ISBN 0-315-56092-4

Hani Fanous

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

July 1989

ABSTRACT

A Blackboard Architecture for Signal Processing

Hani Fanous

The complexity of Signal Interpretation Tasks processed with weak methods can overwhelm computational resources. Knowledge Intensive Programs can be more effective provided the programming environment allows effective representation of all knowledge about the problem and makes effective use of the information during the solution. The Blackboard Architecture proposed in this study uses planning techniques to control its search strategy. The extent of plan development is not fixed, but varies according to the knowledge of the sub-problem that is available to the system. Both control and object knowledge are supplied by the user in a declarative style and the planning process is completely transparent. The system was developed for a speech recognition application, but is suited for a broader category of signal interpretation tasks.

Acknowledgements

I would like to thank All my friends and family who have given me support during the last few years. In particular, I would like to thank Dr. Renato De Mori for his advice and patience. I would also like to thank all the staff of the Centre de Recherche Informatique de Montréal for the help and assistance they provided. Without the extensive use of facilities at CRIM, much of this work would not have been possible. I would like to express special thanks to Radia who was always supportive and encouraging when progress appeared to slow down.

For Radia

Index

List of Figures

1. Chapter one

1.1. Introduction

1.1.2. General description

This thesis describes a blackboard suitable as a vehicle for reasoning in signal processing applications where exhaustive Hypothesize and Test is computationally prohibitive As is common with many such blackboards, the data generated is assumed to be highly structured. The blackboard is implemented in Common Lisp and uses the knowledge representation concepts of CRL (Carnegie Representation Language) [KNOW88] as well as the OPS inference engine provided by the KnowledgeCraft shell. The software organization allows the user to augment the basic data structures manipulated by the blackboard to represent the particular application's objects The same principle applies to the rules which trigger the invocation of knowledge sources

The main contribution of the system, is that it provides run time generated control plans and strategies needed to accelerate the search for solutions, without requiring explicit domain dependent scheduling heuristics, if these are available however, they may be incorporated with advantage. The purpose of control plans is to select a succession of regions in the solution space in which the computational effort will be expended in order to achieve some particular objective. Efficient planning mechanism must be able to devise strategies, modify them and reason about them as the results unfold, without letting the planning effort overwhelm the processing resources. This blackboard goes some way to meeting these specifications.

1.1.1. Motivation for project

The motivation for developing a blackboard arose out of a research project carried out jointly by CAE and CRIM, to devise a system capable of tuning the parameters of an aircraft simulator model The complexity of devising an exact and complete mathematical model to represent the behavior of an aircraft under realistic flight conditions compels engineers to make approximations and use alternative models to simulate different manoeuvres To ensure that a simulator meets its specifications, the generated signals are compared with reference aircraft signals for a set of manoeuvres Unacceptable discrepancies are detected and hypotheses are generated to account for them The system must be able to reason about hypotheses and make the necessary modifications to the model s parameters

Because the development of a problem solver's architecture requires some considerable understanding of the problem to be tackled, and because the domain aspect of the simulator application is still being investigated, the blackboard was developed around an alternative signal interpretation application

Very crude speech recognition techniques generate a lot of uncertainty and consequently very large search spaces The data generated (hypotheses) is readily organized into hierarchies and thus resembles the simulation problem It was felt that an architecture which supports this type of application would be pertinent to the simulator project as well as diagnostic and interpretation applications

1.1.3. Organization of document

The thesis is organized into five chapters

Chapter one introduces the project and then traces the historical development of automatic problem solving, the theories and ideas that shaped production system based architectures and their evolution into modern blackboard models. A considerable portion of this chapter, is devoted to the development of the concept of an Information Processing System (IPS), which was the basis for the use of production systems and later on blackboards, in general problem solving At the end of the chapter the basic organization of a blackboard is presented.

In chapter two, a major production system language and two major blackboard architectures are presented. The concept of planning as the objective of the problem itself is introduced, and modern planning systems are analyzed. In the final part of the chapter, the role of planning as an instrument of control for a blackboard, is proposed.

Chapter three discusses aspects of problem space complexity and some approaches developed to reduce it. The role of planning control strategies in blackboard architectures is introduced and the basic design and operation of the system is described.

Chapter four describes the domain of application, the speech front end, lexical retrieval and the parser

Chapter five describes the implementation of the blackboard, and evaluates the results obtained with the speech recognition application. The applicability of this system to other problems is illustrated by contrasting qualitatively, the models used to solve a well known problem1 with those developed using an alternative blackboard The chapter

---

1 The problem used as an illutration is the one described in HASP/SIAP [NII&FEI82]. The task involves hypothesizing a naval situation board, using accoustic data suplemented with various intelligence reports. The system origially developped was one of the first important blackboard architectures.

concludes with a critical summary and a discussion of extensions and possible developments

1.2. Human Problem Solving

1.2.1. Historical background of blackboard architectures

The origin of blackboard architectures can be traced back to work done during the late fifties and sixties Computer scientists were trying to characterize problems whose solutions did not lend themselves readily to automation The difficulty, it was thought, lay in the structure of problems

"A problem is well structured to the extent that it satisfies the following criteria

- It can be described in terms of numerical variables, scalar and vector quantities
- The goals to be attained can be specified in terms of a well defined objective function
- There exists computational routines that permit the solution to be found and stated in actual numerical terms " [SIM&NEW58]

Any problem not satisfying all three criteria listed above was considered ill structured and thus not amenable to computer processing

The emergence of a number of "heuristic" programs performing tasks such as game playing, theorem proving, symbolic integration etc [FEIG63], demonstrated that in certain cases, solutions to problems not traditionally considered well structured , could still be modelled on computers This class of problem, though lacking well specified algorithms, was otherwise precisely defined A new category was devised, the well formed subset of ill structured problems, for which a precise and formal description of the task could be provided, and for which the goals were readily tested [REIT58]

Newell's point of view however, was precisely the reverse. Structure was not a property of the problem, but a perception of the problem solver [NEWE69]. The inability to automate certain problem solving processes, was due to the bankruptcy of the problem solving model, rather than the nature of the problem in question.

Strongly influenced by Cognitive Psychology, some researchers felt that an understanding of the Human Problem Solving Process, which appears unhampered by problem structuring, would shed light on the difficulties encountered with the computerization of certain tasks [SIM&NEW72].

1.2.2. The human problem solving model

The Theory of Human Problem Solving proposed by Simon and Newell was pivotal to the development of blackboards and other problem solving architectures because it signaled the end of the traditional reliance on "weak methods" (generate and test etc), to solve problems of AI. By investigating the mechanisms by which human beings solve problems, those researchers pioneered the development of knowledge representation and promoted the adoption of the production system model for problem solving This in turn provided the basis for the subsequent development of knowledge intensive programs [NII&FEIG82].

The theory is based on empirical evidence gathered by observing candidates solving tasks such as chess playing [NEW&SIM65], logic puzzles [NEW&SIM57] and cryptarithmetic problems [NEWE67] deemed "problematic", and which had often been, not surprisingly, the subject of earlier "heuristic" programs.

Human problem solving according to this theory can be modeled by an Information Processing System. Once the parameters of the problem have been understood and internal symbolic representation for the initial state and goal conditions generated, problem solving proceeds through a data driven response to partial solution steps. Human reasoning is carried out through the retrieval of pertinent "chunks" of knowledge, stimulated by eternal evidence or by the result of a previous problem solving step Each chunk in turn results in the modification of partial states (and tentative advancement of the solution) The process stops automatically when the specified goal conditions have been recognized in the partial solutions

A full description of the Information Processing System and the mechanisms that are postulated in the theory to account for human problem solving performance, is given in appendix 1

The major consequence of Newell and Simon's theory was a renewed interest in production systems for general problem solving and an attempt to represent human knowledge in a framework of declarative programming

1.3. The Structure of problems

1.3.1. Ill-Structured problems

The theory of Human Problem Solving together with two hypotheses put forward by Newell to classify the problem domain of AI [NEWE 69], expanded considerably the scope of tasks for which a machine model could be attempted

The **Generality Hypothesis** states that *'A General Problem Solver is one that has a collection of successively weaker methods, that demand successively less of the environment in order to be applied Thus a good problem solver is simply one that has the best of the weak methods*

The **Ill-structured Problem Hypothesis**: *"A problem solver finds a problem ill structured, if the power of the methods that are applicable to the problem, lies below a certain threshold"*.

The two hypotheses suggest that solvability is not a property of the problem itself but a perception of the problem solver. The hypotheses further suggest that instead of rigid classifications, we perceive a continuum of problem types.

At one extreme are the problems deemed very "ill-structured", for which the definition of the problem space itself is problematic and the task required is ambiguous and subject to contextual interpretation ("publish or perish!", "make a silk purse from a sow's ear!") The solutions to such tasks is currently exclusively the preserve of humans, endowed with the cultural wherewithal to interpret and make the problem statement succinct

At the other extreme are problems expressed and codified in a rigorous formalism which does not allow ambiguity. In addition, powerful problems solving methods, derived from an awareness and exploitation of the properties of the problem space, make their solution algorithmic and programmable conventionally.

In between these extremes there exists a range of problems (the so-called "well formed subset of ill-structured problems"), for which the problem statement can be clearly expressed (no ambiguity about the goal) and a problem space can be devised, but for which there are no powerful problem solving methods. Much of "heuristic programming" and the class of problems of interest in AI, falls into this category Generally the less is known about the structure of the problem space, the larger the number of states relative to the number of actions. The existence of a cohesive theory, which expresses knowledge about the problem in a concise fashion, often serves to emphasize similarities between seemingly different states [DAV&KIN77] The less is known a priori about problem space constraints, the more general (ie. broadly applicable) and weaker the methods the problem solver must use in order to attempt the solution The penalty paid is in the tentative nature of the problem solving process, hence the search for a solution

Newell illustrates the point by considering the problem of inverting a matrix, a problem for which an effective algorithm exists and which is consequently considered "well structured"

The problem space consists of the elements of matrices of order n, which include the source matrix A, the identity matrix I, and the inverse matrix X, and the basic set of row/column operators E (adding one row to another, multiplying one row by a constant etc.) The problem statement requires that matrix X be found such that

$$AX \quad I$$

or X is understood to be a transformation on A generating the identity matrix I as a goal state Then if the sequence of basic row operations E 1 E 2,E 3, ,E k is such that

$$Ek \quad E3E2E1A \quad I$$

then

$$Ek \quad E3E2E1I \quad A^{1} \quad X$$

the sequence E1,E2,E3 ,Ek is selected from the set {E} What a matrix inversion algorithm provides is the selection of operators Ek E3 E2 E1 A degradation in the power of the method, would result in a search for the most appropriate operator at every stage, the evaluation of the operator being based on the estimate of the resulting state Methods such as Hill Climbing towards I, Heuristic Searches, and finally Generate and Test are increasingly weak (they exploit less effectively constraints of the problem and are consequently more general)

The picture that emerges is not precise and does not involve hard and fast classifications of problem structuring but rather a continuum of methods, at one extreme very powerful (in that they exploit most of the constraints of the task environment) but of low generality, and at the other, methods that make little use of the structure of the problem space but are very broadly applicable. The (human) problem solver's perception of problem structure is not only related to the availability of a method for its solution (which in itself implies that the problem has been stated in a language clear enough for the generation of a problem space), but on the efficacy of that method. The efficacy of the method however is dependent on the degree to which the problem solver is aware of, and uses the constraints when generating the solution, in other words, the extent to which the structure of the problem is reflected in the problem space.

1.4. Automatic Problem Solving

1.4.1. Factors favouring the automation of problem solving

With the new perspective on the nature of problems and the mechanisms involved in human problem solving, researchers began to consider the possibility of automating tasks previously thought ill suited to computer processing. These were achieved without recourse to the traditional weak methods of heuristic programs. Some of the critical observations which prompted these attempts are listed below:

1) The realization that structure was not an intrinsic property of the task but a measure of explicit knowledge about the problem and its representation.

2) The existence of a programming model for human problem solving.

3) The capacity of the programming model to emulate the behavior of the "heuristic" programs already in existence.

4) The realization that the performance parameters of the human mind (in problem solving) was of similar order to that achievable on existing machines (with some reservations).

This marked the beginning of the development of general purpose software environments for emulating the Human IPS. Since the Theory of Human Problem Solving purports to be task independent, a system with a similar organization would in principle, allow any problem to be encoded, and would perform comparably

1.4.2. Comparisons of the machine and the human model

With human processing apparently serial and running at a rate of around 40ms /elementary process. there appeared to be a real possibility of achieving comparable performance with a machine on problems that were perceived to be "problematic" to human beings While the STM[1] of a machine can be considerably larger and does not decay. LTM cannot approach the size of human mind. either in physical size or in knowledge As is suggested by the theory. the LTM is actually production memory and the initial "understanding" phase of problem solving involves not only the creation of the problem space. but the selection of an appropriate set of productions The amount of LTM actively involved in the process of solving a problem is probably small and constant, unless the problem space and methods are changed

As the behavior of the IPS is entirely determined by the content of productions (LTM), the content of STM, and the interpreter, attention was drawn towards the development of general purpose task independent production system languages. Such an environment offered the possibility of emulating the human IPS on a machine, and

[1] STM (Short Term Memory) And LTM (Long Term Memory) are described fully in Appendix 1

therefore of attempting the solution of problems which traditionally did not appear to be suited for computer processing.

1.4.3. Psychological models and Knowledge Intensive Programs

While the primary motivation behind the early work of Newell might have been the modelling of psychological processes [NEWE73], it provided the framework for creating systems equipped with the kind of piecemeal and "informal" knowledge that characterizes human "expertize", and which performed with enough competence (albeit in very narrow fields), to be of practical use. The DENDRAL system [FEIG71], [SMIT72], [LIN&BUC80] which was designed to assist chemical analysis by incorporating knowledge about mass-spectrometry, and MYCIN [DAV&BUC77], which focused on aspects of Medical Diagnosis, were among the first of a line of systems of so called "expert" systems which encoded large amounts of task specific heuristics, in the form of productions.

The objective behind psychological models (PSG, PAS II, VIS) [DAV&KIN77] was the understanding of the mechanisms of human reasoning and perception, and through the emulation of human behavior, the development of theories on human thought. Productions in such systems are used as a formal means of expressing basic symbol processing actions. They are organized into minimal sets which are consistent with characteristics of human performance given certain tasks (including human peculiarities such as mistakes and the effects of forgetting). Analysis of these production sets, when the simulations they produce are accurate, is used to develop theories of information processing psychology [NEWE70].

The goal of knowledge based systems on the other hand, is the attainment of human performance at problem solving, regardless of the nature of human reasoning mechanisms. Productions in this class of systems, are the vehicle for encoding domain knowledge incrementally until adequate performance in achieved. It has been suggested that some measure of the success of AI programs based on production systems is attributable to the equivalence between human cognitive processes and the way in which knowledge is stored and retrieved through productions [DAV&KIN77].

1.4.3.1. Pure Production Systems

A Basic Production System consists of three components.

1.4.3.1.1. Production Memory.

Roughly equivalent to LTM, production memory is the repository of all declarative knowledge (knowledge which remains true regardless of the current state) and consists of rules represented by an ordered pair of symbol strings called LHS and RHS.

1.4.3.1.2. Dynamic Working Memory.

An expanded form of STM which is the sole vehicle for representing temporal (state dependent) information.

1.4.3.1.3. An Interpreter.

This module attempts to match most commonly the LHS of every rule with every symbol in working memory. The completion of this task marks the end of the recognition phase of the problem solving cycle. The set of instantiations or conflict-set, is the set of ordered pairs of satisfied productions (those for which the match was successful) and data elements from working memory. The interpreter selects an instantiation and performs the transformations to the working memory that correspond to the symbols in the RHS. This completes the action phase of the cycle.

1.4.3.2. The reasoning process.

If rules are viewed as a conditional statements then the iteration through several recognize-act cycles constitutes repeated applications of Modus Ponens [NILL80].

Forward chaining occurs when the interpreter matches LHS and modifies working memory according to RHS symbols. This is because the system is reacting to data representing already known states and generating new ones, in the hope of achieving a goal state.

With backward chaining the (hypothetical) goal state is the starting point, matching occurs on the RHS symbols, and the transformations performed are substitutions of LHS symbols. The process is repeated until at least one generated (hypothetical) state matches the initial (given) state. In other words the system attempts to prove the main goal.

Regardless of the direction of reasoning, productions never reference nor even have knowledge of one another. They are in fact just demons that become activated by elements in working memory. The programming discipline imposed by the use of "a unique and universal channel of communication", requires that each rule, as an element of code, leave behind the necessary symbols to be picked up potentially by any other rule. In other words programming is performed entirely by side effect [DAV&KING77]. Such an approach can only be successful provided that the symbols that are generated have a globally consistent interpretation, throughout the body of code and over the span (temporally) of the problem solving process. This forces the programmer to write preconditions that fully specify the states in which the production's application is true, and to write conditions parts that perform universally correct transformations on those states.

The philosophy of a pure production system requires that pattern matching on the LHS consist of literal, possibly variable matching with binding, and simple predicates. The RHS actions should represent elementary actions in the domain. These constraints have been carried over from the early applications of production systems which consisted mostly of string replacement mechanisms, psychological models, and heuristics learning programs [MINS67] [NEWE67] [WATE75]. In Psychological models, the LHS of productions must be confined to simple patterns, in order to model human recognition. This restriction follows from the premise that LTM consists mostly of productions leaving no room for an alternative (program like) mechanism necessary for supporting complex LHS operations. Simple RHS actions are consistent with the notion of elementary processes, and the constraints on the overall structure of productions assures the clarity and modularity that enables a program to read the productions themselves, generate new ones, modify old ones, and thus support learning. Another essential characteristic of pure production systems is the simplicity of the control mechanism.

To implement automatic generation of rules as a learning or correction mechanism, the interpreter must perform rule selection (conflict resolution) in a simple deterministic fashion and the rule generation program must be "aware" of this. The Markov algorithm [GALL70] which selects the highest priority production which is satisfied is the most commonly used rule selection scheme for "pure" production systems It provides a measure of processing economy since the LHS patterns of lower priority rules can implicitly assume the failure of predicates of higher priority rules. Heuristics can be learned experimentally by inserting a new rule just above the one which generated an error when it fired [WATE75], the ordering being in declining priority.

1.4.4. Production system applications in the absence of heuristic knowledge

In situations where the problem space is not completely factorizable, solutions will invariably be tentative. If there is a total absence of heuristic knowledge, the only method available is generate and test (whether the search proceeds forwards or backwards). Representing the problem space using a pure production system is particularly straightforward. The only data structure in working memory is the representation of a state. This in turn consists of the collection of attributes necessary to differentiate one state from the other. The only declarative knowledge is in the form of state generators which are encoded as productions. As nothing can favour one instantiation over another, little is required of the interpreter other than the recognition of a goal state. Rule selection, from the set of satisfied productions can be ad hoc. The constraints of pure production systems are no impediment to programming since there is no incentive to insert any measure of control.

1.5. Performance oriented expert systems

1.5.1. Advantages of production based programming for expert systems

Systems that implement psychological models use production systems as a means of understanding the nature of knowledge itself and the way the human mind exploits it. Expert systems on the other hand are principally concerned with effective use of available knowledge when this is too weak to be compiled into an algorithm.

Among the characteristics of the production format which makes it appealing to the developers of Task Oriented Expert Systems we note:

1) The modularity of rules, which make the system incrementally expendable. Since much of the knowledge incorporated in such systems is informal there is no rigorous theory that ensures completeness and correctness of behavior. During the course of development and maintenance of such systems there is a constant need to augment and possibly modify aspects of the knowledge base. Productions which embody an identifiable element of domain knowledge can readily be added or modified without concern for context.

2) The situation/action or stimulus/response aspect of productions which is well suited to the kind of domain knowledge available for expert system applications. Where profound theoretical knowledge of the domain allows powerful problem solving methods to be developed, they are long prescriptions of actions which require little run time evaluation and are characteristically depended on very few state variables. Powerful algorithms generate few states, their representation can be largely implicit and control flow is heavily emphasized. At the other extreme when domain knowledge is weak, the generation of states can be massive. Actions can only be specified with respect to very detailed state descriptions. Typically states have to be re-evaluted after every action. The production format with its diminished action component on the RHS and constant re-evalution of state through the LHS, is ideal for expressing low level situational problem solving activities,

3) The efficacy of production systems at evaluating the large amounts of data generated because of the weakness of available knowledge. Production systems are much

better suited at evaluating changes in a large numbers of state variables than procedural languages.

To offset the poverty of problem solving methods an expert system relies on intensive representation of every available form of knowledge [FEIG77]. The lack of homogeneity of the knowledge that is available in practice, makes the selection of a formalism to express it and exploit it particularly difficult.

The constraints on production format, which force clarity and precision perhaps at the expense of conciseness [MORA73], and the simplicity of the control scheme which forces the programmer to embody a meaningful "chunk" of knowledge within each independent production [NEWE73], constitute very serious obstacles in performance oriented expert systems.

1.5.2. Inadequacies of the Rule Format

The reasoning step involved in the firing of one production can be measured as the "distance" between the LHS and RHS of a rule. Because designers of psychological models try to analyze elements of human thought, they tend to work with a much finer grain. The human minds they model, are capable of dealing with a huge diversity of problems Considerations of LTM storage capacity favour fine grained "chunks" of knowledge that can be re-used for many tasks. Although the finer grain will require more productions to express complex actions, the control problem generated always remains manageable, because of the very reduced size of state memory. With expert systems however, the task is very specific so the need for re-usable fine grained productions no longer exists. The overhead of matching invoking a large number of "fine grain" productions instead oı a single coarser grained operation can be prohibitive.

The object of expert systems is a cost effective traversal of the search space. Regions where the quality of knowledge is high can be traversed with compiled programs and consequently more efficient programs. These will typically be complex RHS actions which carry the state considerably further than it was on the LHS.

The performance of powerful transformations without intermediate re-evaluation amounts to a definite commitment to a method. As was discussed earlier the power of a method is inversely proportional to its generality, so that the decision to apply a production which performs large transformations in its RHS will likely be applicable, only in very specific situations. Identification of such states will require more complex LHS tests.

1.5.3. Inadequacies of data format

Structure in a problem space that cannot be completely factored, allows state information in one area of the search space to be predictive (at least heuristically) in another [NEWE73]. The system can only make use of this knowledge provided state information is always available to productions which detect it. Where heuristic knowledge of this structure is available, there is strong incentive for maintaining in working memory a complete representation of the search space that has already been explored. Pure production systems as used for psychological models stress simplicity in the symbols of STM. The paucity of information available through data memory symbols together with its reduced size, precludes the use of heuristic knowledge based on histories of paths.

1.5.4. Inadequacies of Control Organization.

The simplicity of control of pure production systems which renders learning a relatively simple process, is due essentially to the reduced size of STM. Productions are invoked as a result of changes of state occuring in working memory. All interaction is forced through this "very narrow channel" of communication [WINO75]. The narrowness of the channel ensures that the level of interaction at any one cycle is kept manageably small. Where productions are used to represent every aspect of available problem solving knowledge, in order to maintain the "openess of the system which ensures that any rule can fire any time " [DAV&KIN77], the width of the channel must not restrict the scope of interactions The scope of interactions supported by a large dynamic working memory, creates the control problem. Not only does the system have to select the production from the satisfied set, but the bindings with which the production is to fire must be determined. In terms of state space search, the control system must select an expandable node and select an arc. The burden of finding the solution economically falls entirely on the control system.

1.6. Development of blackboard architectures

1.6.1. Architectural demands of experts systems

It is always more computationally efficient to compile knowledge into a form ready for computation, than it is to evaluate it through the execution processes during the course of problem solving. Where computational performance is the primary objective and sufficient problem solving knowledge exists, an algorithmic approach will always be the preferred solution. Production systems are therefore used for problem solving, only when the knowledge level is too low to specify, a priori, a sequence of problem solving actions that is computationally feasible. Even modest tasks, in situations where knowledge is weak present formidable search spaces. There is a need therefore to express every modicum of problem solving knowledge that can contribute to accelerating the solution. The power of a system is assessed both in terms of its effectiveness as a vehicle for expression of human knowledge, and in terms of the success with which such knowledge is integrated and used to achieve efficient computation [ROSE77]

A demand for computer systems that can perform very specific tasks with performances comparable to those of humans, prompted a progressive shift away from the pure production system model. While it is true that psychological models achieve comparable performances to humans solving puzzles (or more generally tasks with which they were not familiar), these newer systems are required to achieve the performance of experts solving very specific problems The performance of an individual, at a particular task, improves with experience. This may be regarded as a compilation of knowledge about the problem through observation and analysis of the solving process. Without necessarily being able to generalize the full solution (which would give rise to an algorithm), the expert develops partial programs and local methods, which reduce much of the tentative searching of the novice. An example of such expertise in the game of chess are the opening moves. A beginner, unaware of the special properties of the starting board configuration, is obliged to examine all the available moves to the limit of his capacity. An expert however, with a better understanding of the opening board configuration (and some painfully learned lessons), already possesses compiled (programmed) sequences of moves so that he can select a predetermined program to suit the situation. Any less than optimal response from his opponent assures him of an advantage.

Conventional productions which restrict the RHS to elementary actions, thereby ensuring that the transformation modifies the state minimally, are totally unsuitable for encoding decisions that propel the problem far towards a solution. But this is precisely the kind of "power action" expected from an expert

Limitations of conventional production systems are not confined to the format of productions. For expert systems to be viable, the language in which they are coded, must be effective at representing the knowledge of experts, and efficient at integrating incongruous sources of information and using them.

In the next sections these requirements are examined in more detail and some of the solutions are discussed. The link with conventional production systems must not however be lost. The word blackboard which comes from Selfridge's metaphor of demons (productions) observing the development of the solution (STM) on a Global Blackboard and reacting spontaneously to contribute to the solution [SELF59], emphasizes the tradition of psychological models. These architectures must be viewed as an enlargement of techniques of earlier models to accommodate an evolution in the understanding of knowledge and its use.

1.6.2. Representing knowledge

The literature on psychological models contains very little in the way of representation of knowledge aside from productions. In fact Newell suggests that LTM, the repository of all internal human knowledge is organized exclusively of productions. We know however, that the bulk of human knowledge involves the description, classification and relationships between objects that are used in reasoning. Objects are described in terms of their multiple properties. These objects are often organized into hierarchies for a variety of reasons:

1) The hierarchy provides a compact and economical representation, and accelerated retrieval.

2) Reasoning about elements at different levels of a hierarchy provides an economy in processing, not only by sparing the processors an overwhelming (and redundant) volume of data, but by permitting qualitatively different reasoning (which might not be apparent in a flat database).

3) Hierarchies also facilitate recognition by identifying classification of otherwise unrecognized objects.

In addition to hierarchies, developments in knowledge representation include other kinds of associative representation schemes such as semantic networks which express an arbitrary relationship between objects or classes of objects [BRAC78] These techniques which allow the structure of domain objects to be represented, can be made to symbolize the compiled knowledge embodied in expertise. Another considerable advantage of representation schemes that are flexible enough to reflect the structure existing in the domain, is that they are easier to encode, the mapping to machine symbols being more straight forward and "natural".

It is important to note that the knowledge described above does not correspond to dynamic working memory (or STM). This knowledge is the data component of LTM which represents state independent knowledge of the domain Transient or state information (solution elements) resulting from reasoning about the problem is related to this permanent data through special or user defined relations. Thus in a diagnostic system for instance, hypotheses generated by the reasoning process would correspond to a known (catalogued) diagnostic classification, with known properties, symptomatology etc. The catalogued counterpart is used to assist the reasoning process As a result of further reasoning the hypothesized data could later be removed or modified, without affecting the reference information compiled as part of the system's knowledge.

A specification of blackboard language must provide at least some predefined relations that map items of working memory to the static knowledge base. The properties associated with these relations must be semantically meaningful in the domain of application.

1.6.3. Representing problem solving actions

It is to be expected that during the course of the solution of a problem, an expert will encounter a region in the search space where possibly through recognition, analysis, better quality knowledge or through a theory, the solver is in a position to prescribe with greater commitment an extensive program of actions without incurring the overhead of interrupting evaluations in the middle of the process. An example of such a situation occuring in a game of chess was given earlier. A problem solving system, which is to achieve expert performance, must provide a vehicle for encoding such behaviors. The grain size of an operation must reflect the problem solving knowledge that is being incorporated in the system. On the other hand a decision to proceed with a program without the interruption of state evaluations, is a strong commitment to a particular line of reasoning. Such a commitment can only be made judiciously upon a very thorough evaluations of the state at which it is being made. Thorough evaluations of state may require more than simple pattern matching or predicate evaluation. It is conceivable that small programs may be executed as part of the analysis of state, prior to an informed decision. The distance between RHS and LHS of a knowledge source for an expert system must therefore reflect the certainty of success of the knowledge it embodies.

Operators generate, modify, and delete solution elements. The language used to define operators (independently of their implementation) must be developed with structure of state memory (and consequently the permanent database) in mind. If the data has a hierarchical structure, operators used in reasoning must have interfaces across levels in this hierarchy. For instance a robot movement planning system may reason about the movement of blocks; lower level operators and planning processes must reason about the actual vector movements of the robot's arms and wheels around obstacles, to perform the tasks already reasoned at the blocks and objects level. How such lower level plans affect the higher ones, is a problem for the control system, but the data structure representing the operators must provide the necessary pointers and links for the other modules in the system.

A number of properties and characteristics of operators may be dependent on the domain. For instance a diagnostic system might associate confidence factors with the reasoning of its operators [SHOR76]. A planning system on the other hand, which controls production in a machine tool factory must reason about resources[1] that are utilized when an operator is applied. With so many application dependant requirements, it is virtually impossible to attempt a general formulation that would suit all applications. Instead the effort should focus on improving and developing one particular aspect of the problem, while at the same time incorporating the features that have been tried and tested, and have passed into general use. As techniques are developed they can be expanded and generalized; this task is made easier by the adoption of high level description languages which are widely used in the field, and which provide high level support and a common specification syntax. For this project, KnowledgeCraft©, which

[1] Reasoning about resources is discussed to some extent in chapters 2 and 5

provides utilities such as a schema representation language and a number of inference engines within a LISP programming environment, was used.

1.6.4. Sources of information to support control decisions

The control problem involves the selection of a path in the search space that connects the initial state with a goal state. This path may either be specified in terms of successive states or in terms of successive operators/bindings, depending on the nature of the application; both are equivalent since one is deducible from the other The generation of the path is an incremental process, which is dependent on evaluations during the course of processing. This approach is a consequence of the absence of a general theory and therefore of powerful problem solving methods. The effectiveness of control can be measured by the extent of the exploration of the search space lying outside of the path. In order to maximize its performance, a control system must exploit the compiled knowledge it is supplied with, as well as the run time generated data that is available. Sources of knowledge available to control and to support the decision making process include:

1) The structural description of data objects that belong to the domain.

2) Static state evaluation functions. These heuristics when they are available, are based on approximations which give a measure of proximity of a state to the goal Use of these functions presupposes that they change monotonically with the real distance, a characteristic which cannot always be guaranteed, and that they discriminate effectively between states.

