Sometimes data files have more complicated organizations — e.g., some columns may have multiple entries for a given data item. This tabular view is sufficient for the purposes here, however. As Kent (1978) observes, this is essentially the view taken by the more popular database management models (i.e., Network, Relational).

Note that each data file has three levels of description: the data file name (e.g., EMPLOYEE), the field names (e.g., NAME, AGE), and the data values (e.g., Smith, 37). It is important to note also that a data file represents a *model* of some aspect of the organization, in this case, what are considered to be the important features of employees.

The structure of the data file often carries certain implicit information as well. Often, as in this example, each row of the data file implies the existence of some entity in the environment, in this case an employee associated with the company. The converse assumption is also sometimes made, e.g., if a person's name does not appear in the file, then he/she is not an employee.

Other data files, however, might have different existence assumptions. Consider for instance a file for parts inventories.

PART FILE

ID#	Color	WT	QTY
3	R	10	200
12	B	8	65
7	W	13	0

This file indicates the identification number (ID#), color, weight (WT) and quantity (QTY) on hand of various manufactured parts. In this case, each row of the file does not imply the existence of a part, but only elaborates the features of each generic part type. The existence of actual parts is instead indicated by the QTY field.

These might be called the existential assumptions associated with a file. Other assumptions refer to the possible data values that may appear in a given field, e.g., that SALARY must be less than 50,000.

The basic point, however, is that the data file structure itself is not sufficient to convey all these assumptions. Instead, these appear in the logic of the programs that interpret these data files. Thus, the model of the organization represented in the application system is found not only in the data files but also in the code of the various application programs. This is a problem that has been recognized for some time in database management, and has led to a number of proposals for the separate specification of so called 'data base constraints,' conditions that the data in the database must always fulfill. Such constraints are maintained in a separate table, and verified by each updating program. However, these

approaches do not go far enough. There is a basic problem that remains, which has to do with the very notion of 'data' itself.

In *all* data processing files and database management systems, there is a distinction between *data structure* and the data itself. What we have called the data file names and field names are examples of data structure elements. Thus, for instance, in the above data file for parts, we have in the first row: COLOR = "RED" where the three character string "RED" is the value of the field COLOR. The point is that these data values are regarded as *strings of characters rather than as properties of objects in the environment.* Viewed only as character strings, one is unable to specify even very commonplace inter-relationships between these properties; for instance, that if a thing has a color, it must be a physical object, hence, having weight, physical extension, geographical location, etc.

The basic problem is that the variables in data management models range over sets of *character strings* (so-called 'attribute domains' in the relational model), rather than over *objects* in the environment. For instance, a database constraint that all parts are either red, blue or white would look something like:

PART.COLOR = "RED" OR "BLUE" OR "WHITE"

To recognize that these are properties of objects in the environment, a predicate calculus notation might be used, introducing the variable x to range over these objects:

1. $\forall x$ PART (x) $\rightarrow$ RED (x) OR BLUE (x) OR WHITE (x)

(the symbol "$\forall$" is read "for all"). The point is that in this form, one can begin to elaborate more general properties, i.e., not just of parts, but of anything that has a color.

2. $\forall x$ RED (x) OR ORANGE (x) OR YELLOW (x) OR GREEN (x) OR ... OR BLACK (x) $\leftrightarrow$ COLORED (x)
3. $\forall x$ COLORED (x) $\rightarrow$ PHYSICAL-OBJECT (x)
4. $\forall x$ PHYSICAL-OBJECT (x) $\rightarrow$ $\exists n$ $n > 0$ & WEIGHT (x) = n.

(the symbol "$\exists$" is read "there exists").

Statement (2) is a disjunct of all color names used in the organization, indicated that any of these implies the general feature of being colored, and vice versa, that being colored implies one of these properties. Statement (3) says that anything that is colored is also a physical object (though some physical objects — e.g., glass, mirrors — may not be colored). Statement (4) says that for any physical object there exists some positive number that is its weight (presuming some unit of weight measure).