3) Canned heuristics which select the best operator to proceed with, given a state or a partial path.

4) Dynamically generated heuristics resulting from analysis of portions of the search space that have already been explored. Such an analysis compares the transformations brought about by particular operator/binding combinations and extrapolates predictions on the current state [DUR&LES87]

5) Introspective analysis of the operators available to the system and the ways in which they can be combined and the ways they interact

6) Analysis of the possible bindings of operators and the constraints that can be reasonably applied so as to restrict the range of accessible states within the search space

A good architecture must support the description of all these properties, and must provide mechanisms for reasoning about control decisions, and integrating all these sources of knowledge

1.6.5. Organization of the modern blackboard

The concept of blackboard architectures has evolved from a generalization of the features of a number of innovative expert system environments, which were developed in the late seventies and eighties. These systems were designed to tackle particularly hard problems, for which exhaustive searches were completely intractable. While most of these systems were originally designed to operate in specific domains, it became obvious that the general framework could be suitable for the solution of much broader classes of problems. A number of such systems were recast in a manner that allowed the separation of domain dependent knowledge from the general problem solving framework [illegible]

Domain knowledge is encoded in independent modules called knowledge sources. This knowledge is used to generate or modify data which is part of the general solution of the problem. Knowledge sources are independent of one another and therefore cannot invoke one another directly. Communication between knowledge sources is through windows onto the database containing the solution. The content of a knowledge source is unconstrained; it can consist of one or more procedures, one or more productions or a number of logical assertions which modify the database. Knowledge sources contain at least one condition part, describing situations in the database, where their actions can contribute to the solution

The database or "blackboard" is globally available, for reading and writing, to all knowledge sources. Generally each knowledge source will tend to read and write or modify data, in specific regions of the blackboard. The data which builds up, consists of elements of the solution, which are organized into a structure reflecting properties of the domain. A hierarchical organization for blackboards is the most common, because we tend to organize our own knowledge into taxonomies; the structure adopted however, should reflect the dimensional properties of the problem, and the links representing relations between data objects should be organized into any form that is meaningful in the domain.

Although the knowledge sources are self-invoking (through their condition component), limited processing resources require that an external control system arbitrate among competitors and allocate resources. For the solution to be found in a computationally cost effective manner, the control system must select the most appropriate knowledge source as well as its calling parameters. To achieve the best choice, the control system must examine those regions of the search space (blackboard), to which the knowledge sources, given the available parameters, will modify or write to, and select the one most

most likely to encompass a solution The selected region of the search space in which the control system attempts to find solution elements is called the focus of attention The degree to which the control system can integrate all the available knowledge to make judicious decisions about the focus of attention, determines the effectiveness of an architecture Figure 1 1 depicts the organization of a blackboard

1.6.6. Behavioral characteristics of control

A blackboard system in the process of solving a problem should be able to sustain a line of reasoning while monitoring events and data generated as a side effect When the data justifies it, the system should be capable of re evaluating current decisions or goals. In other words, the stability of its behavior should not make it insensitive to incidental information which would call for changes in its approach In the process of pursuing a task, the system should be capable of exploiting pertinent information gained in another For a data driven system, this behavior requires that a certain discipline be imposed on the system The line of reasoning cannot be dictated by the content of the data used Since most problems generate data which is model based rather than descriptive productions must be able to deal with run time generated conflicting data The system should become aware when multiple instantiations cooperate toward the same conclusion or goal As long as these are genuinely different instantiations (not just a set of similar productions expressing differing levels of specificity) [illegible] strong knowledge in favour of some particular state and would warrant more processing resources.

1.6.7. Characteristics of problems suitable for processing on blackboard architectures

While the blackboard model offers a much more powerful environment for incremental problem solving than does the pure production formalism, the computational overhead involved in assessing and reasoning about the control problem makes these architectures much less interesting unless the magnitude (complexity) of the problem justifies it, and unless there exists some knowledge about the control problem itself.

The relaxation of constraints on knowledge source format, which is an advantage where problem solving knowledge is inhomogeneous across the range of the search space, makes it hard on the system to read its own productions As a result the developer must write interfaces for the control system to communicate with knowledge sources and with the scheduler. The burden of communication between the systems modules is likely to add to the computational overhead of the system. It is therefore a clumsy environment to use where simple productions express adequately the available problem solving knowledge.

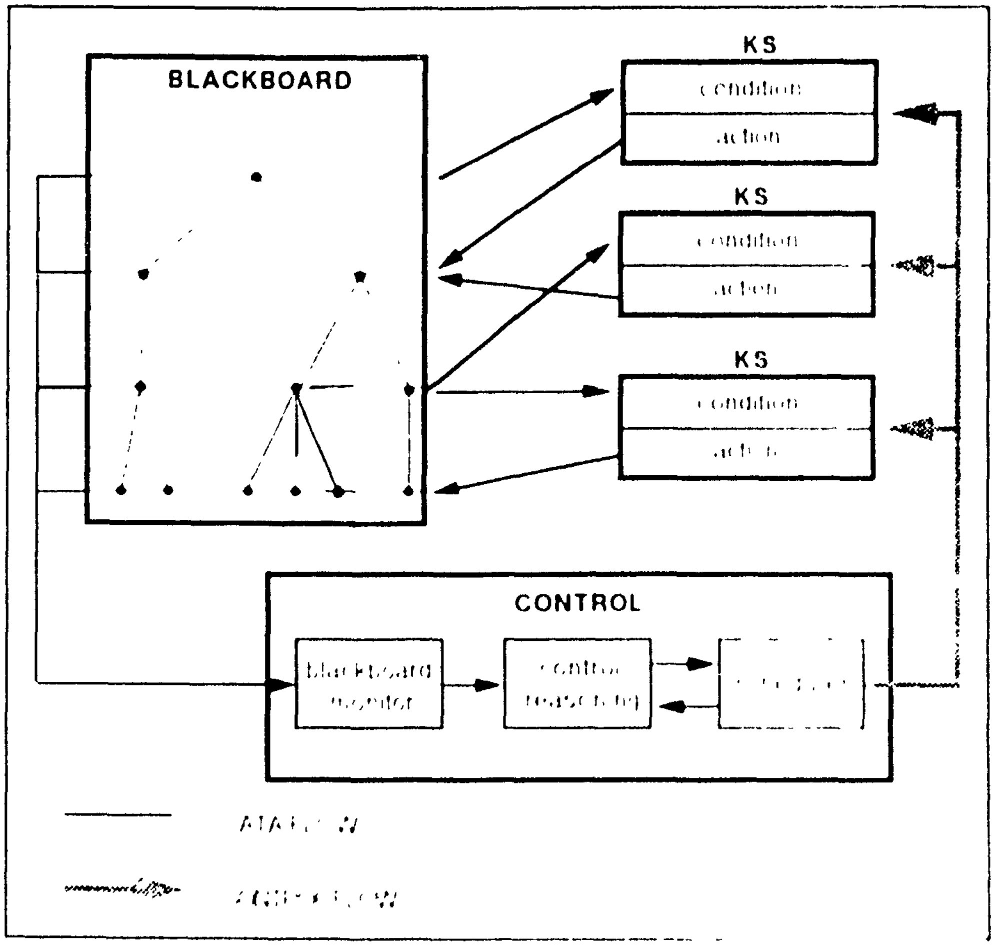


Figure 1.1 General Organization of Blackboard Architecture

The irregular format of knowledge sources also makes [illegible] complicated than the mechanism of simply tracing the reasoning [illegible] and systems that require extensive justification will require [illegible] read the blackboard and contain [illegible] knowledge [illegible] generate explanations.

Finally a blackboard [illegible] knowledge [illegible]

[illegible]

[illegible]

understanding, and as a result, production systems are more easily updated and can serve as tutoring mechanisms.

2. Chapter two

2.1. Development of major blackboard systems and languages

2.1.1. The OPS system

The best known example of a production system language is probably OPS. The original language has spawned a number of variants (OPS4, OPS5, OPS83) and different implementations have been incorporated in a number of expert system development tools (ART, Knowledgecraft). Developed by Forgy and McDermott [illegible] it differs from the other systems described in later sections in that it was intended from the start as a domain independent language. OPS is on the boundary between pure production systems and rudimentary blackboards. It was originally conceived for the Instructable Production System (IPS) [illegible], a psychological model which attempted to emulate human behavior in problem solving as well as learning.

In accordance with Newell's model of human reasoning, knowledge stored in permanent LTM consists exclusively of productions; the results of the reasoning process are stored in a data working memory of modest size. To comply with the characteristics of human performance the software was designed so that performance of the system would not degrade with added productions [illegible] was checked. The architecture of the system can be described in terms of three components: Production Memory, Data Memory, and the Interpreter.

2.1.1.1. Data Memory

[illegible]

constituent slots (attributes). During the reasoning process "instances" of these objects are generated, modified and deleted in dynamic working memory. Schemas in OPS are the exact counterpart of records or structures in conventional programming languages. All the values of the attributes characterizing the instance must be constants; no variables are allowed, neither are pointers to other objects. A time Tag is attached to each instance, identifying the moment of creation or most recent modification. At a grain size beyond the schema, working memory is completely flat and structureless. Object typing also provides a convenient basis for indexing working memory.

2.1.1.2. Production Memory

As the repository of all permanent knowledge, productions must represent small and re-usable increments of problem solving knowledge. The "distance" between LHS and RHS is intended to be small. LHS patterns consist of object class specifications and constraints on the values of attributes. To enable the productions to be general (since they must be shared for multiple problems), variables may be used in the patterns. No complex tests or programs are allowed on the LHS. The implementation involves an interesting pattern matching algorithm, the RETE network, which is based on Petri Nets. Tests on objects which are shared in multiple productions are performed only once. The entire body of LHS tests, for all the productions, is compiled into a network. Changes in working memory which affect these tests, cause tokens to be propagated through the network. There is a test node called a One Input Node for each different test occuring in the productions. The cumulative effect of conditions is aggregated pairwise through Two Input Nodes. For each production there is a terminal Two Input Node. Any (positive) token arriving at the terminal node signals a *instantiation* or combination of working elements satisfying that production. This means that the system can efficiently evaluate the *entire working memory* in terms of *all productions* at every inference cycle. Only changes to working memory involve any processing at all, and the impact of this change is evaluated once for all productions. RHS actions are limited to the creation and deletion of instances of objects, and modification of their attributes values.

2.1.1.3. The interpreter

The interpreter is responsible for coordinating and guiding the inference process. At the start of every cycle, changes to working memory trigger tests at the appropriate One Input Nodes. Indexing speeds this process up considerably. Tokens of the appropriate sign are propagated through the network. Negative tokens remove instantiations from the previous cycle which are no longer valid while positive tokens add new instantiations.

At the end of the Match phase the *conflict set* contains the set of instantiations that is valid for the current state of working memory. Since only one instantiation is acted upon per inference cycle the interpreter must perform the selection or *conflict resolution*. The original versions of OPS did not provide for knowledge based control. Instead the user is provided with two very similar conflict resolution algorithms, consisting of a three stage filtering process:

In the first stage all previously selected instantiations are removed from consideration.

If the conflict set still contains more than one instantiation [illegible] attributed to each one. [illegible] tags associated with the instantiation's data elements [illegible]

If the size of the conflict set is still greater than one, the set of instantiation with the highest *specificity* is retained. Specificity is measured by the number of tests or constraints imposed on the data by the LHS of the production.

If the size of the conflict set is still greater than one, then an instantiation is selected arbitrarily.

The conflict resolution strategy is strongly influenced by the findings of Newell. The first filter, called *refraction* , by analogy with the refractory period of neurons which inhibits triggering for a finite period after firing, ensures that cycles are avoided. The second filter favours the selection of instantiations involving the most recently processed data. Newell had found that STM behaved like a short stack, with older data decaying (unless refreshed) in favour of new data. The third filter specificity, ensures that the most pertinent (least general) production is applied thus carrying the solution further ahead towards the goal.

This basic conflict resolution algorithm is called Lexical Selection The user is given an alternative strategy MEA (for Means End Analysis), which is minor variant of LEX. In the second filter (recency), extra emphasis is placed on the recency of the first condition element. The programmer can select clusters of rules to be enabled together by means of control elements in the first condition of each production. The name Means End Analysis is misleading, as it bears no real resemblance to the reasoning method.

The strength of OPS as a general purp e production language for knowledge intensive problem solving, resides in its efficient pattern matching algorithm, which allows global evaluation of the state of the solution at every cycle The simplicity of the syntax must also contribute to its popularity.

Although it may be ideally suited to the purpose for which it was originally designed (psychological models), OPS suffers from some serious limitations as an environment for knowledge intensive expert systems

2.1.1.4. Expressivity

OPS provides no other vehicle for describing the domain, than basic schemas and productions The rules themselves have a very constrained format not allowing for LHS tests other than comparisons on attribute values. The RHS actions are intentionally limited to ensure fine grain If the knowledge of the domain involves some extensive processing of data (as is often the case), the user must resort to programming artifices to enforce the invocation of rules by one another [DAV&KIN77] These techniques are contrary to the philosophy of the language and result in total loss of modularity and virtually incomprehensible code

2.1.1.5. Control

Psychological models require no control and OPS does not support any. Difficult problems however generate huge search spaces, and problem solvers should go periodically through a phase of introspection to re-evaluate their strategies in the light of the space already explored and their knowledge of the structure of the domain [illegible] terms of the interpretive cycle of OPS this would require an interruption (perhaps periodically) after all instantiations are found and [illegible] processing to decide future strategies

2.1.1.6. Performance

[illegible]

crowding of the workspace. The removal of unpromising elements from memory can result in visited nodes being regenerated unless more programming artifices are employed. With an enlarged workspace, the indexing mechanism becomes less efficient and evaluation at every cycle becomes more demanding. The resulting performance drop can often make the problem unsolvable. In fact the great advantage of globally evaluating the state of the solution which can result in intelligent control is completely lost, but the computational burden is retained. For small problems which do not generate many memory elements and where an exhaustive search is not prohibitive, the computational efficiency of OPS and the simplicity of its use can make it a very convenient tool.

2.1.2. The HEARSAY II Speech Understanding System

The Hearsay II system [HAY-ROT&LES&RED80] like Mycin [SHOR76] was first developed around a specific application and was later generalized with some major enhancements into a domain independent problem solver, Hearsay III [ERM&LON&FICK81]

Hearsay II was developed as a speech understanding system. It was designed as a spoken natural language interface to a database retrieval system. Because of the complexity of the problem, a blackboard architecture was adopted to reduce the search effort.

The search space involves interpretation at various levels of abstraction (signal, phoneme, word, sentence etc...). Knowledge of the domain allows some combinatorial constraints to be applied to these hypotheses. Despite these constraints the magnitude of the problem and performance requirements make processing decisions necessary. The system must identify the most promising hypotheses and the processing options that are available, in order to achieve a solution within acceptable time limits

2.1.2.1. Data memory

Working memory which contains partial solutions is structured hierarchically, each level representing an interpretation of data at a lower level. A phoneme hypothesis for instance is an interpretation of a succession (in time) of signal segments, a syllable an interpretation of a succession of phonemes etc. Because of uncertainty in the interpretive task a number of alternative hypotheses might be generated for each succession of lower level data elements. The cumulative result on the blackboard (working memory) is an AND/OR graph. Conjunctive arcs being formed by successive lower level hypotheses and one of their interpretations and disjunctions between alternative (competing hypotheses)

2.1.2.2. Knowledge Sources

The knowledge sources perform all the processing (interpretation) as well as supplying information to the control mechanism about the pertinence of their activation. Each knowledge source contains two separately executable parts: the precondition and the action

The precondition is more complex than the LHS of productions. It consists itself of two components: a *stimulus frame* and a *response frame*. The stimulus frame responds to changes occurring on the blackboard which provide the necessary input data for the knowledge source to operate. The response frame posts a message to control about the transformation of the input data to be expected from the knowledge source

The action part of the knowledge source is the program that generates and modifies and processes the domain data on the blackboard. Figure 2 1 shows the stratification of the blackboard and the action of some of the Hearsay II knowledge sources.

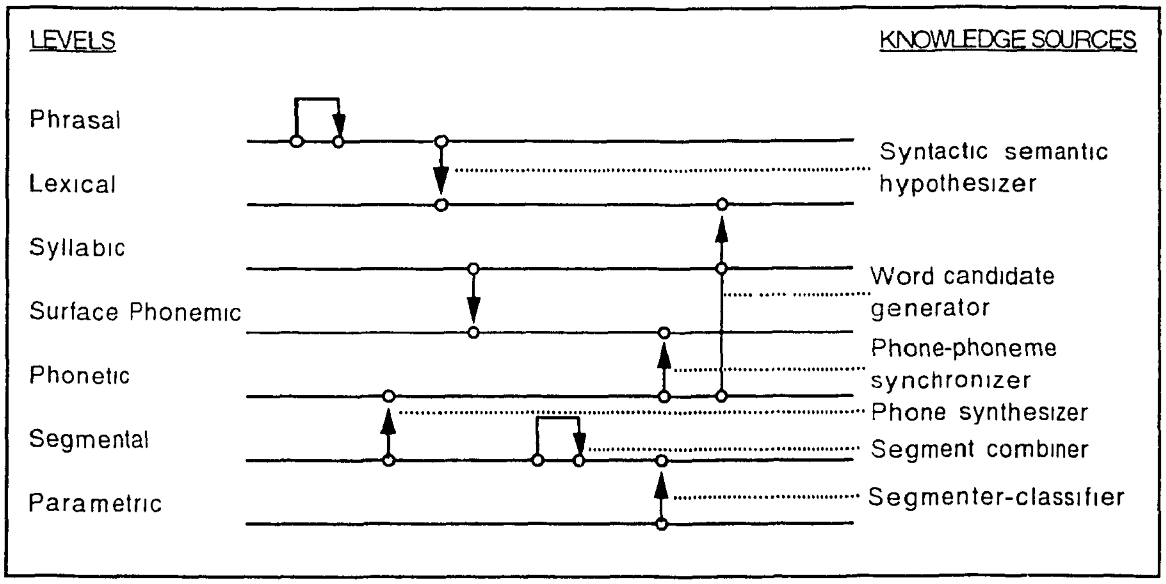


Figure 2.1 - Hierarchical Organization of Blackboard in Hearsay II System

2.1.2.3. The control system

Whenever changes occur on the blackboard (working memory) a *blackboard monitor* detects the changes and identifies knowledge sources that are capable of contributing to the solution, using the new data. The blackboard monitor informs the *scheduler* about newly executable preconditions which are added to the *activities* queue. The activities queue not only contains the current executable preconditions but also pending executable *action parts* from previous cycles.

If the scheduler selects an action part from the activities queue, the blackboard will be modified. If on the other hand the scheduler selects a precondition for execution, the stimulus frame binds the appropriate hypotheses and the response frame sends a message to the scheduler informing it of the type of processing the knowledge source is expected to perform.

The scheduler will use this information to select future activities A cycle during which a precondition is selected for execution does not lead to domain data being processed but is devoted to generating data for the control (scheduling) mechanism. The scheduler evaluates data from response frames using empirical heuristics which have been embedded as procedures The scheduler takes into account the reliability of the processing activities as well as the resources they consume The scheduler reads a database (the focus of control database) which contains information on the resource requirements of knowledge sources given their projected activities, in order to compute priorities Figure 2 2 shows the architectural organization of Hearsay II

Hearsay II is regarded as the first modern blackboard, incorporating most of what is now regarded as essential features of this class of architecture The Blackboard (domain database) is structured to emphasize properties and relationships of the problem space Knowledge source format is unrestricted to facilitate full expression of problem solving programs Control is not bound by a domain independent scheduling algorithm, it exploits heuristics of the application and adapts its strategies to an evolving solution The control system is designed to cope with the high level of interaction between Knowledge sources, that can be expected when reasoning uncertainty is high, and dynamic working memory is expected to grow considerably

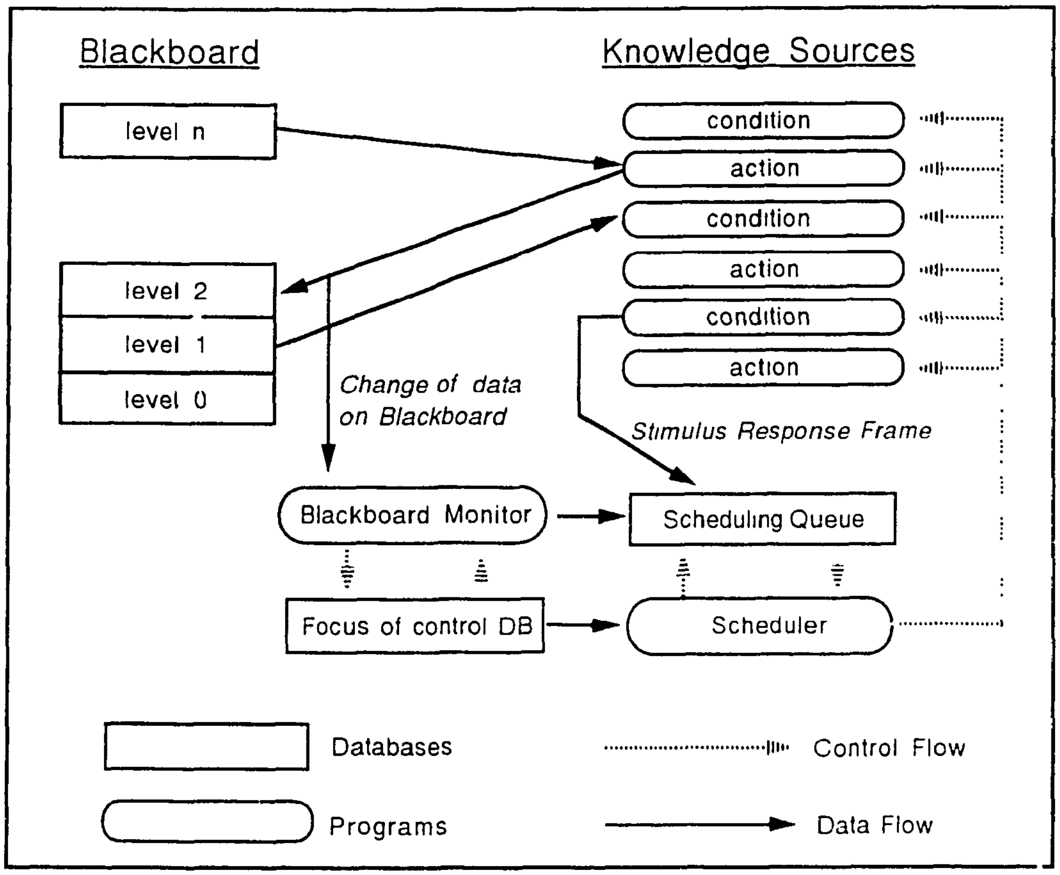


Figure 2.2 - Control Structure in Hearsay II

2.1.3. BB1 (OPM)

The BB1 blackboard [HAY-ROT85] is a direct development of the Hearsay II architecture The innovation in this system is that it addresses the control problem with as much emphasis as it does the domain. The Hearsay II architecture devoted certain cycles to supplying the scheduler with data on potential knowledge source activities. The scheduler itself decided dynamically whether to select a control processing or a domain processing cycle according to fixed heuristics.

In the BB1 system the control problem is solved in parallel with the domain Data pertinent to the control problem is kept on a structured control blackboard and is updated by knowledge sources exclusively devoted to control Domain data is kept on a separate (structured) domain blackboard The interpreter selects either domain or control knowledge sources for execution by using the same evaluation criteria for both classes of activities

The idea of using opportunistic (rule based) programming to solve the control problem was not new. Davis [DAVI80] states

" *We suggest attacking this problem (control) with a general inference mechanism one which will allow the program itself to reason about and deduce the appropriate refinement to use This reasoning is also done at the time that refinement is required and hence is dynamic and responsive to the current state of the problem This appears to be more powerful and flexible than attempting to anticipate the problem a priori and building in fixed strategies*

The novelty of BB1 is that it tackles the control problem with the full apparatus of the blackboard architecture and describes a mechanism for integrating the [illegible] of the problem Another interesting feature of BB1 is that it [illegible] *planning* albeit [illegible], for the control process

2.1.3.1. Data organization

Data memory [illegible]

solution elements [illegible]

[illegible]

illustrated [illegible]

[illegible]

[illegible]

[illegible]

model obscure the description of the system; the interest of BB1 however, is in the control organization which is independent of the application.

At the highest level of the control blackboard, the *problem* level, there exists a control data element created at initialization, instructing the system to solve the problem as a whole. At the next lower level, the *strategy* level, the control data element stipulates an approach to solving the problem in terms of a long-range heuristic, which will select at run time, successive regions of the domain problem space. The next level down is called the *focus* level. Data items at the focus level implement the *current* phase of the strategy. The relationship between a focus element representing the current phase of the processing and its parent strategy node is explicitly represented by a link, as is the relationship between a strategy element and the problem it solves. All control elements contain a description of the condition under which they spontaneously become inactive (the *criterion*). There are in addition, three extraneous levels on the control blackboard that do not appear to belong to any sort of hierarchy, the *policy* level and the *chosen action* level: these will be described along with the interpreter

2.1.3.2. The knowledge sources

Each knowledge source is described by a schema The condition component contains two parts, a *trigger* and a *precondition.* The trigger is a simpler (and computationally less expensive) test which identifies blackboard situations or *events* which might make the knowledge source applicable, while the precondition holds the full set of LHS tests necessary for the input data A slot is devoted to the precondition's variables and another contains the programs performing the knowledge source s actions BB1's knowledge source schema also contains six scheduling variables *trigger weight, action blackboard, action level, knowledge source efficiency knowledge* *source credibility, knowledge source importance* The values of these attributes are used by the scheduler for control decisions. A domain knowledge source uses domain data as its inputs and makes modifications to the domain blackboard, a control knowledge source on the other hand makes modifications to control data elements, but may use data inputs from either blackboards. There are some additional slots describing knowledge sources such as the level on the blackboard at which modifications are made and their blackboard (control or domain) upon which they operate.

2.1.3.3. The interpreter

The interpreter cycle is quite complicated.

1) A blackboard monitor identifies changes on both blackboards which would satisfy the trigger condition of knowledge sources. For each triggered knowledge source a data structure called Knowledge Source Activation Record or KSAR is created.

2) For each KSAR a scheduling rating is computed. The rating is calculated in terms of focus level *control data elements*. Each focus control data element specifies regions of the problem space and contributes a rating component to a KSAR representing a knowledge source whose output is within that region. The algorithm requires that every KSAR be checked against all focus data elements since the ranges specified by the latter may overlap.

3) KSARs corresponding to knowledge sources which [illegible] have their ratings weighted according to [illegible] control data elements [illegible] KSAR [illegible]

dynamically by control knowledge sources; focus decisions change frequently while policy decisions are longer term There might be several focus decisions on the control blackboard at any time, but there can only be one active policy.

4) The KSAR corresponding to the knowledge source with the highest overall priority is selected for execution. Since KSARs are created the same way for control as well as for domain knowledge sources, modifications to both blackboards will occur. Changes to the control blackboard affect the scneduling of subsequent KSARs so control is capable of modifying its behavior.

The Hearsay II system was the first to devote processing cycles to dynamic evaluations of scheduling priorities; Davis's meta-rules introduced the idea of production driven control heuristics; BB1 however pushes dynamic programming much further by completely separating the control problem from the domain. Using the same mechanism for scheduling control and domain knowledge sources, the system should in principle, detect situations where the search problem is getting out of hand, and it should schedule control knowledge sources that modify long or short term control heuristics depending on the scope of the degenerating search. Another novelty of BB1, a by-product of using a structured blackboard for the control problem, is that the system gives the illusion of planning a control strategy. A succession of focus decisions will favour knowledge sources that work in different regions of the problem space. A control knowledge source implementing a particular problem solving strategy which monitors the deletion of a focus decision and generates a new one, is a program that embodies (perhaps weak) knowledge about control flow. It does not constitute true planning which requires laying explicitly partial or complete problem solving steps and reasoning about their interactions prior to their execution. Another serious limitation is that the system confines control to the selection of knowledge sources. The focus of attention should specify explicitly regions of the search space in terms of all the dimensions of the problem. In practice this is equivalent to specifying constraints on values of the input (or output) parameters of knowledge sources. The entire treatment of the dimensions of the problem space is unclear, inconsistent (at least in the publication) and is the reason for much of the opacity of the presentation.

On a practical level, the integration of the control and domain reasoning mechanisms may cause synchronization problems. If the problem solving process runs into trouble, control knowledge sources must be scheduled to change the heuristics, but the focus decisions that are active still bias ratings in favour of the domain knowledge sources creating the problem. It is not obvious that tuning such a system with focus weights can correct this difficulty throughout the problem solving process.

2.2. Planning systems

Planning systems are a subset of General Problem Solvers that generate a partial [illegible] specification [illegible] instructions [illegible] particular task or goal [[illegible] 88].

The generation of instruct[illegible] transformations [illegible]

The [illegible]

The original planning systems based on the STRIPS representation assumed a near perfect model of the world. Communications for such systems are almost exclusively from planner to effector since any completed plan is assumed to succeed without modification. The planner in these systems, requires very little feedback from the executor to complete its plans since the state of world after each planned action is known completely. These systems were intended to generate instructions to a robot executing tasks.

Because STRIPS like systems assume a total knowledge of the world, despite being equipped with a very rudimentary knowledge representation formalism (FOP logic), the worlds they describe are confined to simplified "*Blocks World* " type environments. Even the simplest extension to these models (to make them of greater practical use), involve an enormous computational expense [CHAP87].

2.2.1. Motivation for planning:

There can be a number of reasons for adopting the planning paradigm in problem solving. When the cost of modeling actions is lower than that incurred by the effector tentative problem solving can use planning with advantage. Travel trajectories and robotic control are examples of this class of problem.

Under certain circumstances, the effector might not be able to backtrack at all. In the construction industry for instance, tentative solutions are developed on paper prior to execution.

When the effector also happens to be a computer program as in the case of the blackboard problem solver, the motivation for planning resides in a reduction of task complexity from interleaving planning steps with execution

Appendix 2 contains a detailed review of planning systems and illustrates some techniques that have been adapted to plan control strategies in the blackboard

3. Chapter three

3.1. Saturation

3.1.1. Knowledge source saturation

Complexity results from a lack of perception of the structural properties of the problem. One of the consequences is a reccurse to weak methods (such as statistical techniques), which are unable to discriminate effectively between solutions, another is the inability to select effectively the correct problem solving action to adopt at every stage.

Determinism occurs when only one procedure is considered for invocation at any given time [DAVI80]. The production environment, better geared for situations where knowledge is much weaker, promotes non-determinism, in other words the availability of alternative procedures or multiple chunks of problem solving knowledge, to process data at any one point. This programming style is encouraged to facilitate the expression of all that is known about the domain, no matter how fragmentary or incoherent. Uncertainty about the appropriate action to take, thus compensated, is one reason for the tentative approach to the solution. The retrieval of large numbers of plausible procedures, to process a particular item of data, places a considerable burden on finite resources. Exhaustive application of these knowledge sources not only overwhelms the CPU, but also generates a large amount of incorrect or inappropriate data, which will further aggravate the problem at subsequent cycles. At some point a discriminatory mechanism must be applied.