The direction intended by this example should begin to become clear. Clearly there are many commonplace connections between properties that any organization would agree upon — e.g., the simple physics of colors, weights, physical extent, etc. These rules will hold for any physical object, from peanuts to box cars. Other classes of properties might be restricted to a particular social system — e.g., the number of spouses an employee might have, whether dual nationalities are recognized.

Other classes of properties pertain to specific industries within a given social system —e.g., the accounting practices for banks vs. those for educational institutions. Lastly, there are clearly those properties that are organization specific, such as the ranks of personnel or the parts it manufactures.

Ideally, the inter-relationship of properties at any one of these levels should only have to be developed once —e.g., commonplace physics by a national or world wide bureau of standards, accounting practices by an industry accounting board, etc. Then, the task of any particular organization in developing its application software would only be to specify the ***differences*** of its local practice from that of the standardized models.

The proposal here is, therefore, to offer a predicate calculus (PC) notation as a replacement for the usual data structure view with the claim that it provides a richer framework, capable of specifying the inter-dependence of properties of objects, not just structured organizations of character strings. Related work on the relationship between databases and logic appears in the (Gallaire and Minker 1978, Gallaire et al. 1981).

It should be mentioned that this is not necessarily a recommendation that facts about the environment actually be *stored* in this form — the underlying implementation might actually make use of a more conventional data management model —but rather that the top-most *level* or *view* of the database have the PC form.

It should also be mentioned that a predicate calculus notation is not the only candidate to meet the objectives of abstracting the relationships of general properties. The various graphical representations called 'semantic' or 'associative' network also share this goal. However, the predicate calculus has had a longer history of development and study and, in our opinion at least, is a more robust representation. The predicate calculus is, however, only a *framework*, a meta-theory in which more detailed theories can be described.

It can, for instance, be used to describe theories of mathematics, in which case the variables would range over numbers, or theories in chemistry, where the variables would range over the physical elements. Thus, the real work in pursuing this proposed direction would be to develop a predicate calculus specialized to the problems of administration. This involves, among other things, identifying a set of 'primitive' properties and relationships (i.e., single, multi-place predicates) which identify special classes of entities like people, other physical objects, money, types of contracts, etc.*

* An initial attempt in this direction was made in Lee (1980), which developed a predicate calculus notation, called CANDID, for the description of financial contracts, e.g., loans, leases, options, insurance policies, etc. A formal semantics of CANDID, leading to a contractual theory of the firm, is developed in Lee (1981a, b).

H. COMBINING THE APPROACHES: LOGIC PROGRAMMING

The point of the previous section was to recommend a predicate calculus notation as a richer form of data representation. In section [F], production systems were suggested as a more flexible framework for specifying the potential deductions of an application system. An approach which combines these aspects is so-called 'logic programming' of which the language PROLOG is an example (Clocksin and Mellish 1981, Kowalski 1979a).

Actually, production systems acting upon predicate calculus databases have been in experimental use for some time within the computer science area of artificial intelligence (AI). (See e.g., Nilsson 1980, Infotech 1981). Systems with this design are usually called 'theorem provers,' in that the function of the production system is to prove some 'goal' theorem, based on a set of initial axioms in the database. The term, 'theorem proving,' is not, however, confined to simply proving mathematical theorems. As noted in the previous section, the predicate calculus may be used to represent a wide variety of subject domains beyond mathematics. Coelho et al. (1980) includes examples of applications of logic programming in demography, university administration, biblical family trees, car rallies, biology, electronics parts, travel planning, architectural design, and others.

Very briefly, the basic concept of logic programming is as follows. The classical proof methods for (first order) predicate calculus include a wide number of inference rules (e.g., Suppes 1957:34, 99) These make computational theorem proving difficult because the space of possibilities quickly branches into an exponentially large number of alternatives. Robinson (1965) developed the so-called 'resolution method' which offered equivalent logical power but considerable computational simplicity. This advantage is gained by assuming a syntactic transformation of the logical assertions into 'Horn clauses' of the form:

$$P_0 \leftarrow P_1 \,\&\, P_2 \,\&\, \ldots\, P_n$$

where P_i are predicates whose arguments are either logical constants or universally quantified variables. (Nilsson, (1980:Ch.4) and Clocksin and Mellish (1981:Ch.10) show how arbitrary predicate calculus expressions can be converted to Horn clause form.) Once in this form, only one inference rule, resolution, is needed. This rule basically combines the rules of modus ponens and universal instantiation (unification).