3.1.2. Data saturation

The second source of saturation stems directly from the weak models used in the knowledge sources. The weaker the model the more uncertainty in the result it produces (since it is unable to fully exploit constraint properties). Uncertainty in the results manifests itself in multiple alternative hypotheses which are often accompanied with weights, probabilities, certainty factors etc. Computationally these alternatives can be an even greater source of saturation than the imprecise retrieval of knowledge sources. Reliance on certainty factors alone to select data is insufficient, because they are usually computed on the basis of local knowledge, while the problem requires the integration of elements into an overall solution [DUR&LESS87]. Redundancy in knowledge sources can help, evaluating the same hypothesis from different angles, using different input data and knowledge sources certainly gives a better basis for discrimination. The computational cost effectiveness of such (control) decisions, depends on the branching factor and location of that element with respect to the overall solution. For reasoned decisions regarding processing this information must be available *before* it is required. Conventional architectures that incrementally build up solutions, do not possess this information at the point in time when it is needed.

Within a fragmentary problem solving environment, in which reasoning is performed without a global view of the problem solving process, and in which the quality of the processes that might be invoked is poor, a correct estimate of the computational cost of reasoning with a large set of data versus the effort involved in reducing it can improve the quality of processing decisions and consequently efficiency of the search. Heuristics required to make these estimates, are not however always available.

3.2. Control solutions to saturation

3.2.1. Alleviating knowledge source saturation

Knowledge source application can be viewed as a three stage process. The first stage involves the retrieval of the appropriate KS, the second is the refinement of the selection and the third is application of the final set [DAVI80]. The vast majority of blackboard architectures resolve the conflicts of knowledge source saturation through scheduling. They differentiate themselves in the approach and reasoning used to achieve refinement. From the descriptions given in chapter two, we can see that HEARSAY performs refinement through the execution of precondition programs, which supply data to the system on the activities of knowledge sources, as a function of the available input. The BB1 system selects its knowledge sources by favouring the ones which correspond to the current focus elements. The reasoning of BB1 is more complex, because the focus elements are not generated from local consideration, but as a result of a broader strategy.

Other schemes for refinement include meta-rules which reason about the content of domain rules and contribute scheduling parameters. Meta-rules are organized into hierarchical levels of (meta) reasoning. The interpreter performs a preliminary retrieval of object (domain) rules to form the initial conflict set, then rules, which reason about the conflict set, are retrieved to form a second order conflict set. The process iterates invoking higher level (meta) knowledge sources to resolve conflicts one level below. Higher level knowledge sources represent strategies rather than direct control over the problem solving process. The conflict set shrinks at higher levels, as strategic knowledge is scarcer and of greater scope than local control knowledge.

Procedural attachments attempt to resolve the problem of saturation by ensuring that retrieval remains very specific. Prototypic objects of the problem space are described and the pertinent knowledge sources are associated with them. Instances of the data objects that are generated in the course of the solution invoke the procedures associated with the prototypes. This approach to knowledge representation attempts a level of indexing which is sufficiently focused, to make refinement unnecessary. By making refinement redundant, the object oriented approach is undoubtedly computionally more efficient. Processing however, is not the only consideration, the vehicle for representing knowledge must be well suited to the domain Object oriented programming is probably very well suited to certain domains, in particular those for which it is easy to specify prototypic templates and associate with each a limited number of problem solving procedures To develop a manageable database, which enumerates and describes all prototypes with a grain size fine enough to ensure focused procedural attachments, requires a substantial appreciation of the structure of the problem For domains in which the required level of structure is not apparent, this style of programming is likely to prove difficult and will become cumbersome during processing

3.2.2. Alleviating data saturation

Meta rules, which reason about the content of rules and object oriented programing, resolve to some extent problems associated with knowledge source saturation, they do not however, resolve problems associated with data saturation Although as a secondary effect, more judicious application of knowledge sources, rather than an exhaustive approach, reduces the amount of inappropriate data competing for processing, it does not directly remove the bottleneck caused by large numbers of solution elements

There does not appear to be very much research directed to this problem in general problem solvers The OPS system arbitrated about data on the basis of recency The motivation, however, was the emulation of psychological models rather than efficiency at problem solving It is interesting to note that recency in OPS's conflict resolution strategy was dominant over specificity which placed discrimination of data over and above discrimination of knowledge sources. The HEARSAY II system introduced hierarchical organization of the blackboard which exploits the domain structure to reduce saturation. For an application such as speech interpretation and understanding in which weak knowledge sources generate numerous, competing hypotheses, this was not sufficient, and the HEARSAY II system was given knowledge sources such as WORD-SEQ-CTL and WORD-CTL which imposed an upper limit on the number of competing hypotheses used as input for further processing[1] [LES&ERM&RED74] The BB1 system does not appear to have any mechanism for selecting data All combinations of inputs satisfying the preconditions of the selected knowledge source are processed

Oddly enough, while designers of blackboard systems have been preoccupied with knowledge source selection at the expense of data selection, designers of planning systems appear concerned with the reverse Apart from hierarchical organization, constraint posting, as described in the MOLGEN system [STEF81], appears to be the most powerful method used to cope with the explosion of data generated by weak operators

The technique involves reasoning with objects partially described by variables Allowing partial descriptions has the effect of grouping under one item all instantiations which correspond to alternative binding values. As the reasoning proceeds and the interaction among these generalized objects becomes known, constraints discovered can be applied progressively The application of constraints on variable bindings has the effect, as was described in the previous chapter, of excluding regions of the search space. When constraints are discovered their effects are propagated throughout the hierarchy, potentially excluding more regions from further consideration, and triggering other

[1]The paper by L. Erman does not indicate on what basis the number of hypothesis to be processed is determined. This appears to be decided at run time and is probably a function of the current processing bottleneck and the probability density distribution of the hypotheses.

reasoning mechanisms. The mechanism of posting constraints models the interactions between objects, whether these have been instantiated or not.

Deferred constraints mean that the problem solver can continue the reasoning process to discover other properties of the solution which might allow further refinement, without committing itself on decisions it is not yet in a position to make Arbitrary and uninformed commitment is one of the principal reasons of backtracking Not only do retractions result in wasteful computations, but very often by backtracking, a system will lose elements of the reasoning process which are in part correct.

3.3. Objectives of blackboard design

3.3.1. Representational methods

The difficulty of formulating knowledge about ill-structured problems has been emphasized throughout the first chapter The difficulty of finding a suitable formalism is compounded by the fact that the quality of knowledge available is not consistent in all regions of the problem space. Rather than opting for a particular encoding scheme and applying it universally, it is preferable to select the most suitable knowledge representation method for each aspect of the problem.

The knowledge source format should be as free as possible, allowing arbitrary level of selectivity in the LHS (through patterns and programs to evaluate applicability). The RHS should allow simple modifications to working memory, as well as more complex programs

Constraint based reasoning relies on an explicit representation of dependencies Such dependencies are bound with classes of objects. Solutions to problems solved on blackboards must represent a minimum of structure since the database of partial solutions is hierarchical. In practice, it is possible to express a much greater level of structure and dependency with many applications. Since this structure a priori is expressed in terms of instances, classes and prototype objects, the languages of object oriented programming are ideally suited for that kind of representation The effects of constraint posting which are reasoned rigorously can be propagated along arbitrarily defined relations and links, connecting the different classes of solution elements. As partial or true instances of these classes, these elements inherit the procedures of the prototypes which are invoked automatically by demons, or by messages.

3.3.2. Behavioral objectives

Given that a problem solver's architecture supports the description of all that is known about the domain without loss of the structural information which is so essential to efficient processing, it is necessary that the system make maximal use of the available knowledge, so as to ensure an efficient search.

Some of the factors which compound the difficulty include:

The fragmentary nature of the information represented by independent knowledge sources, which causes a sequencing problem. Permutation of the order of invocation of knowledge sources in the conflict set, when subproblems interact, does not result in equivalent problem solving situations. For instance, the timely invocation of a particular knowledge source which rules out a whole section of the search space can save much useless computation by its competitors.

Diversity in the methods of representation, which are necessary to accommodate qualitatively different kinds of knowledge, complicate the integration of information By contrast, architectures which offer only one method of knowledge representation, such as production system languages, will have a uniform data format. The linkage between knowledge modules for such systems, is only dependent upon the content of data, rather than upon its form.

Redundancy in knowledge, it was argued in the first chapter, is one of the major aids to overcoming uncertainty. In terms of knowledge sources, having several alternative ways to generate the same solution elements helps the system verify the validity of a hypothesis. In terms of data, a path that has been generated, whether it is the correct one or not, together with information about the effect of knowledge sources upon it, can sometimes be used predictively to assess the possibilities in another region of the search space. Redundancy in processing knowledge and unnecessary data elements are responsible for the saturation problems, discussed in the previous section

The increasingly complex methods of control that are used to counter the problems listed above, further aggravate the processing bottle-neck

An incremental improvement in problem solving efficacy, can be brought about by incorporating certain desirable characteristics in the design of a general problem solver's architecture. For this project the emphasis has been placed on the following

3.3.2.1. Self-awareness of knowledge, and capacity to apply the most appropriate action

The cellular format of knowledge sources is designed to facilitate the expression of all available piecemeal information about the problem A consequence of this format, is that the interpreter plays a very passive role The system does not know what knowledge sources cooperate an what knowledge sources compete in the generation of a solution The system merely responds to the bids of individual knowledge sources The problem is further compounded by the lack of communication between knowledge sources. The difficulties resulting from the lack of communication were recognized by Davis and Smith [DAV&SMI81], and were resolved through the development of a protocol of communication between independent knowledge sources. Any procedure requiring a sub-task broadcasts a message through a universal channel. All competent knowledge sources can submit a bid which informs the caller of the extent to which it can solve the problem, as well as the resource cost it will incur. The caller is then responsible for evaluating cost effectiveness and invocation The approach devised further decentralizes control and deprives the interpreter of any capacity to influence the course of the solution The problem solving environment for which this paradigm was developed, is very different from that of a conventional blackboard The system was proposed for a distributed problem solver with multiple processors connected by a network and multiple sensors supplying asynchronous inputs For software architectures designed to run on a single processor, a central control system, which is aware of all the available knowledge and in a position to assess the available resources, is better placed to make appropriate decisions regarding procedure calls

Many recent systems, including some of the ones described in chapter two, encode control know-how within knowledge sources, and apply the same non-deterministic invocation methods that are used for domain programs. Control knowledge sources thus compete with those of the domain for invocation. Not only does this organization introduce considerable overhead in evaluation and processing, but non-determinism tends to make invocation of control knowledge sources less likely, just when there is the greatest need for them, as the conflict set becomes large Another drawback of basing control exclusively on knowledge sources is that all information about object knowledge sources has to be duplicated in the control. This information must, for practical reasons, be abridged and is also subject to errors. The consequent loss of information results in decreased effectiveness

In order to devise a method for expressing the "self-knowledge" that the system should possess, the purpose of each procedure must be stated unequivocally Since a solution most generally expressed, is a set of elements connected by transformations linking a goal state with the initial conditions, control can, to a certain extent, be exercised by specifying more or less narrow regions, in which processing is directed to generate solution elements Even within the narrowest constraints, there might be alternative methods of generating the same solution elements A secondary level of control must then decide, on the basis of cost effectiveness, the most appropriate method of generating hypotheses By implementing primary control through the specification of constraints on the output of knowledge sources, the self-knowledge required of the system takes the form of relationships that different parameter bounds have on one another The methods used for computing these bounds can be embedded within the objects representing solution element classes

3.3.2.2. Planning longer programs of actions to achieve objectives

Problem solving in large search spaces cannot be based on local evaluations, as regions of high certainty and credibility are not necessarily compatible and cannot be integrated into a path A problem solver cannot be a purely reactive system which shifts its efforts from goal to goal and let the tyranny of changing priority ratings redirect its attention from place to place. It must actively pursue a line of reasoning while assessing the consequences of new results on that line of reasoning Planning, which is a (possibly partial) specification of entire paths through the search space, carries the problem solver through those regions of low credibility without distracting its focus of attention It therefore provides the *stability* which was identified in the first chapter as an essential feature of problem solving

3.3.2.3. Monitoring the evolution of a solution and relevant plans.

As plans are executed some of them will very likely fail at some point or other. The effect of a plan's failure and the solutions which depended on its success must be directly available to the problem solver, if it is to avoid unnecessary computational effort in working on partial solutions that can no longer be integrated into a path. More generally, when a plan is formulated, the classes of solutions that it supports must be specified and as the plan is refined, those solutions that no longer fall within the scope of the plan must be excluded. This organization ensures sufficient *sensitivity* to redirect problem solving when it becomes necessary, without sacrificing the stability needed to traverse regions of low certainty

3.3.2.4. Estimating the effort required to achieve plans.

It is likely that a problem solver will have to consider many alternative plans, representing alternative and competing specifications for solution paths. It is part of the function of the control system to evaluate the comparative merits of each. A system should support heuristics that estimate the number of solution elements that may be integrated within each region specified by a plan and heuristics that estimate the cost of achieving these solutions. As a plan is executed, deviations from expectations might shift the advantage to another. The system should be able to monitor execution and revise its strategies without wasting all of the effort spent on its plans

3.3.2.5. Knowledge to reason about processing constraints

While all architectures aim at efficiency in processing, rarely are constraints about computational resources specifically mentioned and used in the reasoning. Generally a system is expected to produce a solution as fast as possible There are many applications particularly those requiring real time processing for which the time to produce a solution is critical Under some circumstances a trade-off between quality or perhaps the number of solutions and the time required to generate them has to be made Such considerations affect processing decisions made by the control system The architecture should provide mechanisms for reasoning about such parameters

3.3.2.6. Cost-effective control overhead

Changes occurring in the search space are incremental Developments must be reassessed in terms of existing control structures rather than invoking global re evaluations and assessments at every cycle

3.4. The Role of planning in control

3.4.1. Problems associated with implementing control exclusively through scheduling

Conventional blackboard architectures resolve conflicts through heuristic scheduling The more sophisticated among them dynamically adjust heuristics ordering functions to reflect current estimates of the next best action to perform

Functions, meta-rules, programs or whatever mechanism is used to evaluate states, must contribute discriminating weights to instantiations or operators that might possibly be scheduled in the current search space front

The preconditions of meta-rules contain references to partial conditions of domain rules that are currently scheduled, as well as references to states upon which they operate Their action parts are weights to be attributed to the corresponding scheduling elements Each meta-rule therefore focuses on one or a group of scheduling elements (be they instantiations or operators). The final ordering of the scheduling queue, which represents control reasoning at that point in the solution process, is the cumulative result of multiple meta-rules. The meta-knowledge base programmer is given a coding format, which is designed to focus on conditions specified by the LHS, independently of the rest of the database. The actions performed are not absolute ratings, but have a relative effect which is meant to differentiate the scheduling elements. Thus, the inclusion of a new meta-rule in the knowledge base requires a comparative assessment of its actions with respect to those of every other meta-rule. This however is precisely what meta-rules were supposed to avoid. In the paper introducing the concept of meta-rules [DAVI80], Davis states "*.. adding the nth. KS (knowledge source) to a system should not require comparing it in detail with all the other n-1 KS*s".

A second drawback of using a meta-rule based control is that the formalism does not easily allow path dependent information to be expressed. Problems often have constraints dependent upon extended path histories. These constraints can be very valuable in reducing complexity, particularly as the solution progresses and the search front widens

Although meta-rules are used to control and therefore to impose constraints on the search, their formats directly express the constraints that they apply in terms of state parameters. Bearing in mind that the complexity of the problem is a direct consequence of unperceived structure, failure to express unambiguously and to apply what little is known about the problem, can only aggravate complexity and obscure the problem solving process.

Static evaluation functions generally attempt, through simplification of the problem, to make estimates of nearness of an intermediate state to the goal. When the space to be traversed is large, it is very unlikely that a single function can make such approximations with any level of accuracy or consistency, this was one of the major reasons for the abandonment of earlier weak methods, such as Hill Climbing and Best First. With multiple evaluation functions, each operating within a specified region of the search space, discrimination is more likely to be accurate. There are however, two problems which make this approach less than satisfactory. A good heuristic function is not easy to find, and requiring many of them, for the majority of tasks, is an impossible demand. The reason for this is that heuristics require analysis of the problem whereas human expertise or competence is acquired through a process of generalization.

Systems which contain multiple heuristics introduce another difficulty While an individual heuristic may discriminate effectively within its area of competence, the problem solver must arbitrate over the whole search front. The control system has no yardstick for comparing the estimates of different heuristics and therefore cannot make reliable processing decisions.

Encoding scheduling knowledge within one or a small number of programs, which is in effect what BB1 does, picks the worst of all worlds. A program is a very tightly constrained method to solve a problem. If this amount of control knowledge is available, a blackboard is not the correct environment in which to find the solution Writing separate control programs does not make the task very much easier when knowledge is weak, even if this knowledge is scattered over a small number of weakly linked procedures. Applying this control knowledge through scheduling might make the system more tolerant of errors, (in contrast with control flow of conventional programs), but it does not reduce complexity as effectively as direct intervention in the application of processes.

3.4.2. Integrating the constraints of the problem space and exploiting them to reduce combinatorial complexity

A problem solver given a description, of domain dependant constraints through object knowledge sources, must have an overriding mechanism for identifying occurrences of constraint parameters and imposing them on elements extending solutions paths. Such constraint posting mechanisms must not compete opportunistically with knowledge sources which extend solutions, since omission of their application potentially allows a combinatorial explosion It should be noted that knowledge of constraints, is object (domain) knowledge although it is used to effect control and not to create solution elements.

To propagate the effect of a newly discovered constraint well ahead of the current search space front, the system places constraints on values of attributes of solution elements still to be created, and which will lie in the same solution path as the source of the constraint. Loss of a piece in a chess game, for instance, can immediately impose constraints on the types of end-games left open to the player, and can thus reduce much of the uncertainty over the choice of moves in the intermediate stages of the game

Expressing such constraints on future elements subsumes knowledge of a class to which these elements belong. This knowledge is always available since it is implicit in the knowledge sources, and is built into the organization of the database (blackboard).

3.4.3. Planning and execution with broad objectives gives more robust plans

The idea of propagating constraints forward down the solution paths, creating partially specified, but uninstantiated objects, which belong to broader classes defined within the system, provides the elements necessary to a control plan.

Planning systems described in chapter two, operated in a world were actions were nearly certainly predictable. In the extreme case of predictability, the problem is solved once the plan is generated. In a world where the result of actions are not so certain, planning ahead of execution can be advantageous where the plan can be generated much faster using simulations of actions, and where the planner is capable of repairing a plan to adapt it to the effective outcome, while monitoring execution. Other situations where planning a program of actions is advantageous include systems where backtracking is undesirable or impossible (eg. controlling a robot system)

For a blackboard whose workings are totally internal, the execution of a plan involves the invocation of the object knowledge sources to generate solution elements. The role of planning is a little more subtle, it is used to provide the problem solver with some foretaste of what lies ahead in the solution path corresponding to the actions that are currently being considered A plan, therefore, partially specifies an overall solution, spanning an entire path from initial condition to goal state. Partial specification of the solution takes the form of local constraints on successive elements in the path.

By reading its available plans, the control system is able to identify those regions of the search space (not yet explored), that are compatible with every existing solution element. Conversely it is able to exclude with certainty considerable regions of the search space when pursuing a particular path.

The view of the search space afforded by plans is global. The system can decide to pursue several intersecting plans simultaneously and monitor the progress of several alternative solutions. This organization opens the ways for a number of domain independent heuristics.

3.4.4. Planning and execution with broad objectives gives more robust plans

Plan elements define regions of the search space by specifying bounds on parameters. The plans themselves consist of linear paths connecting these regions in such a way that individual operators (knowledge sources) can generate elements in one region using data from another. There may be alternative paths spanning a solution and some elements within the plan might be redundant, since alternative knowledge sources can overlap and span different regions of the search space (see figure 2.1 illustrating knowledge sources/solution elements of the HEARSAY II system).

The generation of plan elements themselves affects the behavior of the system. If individual elements are very tightly constrained, there will probably be more alternatives and the chances of failure of a plan at execution time will be greater. At the extreme, if plan elements can only encompass one solution element, the task of generating plans is identical to the task of searching for a solution; it demands as much computational effort and the plan is as likely to fail as any individual path of an exhaustive search. The one advantage of working with plan elements (that of carrying all accumulated constraints down the path), is a result of the method of representation, and could be incorporated in the domain knowledge sources with some modification of format. Conversely, if plan elements specify very broad constraints (each specifies broad regions of the search space), the number of alternative elements will be much smaller. The chances of finding a set of compatible elements (one from each region), under the tighter constraints of the plan execution phase, will be much greater. The drawback, however, is that coarser fragmentation of the search space can result in poorer discrimination and more local search within each plan element.

Advantageous control lies in the correct equilibrium between the two. The planning effort must not overwhelm the processing resources, nor replace the primary task of the problem solver which is the application of the full set of constraints of the problem to identify the correct solution elements.

The problem solver must have, in the second detailed phase of the search for a solution, an exhaustive set of plans which spans all the dimensions of the problem space such that no solution element belonging to a true a path lies outside a plan element. In practical terms, this means that during the planning phase, the generation of plan elements must be exhaustive. Since a detailed exhaustive search at the level of granularity of one or even a few solution elements is ruled out, the generation of solution elements must be sufficiently coarse to be feasible. The method of partitioning the search space into planning elements should not be an arbitrary exercise, but should be determined by the dimensions of the search space, the direction of the solution paths and the constraint knowledge available.

In passing, it should be noted that overly fine plan elements are not only prohibitively expensive to generate, but are more likely to fail at execution time and require backtracking.

3.5. Representing domain data to facilitate control

3.5.1. Representing the dimensions of the problem space

Viewing problem solving, at least partially, as a constraint propagation task, in which the system identifies sequences of regions in the search space possibly containing mutually compatible solution elements, which together integrate into a solution, suggests a certain clarity and homogeneity in the representation of the problem. It is desirable that all dimensions of the problem space (including the levels of abstraction), be represented as parameters in all solution eler its; the goal states would then be specified as constraints on a number of parameters. Unfortunately this is not always straightforward nor even possible. Often techniques incorporated in a problem solver originate from different theories, and the methods of analysis are inconsistent In fact, a case often made in favour of blackboard architectures is that is allows such diverse and disparate techniques to be integrated into the problem solving machine The speech application used for this implementation is a case in point.

This system interprets an acoustic signal which has been segmented and labeled externally[2] Processing involves the generation of elements at the phonetic, lexical, phrasal and sentential levels. The lowest three levels in the abstraction hierarchy (segmental, phonetic and lexical) we.e the result of a particular development, and the integration of words into phrases and sentences is completed by a conventional parser. The vocabulary of parsing and its techniques for analysis and classification are totally different from those of signal processing. The dimensions of a problem space, illustrated in figure 3 2, while adequate for signal processing are not adequate for the paısing problem. Signal segments, phonemes, and words hypotheses can be specified by a label, a start and end time. A parser on the other hand requires different sets of attributes to describe a word. Table 3.1 shows some the attributes required for parsing using an ATN grammar:

[2] Details of the domain and techniques for processing are given in the next chapter.

WORD:	WORD:	PHRASE:	SENTENCE:
LABEL· *noun*	LABEL: *verb*	LABEL	LABEL.
START-TIME·	START-TIME:	START-TIME.	START-TIME
END-TIME·	END-TIME·	END-TIME:	END-TIME·
CATEGORY	CATEGORY:	CATEGORY ***NP***	SUBJECT
NUMBER	FORM·	DETERMINER	DIRECT-OBJECT
CASE.	TRANSITIVITY	HEAD	INDIRECT-OBJECT
TYPE	DESCRIBERS	MAIN-VERB.	
QUALIFIERS	AUXILLIARIES·		
NUMBER	MODIFIERS:		
PERSON.	QUESTION-ELEMENT:		
QUESTION.	VOICE:		
MOOD			

Table 3.1 - Attribute descriptions of parsing entities

Even within one level of abstraction the dimensions can vary It is not particularly meaningful to qualify the "form" of a noun.

Another source of difficulty is that implicit properties of objects connected by arcs at different levels of abstraction might change. For example, a lexical hypothesis (word) is related to its phonetic supports through a relationship of *constituents*, and a phonetic hypothesis to its segmental supports through a relationship of *interpretation*. At the phrasal level of abstraction this breaks down completely, because the constituents of a phrase need not only be at the word level only, but can be an arbitrary mix of words or other phrases.

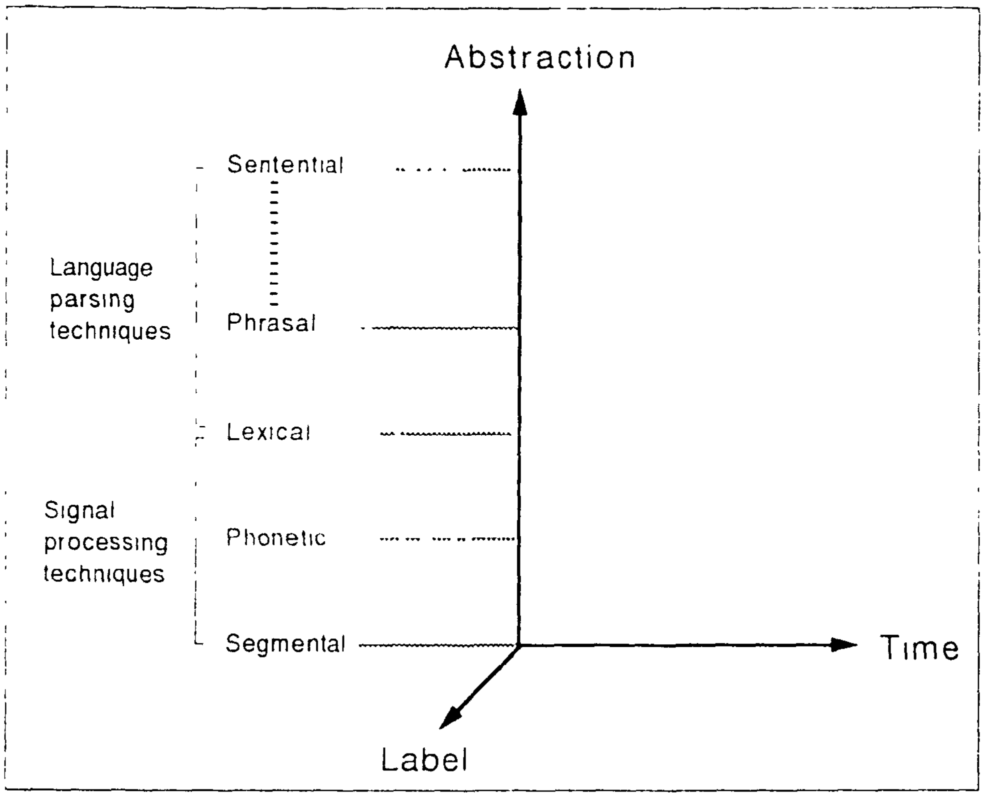


Figure 3.2 - Fundamental dimentions of search space for Speech Recognition problem

It would therefore be misleading to think of the search space as being defined by a global set of problem coordinates; even the notion of a blackboard hierarchically organized along levels of abstraction may not fit every application. Instead, the search space should be defined in terms of prototypic solution elements, each containing the appropriate attributes with domain dependant semantic links connecting them.

Knowledge sources are mapping functions which generate or modify instances of domain prototypes during the solution phase of problem solving. Inputs as well as outputs of knowledge sources are domain elements. The output elements must comply with the constraints of the currently selected plan element. Figure 3.3 illustrates the relationship between prototypes, solution elements and the corresponding plan element. In addition to the relations shown, domain elements typically contain "links" generated by the knowledge sources, which relate the input domain element to the output domain elements of the appropriate knowledge source. The semantics of the links are totally dependant on the knowledge sources and may not be the same across different levels of abstraction

Plan elements are generated during the first phase of problem solving The programs that partition the search space can be stored as procedural attachments[3] . Each plan element is a specification of constraints on one or more attributes of possible domain elements Plan elements designate distinct regions of the search space, and no domain elements can satisfy the constraints of two plan elements When the task is defined, the system contains the initial data and a statement of the goal. The latter is a specification of constraints on values of domain element attributes. Domain elements which satisfy these constraints are identified as goal states. The main goal is therefore, an initializing planning element, just as the given data constitutes the initializing domain elements.

[3] In the implementation plan, elements are not generated by procedural attachments but by planing knowledge sources. This approach has some drawbacks which are discussed in the chapter detailing the implementation.

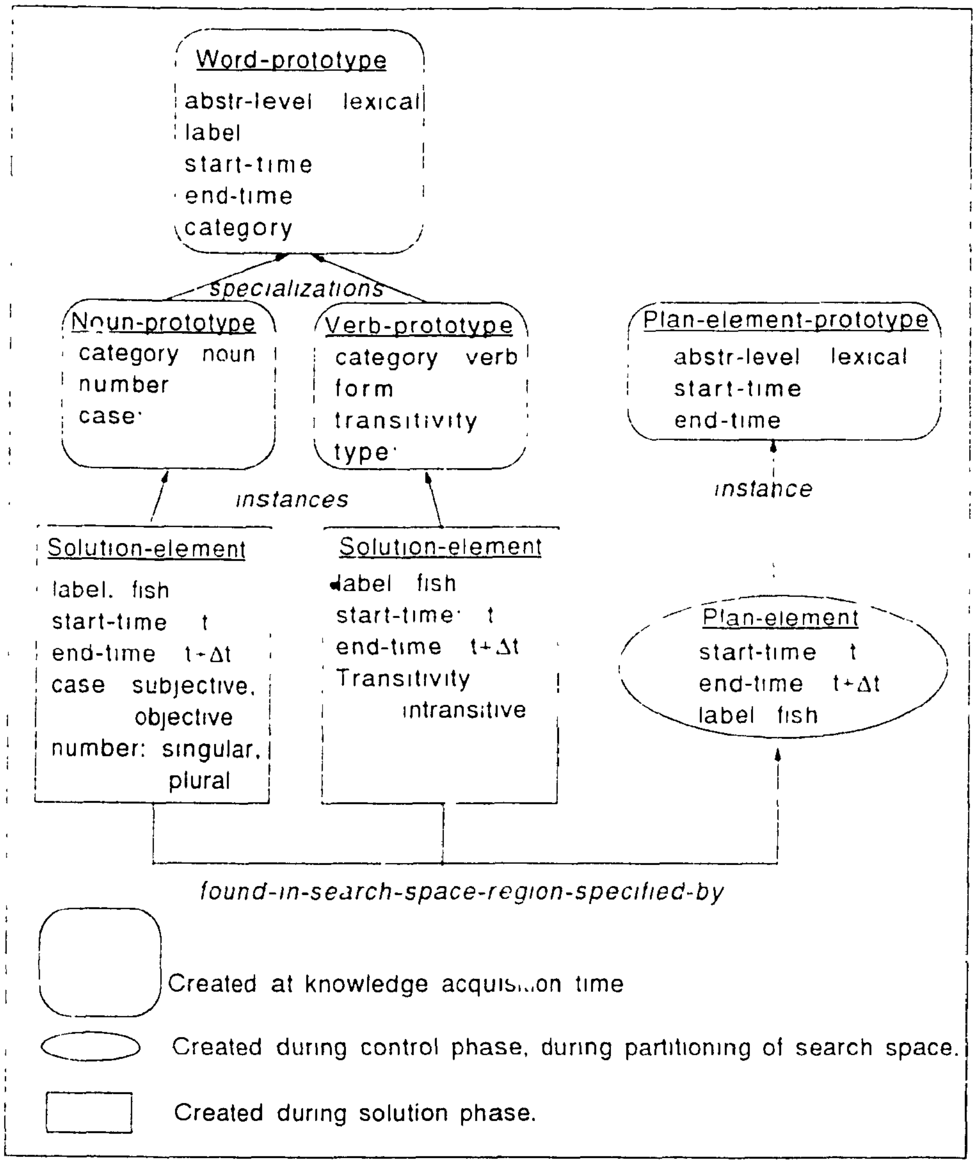


Figure 3.3 - Relationship between Plan Elements and Solution elements

3.5.2. A rigid representational format to impose a problem solving style

Attempting the design of a domain independent problem solver is a juggling act between generality necessary for versatility, and representational rigidity necessary for the application of a problem solving method To achieve any measure of success, it is necessary to separate form from content.