The control structure that is then employed is similar to that discussed for production systems above. However, whereas the PS method was typically to proceed 'forward' from a starting database to the desired conclusion, logic programming designs generally proceed 'backward' from a goal statement to the basic assertions which support it. Thus, the Horn clauses are often read, "if you want to prove P_0 then prove P_1 and P_2 and ... P_n." The system then looks for other rules which have P_1 on the left hand side (called the head of the clause), and attempts to prove P_1. This approach continues recursively until unconditional assertions of the form:

P_k

are found. The proof procedure therefore takes the form of a tree structure, proceeding from the goal theorem through the various theorems supporting this goal until lowest level fact assertions are found.

Disjunction ('OR') is indicated by having multiple Horn clauses with the same head. Thus, if a proof is not found through the branch represented by one Horn clause, the system 'backtracks' to try other Horn clauses with the same head predicate until all alternatives have been exhausted. (If no proof is found, then the system response is 'I don't know.')

An advantage of this type of software architecture is that it allows the specification of so-called 'heuristics,' i.e., a number of overlapping rules for a given situation, which may be more or less effective depending on situational variations. Thus, for instance, a computer program confronted with a particular problem may try to resolve it using one set of heuristic rules; if those do not work, other rules are tried, etc.

In such systems, there may in fact be multiple ways to solve a given problem (or none at all); and it is the job of the program to find a satisfactory solution as quickly as possible. Because the program must search through a number of potentially feasible alternatives for each problem, rather than having a single solution technique pre-selected, these systems are considerably less efficient, though correspondingly more flexible in dealing with highly varied situations. For this reason, application programs using these methods are often called 'expert systems.'

I. THE SEMANTIC PROBLEM

Aside from software adaptability considerations, another effect of representing information systems as a formal logic is to clarify and focus certain linguistic issues.

Regarded as a logical processor, the role of the information system in the organization can be viewed as a linguistic mediator between members of the organization separated geographically and temporally.

However, the information system has an important function not provided by other communications technologies such as memos, the telephone or electronic mail. The information system not only conveys messages in the form they are entered, but it may also do *inferences* on the facts in the database. Database queries are an elementary form of inference. More sophisticated inferences are provided by routines producing summary data and other statistical or mathematical analyses.

In order to draw inferences, the facts must be structured or *formalized*. This is a basic activity in database design. Viewed abstractly, this amounts to expressing these facts in a *formal language*. A formal language is one controlled by fixed, explicit rules, typically including the following (van Fraassen 1971):

I. Syntactic Rules

A. Vocabulary

B. Formation Rules (delimiting well-formed expressions)

C. Transformation Rules (providing inferences)

II. Semantic Rules.

A formal language contrasts with a natural language in that the latter is controlled by the (evolving) consensus of its speaker population, rather than by fixed rules. Note that by this definition, the difference between formal and natural languages is one of authority rather than complexity. Formal languages might someday be invented which are more complex than natural languages, either syntactically or semantically. Note also that so-called 'natural language' interfaces to databases are also formal languages by this definition.

In using the information system, individuals channel certain of their communications through this formal language. Of particular importance is the semantics of these communications, that is, how the formal expressions correspond to the environmental reality.

The foundational work in formal semantics is by Tarski (1956), who focused almost entirely on first-order predicate calculi of the type described above. Tarski regarded semantics as a mapping from symbols, expressions in the language to objects in the world. An immediate problem is that in describing the semantics of a language, we generally cannot physically point to all the objects we wish to designate. Tarski's approach was to make use of another, 'meta-language' for this purpose. The meta-language he adopted was set theory, and nearly all the subsequent work in formal semantics has followed this precedent.

The semantics of a first order language, L, begins by adopting some universal set called the 'domain of individuals', D. A function F, called the interpretation function, maps expressions in L to set theoretic expressions based on D. Logical constants (names) in L map to individual elements of D; one-place predicates map to subsets of D; n-place predicates map to relations on D, and so on.

Tarski called the combination <D,F> a *model* of L (not to be confused with the operational research usage of 'model'; a more familiar term might be an *interpretation* of L). Clearly, a given language L can have any number of models (interpretations). This led Tarski to the distinction between synthetic truths, which are true only under certain models (interpretations) vs analytic or logical truths which are true for all possible models. For example,

a) $\forall x\, \mathrm{LEMON}(x) \rightarrow \mathrm{FRUIT}(x)$

b) $\forall x\, P(x) \vee \sim P(x)$

Statement a) is true only for certain interpretations of LEMON and FRUIT whereas b) is true for any interpretation of P.

In information system applications we are mainly concerned with truths of the first (synthetic) type. The truth of these inferences thus depends on the semantics we attach to these terms. For instance if we interpret FRUIT as the set of all fruit but LEMON as the set of all elephants, then the above implication is false (in this model).

In formal languages, then, semantics is parametric. This is quite different from the way we understand our natural language (e.g., English). In natural language we grow accustomed to a rather fixed, ongoing interpretation to our words.

It is largely by association with natural language terms that the expressions conveyed by the information system have a semantics to the users in the organization. For instance, a database may use terms like EMPLOYEE, SALARY, DEPARTMENT, etc. and we interpret them using their natural language correspondents.

But here we confront a basic problem raised by the distinction between formal and natural languages. The semantics of natural languages is not entirely fixed. The sorts of things we call 'automobile,' for example, are quite different today than fifty years ago. And the interpretation continues to change as each year new automobile designs come into the market. Indeed, it is in the area of social artifacts — economic goods and services — where linguistic change is most rapid. (This contrasts with terms for natural phenomena, e.g., horses, mountains, water, which change more slowly. Social/economic evolution is more rapid than biological or geological evolution.)

On the other hand, the inferences drawn by an information system depend on a fixed, static semantics for their validity. The potential contributions of logic programming and other artificial intelligence innovations will not alter this situation.

The linguist Whorf (1956) is often cited for his observations about the inter-dependence of language and culture. For example, Eskimos have more than a dozen words for snow each of which reflects an important technological distinction in their culture (e.g., good for igloo building, bad for dog sleds, etc.). The Hopi Indians do not have a word corresponding to the English 'time,' nor do their verb tenses make the temporal distinctions between past, present, and future. This is a reflection of their religious and metaphysical beliefs.

Less fascinating but equally important examples abound on the inter-dependence between organizational language and organizational culture (e.g., Kent 1978, Peters 1980). Obvious examples are the names assigned to product lines and specialized, in-house technology. Countless more subtle examples are found in the administrative terminology found in managerial accounting reports, memos, 'shop talk,' office gossip, etc. Thus an organization, to adapt to environmental changes, must also adapt its language, either by introducing new terms or by changing the interpretation of existing terms.

If the information system is to keep pace with this change, inferences which depend on the evolving terminology will have to be correspondingly modified. This presents another important trade-off for the use of information technology in dynamic organizations.

However, strong arguments can be made that these systems will *never* be able to adapt *themselves*.

Putnam (1970, 1978) gives a very persuasive account in what might be regarded as a sociological theory of semantics. Consider my knowledge of the concept 'lemon.' I am not much of a cook, so my understanding is fairly rudimentary. I cannot, for instance, distinguish lemons from yellow limes (if such things exist). But, on the other hand, I do having a working concept that suffices for my needs. I go to the supermarket. Since I know that lemons are a subset of fruit, I go to the fruit section. There, typically, are bins one of which is labeled 'lemons,' and I pick from that. (If they are not labeled, I use the heuristic of picking the yellow, oval-shaped objects somewhat smaller than my fist. But then I may end up with yellow limes of course.)

But how does the supermarket know what to label 'lemons'? They place an order to the fruit distributor indicating they want to buy 'lemons.' How does the distributor know? They buy from the farmers who grow lemons. How does the farmer know? He buys lemon seeds to grow lemon trees from an agricultural supplier. If we continue following this chain, we eventually arrive at the advice of botanical science, which we take to be authoritative.