The representational language which provides the form, must be rich enough to facilitate expression An attempt however, to enlarge on form makes automatic interpretation and processing of language structures too complex and subject to ambiguity The design of such systems, therefore, requires that a concise format be provided for the specification of all aspects of problem solving knowledge utilized by the system Such formats should be accompanied by a description of the kinds of knowledge for which they are intended. A rigid format may not be well suited to represent every kind of information, and in fact this would not be considered a desirable property. A formalism which is tailored for a very specific type of information will assure its proper use

The type of information is related to the function it serves in the problem solving process. If the system can assume that certain data structures contain a particular type of information and the format is invariant, it can use the knowledge without concern for its content. The extent to which the formalism will selectively accommodate knowledge with a specific function determines the extent to which the system can automatically make efficient use of it, and consequently, the extent to which the system is effective as a domain independent problem solver.

3.5.3. Extracting control knowledge from domain descriptions

Two classes of domain dependent knowledge, the object knowledge sources and the search space partitioning procedures, are intimately related Each member of the latter corresponds to a member of the former set. The function of partitioning knowledge sources is to create a number of plan elements specifying non-overlapping regions of the problem space. This process must be performed exhaustively to encompass every possible solution element and it must be completed in one pass; there is no provision for backtracking to modify or delete previously made elements

If the control cycle is to deliver a computational advantage, then exhaustive coverage can only be achieved with coarse granularity. This means that attribute values are specified with broad constraints, or more frequently, plan elements will only specify values for some of the attributes, allowing other parameters to assume any possible value.

The constraint on backtracking implies that knowledge sources responsible for partitioning cannot reason about secondary interactions that invalidate partial paths For example, a plan element which integrates adjacent determiner and noun plan elements to form a noun-phrase are unable to reason about conditions on matching number attributes. This is because, in generating the input plan elements (which encompass words), the category start and end times are retained but not the label, and consequently not the word dimensions (number and case for nouns, form and transitivity for verbs etc). The resulting space defined by the plan element encompasses all the word hypotheses located in the same time interval, and which belong to the same lexical category. The loss of information, which is propagated along the solution path, makes it impossible to detect interactions that might otherwise have been found in later stages of reasoning. The coarse grained partitioning resulting from elements with unspecified attribute values is also under constrained, because missing information relaxes conditions which would rule out paths.

With procedures that generate plan elements, low resolution and under constrained versions of object knowledge sources, the user has a guide for producing this code Closer examination of partition schemes for different types of problems and experimentation with relaxation of different constraints might yield heuristics for automating this process

3.5.4. Using constraint bounds to partition the search space and to represent dependencies

When plan elements are created, the input data elements to the procedures are linked to the new output elements. The partitions are therefore linked with one another in a network of dependencies Since every domain knowledge source which creates new elements (though not those which through applications of post-constraints remove partial solutions), is matched by a corresponding partitioning knowledge source, any link between true domain solutions will find its equivalent in the links connecting the corresponding partitions. By the same token, if two partitions are not linked, the solutions they encompass cannot be integrated into a path. The converse is not necessarily true, since the partition links are established under relaxed constraints.

The initializing plan element or main goal, as was stated in the previous section, is the region containing all acceptable solutions. Any plan elements not included in a path to the main goal can therefore be discarded. Since plan elements have been arrived at with little computational effort, this provides a means of eliminating potentially large areas of the search space from further (detailed) consideration by domain knowledge sources.

Since the database contains only the initial data and the goal plan element before the start of processing, the partitioning problem must involve connecting the given data to the goal partition As we are dealing with only one goal element during this partition phase, the search space topography is clearly a "fan-in" and the preferred direction of reasoning is forward (bottom-up). Processing is exhaustive, so the system is not encumbered with any control decisions

3.6. Representing control plans

The role of the planner in a blackboard architecture is to reason about the effect of actions (knowledge sources) on solution elements, to analyze their interactions and to improve the quality of processing decisions The planner is therefore in charge of control, while the effector actually invokes the knowledge sources, performs the changes on the blackboard (database), and supplies additional or corrected information to proceed with control reasoning.

3.6.1. Planning control strategies to select the focus of attention

The object of a plan is to provide the system with an insight into what lies ahead in the solution before it commits itself to the detailed task of problem solving. Through planned sequences of problem solving actions, the control is system able to evaluate cumulative effects and interactions, and to better select operants. For planning to represent an economy over actual knowledge source invocation, the planner must reason about the effects of knowledge sources without incurring the cost of invoking them.

In the first chapter it was mentioned that in order to be efficient at problem solving, blackboards architectures, must possess a capacity for "introspection", or a capability to base problem solving decisions and actions, upon an assessment and where possible, an analysis of the progress being made. This assessment can be based on heuristic evaluations of state as well as on deeper knowledge of the procedures available to the problem solver

In the speech signal recognition task, the reasoning task is performed along two dimensions, the levels of integration, which include the abstraction hierarchy, and along the time axis. Each plan involves integrating non competing solution elements along the time axis lying at the same level of abstraction. A domain constraint requires that elements of plan be time adjacent; no overlapping or gaps are allowed.

Each element represents a region in space; for this particular application, the abstraction level, the start and the end time attribute values are fixed for all plan elements at levels. Other attributes may or may not have constraints applied to them. There are therefore potentially as many plans as there are possible legal combinations of plan elements at each level, which will span the duration specified in the goal (the initializing plan element).

Executing a step in a plan involves directing the control system to generate solution elements within the bounds specified by the plan element. The selected plan element being worked upon is the current focus of attention. A plan may fail either because the system is unable to generate solution elements within the plan, or because none of the solution elements that have been generated will integrate with existing solution elements under the tighter constraints of domain knowledge sources.

3.6.2. Representing hierarchies of plans which are mutually exclusive

A plan for the speech application, as was mentioned in the previous section, is composed of regions of the search space, specified by plan elements which lie on the same level of abstraction, and which span a portion of the signal (denoted by the time axis), without overlap or gaps At the highest level, the plan contains only one region (that specified by the goal). At lower levels these regions specify shorter time intervals, since the mechanism of solving this problem involves integrating shorter interpretational entities into larger ones. The problem is solved if a solution element is found at the highest level plan, as this element will necessarily satisfy the constraints specified by the goal. Initially the solution elements are only available at the lowest levels in the hierarchy. Plans will therefore have to be developed and executed at lower levels.

Despite the coarse grain of plan elements, at the lowest levels in the abstraction hierarchy, the number of distinct plans increases as does complexity. Detailing and reasoning about all plans can become quite a formidable task. To maximize efficiency the system should perform all the analysis that is necessary to ensure that the *next decision* is made as correctly as possible, given all that is currently known about the problem. The system should not perform work for plan steps scheduled in the future, which might never be executed if the plan is aborted. The planning process is therefore refined incrementally. A plan however, even at the very first stage of refinement, spans the space necessary to achieve its goal.

It is crucial for a particular planning task to identify the information needed for the decision at hand, to achieve a representation that makes this information readily available without incurring the cost of unnecessary computation, and to make use of information which has already been computed in previous cycles and remains unchanged.

As no two plans contain the same set of regions and no solution element may lie in two regions, each plan will generate a different overall solution. Competing plans therefore generate mutually exclusive solutions.

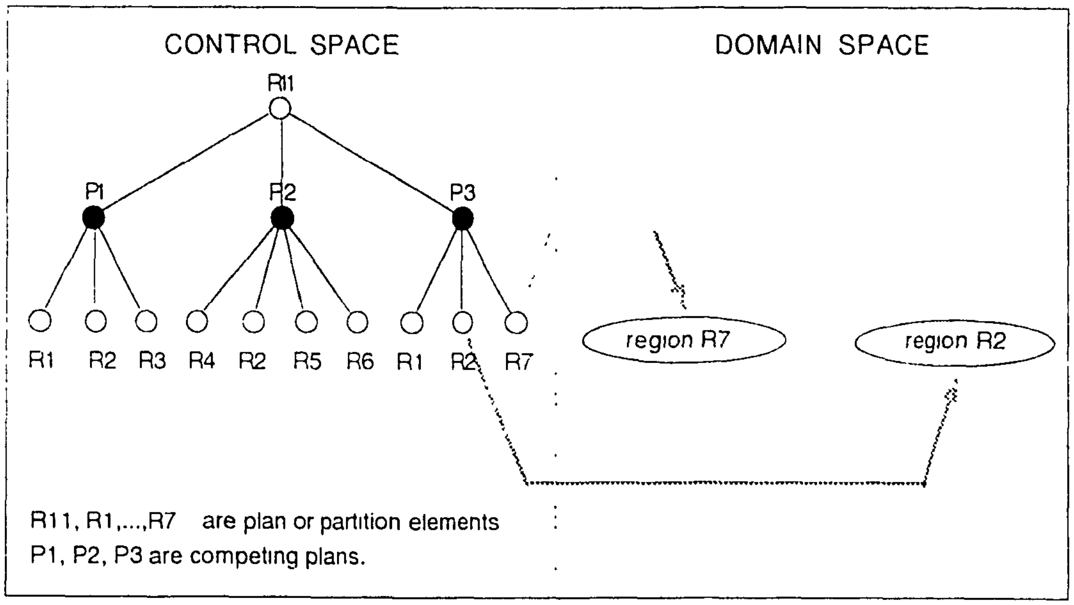

CONTROL SPACE
DOMAIN SPACE
R11
P1
P2
P3
R1
R2
R3
R4
R2
R5
R6
R1
R2
R7
region R7
region R2
R11, R1,...,R7 are plan or partition elements
P1, P2, P3 are competing plans.

Figure 3.6 - Relationship betwen Control Space and Domain Space

Figure 3.6 shows the relation between plan elements and the domain search space. Plans P1, P2, P3 compete to generate disjoint sets of solution elements, in the region of domain space corresponding to plan element R11.

3.6.3. Planning on the basis of least commitment

For each plan element which does not encompass enough domain solutions, a plan spanning the dimensions of the region is created. In the speech application, generation of solution elements is an interpretation of its constituents one level below in the hierarchy (phonemes in terms of signal segments, words in terms of phonemes, phrases in terms of simpler phrases and words, and finally sentences in terms of phrases). A plan is created because the knowledge source in charge of integrating constituents has failed to meet its target goal of solution elements. The failure can only be due to an insufficient number of suitable constituent elements being available as inputs. The objective of the plan is to identify the regions encompassing every constituent, and to ensure that enough elements are created in each. As the control system is likely to have many alternative plans to choose from, each potentially generating completely different solutions, the system must first select a plan. A policy of *least commitment* is applied. As plans will often intersect, the control system can "hedge its bets" by electing to process first those elements shared by the largest number of plans. Success then assures that the problem solver retains the largest number of options Failure, on the other hand, rules out all regions of the intersecting plans and eliminates a larger portion of the search space, thereby reducing complexity.

In the previous section it was stated that the control system does not attempt to generate all the possible plans from the beginning. Instead, it defines plans only as far as it needs them to make immediate decisions. There are therefore as many plans elaborated to develop solution for a "parent" region further up in the solution process, as the system has decisions to make at any one step.

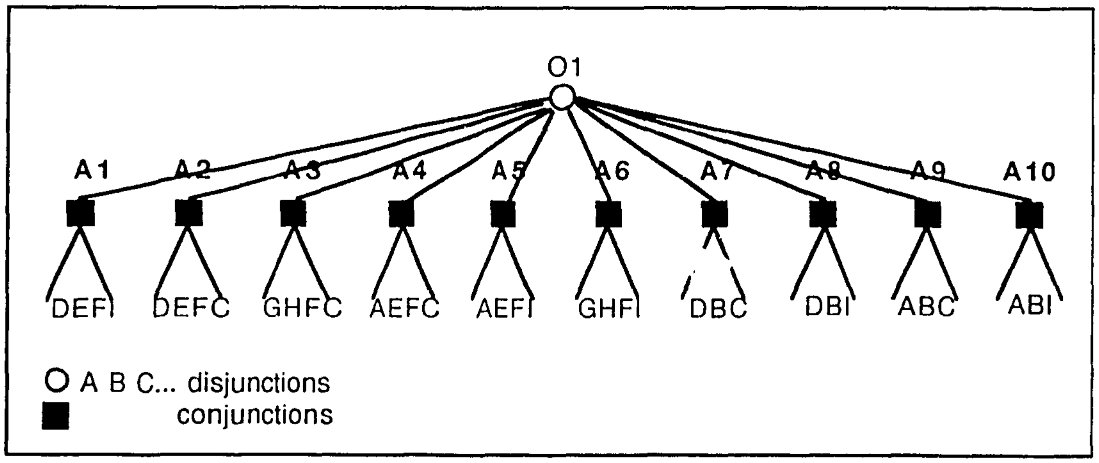
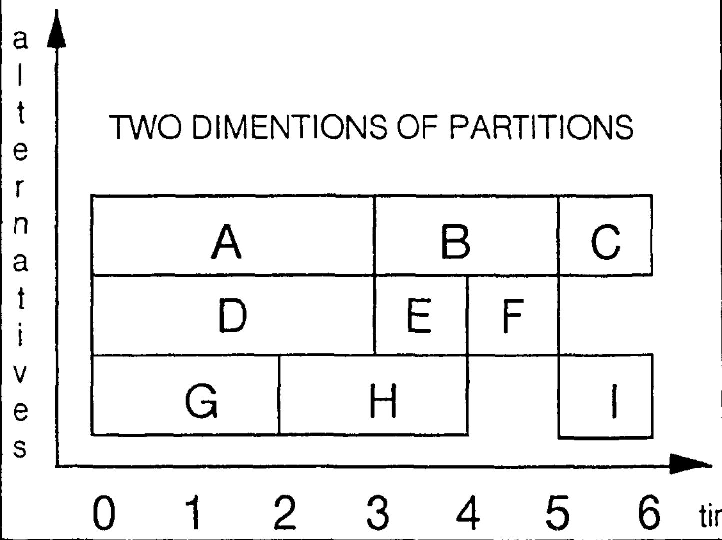


Figure 3.6 - AND/OR graph of Planning Space

Figure 3.6 shows a set of disjunctions { A, B, .. , I } all pointing to partitions at the same hierarchical level. The disjunction labelled O1 points to a partition at a hierarchical level above the others. Conjunctions **A1,...,A10** all point to the same partition as O1. They will all however generate different solution elements.



OCCURENCE OF PARTITIONS WITHIN PARENT NODES

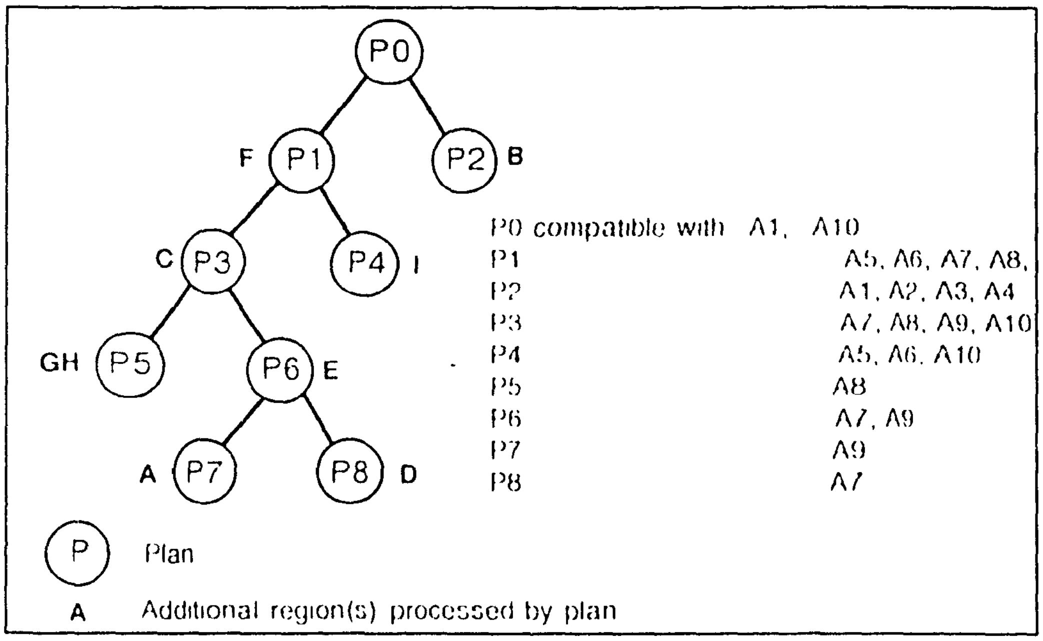
A occurs in	4 conjunctions
B	4
C	5
D	4
E	4
F	6
G	2
H	2
I	5

Figure 3.7 - Conjuncts in Domain Space

Figure 3.6 shows three levels of the AND/OR graph with nodes pointing to partitions occuring at two successive levels of abstraction. Figure 3.7 shows a table listing the number of conjunctions that each partition occurs in, as well as two dimensions of the partitions A, ..., I. The horizontal axis in the later figure, shows the signal time and the vertical shows the label dimension.

According to the table in figure 3.7, under a policy of least commitment, control would choose region F to work on first, since it is shared by the largest number of conjunctions. A decision to process F however excludes partition B (since the time segments of signal they span overlap and solutions from one cannot be integrated with solutions from the other). Because the planner must decide which between two alternatives, separate plans are generated each encompassing one of the choices.

Assuming the plan encompassing partition F has been selected and that the required number of solution elements in that region have been found, the planner continues processing solutions in other regions and integrating them with those in F Under a dominant policy of *island driving* [4] · regions { E, H, I, C } would be considered as candidates for the second step of the plan encompassing region F Since E and H compete with one another as do C and I, the plan P1 is refined into two further plans.



P0 compatible with A1,	A10
P1	A5, A6, A7, A8,
P2	A1, A2, A3, A4
P3	A7, A8, A9, A10
P4	A5, A6, A10
P5	A8
P6	A7, A9
P7	A9
P8	A7

Figure 3 8 - Successive Plan Completions

Figure 3 8 shows the succession of plan completions Plan P0 the initial root plan does not represent a commitment to process any particular region other than a decisions to integrate solutions into the partition O1, it is therefore compatible with all the conjuncts A1, . A10 grouped by O1 Plans P1 and P2 encompass the first processing decision to

[4] Island driving is a policy that has been adopted in a number of signal processing applications including the HASP/SIAP system [NII&FEI78], the HEARSAY II system, the SPARSER system [BATES75] and the "Distributed Vehicle Monitoring Test Bed" [LES&COR 83]. Under island driving a system processes radially from anchor points of certainty, then attempts hypotheses to connect islands into complete theories

process F or B. Each of theses partitions is rated in table 3 9 according to its occurrence in the conjunctions grouped under O1

Partition	Occurrences
A	4
B	4
C	5
D	4
E	4
F	6
G	2
H	2
I	5

Table 3.9 - Occurence of Conjuncts

In the absence of other knowledge F is rated most highly and therefore plan P1 is processed first. Success in generating the required number of solution elements will cause plans P3 and P4 to be sprouted expanding the RHS from of the solution island Plan P5 which extends the island into partition also involves processing partition G since the latter is a consequence of the decision

3.6.4. Meta-level reasoning about plans

In the previous two sections it was stated that a plan need not be investigated beyond the next step, and that a least commitment policy minimizes the refinement of plans. This statement requires some qualification. This policy which is well supported by the system, minimizes unnecessary computation and maximizes the odds of successful solution elements being found early during processing. It is a good approach to take, when the level of knowledge about the sub-problem, or more specifically about plan selection is low.

in situations where the search is better informed and has *expectations* about the form of the problem the system may want to discriminate early on about possible alternatives This principle is in tune with a more flexible planning philosophy were where the planner under certain circumstances has a better model of the world than in others The interaction between planner and effector must retain the flexibility necessary to allow dynamic adjustment of the extent of plans A data driven mechanism is a good vehicle for identifying opportunities for the planer and triggering its mechanisms

From a knowledge engineering point of view, the planning activity is transparent the system allows the user express meta knowledge which extends the planning capacity

Consider for example a speech understanding system to control robot commands. The system has access to the robots understanding of the world in terms of FOP logic predicates The current state indicates among other things that there is a box in a shelf and another box on the table with two predicates

(ON Box Shelf)   (ON Box Table)

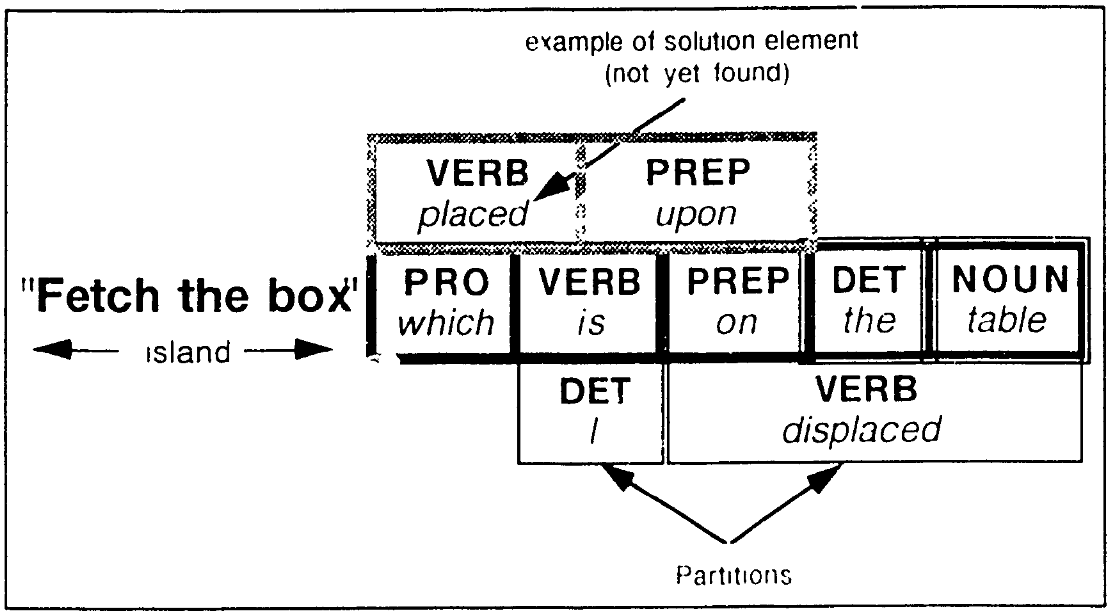


Figure 3 10 - Example of Meta-level Reasoning at the lexical level

Figure 3.10 illustrates a situation were a solution island at the lexical level has been built up and is being extended to the right. The system attempts to validate the partial hypothesis by checking if BOX is an object in its world and which can be an argument to to the command FETCH, in order to determine if the island is worth pursuing

A check through the predicates identifies two occurrences of BOX and confirms that such a symbol may be used as argument to the command FETCH Since there are two predicates containing the argument and the command must be interpreted without ambiguity, the system searches for a prepositional phrase to specify the predicate

Further reasoning reveals that the second argument to the predicate occurs as the noun at the end of the prepositional phrase. This reasoning can be used to restrict the processing options opened to the planner Extending the island into the successive partitions PRO, DET, VERB would not provide a satisfactory hypothesis

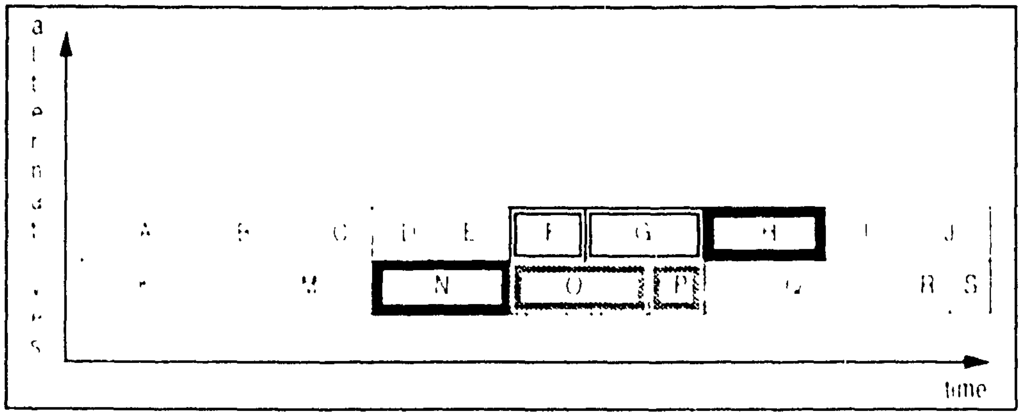


Figure 3.11 Example of planning with meta level reasoning

Figure 3.11 represents shows diagrammatically the planner's perspective. Solutions in partitions N and H must be integrated. The system can easily trace two distinct partial paths {N, O, P, H} and {N, F, G, H}. Figure 3.12 shows the (extend) plans that result from the meta level constraint.

Although meta level constraints impose paths between N and H, the region I is included in plans P112 and P122, because any path including region H must also pass through I.

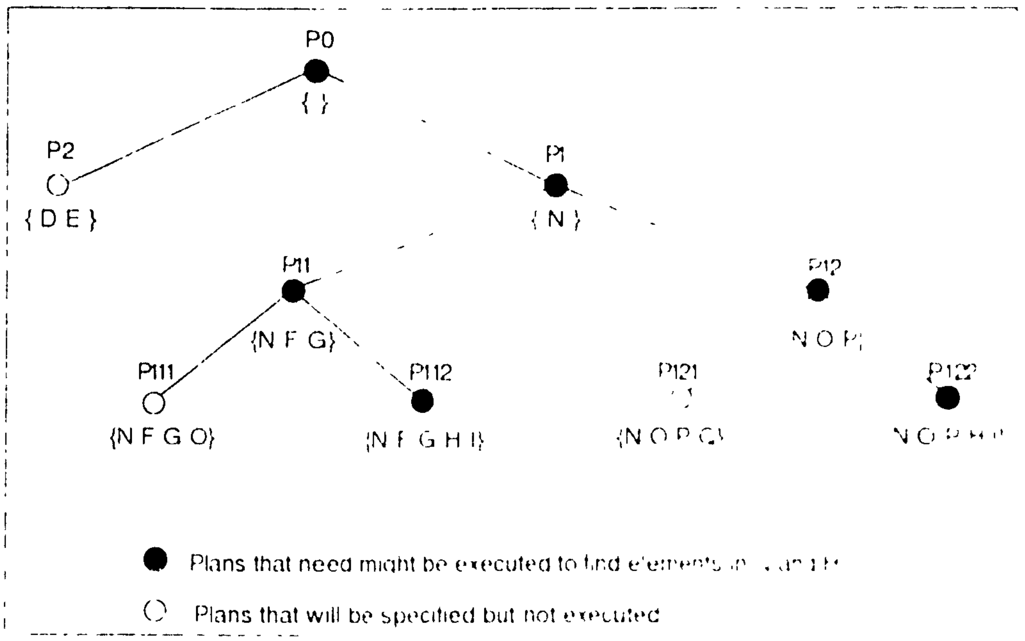

P0
{ }
P2
{D E}
P1
{N}
P11
{N F G}
P12
{N O P}
P111
{N F G O}
P112
{N F G H I}
P121
{N O P G}
P122
{N O P H I}
● Plans that need might be executed to find elements in N and H
○ Plans that will be specified but not executed

Figure 3 12 - Plans generated through application of Meta Knowledge

Figure 3.12 shows as black and white nodes the plans that would be generated (refined) to find solutions in regions N and H Only the plans corresponding to nodes in black need be executed.

The representation is also useful for keeping track of plans that fail Plan P1 encompasses all paths through N Each sub-tree refines a different path Failure of a plan eliminates the entire sub-tree To examine the remaining paths the system develops the other branches. The representation assures the availability of information generated by previous cycles in a form that is readily usable by the decision making processes Having this information, the system can, if control knowledge is available, make better informed decisions without incurring the cost of unnecessary re-evaluations The system updates its knowledge of the search space to reflect changes, rather than re-considering the problem globally.

Control is implemented through the reasoned generation and refinement of plans, the selection among competitors, the ordering of plan steps when more than one is specified, and through the management of knowledge source/data arbitration to generate individual plan elements.

3.6.5. Incorporating control heuristics

When higher level control reasoning has invoked the necessary level plan refinement, the system will be required to arbitrate among competitors if more than one choice is available (as for instance in the example of figure 3.12, with plans P112 and P122). The basis for selection depends on the processing constraints imposed on the machine.

If the system is operating with restricted resources (CPU, memory, processing time etc.) the merits of each plan must be evaluated taking into account these parameters. Some work may already have been achieved towards a plan in a previous phase of problem solving and solution elements might already exist. With constraints on processing time the system might opt for plans which will involve less work. Again the hierarchical blackboard representation allows the system to evaluate quickly the number of solution elements associated with each plan element using the pointers of the semantic network.

Lesser and Durfee have described a mechanism for developing heuristics for the Distributed Vehicle Monitoring Testbed [DURFEE 87] to estimate costs and result quality. The method suggested here relies on the same principles but has been adapted to the organization of the blackboard.

To estimate plan costs, for each evaluation criterion such as processing time or memory requirements, critical attributes are identified. A function retrieves, from the set of plan elements which have been processed, a plan element whose critical attributes values are closest to the plan element being evaluated. The total cost of the plan is the sum of the costs of the steps involved. The heuristics are intended to differentiate between the choices that are available; in the example of figure 3 12, the heuristics can be used to compare the costs of P112 and P122

To estimate the likelihood of success of a plan, control retrieves the nearest matching region to the one being processed by the current plan In the speech application for instance, if the plan is designed to generate hypotheses at the word level, the plan steps being regions at the phonetic level, the system would retrieve a closely matching word region which had already been processed The attributes upon which the match is performed would have to be selected after careful study of the application The relationships between the completed plan, represented by the matching word region and the plans under consideration can be used as yardsticks for evaluation The number of plan steps, the parameters of individual regions within the plan, criteria dependant on local properties of plan elements, and global properties of the entire plan, can for the used for predictions

3.6.6. Finer control within individual plan steps

Once a plan has been chosen and the system has selected a step for processing assuming enough hypotheses exists within the region and that sub-plan are not required, a processing conflict will arise as hypotheses compete for integration into the solution The level of control that has to be exercised at that level, will be very much dependant on the application and the cost of processing. If the knowledge sources are very expensive, the system will have to be selective, choose the best hypotheses first, and limit the number of solutions. If the knowledge sources are not computationally demanding, exhaustive processing within the bounds of the region and the integration of all available hypotheses can probably be considered. Even if selective control is required, the task is sufficiently simple to be implemented with a conventional scheduling queue.

3.7. The complete problem solving process

3.7.1. The complete problem solving process

The system is initialized with starting data and the goal. The goal, as was said earlier, is a plan element which forms by itself the principal root plan.

Either by explicit procedure invocation or through demons, depending on implementation, instances of appropriate regions (plan elements) are created which encompass all the initial data. Whether at this lowest data level several plan elements map into one region or whether the correspondence is one to one again depends on implementation and the domain.

From this point on, until all the search space has been partitioned into plan elements, reasoning is carried out in terms of regions. Processing during this phase is exhaustive and data driven. Whether it is implemented with object programming or through productions depends on the software environment. In retrospect, object oriented programming would appear the better choice because it emphasizes more explicitly the structural relations involved, and because it focuses the programmer's attention on generic properties of the data being manipulated. At the end of this first phase of processing (which is performed bottom up, or forward chaining in the case of productions) the search space will consist of a network of arcs and region nodes. An AND/OR relationship exists between these nodes. Any nodes not connected to the root region (goal node) can be discarded as dead ends.

The second phase of processing interleaves the elaboration and execution of plans. The first plan to be specified consists of only one step, the root goal. At this stage there are no solution elements corresponding to the goal region. Plans must be elaborated to generate hypotheses within the constraints of that region. In the interpretation task, hypotheses are generated by integrating elements from compatible regions one or more levels below. If any region requires additional solution elements which can be integrated with adjacent neighbors, then a local plan is created to generate them Planning is therefore a recursive process If a region contains enough suitable hypotheses, the planner proceeds with the integration of these elements into an island eventually spanning the parameters of the parent region This last process models the interactions between solution elements

If the number of hypotheses that are generated is very large, the resources can be overwhelmed Rather than allowing all solution elements to be generated within one region, a beam of the best solutions is retained If the correct solution element is missed, the system backtracks and generates plans encompassing partially processed regions The system re-plans, because the selection and ordering of steps can change as a result of additional information about the search space, discovered in the preceding pass at the solution

3.7.2. The interpretive cycle

To describe the interpreter's cycle, it is first necessary to describe some of the procedures it invokes and to specify some of the data structures that are referenced

Plans can be described by the following schemas. The examples refer to ' ` problem illustrated in figures 3.11 and 3.12:

{{ **Plan** P112

Generates-solutions-for-region·

Regions-to-process: N F G H I

Regions-already processed

Start time-of region-for-which-solution-is sought x

End time of region-for which solution is-sought y

Compatible paths S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8

Status Processing

Is refined by

Refines P11

}}

{{ **Plan** P11

Generates solutions for region

Regions to process N F G

Regions already processed

Start time of region for which solution is sought x

End time of region for which solution is sought y

Compatible paths S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8

Status Processing

Is refined by P112

Refines P1

}}

{{ **Plan** P1

Generates solutions for region

Regions to process N F G

Regions already processed

Start time of region for which solution is sought x

End-time-of-region-for-which-solution-is-sought· y

Compatible-paths: S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8 ·· ·

Status. Processing

Is-refined-by: P11

Refines·

}}

where S_1 = A B C { N F G H I } J

S_2 = A L M { N F G H I } J

S_2 = K L M { N F G H I } J

S_2 = K B C { N F G H I } J

S_5 = A B C { N F G H I } R S

S_6 = A L M { N F G H I } R S

S_7 = K L M { N F G H I } R S

S_8 = K B C { N F G H I } R S

The refinement of plan P1 before it is processed is an anomalous situation which resulted from the intervention of meta knowledge Normally plan P1 would be executed first and if successful, would be refined into P11 and P12, and so on In this example however, because of meta-knowledge constraints, plan P112 is processed directly, and as soon as a step has been completed, the updates are made to the ancestors P11, P1 This ensures that the system can recover in case of plan failure

Two separate procedures are required. The first generates a seed plan where none exits, the other refines an existing plan which does not yet span the parameters of the region, for which solutions are sought

```
Function Create-plan-from-scratch (region)
/* Creates a plan where none exists  Because the root plan does nct commit the
search to any region, this function must also perform a first level of refinement
and generates all the alternatives resulting from an initial processing decision
The function returns a list of plans representing the first level of refinement  */
Begin
   create-root plan (region)
   Anchor    get anchor (region)
   /* May activate meta-knowledge sources */
   alternatives to anchor     get alternatives to anchor (anchor, region)
   for i in alternatives do  begin
                              path     catenate (
                              /* Catenate the path from the right and from left  */
                                   extend path (i, region, extention to left)
                              /* Extend path to the left  until 1st decision point  */
                                   extend path (i, region, extention to right) )
                            /* Extend path to the right  until 1st decision point  */
                              Generate plan (region  path),
                            end /* for */
   Return (root plan and generated plans)
end /* main */
```

```
Procedure Refine-plan (plan, direction-of-path-extention)
/* Takes any plan which does not yet span the parameters of the region for which
it is expected to generate solution elements, and refines it so that it incorporates
additional sub-regions needed to generate solution elements at the parent level.
As Island driving and least commitment are always the underlying policies, the
system will always extend the plan either from the right or the left of the existing
island.
The side (left or right) which is selected for extension is the one containing the
fewest choices. The plan is extended in that direction to include all adjacent
elements up to the point where the path splits. */
Begin
   if direction-of path-extention = right
   then Neighbours := get-neighbours-to-right (plan.regions-to-process)
   else Neighbours := get-neighbours-to-left (plan regions-to-process)
   For i in neighbours do
      create-a-plan-refinement-extending-path-to (direction-of-path-
extention,plan)
End /* refine */
```

The paths listed in the attribute "compatible-paths" of the plan schema, are retrieved from the "links" connecting the regions generated during the partitioning of the search space. Every sub-region within a path is a conjunct, and alternative paths are disjuncts. The start and end-times refer to the dimension along which the planning process takes place. For the speech application, this represents the duration of the signal. The actual values appearing in the plan, are the same as those of the region for which the plan is generating solutions. A plan in which constituent sub-regions do not span the the interval specified by these attributes, will require further refinement. As eacl successive plan refinement extends the path(s) of its predecessor, it will inherit the values of the slots "Regions-already-processed" and "Regions-to-process" of its parent plan. Each plan represents a commitment to a partial path, which represents a processing decision. The regions to be processed by a plan involve no choice. Normally a plan is refined only after all its regions have been successfully processed. Meta knowledge can intervene to enforce arbitrary path constraints and therefore telescope several decisions at once. Because the only provision for meta-knowledge to specify constraints is in terms of paths, the information is available when the (first) root plan is formed by the procedure "Create-plan-from-scratch", and meta-knowledge rules need only be invoked once for each plan If meta level knowledge imposes a path encompassing several decision points, the "dummy" parent plans are created The leaf plans are executed, but the parent's 'regions-already-processed" and "regions-to-process" attributes are updated. This is necessary to track the portion of the path that succeeded in case the leaf plan fails. A plan fails if no solution element can be integrated into the island

Plan execution:

```
Function Execute-plan (plan) ;

/* This function is invoked by GENERATE-SOLUTIONS when the system is unable to
generate solutions elements. It executes one of the alternative plans PLAN in order to
generate the components (input-data) needed by the knowledge sources to create solution
elements in the target partition. The function returns one of three values "success+fully-
refined" if the plan was executed in full (no further refinement is possible) and generated
enough component data to produce the required solutions in the target partition A value
"success" is returned when the plan for a plan which is not fully refined but which has
been executed successfully. The value "redundant" is returned when the plan is aborted
during the course of execution because domain knowledge sources were able to generate
the required solution elements The plan has in this case become redundant. The function
returns the signal "fail" whenever the processing of a sub-partition fails to yield at least
one solution element (input data item) necessary for the target partition's solutions. The
solutions elements are generated as side-effects. */

LOOP
if no-regions-left-to-precess and plan-is-fully-refined-to-right and plan-is-fully-refined-to-
left
then return (plan fully-refined-and-successfully-executed)
else
   if no-regions-left-to-process
   then return (success)
   else   if no-goal-is-attached-to-region-for-which-solutions-are-required
/* The plan is not yet completed but the goal which drives the KS responsible for
generating solutions in the target region has been removed by demon action */
          then return (redundant)
          else   if call to GENERATE-SOLUTIONS (current-region) = 0
                 then return (fail)
                 else continue processing another sub-partitions
go LOOP
```

```
End /* execute-plan */
```

```
Function Generate-solutions (Region) ;
/* Generates solutions within Region and returns the number of them which were
created. */
Goal .= Create-goal-element (region)
/* the goal specifies the number of solution elements required in the region  When
these have been created a demon removes the goal in that region */
Activate-domain-knowlede-sources
if goal-element-still-exists (goal) and get-level (region) > lowest-level-in-hierarchy
then plan-list := create-plan-from-scratch
LOOP
   current-plan .= pop (plan-list)
   if null (current-plan) and no-solutions-generated-satisfying (goal)
   then    begin
               delete-goal (goal)
               return (nil)
           end
   else    if null (current-plan)
           then    begin
                       delete-goal (goal)
                       return (no-of-solutions)
                   end
   plan-execution-result := execute-plan (current-plan)
   if plan-execution-result = success+full-refined
   then activate-knowledge-sources
   if goal-removed (goal)
   then    begin
               delete-all-plans (region)
               return (t)
           end
   else if plan-execution-result = success
         then    begin
                     best-direction-for-plan-refinement :=
                    get-direction-of-plan-refinement (current-plan)
                  new-plans := refine-plan (current-plan,
                  best-direction-for-plan-refinement)
                  push (new-plans, plan-list)
                end
      else if plan-execution-result = redundant
                then  begin
                          delete-all-plans-for-partition (region)
                          return (t)
                      end
                else if plan-execution-result = fail
                      then delete (current-plan)
                          /* and process next plan in loop */
Go LOOP
if goal-element-still-exists (goal) /* level of region is leaf, no planning possible */
then    begin   if number-of-solutions-corresponding-to (goal) = 0
                then  begin
                          delete (goal)
                          return (nil)
                      end
                else  begin
                          n := no-of-solutions-corresponding-to-goal (goal)
                          delete (goal)
                          return (n)
                      end
        end
else return (t)
end /* generate-solutions */
```

Plans are created to generate solution elements in a region. Each competing and complete plan represents a combinatorialy different selection of sub-regions. Elements from each sub-region may be combined, subject to additional constraints not specified in the plan, to generate a solution element in the parent region. The objective of the plan is to select a set of compatible regions.

Because the constraints imposed on the aggregation of elements tend to increase with the length of the plan (number of steps), the planner attempts to generate a number of islands. This number declines as the plan progresses and cumulative constraints reduce the number of extendable partial solutions If there are not enough solution elements in one sub-region to generate the required partial solutions, the system will generate and execute a plan for that particular sub-region, unless that sub-region is at the level of the initial data If this occurs the plan fails and the failure is passed up to the point where an alternative plan may be selected

3.7.3. Heuristics for plan selection

Plan selection is a commitment to a restricted set of paths and therefore a subset of possible solutions Heuristics can be developed that evaluate the comparative merits of alternative paths supported by the plans These heuristics differ qualitatively from the ones used in traditional hill climbing, in that they do not attempt to measure the relative distances of states to the goal, but estimate the comparative likelihood that different sets of constraints will contain solutions For instance if the region to be analyzed represents a noun-phrase, one plan might impose the following constituents.

DETERMINER + ADJECTIVE + ADJECTIVE + NOUN + PREPOSITIONAL PHRASE

while the other, of course also spanning the same signal, might require:

DETERMINER + NOUN

Evaluation can be on the basis of parameters internal to the plans, for instance consideration of the signal length versus number of words, or they might involve context based evaluations or expectations. Plan selection therefore provides another means of expressing knowledge based control which serves to make searching more efficient.

3.8. Reassessment of behavioral characteristics

3.8.1. Two grain sizes of reasoning

The system allows reasoning to be performed at two levels. A coarse grained level enables a rapid and economical elimination of large sections of the search space Formalizing this knowledge is a relatively simple task. Regions of the search space, which are the data elements used by the partitioning procedures, bear a meaningful relation to the solution elements they encompass. For instance if three adjacent regions at the phrasal level , a noun-phrase, a verb and a second noun-phrase combine to form a sentence, the verb and each of the noun phrases may correspond to a large number of distinct (domain) hypotheses. However all that is required to proceed with the reasoning is the classification of the word or the phrase It should be noted that a consequence of the "blurring" of detail which occurs when reasoning is based on regions as opposed to individual object hypotheses, results in loss of information which might have eliminated candidate plan elements. This loss of information is propagated along with the reasoning. Generally however, the reduction in complexity resulting from reasoning at coarser resolution far outweighs the increase in complexity due to relaxed constraints.

This provides a clue to the methods of partitioning the search space. To integrate phrases into a sentence, the system requires precise signal time parameters in order to determine adjacency of constituents, and it requires coarse classifications. Second order features such as the form or type of verbs, and the number of noun-phrases are ignored.

3.8.2. Goal driven processing

Problems which are subject to data saturation cannot be effectively resolved with data driven processing. Processing effort must be focussed in those areas which have a direct bearing on the objectives The second phase of processing to generate solution elements, is goal driven, and under the control of plans Processing starts with a requirement to generate solutions at the main-goal level If such a requirement cannot be immediately satisfied, the system attempts to analyze the alternative sets of sub goals that are necessary to generate solutions Plans are generated as a means of reasoning about sub-goals in a manner which is more effective than that which is possible with static heuristic functions The plan construct provides a language for expressing domain dependant control knowledge (meta knowledge) Like conventional planners, control is applied by reasoning about goals Unlike conventional planners however, the system is intended to cope with uncertainties, so the extent of planning is constrained

3.8.3. Planning capability to achieve objectives

The decomposition of the problem into sub search search spaces, while at the same time recording their mutual dependencies, confines the planning problem With the scope of each individual plan limited to one decision point, the system avoids the twin traps associated with planning in uncertain environments The system avoids developing extensive plans that are never used because of an early failure and it is spared the management of increasingly complex scenarios, which result when the planner decides to hedge its bets Complexity is controlled by interleaving execution at every decision point, while at the same time retaining, through regional dependencies, a longer range view of the problem. Through the plans, the system is capable of reasoning about the interactions between goals and about the implications of control decisions as far as the knowledge of the problem allows.

3.8.4. Meta level reasoning

By providing explicit representation of structural properties of the solution, the problem solver favours the expression of meta-knowledge, where it is available. When meta-knowledge represents information about dependencies between the plan's conjuncts, it refines the plan several steps without invoking execution of domain knowledge sources. The system therefore translates meta-knowledge directly into control knowledge, by extending its planning decisions beyond the scope of its regular processing Meta-knowledge, which is knowledge about the applicability of problem solving processes or knowledge sources, should reduce the uncertainty about invocation. Extending a plan is exactly a commitment to a specific program of problem solving actions. The organization of plans ensures that processing decisions are not extended beyond the level of knowledge available to support them At the same time, as meta-knowledge is itself subject to error, the system allows for backtracking out of plans without losing the information gained through processing.

The level of meta-knowledge which is not well supported, is that which applies additional constraints *within* one processing step. For example, when combining Noun-phrase + Verb + Noun-phrase to form a sentence, meta knowledge sources might be able to specify the number of the first noun-phrase. The system does not directly support this level of intervention because control reasoning is entirely in terms of regions and their interactions. Within a region it has been assumed that the problem is small enough to be effectively controlled by a conventional scheduling mechanism. Under these circumstances, layered meta-rules for refining the conflict-set, would probably be adequate

3.8.5. Minimized control overhead

The effort expanded to reason about control is justified, provided it is commensurate with the effort it saves in the domain. In order to be effective, and minimize evaluation, the data must be available in indexed, and compiled form The partitions of the search space provides an effective basis for indexing The tree representation of plans allows incremental evaluation of control decisions, and effective backtracking when control actions fail

3.8.6. Reasoning about processing and resource constraints

Although this aspect of problem solving has not been discussed in this chapter, usage of finite processing resources is at the heart of all AI problem solving If limitless resources were available then the weakest method, "generate and test", would be adequate for all problems Although problem solvers do not always work in an environment with predefined limits, performance becomes progressively less acceptable as the resources consumed exceed expectations Other systems which perform real time computations must achieve results within specified time limits. No matter what the operating constraints are, a problem solver should have expectations of the effort required to achieve a particular goal, it should be able to monitor progress and if necessary alter its course of actions Reasoning about resources is important when the system contains redundant knowledge. The problem solver can trade off quality or certainty in the solutions for resources requirements The system can make estimates of the work required for each plan by estimating the number of additional solutions elements needed per region (and sub-regions) The system may decide that it will proceed with fewer solution elements, thereby reducing the likelihood of successful completion, in order to satisfy resource limitations.

Referring to the example in figure 3 10, the system may, if estimates the time required to process solutions in each partition exceeds that which is available for processing, decide to generate the solution elements only within the noun partition, or within the noun and preposition. Such decisions involve a risk of error but the economy of resources may be considered worthwhile.

3.8.7. Expectation of effort required to achieve a solution and execution monitoring

The constraints of a plan are cumulative. Even if each component region has the same large number of solution elements, the number of islands that can be built up declines with the length of the plan (the number of processed components). It seems reasonable to suppose that an incorrectly selected plan (for instance one that partitions the signal time wrong), would show an accelerated decline in the number of islands. This opens the possibility of of monitoring progress as a means of heuristic evaluation of plan choice. Thus a plan not only serves as a control device but also the basis for the development of heuristics

4. Chapter four

4.1. Background to the application aspect of the blackboard

4.1.1. Objectives of the application

The original objective of the project was the development of a "general" purpose problem solving environment that might eventually be suitable for the flight simulator application. Because ideas on problem solving techniques cannot be developed without focussing on the needs of a particular problem, and because expertize in speech recognition was available locally, it was decided that the blackboard would be developed around this application

The speech domain is however very complex, and the problem is still actively researched With processing techniques differing widely, and it was evident from the start, that the software needed in order to achieve state of the art performance would not be available The goal in any case, was not a speech recognition system but a blackboard architecture

Blackboards and productions systems, which require cyclical evaluations of large data bases, are not well suited to problems requiring very fast response If the yardstick for the evaluation of speech recognition systems is human performance then the blackboard model, with its extensive use of working memory and varied assortment of processing techniques, is a very poor approach and not a suitable model of the biological system

These general purpose problem solvers do nevertheless, manage reasonably well large quantities of intermediate results, generated by weak processing methods The absence of a good theory, which might have given rise to powerful solution methods makes the speech problem, a good candidate for the development and evaluation of blackboard type problem solvers. While production systems therefore are poor performers for speech recognition, the domain provides an excellent application for testing and evaluating their architectures.

The weakness of speech interpretation techniques makes the task particularly "hard" and problematic. The knowledge available consists of collections of different processing techniques, which individually yield uncertain and partial results; only through cooperative effort between these individual knowledge sources can any measure of success be achieved at reducing uncertainty and at attaining the overall goal. Another characteristic of knowledge about speech is that is fragmentary. The structures used to represent sounds, words, phrases, sentences and the information applied from sentences are all different. They are in fact organized into a hierarchy of interpretation levels. This stratification of information facilitates the complex process of mapping structures, which is the essence of interpretation. Figure 4.1 below shows commonly recognized levels in hierarchy and the classes of knowledge that perform the mapping of structures.

One prossible explanation for stratification, is that it is considerably easier to develop knowledge that transforms structures that are conceptually close, than it would be say to map speech signals directly into the current model of the world. As a result of this fragmentation of the problem space, knowledge sources function within their specialization levels, independently of one another. This modularity makes speech an ideal candidate for blackboard processing.

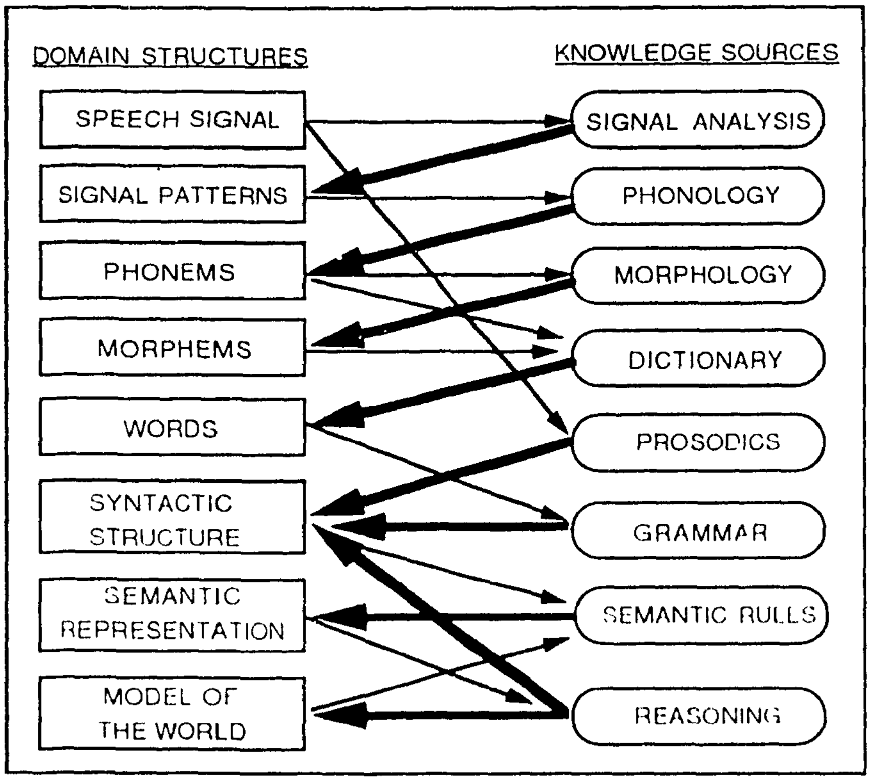


Figure 4 1 - Domain Objects in Speech Understanding Problem[1]

While the diagram of figure 4 1, suggests a preset stratification for speech and language structures, there are in fact no hard and fast rules; techniques exist for instance, to interpret signal patterns directly into words, and strings of words can be mapped directly into a semantic level without recourse to parsing when dealing with proverbs or sayings Often the model of the world and its expectations are used to eliminate candidates at the word or syntactic level etc

While stratification of structure facilitates considerably the task of devising processing techniques, there is no concensus on the best approach for interpreting data any level Bearing in mind that the evaluation of speech interpretation techniques, is not the goal of this thesis, and that a valid, or at least state of the art speech understanding system is not achievable, domain modules were confined to a very modest set of techniques,

[1] Based on T Winograd's representation in [WINO83]

which were readily available. The weakness of the domain methods available, makes the problem "harder", by introducing additional uncertainty. This presents an even greater challenge for the control system.

Recognition of an utterance for the purposes of this system involves synthesis through four levels of abstraction. At the lowest level the signal is processed through segmentation and analysis into Primary Acoustic Cues (PACs). Primary Phonetic Features (PPFs) are then hypothesized from the PACs using Markov Models. From the PPFs and a lexicon, candidate words are generated. Finally theories consisting of time adjacent words are parsed to screen out invalid sequences. Any successfully parsed sequence of words is accepted as valid interpretation of the signal. Since there is no processing at the semantic level, and there is no model of the world to update, as a result of interpretation, the system has no "expectations" and no means of excluding successfully parsed but incorrect hypotheses. The absence of real test data, adds another serious limitation, to the evaluation of the speech recognition performance of this system. As there were no facilities for recording and processing speech, input data was in fact built up artificially, by catenating signal segments generated from individually spoken words. This data was made available from another project.

4.2. The speech front-end

4.2.1. Processing the signal

PACCODER a program implemented by R. Cardin and described in [MATH87], segments the signal and labels each segments according to recognized features and attributes of the signal (eg. peaks and valleys etc... in the change of energy with time) [DeMOR&LAM&GIL87]. The labels generated are descriptions of that interval and are independent of adjacent segments. Table-4.2 lists the set of PACs and describes the characteristics of signal associated with them. Table-4.3 gives the description of the symbols in table-4.2.

PACCODER makes absolute decisions about segmentation and labeling so the entire signal generates only one sequence of PACs. As no search is involved in this first step, the system accepts the sequence of processed PACs as its starting data for interpretation.

PAC symbol	Attributes	Description
LPEAK	tb, te, m1, zx, rmin	long peak of total energy (TE)
SPEAK	tb, te, m,1 zx, rmin	short peak of TE
MPEAK	tb, te, m1, zx, rmin	peak of TE of medium duration
LOWP	tb, te, m1, zx, rmin	low energy peak of TE
LNS	tb, te, zx, rmin	long sonorant tract
SNS	tb, te, zx, rmin	short sonorant tract
MNS	tb, te, zx, rmin	sonorant tract of medium duration
LVI	tb, te, m1, zx, rmin	long vocalic tract adjacent to a LNS, a SNS, or an MNS in a TE peak
MVI	tb, te, m1, zx, rmin	vocalic tract of medium duration adjacent to an LNS, a SNS, or a MNS in a TE peak
SSI	tb, te, m1, zx, rmin	very short vocalic tract adjacent to an LNS, an SNS, or a MNS in TE peak
LDEEPDIP	emin, tb, te, zx	long deep dip of TE
SDEEPDIP	emin, tb, te, zx	short deep dip of TE
LMEDDIP	emin, tb, te, zx	long dip of TE of medium depth
SMEDDIP	emin, tb, te, zx	short dip of TE of medium depth
LHIGHDIP	emin, tb, te, zx	long non-deep dip of TE
SHIGHDIP	emin, tb, te, zx	short non-deep dip of TE

TABLE- 4.2: Primary Acoustic Cues[1]

Attribute	Description
tb	time of beginning
te	time of end
m1	maximum signal energy in the peak
emin	minimum total energy in the peak
zx	maximum zero-crossing density of the signal derivative in the tract
rmin	minimum value of the ratio between high and low frequency energies

TABLE-4 3: Attribute Descriptions

4.2.2. Phonetic alphabet

The labeled signal segments are interpreted into phonemes Instead of the standard English phonemes, the signal segments are interpreted into a set of coarse phonetic classes, called Primary Phonetic Features (PPFs) [DeMOR&LAF&MON85] The set of PPFs consists of five basic phonetic classes and a further fourteen auxiliaries

The main five PPF classes are the following:

[1] The description of PACs is taken form the Msc thesis of Luc Mathan, School of Computer Science, McGill University (1987). A more complete discussion on segmentation and labelling can be found in [DeMOR&LAM&GIL87]

NC	non-sonorant continuant consonants (fricatives)
NI	non-sonorant interrupted consonants (stops)
SON	sonorant consonants
V	vowels
WF	weak fricatives

The relation between the standard ARPAbet phonemes and PPF symbols is shown in tables 4.4 to 4.9 :

Phonemes mapping into PPF **NC**

Phoneme	ARPAbet transcription
s as in "sat"	S
z as in "zoo"	Z
sh as in "shut"	SH
z as in "azure"	ZH
h as in "hat"	HH
f as in "fat"	F

Table-4.4

Phonemes mapping into PPF **NI**

Phoneme	ARPAbet transcription
p as in "pet"	P
b as in "bet	B
t as in "ten"	T
d as in "debt"	D
k as in "kit"	K
g as in "get"	G

Table-4.5

Phonemes mapping into PPF **SON**

Phoneme	ARPAbet transcription
m as in "met"	M
n as in "net"	N
ng as in "sing"	NX
w as in "wit"	W
wh as in "which"	WH
y as in "you"	Y
l as in "let"	L
r as in "rent"	R
oo as in "boot"	UW
oa as in "boat"	OW
ou as in "bought"	AO
o as in "Bob"	AA
e as in "roses	IX

Table-4.6

Phonemes mapping into PPF **V**

Phoneme	ARPAbet transcription
ea as in "eat"	IY
i as in "bit	IH
ai as in "bait"	EY
e as in "bet"	EH
a as in "bat"	AE
a as in "about"	AX
u as in "but"	AH
oo as in "book"	UH

Table-4.7

Phonemes mapping into PPF **WF**

Phoneme	ARPAbet transcription
th as in "that"	DH
th as in "thing"	TH
v as in "sing"	V

Table-4.8

Phonemes mapping into PPF **NC**

Phoneme	ARPAbet transcription
ch as in "church"	CH
j as in "judge"	JH

Table-4 9

The auxialliaries are formed by the catenation of two or more basic PPFs:

SV for catenations of SON-V
SVS for catenations of SON-V-SON
VS for catenations of V-SON
VSV for catenations of V-SON-V
VV for catenations of V-V
WV for catenations of WF-V
NIF for NI at end of word
NIS for catenations of NI-SON
SVSV for catenations of SON-V-SON-V
VSVS for catenations of V-SON-V-SON
SW for catenations of SON-WF
VW for catenations of V-WF
NCF for NC at end of word
NCS for catenations of NC-SON

4.2.3. Signal to phoneme interpretation using Markov Models

Given a set of phonemes:

$$V = \{ v_1, v_2, ..., v_n \}$$

and acoustic evidence O generated by one member of the set *V*. The task of phonetic interpretation is that of identifying **V**, a member of *V*, such that:

$$\boxed{P(\mathbf{V}|O) = \underset{i=1}{\overset{n}{M}} P(v_i|O)}$$

Where M is the maximum conditional probability. The signal interpretation knowledge source searches for the most likely phoneme label, given the acoustic evidence O.

From Bayes's formula:

$$\boxed{P\ (V|O) = \frac{P(V)\ P(O|V)}{P(O)}}$$

where P (V) is the a priori likelihood of occurrence, of a particular phoneme in the language,

P (O) is the average probability that acoustic evidence O is observed in the language in other words:

$$\boxed{P\ (O) = \sum_{i=1}^{n} P(v_i)\ P(E|v_i)}$$

For an observed acoustic evidence O, we therefore have:

$$\boxed{P(\mathbf{V}|O) = \underset{i=1}{\overset{n}{M}}\ P(v_i)\ P(O|v_i)}$$

The learning process involves recording the statistical distribution of acoustic observations associated with uttered phonemes. The database accumulated by learning is P (O | V).

4.2.4. Acoustic models based on Hidden Markov Models

Hidden Markov Models (HMMs) can be used to model stochastic processes that are not directly observable (hidden), but which can be perceived though an auxiliary set of stochastic processes, producing a sequence of observations. In speech recognition, the observations are acoustic evidence, in this case, the sequence of PAC symbols generated by the utterance. The hidden process is the speech utterance itself.