The point is that many of our concepts are not understood by each of us individually, but rather through a complex social network. As Putnam remarks, we tend to think of words as tools, but many are not like hand tools that we use individually, but rather are more like an ocean liner that requires a crew of hundreds for its operation.

In the example of 'lemon,' the social network led to a certain scientific authority, the botanist, who is assumed to know all there is to know about lemons. We seem to regard science as the base authority for most of the terms used for physical and biological phenomena.

This is not the case for psychological and social phenomena. Many of us are skeptical that 'intelligence' is what an IQ test measures. Nor do we easily accept the psychophysics definition of 'anxiety' as Galvanic Skin Response.

More relevant to the activities of commercial organizations are terms for mundane, social artifacts. For instance, consider the concept 'chair.' What definitions are available? Consider examples like office chairs, an over-stuffed easy chair, beach chairs, bean-bag chairs, etc. Perhaps the only common characteristic is that we sit on them. But then consider what you sit on during a picnic or while visiting a house in Japan.

What counts as a chair, it seems, depends much more on social context than on any physical characteristics of chairs. It is closely tied to the sociology of sitting, which we each learned through a long acculturation process. Further, the concept of chair is only partially known to each of us, and probably has a larger and different concept for each social aggregate we examine, (e.g., the concept chair in New York vs Tokyo vs Europe, etc.) Further, the concept is constantly changing. Indeed, the marketing strategy of certain furniture design companies depends on extending and altering our conception of 'chair.'

If we are persuaded by this view of semantics, then we are forced to admit to an eventual limitation to the technological promises of production systems, logical databases, and logic programming.

We tend to underestimate the magnitude of the semantic problem, particularly as regards terms describing objects and phenomena in the social sphere. (And it is these aspects that are of primary concern to most organizations.)

Organizations, as sub-cultures, represent an intermediate case between the individual's understanding of these terms, and the understanding of the society at large. However, it is based on a dual membership of individuals, in the organization and in the society, that the organization itself maintains an effective relationship with its social environment. Computer systems are not likely to duplicate the mechanisms by which we adapt our language unless they too attain a similar social membership. Only then can they learn why, for instance, "ring around the collar" implies embarrassment at cocktail parties).

J. CONCLUDING REMARKS: THE THIRD WAVE SCENARIO

Referring to the two dimensional taxonomy (complexity vs uncertainty) discussed earlier, Jay Galbraith (1973, 1977) observes that hierarchical management structures tend to be oriented towards aspects of the organization's activities which present the greatest uncertainty. Thus a manufacturing company, where technology is dynamic, will tend to have a functional organization, whereas an insurance company will probably have divisions based on customer type (reflecting dynamic insurance needs). Organizations which face high uncertainty in multiple aspects may find it useful to adapt a matrix type of organization with dual hierarchies. For example, a computer manufacturer might have management hierarchies based on technological aspects and another hierarchy based on customer differentiation.

In these cases, the lower level operating departments have two sets of superiors in the authority structure and often are in an arbitrating role rather than a strictly subordinate one.

In *The Third Wave*, Toffler (1980) creates a fascinating scenario which carries this trend to the level of economic organization. The first wave 'economic' structure of primitive societies is mainly agrarian, with largely manual and animal technology. Population is limited to small villages that are economically self-dependent. Production and consumption are closely associated.

The 'second wave' is characterized by heavy industrialization. Economies of scale lead to the formation of large, centralized organizations, and, as a consequence, population concentrations in large cities. Statistical, mass marketing becomes the link between producer and consumer.

We are now, says Toffler, entering into the transition from industrial society to a new, 'third wave.' As material demands become satiated with cheap manufactured goods, other social and environmental factors take on increased importance. We become concerned about pollution and

destructive alterations to the environment. Product safety and the quality of food and drugs become issues. We begin to resent living in noisy, crowded cities and working in large, impersonal work environments. Special interest groups form around each of these themes and each exerts its own political/economic pressures. To respond to the wide, dynamic variations in demand, organizations 'un-bundle' their products and operations and become de-centralized, loosely-connected systems.

This permits demographic shifts back to smaller, more human-sized villages. But these are not the isolated villages of the first wave, but rather are inter-connected in a large social network.