The HMM is trained by recording multiple instances of the sequence of observable symbols (PACs), generated by repetitions of the same utterance. A separate model is trained for each PPF. Recognition is achieved by "scoring" the sequence of observed symbols against every trained model. The highest scoring model corresponds to the most likely spoken string.

A HMM consists of a number of states and a set of allowed transitions. Within a state, certain properties of the signal remain constant. An event occuring at time *t*, is characterized by a transition, possibly to the same state, according to a probability density distribution, and it is detected by an observation. The symbol is also generated according to a probability density distribution which is dependent upon the current state.

A Markov model λ , is built for every phoneme. The first parameter of I, is the the matrix of state transition probability distribution **A**, such that element $a_{ij} = P(q_j \text{ at } t+1 \mid q_i \text{ at } t)$, q_i denotes the state i in the model. The second parameter of λ, is the observation symbol distribution, for every state B, where $B = \{ b_{ij}(k) \}$ and $b_{ij}(k) = P(v_k \mid q_i \text{ at } t, q_j \text{ at } t+1)$.

Given a sequence of observations $O = O_1, O_2, \ldots, O_T$, corresponding to an unseen sequence of transitions $I = i_1, i_2, \ldots, i_T$:

$$P(O \mid I, \lambda) = b_{i1}(O_1)\ b_{i2}(O_2) \ldots b_{iT}(O_T)$$

$$P(I|\lambda) = a_{i_1 i_2}\, a_{i_2 i_3} \ldots a_{i_{T-1} i_T}$$

$$P(O|\lambda) = \sum_{\text{all i}} P(O|I,\lambda)\, P(I|\lambda)$$

Recognition is achieved by scoring the observe sequenced of symbols O against all every HMM λ. The HMM which appears to model the observation best, corresponds to the most likely interpretation. The complexity of the summation grows exponentially with the number of observed symbols. The forward-backward procedure [RAB&JUA86] can be used to compute P $(O|\lambda)$ with a complexity of the order of N^2, where N is the number of observed symbols. Alternatively one can use the Viterbi algorithm, which gives similar discrimination for speech applications, but requires fewer calculations.

Training of HHMs is achieved iteratively using the Baum-Welch re-estimation formulas. A full description of the algorithm is found in [RAB&JUA86].

4.2.5. Hidden Markov Models used to model Primary Phonetic Features

For each main and auxiliary PPF a three state Markov model is constructed and the diagram below shows the allowed transitions:

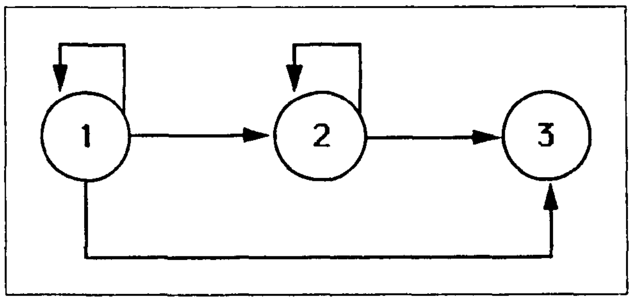


Figure 4.10 - Three state Markov Model

The initial state is always **1** and the final state **3** has no outgoing arcs. A PAC observation is associated with each transition. A PPF can therefore be hypothesized from one (with a 1 --> 3 transition) or more PACs.

For a given model λ, the probability of a transition from state i to j denoted by P_λ (i --> j) is a_{ij} and the conditional probability of observing a PAC V_k, given this transition

$$P_\lambda (V_k \mid i \text{-->} j) = S_{V_k}^{\,j}.$$

For say a 3 PAC string $\{V_k, V_l, V_m\}$, there are three possible sequences of transitions generating such a string:

$$I_1: 1\text{-->}1 \; ; \; 1\text{-->}1 \; ; \; 1\text{-->}3$$

$$I_2: 1\text{-->}1 \; ; \; 1\text{-->}2 \; ; \; 2\text{-->}3$$

$$I_3: 1\text{-->}2 \; ; \; 2\text{-->}2 \; ; \; 2\text{-->}3$$

where:

$$P_\lambda (I_1) = (a_{11} * Sva^1) * (a_{11} * Svb^1) * (a_{13} * Svc^3)$$

$$P_\lambda (I_2) = (a_{11} * Sva^1) * (a_{12} * Svb^2) * (a_{23} * Svc^3)$$

$$P_M (I_3) = (a_{12} * Sva^2) * (a_{22} * Svb^2) * (a_{23} * Svc^3)$$

More generally for an *n* PAC interpretation of a PPF there are *n* possible paths. The diagram below shows all the possible paths for one, two, three and four PAC interpretations.

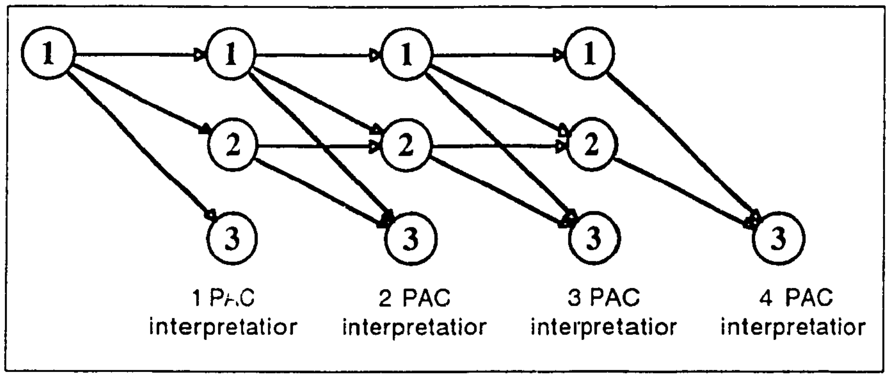


Figure 4.11 - Graph of possible paths for 1, 2, 3 and 4 PAC interpretations of PPF

4.2.6. Compiling the training data

The training data is available in the form of text files listed in [MATH87]. For each model (PPF), the prior probabilities of the transitions are given as well as the conditional probabilities of occurence of PACs for each transition.

Compiling the training data involves computing P (O | λ) for every possible sequence of observations and every λ (PPF). In practice the probabilities of PAC sequences longer than four was found to be negligibly low, so that it was sufficient to compute the expression for every possible one, two, three four PAC strings for each PPF and storing the results in tables.

In order to obtain a number of competing interpretation, a mapping of PAC sequences, to list of PPF hypotheses and their respective probabilities is generated from the conditional probabilities. Details of implementation are given in chapter five.

4.3. The lexical level

4.3.1. Generating word hypotheses from PPFs

The next level in the synthesis is the interpretation of sequences of phonemes into recognizable words. This step is achieved through a lexicon developed by Professor C. Suen. The lexicon originally developed for speech synthesis, contains the transcription of some 9000 words in terms of the PPFs described earlier. The original lexicon was modified using the following transformation rules suggested by E. Merlo. The transformations involve all possible decompositions of catenated PPFs such as SV, SVS, SVSVS, VS, VSV, VSVS, NIS, NCS, VW, SW, VV, WV. For example:

SVS	—>	SON V SON
SVS	—>	SV SON
SVS	—>	SON VS

Every occurrence of one of these phonemes is replaced by all its possible constituents. Multiple occurrences require all combinations to be included. While vocabulary available to the system remains the same, the number of keys to the words increases enormously (over 70000).

The resulting lexicon is stored as an inverted dictionary, so that given a string of PPFs it is possible to retrieve all the candidates words that share the same phonetic description.

Each word hypothesis contains the parameters of time corresponding to the signal it interprets.

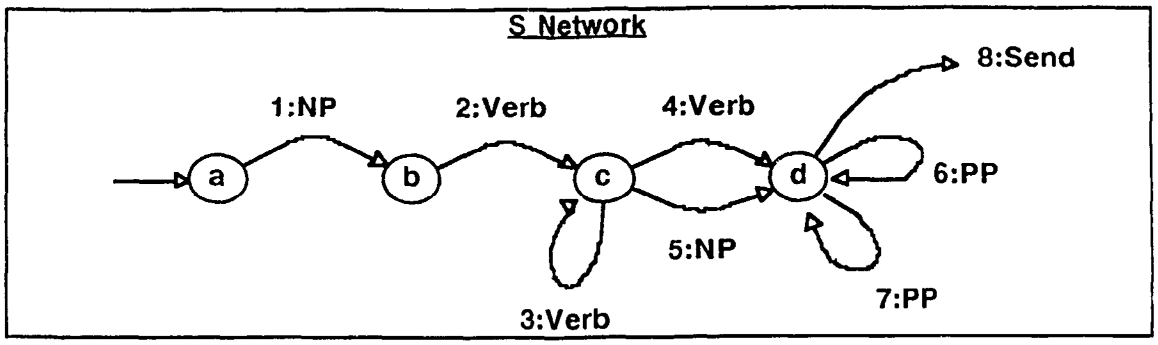
4.4. The phrase level

4.4.1. The grammar

The final step in processing for this system is a parser that screens theories of words that are incompatible with the rules of grammar. The grammar used is based on ATN grammar described by Winograd [WINO83]. While the syntax is described in terms of an ATN the parser it was actually implemented as a rule based parser. Figures 4.12, 4.13, 4.14[1] describe the grammar.

---

1 ATN grammar taken from T. Winograd "language as a cognitive process" Addisson Wesley (1983).



Roles: Subject, Direct-Object, Main-Verb, Auxiliaries, Modifiers

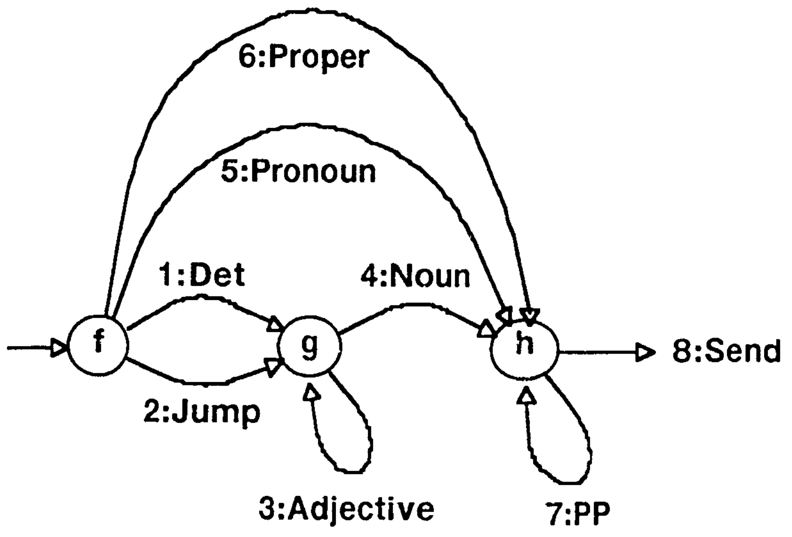
Feature Dimensions: Voice: Active, Passive: *default,* Active

Initializations, Conditions and Actions:

$_{a}\mathrm{NP}_{b}$	**A:**	Set subject to *
$_{b}\mathrm{Verb}_{c}$	**A:**	Set subject to *
$_{c}\mathrm{Verb}_{c}$	**C:**	The type of the Main-Verb is Be, Do, Have or Modal
	A:	Append Main-Verb to Auxiliaries. Set Main-Verb to *
$_{c}\mathrm{Verb}_{d}$	**C:**	The form of * is Past-Participle and the type of Main-Verb is Be
	A:	Set the Voice to Passive. Append Main-Verb to Auxiliaries. Set Main-Verb to * Set Direct-object to subject. Set Subject to a dummy NP.
$_{c}\mathrm{NP}_{d}$	**A:**	Set Direct-object to *
$_{c}\mathrm{PP}_{d}$	**A:**	Append * to modifiers.
$_{c}\mathrm{PP}_{d}$	**A:**	Voice is Passive and Subject is a dummy NP and the word in the Prep of * is *by.*
	A:	Set Subject to the Prep-Object of *.
$_{d}\mathrm{Send}$		
"*"		refers to the most recent node role or feature attributes of the most recent constituent or node (ie. the one just processed by the current arc.

Figure 4.12 - "S" Network Grammar

NP Network



Feature Dimensions: Number: Singular, Plural; *default*, -empty-

Initializations, Conditions and Actions:

NP-1: $_{f}$**Determiner**$_{g}$	**A:** Set Number to the number of *.
NP-4: $_{g}$**Noun**$_{h}$	**C:** Number is empty or number is the number of *. **A:** Set Number to the Number of *.
NP-5: $_{f}$**Pronoun**$_{h}$	**A:** Set Number to the number of *
NP-6: $_{f}$**Proper**$_{h}$	**A:** Set the Number to the Number of *.

"*" refers to the most recent node role or feature attributes of the most recent constituent or node (ie. the one just processed by the current arc.

Figure 4.14 - "NP" Network Grammar

A brief description ATN parsers is necessary to be able to facilitate reading the grammar; the parser actually implemented is not an ATN.

4.4.2. ATN parsers

An ATN grammar consists of labeled networks consisting of nodes and arcs An arc specifies a *word*, a *lexical category*, a *syntactic constituent* which must occur at the input if the arc is to be traversed. There is a also a *jump* arc which may be traversed without consuming any input ATN's are generally top down / left to right parsers although Bates [BATES75] has shown how an ATN may be adapted to work from the middle out for island driven recognition systems and can be designed to function top down, bottom out or bi-directionally. If the processing reaches a *send* arc, then a constituent has successfully been processed. If the parser fails to find a suitable arc at the current point of input then parsing of the current constituent has failed and the system must backtrack

The nodes of an ATN represent intermediate states during parsing During the processing of a constituent, the parser may perform actions which include the setting or modification of a number of attributes of the constituent being parsed These attributes are called *roles* For instance the roles of the Sentence constituent in this grammar are Subject, Auxiliaries, Main verb, Indirect object, Direct object The Roles of a constituent are filled by words or by whole constituents For instance the Subject Role of a sentence can be a Noun Phrase

In addition to role attributes, constituents can have a another class of attributes called *features* The values of these feature attributes, are characteristics of the constituent as a whole which the parser has arrived at as a result of its analysis of the words and structure of the constituent The values of role attributes are therefore *words* or constituents of the parse while the values of feature attributes are properties of the constituents

Arcs can have conditions attached to them. The constraints imposed by conditions must be satisfied by the current state if the parser is to proceed along the respective arc. Conditions generally test the current values of role and feature attributes.

The result of parsing is a tree structure whose leaves are the individual words of the sentence being parsed. The structure is built up through the values of role attributes which are pointers to constituent and words. Figure 4.15 below shows the structure resulting from an ATN parse of the sentence *the speaker was given a standing ovation by the audience* .

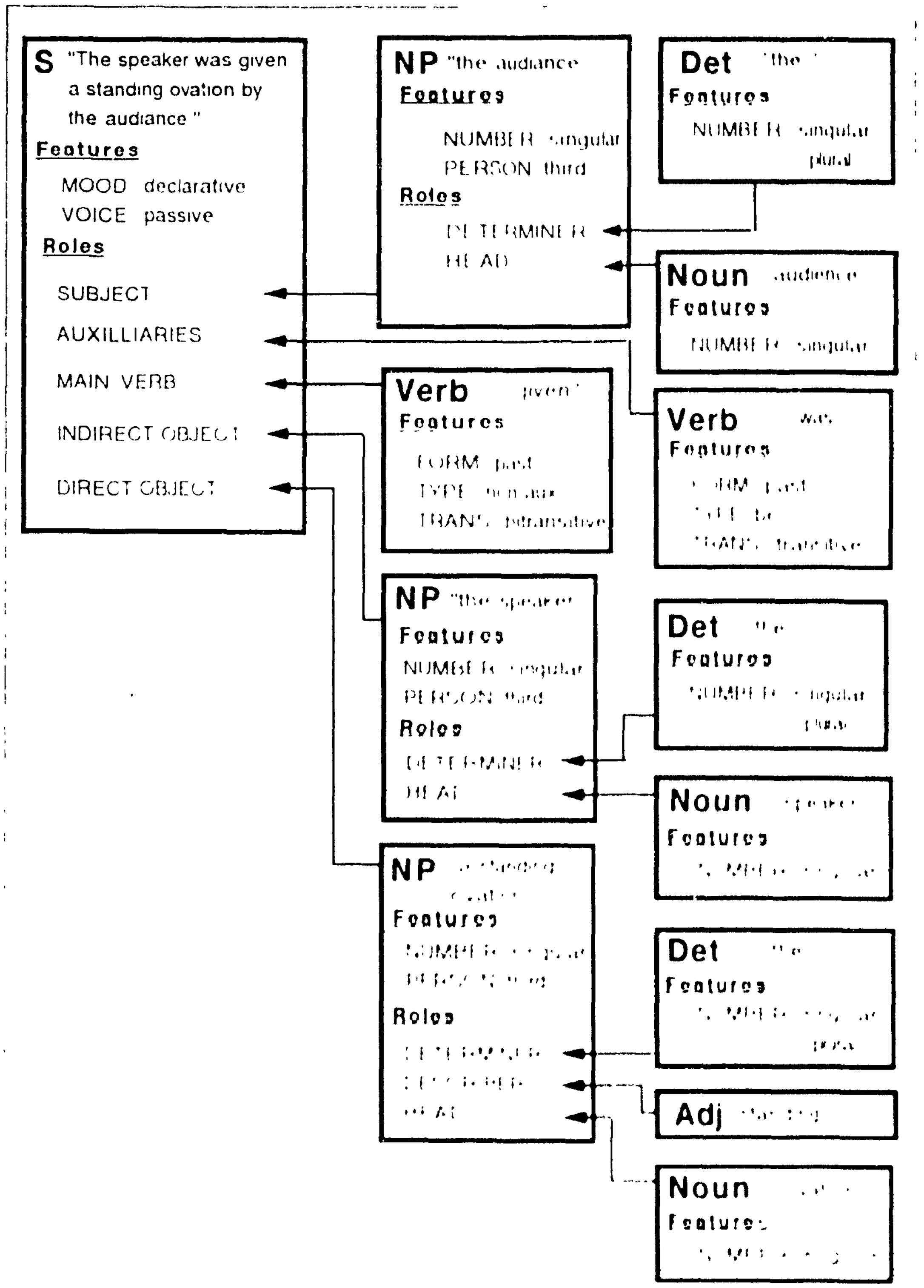


Figure 4.11 Structure generated by parser

4.4.3. The grammatical dictionary

From the grammar it is clear that the values assumed by the feature registers, although sometimes initialized before the start of parsing, are modified according to the properties of the words consumed during the course of processing. The parser must therefore have access to a dictionary where the category and all the features of the word can be accessed before an arc is taken. Table-4.16 below shows a sample of the dictionary.

Word	Category	Features
the	determiner	
speaker	noun	
was	verb	Form: past
Type: be, non-auxiliary		
given	verb	Form: past-participle
Transitivity: intransitive		
a	det	Number: singular
standing	verb	Type:
Transitivity:		
adjective		
noun	Number: singular	
ovation	noun	Number: singular

Table-4.16 - Sample Dictionary entries

In addition to the grammatical dictionary it is often practical to keep a table of default attributes and values such as the one listed Table-4.17.

Category	Dimension	Choices	Default
Adjective			
Determiner	Number	Singular, Plural	Singular
Question	Yes, No	No	
Noun	Number	Singular Plural	Singular
Case	Subjective, Objective	Subjective Objective	
Preposition			
Pronoun	Number	Singular, Plural	Singular
Person	1st, 2nd, 3rd	3rd	
Case	Subjective Objective	Subjective Objective	
Question	Yes, No	No	
Proper	Number	Singular, Plural	Singular
Case	Subjective Objective	Subjective, Objective	
Verb	Form	Infinitive Present	Present Infinitive
Past, Past-participle			
Present participle			
3rd Singular present			
Transitivity	Intransitive Transitive Bi transitive	Transitive	
Type	Be Do Have	Non aux	
Modal, Non aux			

Table 4.17 [illegible] default values [illegible]

4.4.4. Use of ATN parsers in blackboard architectures

ATN parsers were intended for [illegible] processing, [illegible] important features of many blackboard architectures [illegible] for [illegible] a disadvantage [illegible] processing [illegible] the hypotheses are reasonably certain, and then to expand laterally in both directions until compatible islands merge. A system understanding system developed by Bates [BATE75], was one of the first to modify ATN parsers to allow processing in both directions and possibly bottom-up. Given a word or a sub-network hypothesis, the anchor point, the system retrieves all possible "lead-in" transitions from all sub-networks. As processing continues towards the left some sub-network candidates are eliminated.

4.5. Search space partitions in the speech application

4.5.1. Partitions at the phonetic level

The finest resolution at the phonetic level is the PPF. These phonetic features do not however, represent the full extent of discrimination possible within the domain as each of them groups multiple English phonemes. Vowels for instance are all represented by one HMM "V". The result is that the lexicon maps each phonetic description into a number of alternative words. While some reduction of these candidates is possible through parsing, the number of alternatives still remains large. PPFs in fact should be considered as partitions at the phonetic level rather than domain hypotheses. Domain solutions should refine "V" hypotheses into individual vowels. Efficient partitioning requires good familiarity with the domain, the lack of familiarity with these phonemes makes it it difficult to make suggestions.

4.5.2. Partitions at the lexical level

The lexical level being more familiar, suggestions ar a little easier to make. The time or signal attribute must be exact in all partitions, since this is the dimension along which plans are developed. The system can proceed with the partitioning phase bottom up without differentiation between alternative words hypotheses belonging to the same lexical category and spanning the same signal time. This implies some loss of information, for instance the features which are tested by the parser, but the reduction in complexity obtained by broadening the label constraints, outweighs (substantially we feel), the reduction in complexity that this information would contribute, at subsequent stages in processing

4.5.3. Partitions at the phrase level

The partitioning process is performed exhaustively and therefore with a minimum of control. Processing must be economical and efficient Since loss of information at the lexical level makes the tests appearing in the conditions of the ATN grammar impossible, a simplified rule based parser which generates all the constituents produced by the ATN parser is sufficient. At each level in the phrasal hierarchy, alternative hypotheses (constructs) of the same type, and which span the same segment of signal, are grouped together before being incorporated into higher level constituents. Again the partitioning is the same as it was at the lexical level, with constraints being relaxed on the label, but kept on the signal time and on the class.

5. Chapter five

5.1. Software environment

5.1.1. The programming environment

The software was developed on a TI Explorer II Lisp machine. The KnowledgeCraft© program augments Common lisp with a schema based description language, object oriented programming, and two inference engines. The forward chaining engine is a specially adapted version of OPS5, and the backward chaining is a modified form of Prolog. KnowledgeCraft features are available as function calls and the system is otherwise a conventional LISP machine interpreter.

Invocation of schema generating functions creates as a side-effect, special data structures which can be retrieved, modified or deleted by other KnowledgeCraft functions. These data structures are used to represent all the basic elements of the blackboard (solution elements, partitions, plan elements etc...).

In the KnowledgeCraft environment schemas are generated, modified or deleted by three mechanisms all of which are utilized by the blackboard:

1) Through explicit function calls initiated by the interpreter or by other functions.
2) Through demons and other object oriented programming techniques. These occur as a result of modifications to schemas.
3) By the actions of inference rules.

The transfer of control from one mechanism to the other is crucial to the function of the blackboard.

The dominant mechanism is object oriented programing, any changes to attributes which cause the activation of demons will cause a transfer of control to the procedures or functions associated with demons or procedural attachments.

The second dominant mechanism in the KnowledgeCraft environment is the pattern matcher of the OPS engine, the RETE[1] network. The network represents all the tests expressed in the left hand side of OPS rules. With regards to rules set, modifications to schemas tested in the left hand side, represent changes of state. As soon as demons have executed their routines, the final effect of these changes is propagated through the network and the potential effect on instantiations is evaluated. No rules may be fired (institute changes to the database) however, without explicit launching of the inference engine.

5.1.2. The system's data structures

With KnowledgeCraft all the properties associated with schemas are described by other schemas, which may be user defined or specializations of schemas pre-defined in the software. Relations between prototypical schemas, demons etc. are declared [illegible] uniform formalism. Basic properties required for knowledge representation [illegible] the fundamental relations IS-A, INSTANCE [illegible] are supported and the [illegible] with schemas connected by such relations [illegible] defined. Because a comprehensive description of KnowledgeCraft [illegible] important the declarations of data structures used [illegible] will be accompanied by explanations.

[1] A [illegible]

[illegible]

[illegible]

```
{{ Partition            /* partition and region are used synonymously */
LABEL:
            START-TIME:
            END-TIME:
            LEVEL:
            HAS-DOMAIN-MEMBERS:
            SUB-PARTITIONS:
            OCCURENCE:
            TARGET-NUMBER-OF-DOMAIN-MEMBERS:
            DATA-GROUPING-NODE:
CF:}}
```

The LABEL slot takes the value of the lexical class for "word" level partitions, its value for partitions at the constituent level is the syntactic constituent class itself (NP, PP, S...). START-TIME and END-TIME refer to the time values of the signal spanned (PACs). LEVEL is the level of abstraction in the blackboard. Because a phrase level hypotheses can be composed of other phrase level constituents, it does not necessarily follow that the components of a partition belong to the next lower level. HAS-DOMAIN-MEMBERS is a pointer to solutions elements within the partition. SUB-PARTITIONS is a pointer to lower level component partitions which are used for abstraction. Partitioning of the search space results in an AND/OR graph of partitions. During the forward phase of reasoning, time adjacent partitions are integrated into higher level ones. Non intersecting spaces that compete but cannot be differentiated by forward knowledge sources are merged by broadening some of the constraints. The objective here, is to reduce the number of (redundant) elements with which the forward productions have to reason. Rather than modify the parameters of the elements themselves, nodes to be merged are connected through a OR node. The input data for forward (partitioning) rules is always composed of these OR nodes and the output is always AND nodes The partitioning process therefore alternates synthesis (AND nodes) with data reduction (OR nodes) Figure 5 1 illustrates a portion of the AND/OR graph In the program the same data structure is used to represent both types of nodes The slot DATA-GROUPING-NODE, takes a boolean value which if true indicates an OR node The slot HAS-DOMAIN-MEMBERS is only filled for AND nodes, as the plans are specified in terms of conjuncts The SUB PARTITIONS slot contains the pointers to conjuncts in the case of AND nodes, and disjuncts in the case of OR nodes.

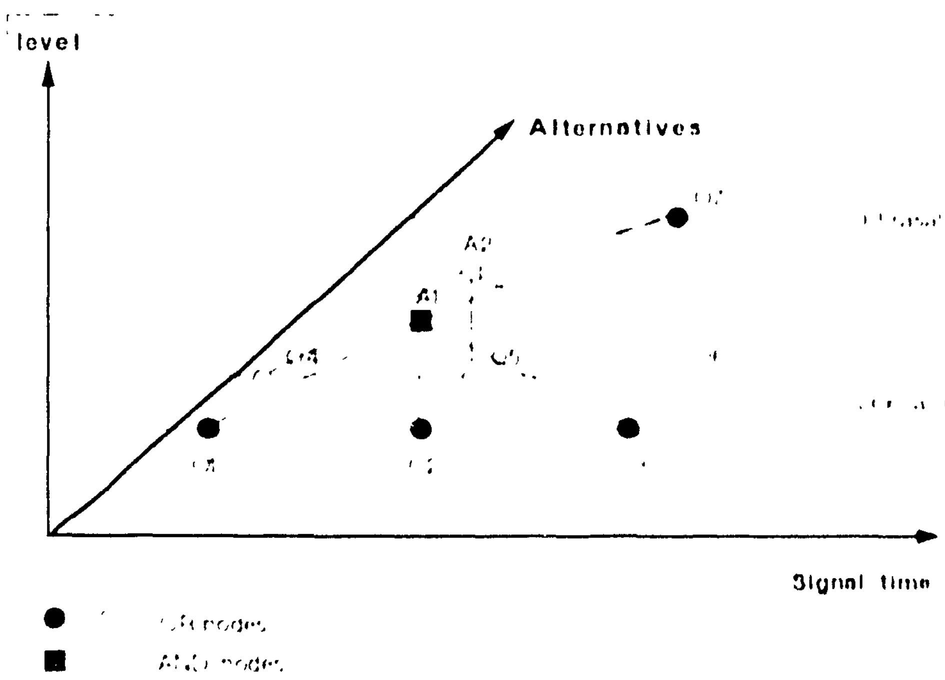


Figure 5 1 Graph of control nodes

All pointers are defined in terms of [illegible] and the system [illegible] establishes the inverse pointers. The [illegible] slot takes an integer value which is incremented whenever a [illegible] AND node [illegible] calculate [illegible] [illegible] phonetic level. In the absence of heuristic evaluation functions at higher levels of abstraction, these values were propagated to word and syntactic constituent hypotheses using the formula:

$$\boxed{CF = \sqrt[n]{\prod_{i=1}^{n} h_i}}$$

where h_i is the CF of the ith. hypothesis (conjunct) incorporated into the higher level interpretation.

The significance of these certainty factor values is, however, very doubtful and provides a very weak basis for discriminating between hypotheses at the nigher levels. TARGET-NUMBER-OF-DOMAIN-MEMBERS, is used in the plan elaboration phase. Partitions that are processed in the earlier stages of a long plan, should allow for more solution elements to be integrated into islands, as the chances of failure with cumulative constraints are greater. This slot contains the estimate of the number of solution elements required to retain enough islands up to the completion of the plan.

The prototype schema for a plan is as follows:

```
{{ Plan
            PLAN-LEVEL:
            PENDING-PARTITIONS:
            COMPLETED-PARTITIONS:
            PLAN-VALUE:
            RESOLVES-PARTITION:
            COMPATIBLE-SOLUTIONS:
            SUB-PLANS-FOR-CURRENT-STAGE:
            STATUS:
            START-TIME
            END-TIME
            CURRENT-GOAL-GENERATED-BY-PLAN: }}
```

The PLAN-LEVEL level value is the same as level value of the solution elements it is intended to create. PENDING-PARTITIONS, contains the list of partitions that still have to be processed under the current plan COMPLETED-PARTITIONS, contains the list of partitions for which solutions have already been found (under the current plan) Unless the plan has failed, there will exist at least one island spanning all the complete partition of the plan PLAN-VALUE contains an estimation of the plan's potential for fulfilling its goals given the current processing constraints and policies This value is used to select a plan from the set of competing alternatives, which will generate solutions within the same partition RESOLVES-PARTITION, contains a pointer to the partition for which the solution elements are being generated. COMPATIBLE-SOLUTIONS contains pointer to the disjuncts (partition nodes) which are compatible with the processing decisions represented by the plan SUB-PLANS-FOR-CURRENT-STAGE, points to other plans which have been elaborated to generate solution elements within the partition in which solution islands are being extended The plan's STATUS slot informs the system whether a plan has been completed (no further processing required), whether it has failed (no island could be extended to span the entire plan) or whether it is still processing START-TIME and END-TIME has the same value as the start and end times of the partition for which the plan is generating solutions A plan that has generated the required number of islands spanning its start and end times is complete

Control over domain knowledge sources is exerted through goal elements generated by the plans being executed Goal schemas are directives to generate solution elements in a particular partition of the blackboard, or to integrate solution elements of a particular partition into islands The prototype goal is defined as follows

{{ **Goal**
ACTION-LEVEL
REQUIRED-SOLUTION-ELEMENTS
SOLUTION-ELEMENTS-STILL-REQUIRED
PARTITION-UPON-WHICH-GOAL-IS-FOCUSED

PLAN-WHICH-GENERATED-GOAL:}}

ACTION-LEVEL refers to the hierarchical level on the blackboard at which domain knowledge sources are required to output their solutions. To control the number of hypotheses that are generated, knowledge sources only output one solution element per invocation. The number of required solution elements is recorded in the slot REQUIRED-SOLUTION-ELEMENTS, every time the goal is involved in the invocation of a knowledge source, and results in the creation of a hypothesis, the value in the slot SOLUTION-ELEMENTS-STILL REQUIRED is decremented. When the value in this latter slot becomes zero, the target number of solutions has been reached and the goal is removed by a demon, effectively inhibiting further processing The slot PARTITION-UPON-WHICH-GOAL-IS-FOCUSSED points to the partition element, to which the solutions output will belong. PLAN-WHICH-GENERATED-GOAL is a pointer to the plan schema which generated the goal element.

With the exception of the top level, the purpose of goals is to generate solution elements that may be integrated into higher level hypotheses. When the required number of solutions at a particular has been created and the demon attached to the slot SOLUTION-ELEMENTS-STILL-REQUIRED is triggered, it should not only remove the goal in question, but all the sub-goals that were created at lower levels in the hierarchy.

The attributes of domain elements differ according to the hierarchical level on the blackboard, several specialization of the domain element prototype are defined.

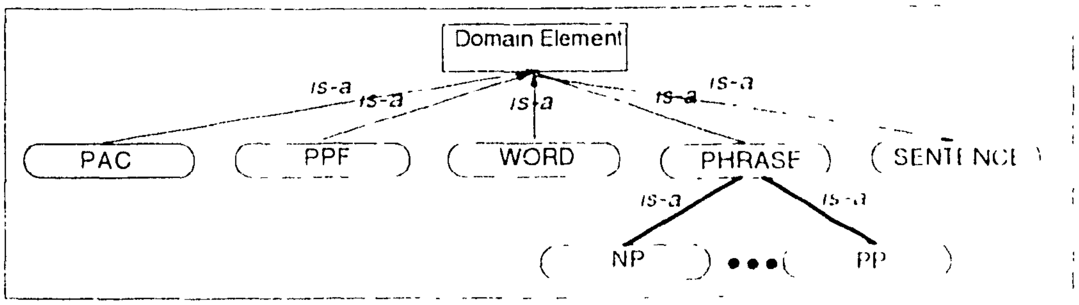


Figure 5 2 - Domain Object declarations

Each specialization schema shown in figure 5 2 inherits all the attributes of the more general parent DOMAIN-ELEMENT schema

{{ **Domain-element**

LEVEL

LABEL

PARTITION-LABEL

BELONGS-TO-PARTITION

START TIME

END-TIME

WORK DONE

CF

PLAN-THAT CREATED IT

GOAL RESPONSIBLE FOR CREATION

SUB ELEMENTS

PARENT ELEMENTS }}

The LEVEL slot records the hierarchical level of the solution element LABEL records the class of the domain element In the case of PACS, *[illegible]* *[illegible]* For PPFs *V W SON* at the lexical level the label is the word itself at the phrasal level possible label values include *noun phrase, prepositional phrase* etc The PARTITION-LABEL attribute is the label of the partition to which the domain element belongs For the *word* level hypothesis "dog", for instance the partition label would have the value "noun". BELONGS-TO-PARTITION is a pointer to the partition schema itself. START-TIME, END-TIME are the subtended signal times. WORK-DONE is an integer value which represents the cumulative computational effort to arrive at the hypothesis. In this particular application, it is assumed that every domain knowledge source invocation is equivalent, so that the work done is one plus the sum of the work done for each of the constituent (lower level) hypotheses. The CF is the confidence factor associated with the hypothesis. PLAN-THAT-CREATED-IT is a pointer to the plan schema which generate the goal necessary for the invocation of the knowledge source. CREATION-CYCLE contains the cycle number at which the hypothesis was created. GOAL-RESPONSIBLE-FOR-CREATION points to the goal which enabled the instantiation and was eventually responsible for the creation of the hypothesis. SUB-ELEMENTS and PARENT-ELEMENTS point respectively to constituent hypotheses and integrations of adjacent hypotheses.

In addition to the slots of the **Domain-element** schema, hypotheses inherit slots which depend on their hierarchical relations:

```
{{ Pac
        IS-A: domain-element
        PREVIOUS-PAC:
        NEXT-PAC:
        LEVEL: 0 }}
{{ Ppf
        IS-A: domain-element
        LEVEL: 1
        NO-OF-PAC-SUPPORTS:
        BETTER-HYPOTHESIS:
```

WORSE HYPOTHESIS

PREVIOUS-PPFS

NEXT-PPFS }}

PREVIOUS-PAC, PREVIOUS PPF point respectively to the PAC or PPFs whose signal times terminate at the starting point of the current hypothesis. WORSE HYPOTHESIS points to the ppf (domain element at level 1) schema supported by the same PAC evidence with the next best scoring value (the second best phonetic interpretation for the same succession of signal segments).

{{ **Phrase**

IS A domain element

FEATURES

LEVEL 6 }}

{{ **NP**

IS A phrase

DETERMINER

HEAD }}

{{ **PP**

IS A phrase

PREP OBJECT

PREPOSITION }}

The prepositional phrase and the noun phrase schema [illegible] element as well as those of the phrase prototype [illegible] types differ a prototype schema [illegible]

5.2. Knowledge sources used to partition the search space

5.2.1. Partitioning knowledge sources at the phonetic level

The initial data consists of a sequence of PACs spanning the entire signal. The first step in processing involves generating PPF hypotheses. Examination of HMM models shows that the relative probabilities of phonetic interpretations of strings of PACs greater than four, are virtually insignificant. At the same time the compiled tables required for 5 or more PAC interpretations of phonemes become unmanageably large making them totally non cost-effective. For these combined reasons phonetic interpretations comprising more than 4 PAC segments are ignored.

Given a sequence of PACs the system will generate all possible phonetic interpretations for one, two, three and four segments. For instance given a five PAC sequence {a b c d e} where the the PAC identified as a is the first, followed by b etc...

Possible 1 PAC per PPF interpretations are: $\{H_a\}$ $\{H_b\}$ $\{H_c\}$ $\{H_d\}$ $\{H_e\}$, where $\{H_a\}$ denotes the set of hypotheses which can be generated from the label of PAC a

Possible 2 PAC per PPF interpretations are $\{H_{ab}\}$ $\{H_{bc}\}$ $\{H_{cd}\}$ $\{H_{de}\}$, where $\{H_{ab}\}$ denotes the set of hypotheses which can be generated from the labels of the two successive PACs a and b.

Similarly $\{H_{abc}\}$ $\{H_{bcd}\}$ $\{H_{cef}\}$ $\{H_{abcd}\}$ $\{H_{bcde}\}$ are generated.

These sets of hypotheses are generated by four functions (one for 1 pac hypotheses another for 2 PAC etc...) which iterate through the list representing the successive PACs.

The functions access a LISP hash tables. There are four hash tables, one for 1 PAC interpretations, one for 2 PAC interpretations etc.. . The key to the hash table is formed by catenating PAC labels. The hash tables are built up by scoring every Markov model against a given observation. For example, a three PAC observation [illegible] can be generated by the following state transitions given the Markov models illustrated in figure 4.10

$$1 \to 1 \to 1 \to 3 \text{ or } 1 \to 1 \to 2 \to 3 \text{ or } 1 \to [illegible]$$

The paths can be traced from figure 4.11

$$P(\text{[illegible]}) = \frac{P(\text{[illegible]})\,P(\text{[illegible]})}{P(\text{[illegible]})}$$

As the comparison is made across all [illegible] keeping [illegible] the denominator is eliminated [illegible] is assumed to be the same [illegible]

$$P(\text{[illegible]}) \propto P(\text{[illegible]})\,P(\text{[illegible]})$$

$$P(\text{[illegible]}) = \text{[illegible]}$$

$$P(\text{[illegible]}) = \text{[illegible]}$$

$$P(\text{[illegible]}) = \text{[illegible]}$$

[illegible] The a priori [illegible] phonemes [illegible] English text [illegible]

Had paths longer than 4 PACs been taken into consideration, the compilation of the hash tables could have been based on the Viterbri algorithm which selects the maximal path incident upon a node of the network in figure 4.11, instead of the sum over all possible paths. The Viterbri algorithm gives more or less the same discrimination between models, though with different relative weights.

There are on average about eight phonetic hypotheses for every one, two, three, and four PAC segment of signal. The combinatorial complexity, even at these initial stages of processing, is such that exhaustive processing cannot be carried without reduction of data. It was decided to group similar PPFs into a smaller number of classes (constraints broadening of the label dimension), before generating partitions at the lexical level. The grouping is as follows:

FRICATIVE	<--	{ NC NCS WV VW WF }
RESONANCE	<--	{ V VS VSV VV VSVS SON SV SVS SW SVSV }
SILENCE	<--	{ NI NIS }
FINAL	<--	{ NIF NCF }

During the partitioning phase a function is invoked to group phonetic hypotheses that span the same signal times and belong to the same class. The phonetic classes are then used to generate partitions at the lexical level. The confidence level associated with the partition is the same as the highest confidence level of its member hypotheses.

5.2.2. Partitioning knowledge sources at the lexical level

Generation of lexical partitions is achieved by means of rule based knowledge sources The input elements are partitions representing phonetic classes An output element is generated for every lexical category represented by time adjacent integration of phonemes. In the phonetic dictionary the phonetic description varies from single to six phoneme words. There are therefore six OPS rules that abstract phonetic partitions to lexical ones.

Lexical partitions generated by OPS rules represent lexical classes. A rule which integrates a sequence of phonetic classes that have interpretations belonging to more than one lexical class will generated all the necessary partitions. Because however there may be different sequences of phonetic classes spanning the same segment of signal, there might be several partitions at the lexical level which span the same segment and represent the same lexical category, but which are based on a different [illegible]. The loss of information already incurred by the [illegible] partitioning [illegible] differentiation between constituents with alternative support [illegible]. Further reduction of computational complexity can therefore be achieved [illegible] penalty by grouping lexical partitions belonging to the same category [illegible] the same segments of signal. This task is also performed by [illegible].

The dictionary [illegible] mapping [illegible] package [illegible] the value of a [illegible] representing the phonetic group.

Example

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

5.2.3. Partitioning knowledge sources at the phrasal level

Integration of partitions representing lexical solution elements is achieved by a rule based parser. The grammar used is compatible with the ATN grammar described in chapter four. This rule based grammar however does not apply all the tests and constraints as much of the needed information has been lost at the current (lexical) level. The parser must however generate a super-set of the constituent structures of the ATN. Some of the structures of the rule based parser will inevitably fail if processed by the ATN, because of the tighter constraints. Table 5 3 below lists the rules

NOUN-GROUP	-->	NP
DET + NOUN-GROUP	-->	NP
ADJ	-->	ADJ1
ADJ + ADJ1	-->	ADJ1
ADJ1 + NOUN	-->	NP
DET + ADJ1 + NOUN	-->	NP
NP + PP	-->	NP
VERB	-->	VERB1
VERB + VERB1	-->	VERB1
NP + VERB + VERB	-->	S
NP + VERB + VERB1 + VERB	-->	S
NP + VERB + VERB1 + NP	-->	S
NP + VERB + NP	-->	S
PREP + PP	-->	PP
S + PP	-->	S

Table 5 3 - Context Free Grammar

5.2.4. Flow of control and flow of data during search space partitioning

Each rule is considered a separate (partitioning) knowledge source [illegible] es, with processing at lower levels, the inputs are conjuncts which group individual hypotheses with the same label and spanning the same portion of signal. The outputs are again regrouped before they can be used as inputs for further syntheses. The LEVEL slot of partition schemas determines whether the element can be used as input for a synthesis (partition) knowledge source or whether it can be used as input to a partition merging knowledge source. Table 5.3 lists the partition levels and their function.

Level	Partition type	Used by knowledge source type
8*	Groups all solutions	
7	Constituents spanning entire signal	[illegible]
6*	Syntactic constituent groups	[illegible]
5	Syntactic [illegible]	[illegible]
4*	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]

[illegible]

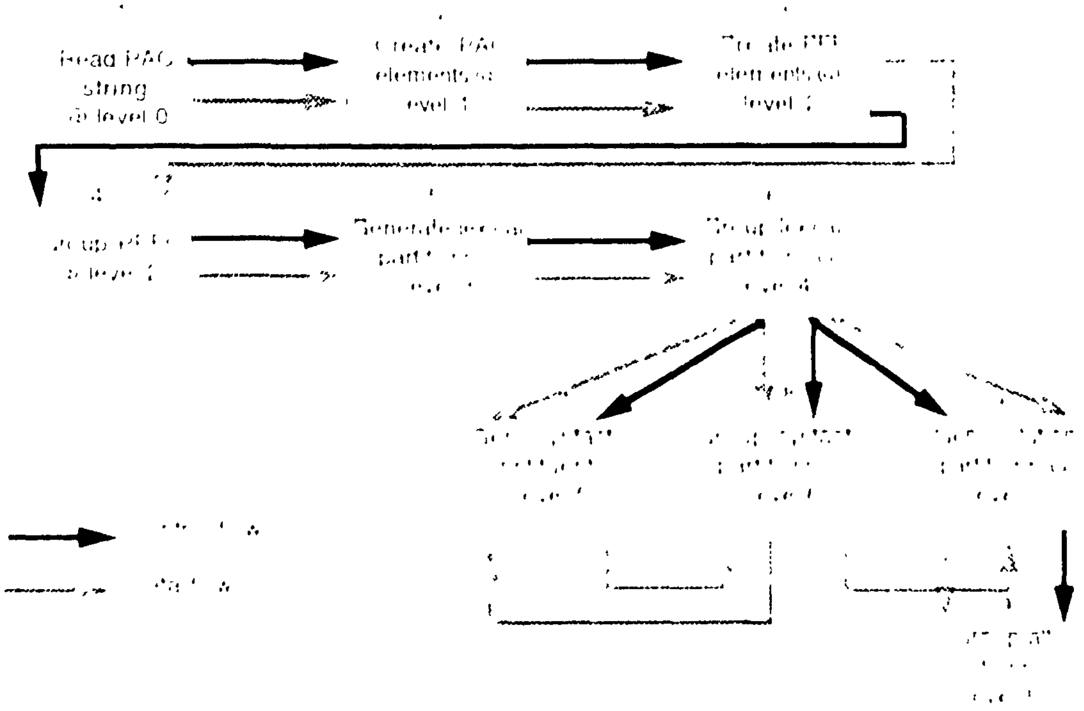
[illegible]

Level	Domain element type	Used by knowledge source type
8	Complete utterance (solutions)	
6	Syntactic constituents (S, NP,PP...)	Inputs to parser or sol. grouping KS
4	Lexical groups (Noun, Verb, Prep,. .)	Input to parsing rules
2	PPFs	Input to lexical retriever
1	PACs	Inputs to HMM hypothesizer
0	Speech-string (initializing data)	Input to PAC schema generator

Table 5.6 Hierarchical Levels and knowledge sources in domain space

Table 5.6 lists the domain element types and their function with respect to domain knowledge sources. Although PPFs are regarded as domain elements for the purposes of this application, The phonetic discrimination they provide is insufficient to allow interpretations of small sub-dictionaries. They should in fact be regarded as partitions within which the search for a particular finer grained phoneme is performed. As is it was not possible to train develop HMMs for more focussed phonemes, it was assumed that PPF hypotheses represent the finest degree of resolution at that level of abstraction.

Figure 5.7 illustrates both control and data flow in the partitioning phase of the solution Steps 1, 2, 3 and 4 are achieved by direct function calls. Step 5 is executed by the enablement of a set of 7 OPS rules running in parallel; six of these rules generate lexical partitions and the seventh carries control to the next step. After step 5 has been completed, the only rule that can fire (the 7th) disables the set of 7 rules (including itself), it invokes a function to group all lexical partitions (effectively completing step 6), and enables a new set of to achieve concurrently steps 7, 8 and 9 Although the flow of control is such that the three steps are enabled concurrently, the parsing rules require input data that has been grouped to eliminate redundant partitions. The input data can consists of lexical categories (grouped) at level 4 and syntactic constituents (grouped) at level 6. The output data is either larger syntactic constituent (ungrouped) at level 5, or a complete sentence spanning the entire signal (ungrouped) at level 7. Processing up to level 4 proceeds incrementally up the level hierarchy, levels 6,7 and 8 interact in a data dependant fashion, hence the flow of control organization. The last rules to fire in that set disables the parsing productions and transfers control to the rule which combines all solutions (the root of the AND/OR graph) and enventually terminates the partitioning phase.



[illegible]

[illegible]

that must be enabled together (including the rule which disables them) specify the same value for "current-step". The rules which perform data transformations, for instance synthesis of PPF groups to lexical partitions, will contain one or more additional condition elements (in the LHS). The specificity condition of OPS (described in chapter 2), ensures that the instantiations associated with rules with more complex LHS condition elements, are selected first. The simplest rule which has only the STEPPER condition element, will fire last and its RHS modifies the value of the STEPPER schema thereby enabling another set of rules.

Once the partitioning of the search space into an AND/OR graph of partitions is completed, all partitions not connected to the root node can be eliminated. These occur when syntactic constituents parse correctly but cannot account for the entire signal. The blackboard will then commence the planned search for a solution selecting compatible sets of partitions to investigate

5.3. Domain knowledge sources

The shortcomings of this application are primarily due to the weakness of domain knowledge sources. Effective control in searches requires a problem solving framework that allows the expression of every modicum of application related knowledge, and a program organization that integrates effectively this information, to optimize problem solving actions. Evaluation of a problem solver requires not only a complex problem but also multiple sources of information, evaluation techniques and a diversity of methods for building solution increments. The problem of speech recognition and understanding is certainly complex enough, interpretation techniques and analysis techniques abound, but their incorporation and evaluation requires thorough familiarity with the field and very broad based expertize. This problem was underestimated initially and the result is that what knowledge sources were incorporated are insufficient to test fully the features of the problem solver.

The blackboard was designed to support island driving. This type of strategy is advantageous as interpretation tasks are bound to span partitions of uneven certainty. Island driving however, requires reasonably meaningful certainty factors at every level where islands are built up. With the knowledge available certainty factors are reliable at the phonetic level, but there are no reasonable evaluation heuristics at the lexical or phrasal level. Propagating the values of lower level hypotheses to the higher levels, dos not contribute additional information about newly synthesized data.

The initial survey of literature confirmed that island driving was very common in speech and signal processing systems employing blackboard type architectures. This fuelled the motivation to adopt this approach without proper consideration of the knowledge sources required to test it. In the end the implementation was modified to some extent to accommodate a functioning of the available knowledge sources.

5.3.1. Domain knowledge source for parsing

The original intention was to implement an island driven ATN parser similar to that described in [BATES75]. These parsers given an initial good word hypothesis as an anchor point, retrieve the lexical categories from a grammatical dictionary. For each category the sub networks in which it might occur are retrieved.

It can be shown that there exists a context free grammar for every RTN (Recursive Transition Network) grammar [WINO83]. Similarly the augmentations of ATNs have their counterpart in Augmented Phrase Structure Grammar (APSG). It is possible to express an augmented phrase structure grammar which is strongly equivalent to the ATN, that is the structures the parser generates are the same. In partitioning the search space at the phrasal level a context free grammar was used. As the structure generated during the partitioning of the search space is used as the basis for control in the planned search phase, it is imperative that the structures generated by the two parsers, for a given string, be identical. For this application, the design alternatives were to use an RTN for partitioning, and an ATN for the detailed search, alternatively to use a CFG parser for partitioning and a strongly equivalent ATN for the planned search, or finally to use a CFG for partitioning and an APSG for the detailed search. The CFG based parser is advantageous for partitioning because it is readily implemented using the existing OPS engine. For the second phase of processing, the constraints of natural language require either an ATN or an APSG based parser. Of the two, the ATN can exploit the control properties of the blackboard more fully. When implemented for island driving, the control decisions required of an ATN parser are the selection of the edge to develop (left or right) and the selection of arcs, at nodes where a choice exists. This is exactly the type of decision performed by the blackboard's planner. The disadvantage of an island driven ATN however, resides in the comparative complexity of its implementation and in the uncertainty that the structures generated will map in to those of the partitions The ease with which the augmentations can be incorporated into the the CFG parser, using the extended OPS available with KnowledeCraft favoured the implementation of the APSG parser.

Rule based parser consume all the elements of a phrase simultaneously, rather than element per element. The blackboards planner was designed to provide a finer level control. A fully refined plan specifies the set of adjacent partitions which must each contain at least one solution element in order to attempt the parsing of a grammatical constituent. The planner described in chapter three however, only refined its plans to the extent of one decision point before invoking domain knowledge sources. The modification that were made to the design of the blackboard are described in the next section.

The left hand side tests of the rules used to implement the APSG parser, are a super-set of the conditions used for the CFG parser. There is a one to one mapping between the rules in each parser. To create a rule for the APSG the corresponding CFG rule is taken The augmentations are found by taking the appropriate path in the ATN network and applying all the tests and actions as augmentations For example the path (f, g) in the NP network of figure 4.14, specifies that the number feature of the Noun Phrase, is set to the number value of the determiner consumed by arc $_f$determiner$_g$ The second arc $_g$Noun$_h$ requires that the number register of the NP be either NIL or match the number of the noun word consumed. Because of the action of the first arc the net constraint on the path is that the the number of both the determiner and the adjacent noun must concord. The noun phrase resulting from a successful parse will assume the same number feature. The tests and augmentations only restrict the construct that might be generated (with respect to the CFG grammar), there is no risk of generating a new construct. Augmenting the the CFG parser is a very straight forward task and avoids the risks of using grammars that are not completely equivalent.

5.3.2. Domain knowledge source for lexical retrieval

The input to this domain knowledge source is a string of PPF symbols and the output is one or more words. As with partitioning knowledge sources, the phonetic dictionary is stored as strings of PPFs which evaluate into a list of candidate words within a dedicated package. This method of storage is incompatible with true island driving, as the entire string of PPFs must be available to generate lexical hypotheses. A discrimination net with variables provides an easy way to retrieve all hypotheses consistent with an arbitrary sub-string

The grammatical dictionary which is much smaller, containing some 50 words, (the phonetic dictionary contains some 7000 words) is stored in a hash table, its key is being the word

Example

```
(gethash 'fish *gram-dict*)

-> ((NOUN (NUMBER (SINGULAR PLURAL))  (CASE (SUBJECTIVE OBJECTIVE)))
    (VERB (TRANSITIVITY INTRANSITIVE))  (FORM (INFINITIVE PRESENT)) ) )
```

The outer list contains sub-lists, each of which is headed by a lexical category (noun, verb, adjective etc .) Within each lexical category, every dimension is expressed as a sub-list whose head is the dimension, and whose tail is the value of that dimension. The value of a dimensions is an atom if it is single valued and a list if it is multi-valued. Although it was suggested in the fourth chapter that default values for the dimensions of each lexical class should be kept in a table to minimize storage. In practice because the grammatical dictionary had few entries, and it was more difficult when compiling it to remove entries for default values, all the properties were entered

5.3.3. Domain knowledge sources for generating hypotheses at the phonetic level

It would have been preferable to use HMMs trained on the broader classification of PPF groups for partitioning and to use finer grained HMMs, including models for vowel recognition to perform the domain search. For a fixed size training set, models based on a broader phonetic classes would have yielded better discrimination. Similarly finer grained classification for the planned search would have resulted in fewer and more precise solutions As however, training HMMs was not possible, the lowest level partitions were generated artificially by compressing phonetic (PPF) data.

5.4. Implementation of control and planning

5.4.1. Departures from the original blackboard design

While the use of rule based triggers for knowledge sources, imposes some changes on the synchronization of the blackboard, the modifications are less drastic than might be expected From figure 3 8 one can see that only after a plan has been fully refined, in other words when the sub-partitions it spans, encompass the signal time of the parent node, can a domain rule be triggered at that level Executing a refined plan ensures that the solution elements required by conditions of the knowledge source, are generated

A capacity for testing partial triggers (a subset of the conditions of the knowledge source), would enhance the performance appreciably With the blackboard functioning in beam-search mode, the system could detect failure or perhaps a drop in the survival rate of solution islands, and could consequently repair the plan by broadening the beam Similarly if the system were required to search exhaustively target conjuncts, early detection of plan failure would save much needless searching in the lower sub goals.

Testing partial triggers, provides a capacity for interleaving plan refinement with plan execution Reducing this capacity, diminishes the amount of information that is available to control which could be used to influence its decisions One could envisage more sophisticated domain knowledge sources, which exploit information available to control through the execution of partial plans, to generated more appropriate hypotheses

The reduction efficiency which results from coarse sequencing of plan-elaboration/plan-execution, is to a certain extent minimized because partial paths are shared by competing plans. This characteristic is emphasized because plan development is heavily biased by the policy of least commitment.

5.4.2. Control interface with the OPS engine

Any architecture built on top of an OPS engine, which imposes an alternative control discipline, must program its inference cycle in terms of a number iterations of the basic engine's interpreter. This does not allow optimal low level efficiency. More recent implementations of the Rete algorithm (OPS83) provide the programmer with information on instantiations, and allow the selection of the rule to be fired This blackboard was originally developed with an older version of KnowledgeCraft which only provided the two traditional OPS5 conflict resolution strategies A newer version of the software now provides some information on the conflict set and allows the user to customize the conflict resolution strategy, but it still does not provide the full functionality of OPS83 As the blackboard was modified to run with the new software a description of both interfaces follows

5.4.2.1. Interface with the older version of OPS5

Two additional data structures are required for this interface.

{{ **Instantiation**

STATUS:

CYCLE-NO:

ASSOCIATED-GOAL:

LEVEL-OF-ASSOCIATED-GOAL.

CONFLICT-SET-TO-WHICH-INSTANTIATION-BELONGS:

KNOWLEDGE-SOURCE:

INSTANTIATION-SELECTED-P. }}

```
{{ Agenda
        CYCLE-NO
        CONFLICT-SET
        SELECTED-INSTANTIATION }}
```

The knowledge sources are coded using two rules The first tests the conditions that make the knowledge sources invocable and returns this information to the blackboard's control, while the second allows the control system to invoke the selected instantiation

The LHS of the first rule of a knowledge source contains all the conditions pertinent to the knowledge source itself The GOAL condition ensures that the knowledge source remains dormant until the blackboard requires its output, and creates a goal schema instance with specifications that correspond to the desired output partition When such rules fire, the system is informed of the knowledge source instantiations that can potentially fulfill a data processing requirement This information is passed to the blackboard's control by the creation of an instance of the INSTANTIATION schema containing all the relevant information (knowledge source name, goal schema that the instantiation satisfies, input data reliability and efficiency parameters were they are relevant etc .) This schema is created by the RHS of the first OPS rule Because the system must know all the instantiations that are available at every blackboard cycle, the OPS engine is allowed to fire all the first rules of the knowledge sources together with all the bindings that exist (OPS instantiations)

The blackboard must then consider its processing priorities Pointers to the instantiation schemas are stored in an instance of the AGENDA schema Such an instance is created at every blackboard cycle, and is retained for tracing purposes The best instantiation in the current agenda is selected by setting a boolean attribute

The second rule of each domain knowledge source has a condition element that requires this attributes value to be true. Since only one instantiation is set at every cycle only one invocation of a knowledge source is performed per blackboard evaluation cycle. The organization of the systems allows this evaluation to be performed on any basis Meta rules could be used for evaluations, and there is no fundamental reason why only one instantiation should be selected at every cycle. In fact, evaluation should be regarded as a costly blackboard function, to invoked frequently, only when major changes of state are expected.

From the point of view of implementation, the blackboard has no means of determining from cycle to cycle, what instantiations have become invalid as a result of the most recent knowledge source invocation. All currently valid instantiations, must be regenerated at every cycle. OPS's refraction strategy however inhibits repeated firings of a rule with the same set of parameters. A change of value of one attribute in a condition element however, constitutes a new set of parameters and permits invocation. Each rule in the system contains a STEP condition element which is used to control the blocks of rules that are enabled during any OPS cycle A single instance of the step schema is created for the entire run. During the partitioning phase, this step schema is used to enforce flow of control when rules are invoked. During the planned search, the system alternates between the generation of instantiation schemas and the invocation of knowledge sources. The step schema enforces these changes. By changing the attribute value of the step schema the rules generating the instantiation schemas are "refreshed" and can fire again at the next cycle with the same domain parameter, when the latter are still valid.

5.4.2.2. Interface with the newer version of OPS

The OPS engine in the newer version of KnowledgeCraft provides a function for determining a number of parameters of every instantiation including overall specificity, recency and the rule name There is no information on the working memory (input)elements involved It is also possible to define a conflict resolution strategy which is based on the available information

In the newer version of the blackboard a knowledge source Is represented by a single rule The condition elements are based on the goal element and the partitions which encompass the knowledge source's input data and its output There is no explicit reference to domain elements To prevent a rule from firing without there being any domain elements to provide input data the OPS engine is allowed to "see" the slot HAS-DOMAIN-MEMBERS of the partition schemas Unless this slot is non nil for every input partition the rule cannot fire. The creation of any new domain elements refreshes this slot and allows the rule to fire without further interference from the blackboard. The system can however monitor the progress of solutions by reading the same slot in the partitions, and take appropriate actions should the situation require it Since the invocation of a domain knowledge source si a one rule process, there is no need of the STEP condition element, all the control is performed by the GOAL schema. The simpler format of theses rules (there is no reference to domain elements), reduces the time required for pattern matching considerably

5.4.3. Knowledge source enablement priorities

The objective of the problem is to generate solutions satisfying constraints at the highest level on the blackboard Generally, the higher level the solution element, the greater the integration of partial solutions and the closer to the main goal A top level solution element corresponds to a very small set or possibly even a unique set of lower level hypotheses. The generation of large number of hypotheses is associated with the uncertainty and weakness of the knowledge sources. The system must always remain responsive to the primary objective which is the top level knowledge source.

During the top down generation of solution elements, it is conceivable that the planner might attempt to keep generating lower level solution elements to satisfy its targets for each partition while there might already exist a competing top, or at least higher level, instantiation. For this reason competing instantiations in an agenda, are always weighted to favour higher level invocations The information given by the structure of plans could be even be used more efficiently to favour parent hypotheses over their children but perform meta level reasoning when comparing hypotheses not connected by direct ancestry

As the interface with the newer version of OPS does not include an explicit knowledge source instantiation selection cycle, the conflict resolution strategy is programmed to favour rules with a lexically smaller name. The user then ensures that rules are named appropriately.