Interestingly, the key factor that Toffler sees to enable this transition is information technology. Small-scale, custom manufacturing becomes possible through flexible automation. Geographical decentralization is supported by tele-work and electronic networks. Education and research advance through electronic libraries, computer-aided instruction and expert systems. The Japanese seem to have a similar scenario in mind in their development plans for '5th generation' computing technology (Moto-oka 1981).

The world of the third wave is an attractive one. It portrays the relief of numerous social tensions and anxieties not only for technologically advanced countries but for developing countries as well. It does however place a tremendous import and responsibility on issues raised in this paper, particularly on the semantic problems discussed in the last section.

An organization may define (hence fix) its terms to suit its special interests, and construct complex information systems based on that vocabulary. As discussed, the continued use of the information system relies on the permanence and stability of these definitions, so that the use of the technology results, ultimately, in a trade-off much like that of bureaucratic rationalization. By fixing its language, the technology aids in coping with complexity (including foreseeable variations) but loses its usefulness as the organization is confronted by unexpected and surprising phenomena.

The problem is all the worse in the inter-connected society predicted by the third wave scenario. Here the request is for technologies that not only aid in the management of complexity, and are adaptable but must also be compatible with a number of over-lapping organizational/societal hierarchies.

The suggestions here have been that the new technologies promised by artificial intelligence research will indeed by an important factor. But they will not be the only factor in the Brave New World of the Third Wave. The limits to information technology are bounded by the limits of bureaucratic rationalization.

REFERENCES

Abrahamsson, B. 1977. *Bureaucracy or Participation: The Logic of Organization*. London: Sage Publications.

Burns, T. and G.M. Stalker. 1961. *The Management of Innovation*. London: Tavistock Publications.

Clocksin, W.F. and C.S. Mellish. 1981. Programming in Prolog. Berlin, Heidelberg, New York: Springer-Verlag.

Coelho, E., J.C. Cotta and L.M. Pereira. 1980. How to Solve it with Prolog. Lisbon: Laboratorio Nacional De Engenharia Civil.

Davis, R., and J. King. 1975. An Overview of Production Systems. Stanford AI Lab Memo AIM-271, Stanford Computer Science Report. STAN-CS-75-524. Stanford, California.

DeMillo, R.A., R.J. Lipton, and A.J. Perlis. 1979. Social Processes and Proofs of Theorems and Programs. *Communications of the ACM*, 22(5):271-280.

Deal, T.E. and Kennedy A.A. 1982. *Corporate Cultures*. Reading, Massachusetts: Addison-Wesley.

Downs, A. 1967. *Inside Bureaucracy*. Boston: Little, Brown & Co.

Fick, G.P. 1980. Small Computers in Organizations: Issues and Arguments — or — How to Fight With Computer-Enhanced Bureaucracy. WP-80-146. Laxenburg, Austria: International Institute for Applied

Systems Analysis.

Galbraith, J. 1973. *Designing Complex Organizations*. Reading, Massachusetts: Addison-Wesley.

Galbraith, J. 1977. *Organization Design*. Reading, Massachusetts: Addison-Wesley.

Gallaire, H., and J. Minker. eds. 1978. *Logic and Data Bases*. New York and London: Plenum Press.

Gallaire, H., J. Minker and J.M. Nicolas. eds. 1981. *Advances in Data Base Theory* Volume 1. New York and London: Plenum Press.

Hampton, D.R., C.E. Summer and R.a. Webber. 1978. *Organizational Behavior and the Practice of management*. Third Edition. Glenview, Illinois: Scott, Foresman and Company.

Infotech. 1981. Machine Intelligence. Infotech State of the Art Report,

Kent, W. 1978. *Data and Reality*. Amsterdam: North-Holland.

Kowalski, R. 1979a. *Logic for Problem Solving*. New York and Oxford: North Holland.

Kowalski, R. 1979b. Algorithm = Logic + Control. *Communications of the ACM*, 22(7):424-436.

Lee, R.M. 1980a. Applications Software and Organizational Change: Issues in the Representation of Knowledge. WP-80-182. Laxenburg, Austria: International Institute for Applied Systems Analysis.

Lee, R.M. 1980b. CANDID: A Logical Calculus for Describing Financial Contracts. Ph.D. dissertation, available as WP-80-06-02, Philadelphia, PA: Department of Decision Sciences, the Wharton School, University of Pennsylvania.

Lee, R.M. 1981a. A Formal Description of Contractual Commitment. WP-81-156. Laxenburg, Austria: International Institute for Applied Systems Analysis.

Lee, R.M. 1981b. CANDID Description of Commercial and Financial Concepts: A Formal Semantics Approach to Knowledge Representation. WP-81-162. Laxenburg, Austria: International Institute for Applied Systems Analysis.

March, J.G. and H.A. Simon. 1958. *Organizations*. New York: Wiley.

Michael, D.N. 1977. Planning's Challenge to the Systems Approach. In H.A. Linstone an W.H.C. Simmonds, eds. *Futures Research*. Reading, Massachusetts: Addison-Wesley.

Newell, A., J. Shaw, and H. Simon. 1963. Empirical Explorations of the Logical Theory Machine. In *Computers and Thought*, E. Feigenbaum and J. Feldman, eds., New York: McGraw-Hill, pp.109-113.

Nilsson, N.J. 1980. *Principles of Artificial Intelligence*. Palo Alto, CA: Tiogo Publishing Co.

Moto-oka, T. ed. 2981. *Fifth Generation Computer System*, Proceedings of the International Conference on Fifth Generation Computer Systems, Tokyo, Japan, October 19-22, 1981. Amsterdam: North-Holland.

Peters, T.J. 1980. Management Systems: The Language of Organizational Character and Competence. *Organizational Dynamics*. Summer.

Putnam, H. 1970. Is Semantics Possible? In: H.E. Keefer and M.K. Munitz, eds, *Language, Belief, and Metaphysics*. New York: State University of New York Press. Also reprinted in: S.P. Schwartz, ed, *Naming, Necessty, and Natural Kinds*, 1977. London: Cornell University.

Putnam, H. 1978. *Meaning and the Moral Sciences*. Boston: Routledge & Kegan Paul.

Robinson, J.A. 1965. A Machine Oriented Logic Based on th Resolution Principle. *Journal of the ACM*, 12(January):23-41.

Shortliffe, E.H. 1976. *Computer- Based Medical Consultations: MYCIN*. New York: America Elsevier.

Simon, H.A. 1955. A Behavioral Model of Rational Choice. *The Quarterly Journal of Economics*, Vol. LXIX.

Suppes, P. 1957. *Introduction to Logic*. New York: D. van Nostrand Company.

Tarski, A. 1956. The Concept of Truth in Formalized Languages. In A. Tarski, *Logic, Semantics, Metamathematics* (translated by J.H. Woodger. Originally presented to the Warsaw Scientific Society, March 1931 in Polish. Oxford: Clarendon Press.

Thompson, V.A. 1961. Bureaucracy and Bureaupathology. In: Hampton, et al. eds. *Organizational Behavior and the Practice of Management*. Glenview, Illinois: Scott Foresman.

Toffler, A. 1980. *The Third Wave*. New York: Bantam Books.

van Fraassen, B.C. 1971. *Formal Semantics and Logic*. New York: Macmillan.

Weber, M. 1956/1978. *Economy and Society*. Berkeley, California: University of California Press, translated from *Wirtschaft und Gesellschaft*. 1956. Tuebingen: J.C.B. Mohr.

Whorf, B.L. 1956. *Language, Thought and Reality: Selected Writings of Benjamin Lee Whorf*, edited by J.B. Carroll. New York: Wiley.

Williamson, O.E. 1973. Markets and Hierarchies: Some Elementary Considerations. *American Economic Review*, 63(2):316-325.

Winograd, T. 1975. Frame Representations and the Declarative/Procedural Controversy. In D.G. Bobrow and A. Collins, eds., *Representation and Understanding*, pp.185-210. New York: Academic Press,

Winograd, T. 1979. Beyond Programming Languages. *Communications of the ACM*, 22(7):391-401.

Wulf, W.A. 1977. Some Thoughts on the Next Generation of Programming Languages. In A.K. Jones, ed., *Perspectives on Computer Science*. New York: Academic Press.