5.4.4. OPS efficiency considerations

The OPS engine was designed to solve problems in which the transformation expressed by the action (RHS) part of rules represent small changes of state. In fact the efficiency of the RETE algorithm rests on the fact that these changes are incremental and only partial re-evaluation is required from cycle to cycle. The blackboard on the other hand shifts its focus of attention by means of goal schemas at every cycle thus forcing extensive changes in the RETE network. This probably explains why none of the blackboard architectures described in the literature appear to be built on top of OPS

5.4.5. Performance and results with speech application

The results plotted were obtained by processing a sixteen segment (PAC) utterance created by the catenation of signals labels used in single word spotting experiments. The dictionaries used were restricted to the samples of words available from the word spotting data files (some 70 of them) The blackboard was set to search the selected partitions exhaustively

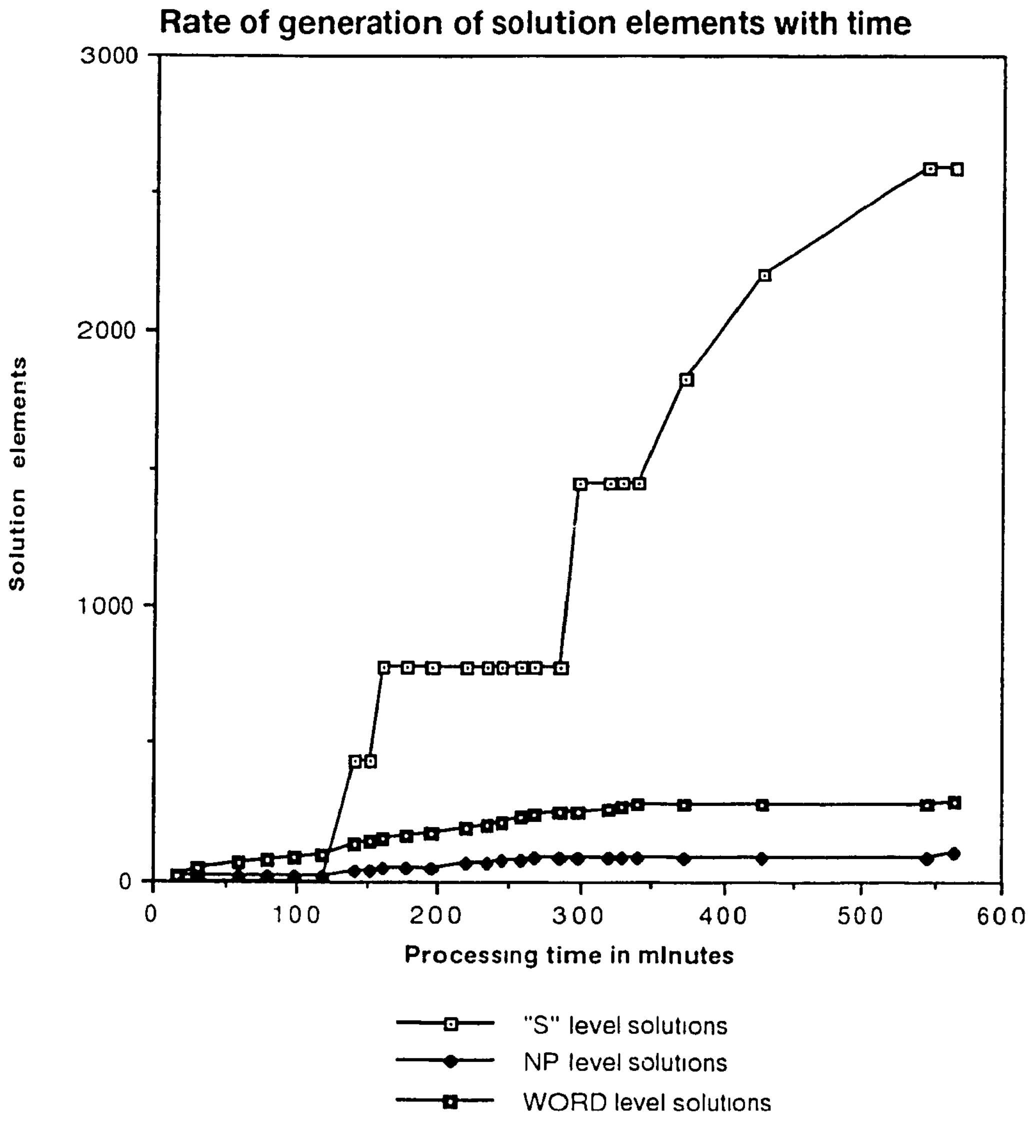


Figure 5.8 - Graph of solution elements generated with time

Integration of lower level solution elements into overall solutions

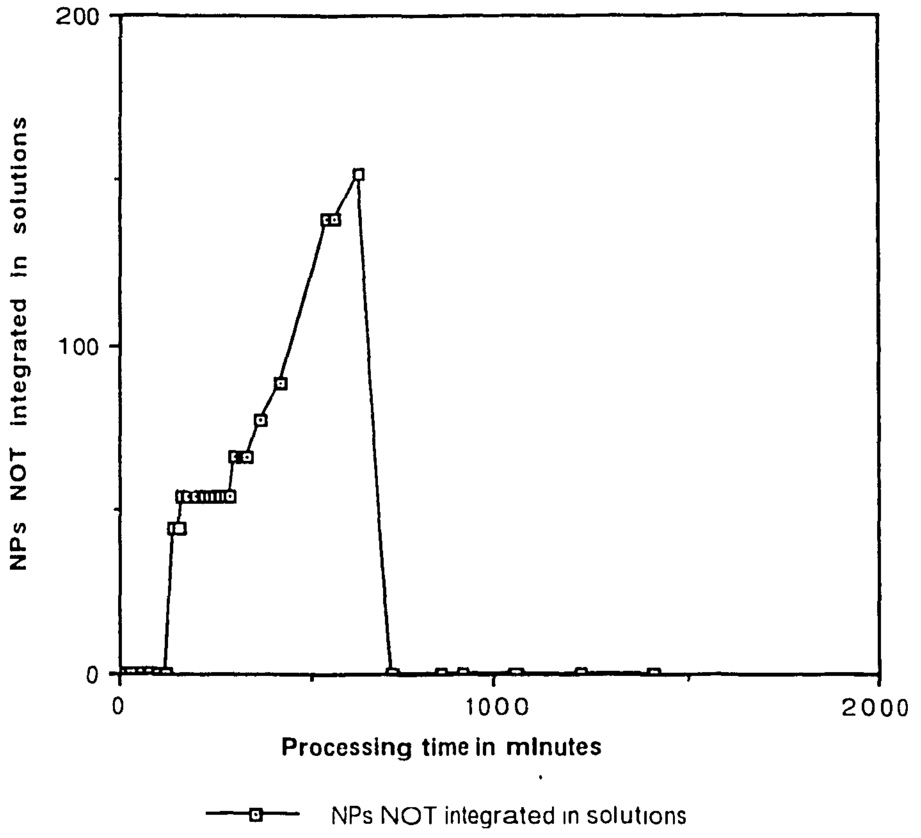


Figure 5.9 - Number of partial solution elements which failed to integrate VS time

5.5. Suitability of the blackboard organization for the application

5.5.1. Relationship between the application and the problem solving architecture

The majority of blackboard systems that are documented, were designed and optimized around a particular application. The developments of Hearsay II [LES&ERM&RED74] [LES&ERM77], HASP/SIAP [NII&FEI82], and CRYSALIS [TERRY83] were each designed to solve specific hard problems. With the exception of CRYSALIS the problems incorporated a considerable amount of "expertize". The performance was attributable to the knowledge intensive nature of the system rather than refined computational efficiency. The momentum for blackboard architectures grew out of the realization, that difficult problems could solved, with proper use of control information. Through the historical line of development, one can trace a two fold development. On the one hand increasingly complex mechanisms to support on the fly analysis, and evaluation of processing options, and on the other growing sophistication in the formalisms to express knowledge and the reveal structural properties of problems.

Innovative architecture design is spurred by the untapped wealth of problem reducing knowledge. The relationship between the domain and the programing environment is symbiotic. A powerful and flexible environment promotes the formulation and integration of available knowledge, and application knowledge itself, provides the necessary pointers and clues to the system's designers. Given a refined and optimized system, it is then possible to generalize the constructs it provides to accommodate alternative applications

For this project the situation was completely reversed. There was a requirement for an architecture to solve that was not yet properly formulated. The data to be represented and processed had not yet been specified, the structural relations in solutions elements could therefore not be expressed and the processing methods were totally unknown (to us at any rate).

The need for a precise problem of adequate complexity, was answered by the speech application. The task had been attempted before using a blackboard (HEARSAY II), and some of the knowledge sources could be coded relatively quickly. The drawback however has been the poverty of knowledge both for control and in the domain. The redundancy in processing techniques which so greatly assists in reducing complexity, was in this case, completely absent.

The problem is itself artificial as there are no physical means of generating the continuous data required as input. Test data has been constructed, by catenating signal segments from files used for discontinuous word recognition. The knowledge sources that reduce complexity such as prosodic analysis, expectations of words, context, evolving models of the world which can be used to eliminate some hypotheses, are all absent.

The awareness that such knowledge sources exist in all applications justifying the use of blackboards, has resulted in provisions being made for such knowledge sources at several levels in the problem solving process. The the control mechanism is built upon an AND/OR decomposition of the task, which is a very general and domain independent representation of problem solving processes. In addition the system can exploit parallelism as well as cope with interactions between conjuncts. The generalized features of the problem (those actually implemented and easy extensions) will be discussed in a latter section of this chapter.

5.5.2. Suitability of architecture for parsing

Having suggested the features that make this architecture of interest as a general purpose problem solver, one should evaluate its suitability for the type of task, around which it was built. The problem posed by this application of speech recognition, is an integration of two tasks quite commonly encountered in AI, namely parsing of natural language and data synthesis.

To evaluate this problem solver's capacity to support non-deterministic parsing, we should examine the most common types of parsers, the software characteristics that are essential to their function, and the computational schemas that enhance their performance.

5.5.2.1. Parsers based on context free grammars

Non-determinism in context free parsers occurs not only because there is a choice in the rules to select at every stage, but because there are a number of dimensions in processing which must be decided, and u⌐on which the efficiency of processing dependents. A basic parsing procedure can parse in parallel or sequentially (depth first, or breadth first). Processing can be top-down or bottom-up. Given any of the above combinations there still remains the choice of nodes to expand within a rule. Selection of nodes can be strict left-to-right, right-to-left or island driven.

The blackboard itself allows all these degrees of freedom. Top-down processing is achieved by writing knowledge source preconditions in terms of the LHS of parsing rules, bottom-up achieved by using the RHS of rules as preconditions (knowledge sources written both ways are required to for bi-directional processing). The choice of nodes to expand is entirely within the control of the planner and it can be fixed or decided dynamically.

Generally top-down parsing ensures that nodes that are processed fit together in the overall structure, while bottom-processing ensures that the structures that are built up, are complete at the lower level. A backtracking search often requires re-parsing a common constituent encountered in several paths, while parallel processing can overwhelm resources. Active chart parsers combine the best of all strategies by parsing every constituent only once and processing only relevant nodes.

The blackboard itself provides a sophisticated active chart by retaining all processed "well-formed sub-strings"; the system will actually block any attempt to reprocess a sub-strings with the same rule while at the same time making the result globally available. The plan generator fulfills all the functions of proposing pending edges and combining completed ones, but it is also able to perform more judicious selection by analyzing the global consequence of its choices In addition the planner is able to respond to external constraints discovered form levels of reasoning beyond the functions of the parser.

In the absence of expectations or knowledge about the problem, decisions between competing hypotheses of equal value can only be made arbitrarily. However when a parser is used in speech recognition and understanding there is a level of processing hierarchically below the parser (as was the case with the application used in this project), and there would be a level of processing above the parser which utilizes the tree structures that are generated, to extract meaning, and modify the system's model of the world. A system starting with a model of the world (as a higher level hypothesis) and with some knowledge of what constitutes feasible, or rational changes of state, has the capacity of discriminating between different structures that could be generated by the parser.

5.5.3. General adaptability to other applications

In order to claim that this architecture is generally adaptable to a broader class of problems, it is interesting to consider whether the problem solving principles employed can be adapted, to model effectively, seemingly different applications. A quantitative evaluation is of course, completely out of the question. All that is attempted is a qualitative representation of the problem using the constructs and the paradigms described earlier.

The application considered is taken from the literature and was solved using one of the earliest blackboard architectures. The system published under the name of HASP/SIAP [NII&FEI82] was developed for the US department of defence.

The task was to develop and update a "situation board" to show the positions of all naval vessels. The system receives data primarily from hydrophones placed at several locations in the oceans but also intelligence reports from multiple sources. Acoustic data is fuzzy with a low signal to noise ratio and can be misleading because of echoes from uneven ocean floors and acoustic camouflage of military vessels.

The database is organized into hierarchies, at the lowest level the hypotheses are acoustic segments, then lines, harmonic sets, sources for acoustic data, platforms (vessels), and fleets Hypotheses are instances of prototypes of different levels A hypothesis at the highest level in the HASP system is a snapshot of the entire board, identifying and locating all vessels at an instant in time. The system concentrates its efforts on one global hypothesis called the CBH (Current Best Hypothesis). The CBH evolves with time, according to expectations (such as anticipated movements intelligence reports etc) and events (such as new signal detections). While local hypotheses are allowed attributes with multiple values to account for uncertainty, the parent hypotheses encompasses all allowed variations in the children; the resulting graph is composed of AND arcs but no ORs. Hypotheses that conflict with their parents are ignored unless or until the evidence becomes overwhelming forcing a reconsideration of the CBH.

Most of the knowledge sources (there are some 50 of them) perform bottom up synthesis, although some of them reason from higher levels to lower or modify hypotheses at the same level of abstraction.

The system has two modes of reasoning:

1) Reasoning based on *data events* prompted by occurrence of incoming data (time stamped). The incoming data is mostly acoustic evidence but can also include higher

level (intelligence) reports All incoming data is stored in an event list. Event based (data driven) reasoning involves bottom-up processing.

2) *Model based reasoning* initiated by the CBH The CBH at any time incorporates velocity vector for the identified vessels. This information is used as a basis for predictions and expectations as platforms are expected to pass close to hydrophones. Tests and verifications can be scheduled in the problem list.

Event based processing is controlled by a special (control) knowledge source called Event Driver, and Model based processing is controlled by an equivalent Expectation Driver A higher level control knowledge source called a Strategy knowledge source selects the mode of reasoning

The Event Driver if invoked (by the Strategy Knowledge source) selects the instantiation (data item and the knowledge source to process it). The Expectation Driver when invoked scans the Problem list for the most useful candidate that can be tested at that stage in time

Solving the HASP problem is a matter of extending the CBH correctly over time The criterion for correctness is assessed in terms of consistency between perceived (and interpreted) data over time on the one hand, and plausible tracks for vessels on the other. This task of reconciliation is embodied in the Strategy knowledge source. Difficulties arises because the signal to noise ratio is low and some uncertainty exits exists over interpretations of signals. This means that the tracks of vessels can be lost, or become fainter and misleading over periods of time. When traversing such regions of uncertainty, the extension of a track is very prone to errors.

Again the advantages of island driving, manifest themselves in a very concrete manner. It is much easier to *extend* a track starting from a known path, with the support of acoustic or other evidence, than it is to develop an entirely new one, based entirely on sensory data. The history of state transitions imposes constraints on plausible and possible alternatives. The blackboard's plan construct, can be used to represent changes of states occurring at discrete points in time This representation is the same as the concept *chronicles* used in planning [McDER82] [McDER85], to represent states during which physical parameters are invariant in time, and state transitions as points in time where chronicles change [PEDN86] In fact the HASP problem bears a relationship with a conventional planning problem In traditional planning the problem solver has definite goals (state specifications) and attempts to achieve them through a repertoire of actions. In HASP the problem solver is a passive observer of plans from an obscured viewpoint Its objective is to try to determine the sequence of actions Weak knowledge of potential goal (inferred), and the interactions of plan steps, is used to supplement noisy observation data.

Figure 5 9 shows a component of the CBH (based on an example given in HASP's description). The complete description of the corresponding HASP data structure is given in figure 5.10

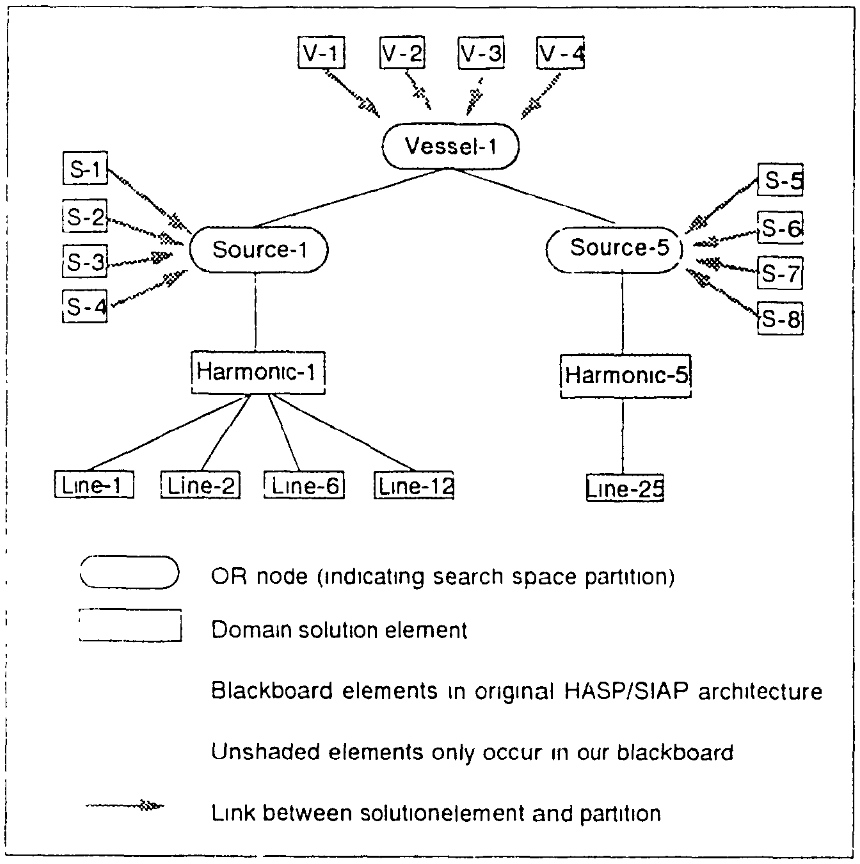

V-1
V-2
V-3
V-4
Vessel-1
S-1
S-2
S-3
S-4
Source-1
Source-5
S-5
S-6
S-7
S-8
Harmonic-1
Harmonic-5
Line-1
Line-2
Line-6
Line-12
Line-25
OR node (indicating search space partition)
Domain solution element
Blackboard elements in original HASP/SIAP architecture
Unshaded elements only occur in our blackboard
Link between solutionelement and partition

Figure 5.9 - Blackboard objects with HASP architecture and with proposed representation

The data generated by HASP in figure 5.9 is interpreted as follows.

Acoustic data represented by Line-25, which is independent of any previous acoustic segment, is therefore assigned to a new harmonic group, Harmonic-5 This is in contrast to Line-1, Line-2, Line-6 and Line-12 which are successive segments (5 second sampling) of the same signal or harmonic group, Harmonic-1.

A source level hypothesis associates one (or more) generators to the signal. According to the examples given, there are three possible generators of acoustic signals. Propellers, propeller shafts, and the reciprocating parts of engines. HASP/SIAP's knowledge base contains data and programs that enable the classification of signals according to the propellers, shafts or engines of each class of vessel. A source level hypothesis associates the generation of a harmonic group with a particular vessel's component.

At the vessel level the system has identified a platform's class, its location, its course and speed, and is therefore able to extrapolate its path into the future.

At the fleet level, all vessels have been located and their displacement vectors are known with certain tolerances. The CBH (Current Best Hypothesis), is the best fleet level hypothesis at a particular moment of time.

In addition to signal recognition programs, the HASP/SIAP system contains information about shipping lanes, geography, military logistics, performance and limitations of vessels, it is provided with additional intelligence on naval traffic from alternative sources and is capable of performing some common sense reasoning, about the continuity of ship's tracks, etc.

CBH at time 20455

Vessel-1

CLASS: (OR cherry 8.4) (iris 6.9) (tulip 6.2) (Poppy 4.8) 20455 ...)

LOCATION: (lat 37.3) (long 123.1) (error 37)

SPEED: 15 7

COURSE. 135.9

SOURCES: (AND Source-1 Source-5)

Source-1

TYPE: (OR (cherry propeller 5.5) (poppy shaft 2.5)
(poppy propeller 2.0) (cherry shaft 2.5) 20455)

DEPENDANCY: Unknown

REGAIN· (20230)

HARMONICS. (Harmonic-1)

Harmonic-1

FUNDAMENTAL: (224.5 20520)

EVOLUTION· (fade-in 20230 fade-out 20210)

LINES (AND Line-1 Line-2 Line-6 Line-12)

Source-5

TYPE: (OR (Cherry shaft 6 0) (Poppy shaft 4 0) (Iris propeller 5 0) (Tulip propeller 2.0) 20455)

DEPENDANCY· 6

HARMONICS: (Harmonic-5)

Harmonic-5

FUNDAMENTAL: (162 4 20455)

EVOLUTION: (fade-in 20455)

LINES. (AND Line-25)

ASSIMILATION: (RATIO Source-1 Source-5 .5) 20455)

Problem-list

(EXPECT Vessel-1 (SUPPORT cherry (Dependency propeller 5))

(EXPECT Vessel-1 (PRED.LOC (Lat. 37.2) (Long 123.) (Error 41.3))

```
(REPORT REPORT-GEN  Rose (Signature (engine 30 166.7).... ))
```

Figure 5.10 - Solution elements in HASP problem

The authors of the HASP/SIAP state that the system must decide whether to execute computationally costly but accurate identification programs or whether to select less accurate, but more expedient knowledge sources. This suggest that the system is not only saturated by data, but also by procedures and knowledge sources. Despite this the control system appears very poorly developed. There is mention of the strategic knowledge source which decides whether to proceed with model driven processing (top-down), or whether to proceed with data driven processing (bottom-up). Based upon this decision, a specialist knowledge source selects the most beneficial instantiation from the appropriate conflict set. There appear to be few special control data structures that are needed to build a layer of control reasoning, and to reflect the dependencies of alternative processing decisions.

The HASP/SIAP system interestingly enough performs a similar data reduction to the one proposed in the previous chapters. The example data of figure 5.10 shows that the attribute values for sources and vessels are multi-valued (with an OR) to reflect uncertainty in data processing. These data structures are reminiscent of the search space partitions described earlier. They allow for several alternative hypotheses shown in figure 5.9 unshaded, but not represented in HASP. The authors were probably compelled to process sensory data under relaxed constraints to avoid overwhelming the system.

Top-down processing options represented by HASP's PROBLEM-LIST (partly listed in figure 5.10), specifies the task of searching for a precise hypothesis corresponding to Vessel-1 ({EXPECT Vessel-1 (SUPPORT Cherry) (Dependency Propeller 5)....}). The sparse data structure, does not express how this task is to be achieved, although much of the work required to express this has already been performed.

The representations of HASP leave many questions unanswered.

- Where should the supporting hypotheses be searched for in the search space?

- What are the alternative hypotheses than Cherry class?

- Are the alternatives consistent with the past track?

- If further evidence for this hypothesis cannot be found in the immediate future time segments, at what stage would one expect an opportune situation for disentangling acoustic signals (a time frame where competing signals would be less concentrated?

- The low signal to noise ration, coupled with the intent on stealth of military vessels, means that platforms will pass through partitions where they are more weakly sensed. At what stage in time is a hypothesis, that is not showing promise definitely eliminated from further consideration, and if it is not eliminated where and which signals should be further processed to help reaffirm the hypothesis?

- If processing performed in support of one hypothesis failed to substantiate it what are its implications for other hypotheses?

Answers to these questions, would greatly enhance the performance of the HASP. The system could easily incorporate this knowledge, and some of the information is extractable from already existing data structure. Unfortunately the HASP architecture does support the representation of this information explicitly. The application programmer would have to hard-code routines to generate such information. Such a recourse would make it difficult for the system to make use of the additional information to improve its performance.

The architecture used for speech recognition can, with a few modifications, be adapted to process a continuous stream of data, and accommodate the constraints of real time processing An incoming signal queue or buffer is needed to hold time stamped acoustic data until resources can be devoted to their processing. All incoming data is processed under relaxed constraints to generate search space partitions. These partitions specify time slices, and geographic areas at all hierarchical levels. Relaxation of constraints on the "label" attribute (by grouping similar classes of hypotheses) at the higher levels of abstraction, is probably an essential aspect of partitioning in this problem. This appears to be verified by the approach adopted in the original HASP/SIAP system. Rather than associating each label with an independent hypothesis, the original system allows label attributes to assume multiple, competing values. HASP reasons initial on the basis of the best (most likely) value. Determining whether partitions with relaxed constraints are useful at the lower (signal) levels of abstraction, requires a more thorough understanding of the signal processing front-end and the available processing resources.

Re-organization of the blackboard to accommodate the planning approach used in for our speech application is to some extent dependant on the dominant reasoning approaches of the HASP domain.

The organization of the blackboard at the higher levels of abstraction in tables 5.11 and figure 5.12 puts the emphasize on different domain dependant properties. The hierarchies of both schemes, from the vessel level down, are the same.

Figures 5.13 and 5.14 represent sets of hypotheses at every level, according to the blackboard organizations of tables 5.11 and 5.12 respectively. Processing decisions taken at high levels have a much greater impact than those taken at lower levels. The task of the planner is to select one partition from a set of candidates for each one of the conjuncts at a particular level. within each of the partitions a solution element must be found such that the complete set can integrate into a hypothesis at the parent level. The planner is successful to the extent that it can select a compatible set of partitions that will yield lower level solutions that can eventually be integrated. The organization of table 5.11 and figure 5.13 gives a different viewpoint than that of table 5 12 and figure 5.14 The partitions of the highest level plan involve alternatives for the track of each fleet. the planner task is to select a set of alternative search space partitions that is mutually compatible. The decisions made by the planner using this organization, will be founded on the plausibility of a particular partition for the track of one fleet, given that another partition has been selected for search of another fleet. For instance, if the planner has a solution for fleet A which is located in partition 1, it is about to select a partition for fleet B. A candidate partition 3 which might have been selected as the region containing the track of fleet B, might be rejected on the grounds that tracks encompassed by this region, are incompatible with the tracks encompassed by previously selected regions such as partition 1. Since partition 1 and partition 3 cannot belong in the same processing plan, two alternative plans must be elaborated and compared for plausibility. The grounds for selecting a blackboard organization are dependent on the capacity to discriminate between compatible sets of conjuncts as well as the availability of meta-knowledge to evaluate the plausibility of competing plans. Looking at the next level down in the hierarchy defined by table 5.11 and figure 5.12, we have conjuncts consisting of the tracks of individual vessels within each fleet. At this level the planner is required to select compatible regions each representing coarse descriptions of tracks of individual ships within the same fleet. Knowledge of the functions of vessels, within a moving fleet can be used to decide the set of compatible partitions.

The organizations of blackboard defined by table 5.11 and illustrated by figure 5.13 represents global snapshots. At the highest level, the conjuncts represent snapshots of *all* the fleets at [illegible] successive instants in time (or within a comparatively short time interval). The level below contains conjuncts, each representing a snapshot of a fleet, all of which are taken at the same time. A plan at the top level attempts to combine a set of regions each represents a coarse description of the possible situation board at an instant in time. The criterion for combining conjuncts (regions) into a plan and rejecting other combinations is based on the legality or the possibility of successive regions. For instance having selected one regions to represent the configuration of fleets at time t_n, and perhaps having found viable hypotheses within that region, the planner might reject a candidate region for time t_{n+1} on the grounds that the changes with respect to the previous conjunct, imposed by the regions parametric constraints are too considerable, given the elapsed time interval.

Because the second scheme breaks hypotheses down into conjuncts which are time dependent, the plans that it generates (at the top two levels) are formed on the basis of legal time dependent changes. The first scheme by contrast decomposes hypotheses into time independent conjuncts, each of which represents functional agglomeration of target objects (agglomeration of ships into fleets). The type of reasoning that is involved in elaborating such plans is qualitatively different from that in the first scheme. In this latter case the planner must selects regions for each conjunct, on the basis of mutual compatibility of the displacements [illegible] .cks) of every fleet. One level below, the same reasoning is carried out with finer detail. Given the track of one ship, is the track of another ship, broadly defined by the partition, compatible knowing these ships belong to the same fleet and function in unison?

The constraint based reasoning that determines compatible partitions and is used to elaborate processing plans, is not the only knowledge that differs from one blackboard hierarchy to the other. The meta-knowledge used to compare competing plans and select the most plausible is also qualitatively different. Selecting from competing plans at the second highest hierarchical level in the representation of figure 5.14, is a decision about alternative sets of paths taken by individual ships moving within the same fleet. Knowledge that is required is about the internal workings of a fleet the performance and functions of the ships that form it. Selecting from competing plans at the second highest level in the hierarchy illustrated by figure 5.11 does not necessarily involve knowledge about the internal workings of a fleet, but about the likely configurations assumed *between* fleets Strategic and tactical knowledge taking into consideration objectives, can be used to determine the most likely naval formations.

The order in which the events (signals) occur need not dictate the order in which processing is performed. Knowledge of the search space that is built up through planning, can be used to decide whether processing of a node should be deferred or not For instance, if the system losses track of one vessel at some stage because its signal is drowned in noise, and if attempting discrimination would require too many processing resources, it can differ that search for solution element until a latter time slot, when demand on resources relaxes, or when the traffic jam eases. Similarly the extent of processing and the resources that should be devoted to a particular task are most efficiently decided in the context of the total solution. A weak or poorly discriminated signal might at first require extensive and processing to strengthen a hypothesis. If subsequent tracking of the source renders the dependance on the initial data less critical, the system can with confidence ignore the initial ambiguity. If on the other hand subsequent tracking still remains ambiguous, the system can return to the particular hypothesis and process it more thoroughly

The process of planning at successive levels, no matter what organizational scheme is used, involves zeroing in on initial data by fixing successive constraints as regions are selected down the hierarchy At each stage the planner draws on its knowledge of conjuncts (at the same level in the hierarchy) to support a decision

ABSTRACTION LEVEL	CONJUNCTS	PURPOSE OF PLAN
Situation Board (all fleets at successive time intervals)	All fleets at successive time intervals	To integrate movements fleets at successive time interval
Fleets at an instant in time	Individual vessels at an instant in time	To integrate the location of individual vessels which form the elements of a fleet
Individual vessels located at an instant in time	Sources of perceptual evidence and intelligence reports	To integrate sources of perceptual evidence and reports in support of a vessel hypothesis
Sources of evidence (acoustic and intelligence reports)	Harmonic groups	To amalgamate harmonic sets of signal into a source hypothesis
Individual frequency lines extracted from sonograms		

Table 5 11 - Model (model 1) for HASP problem using blackboard

ABSTRACTION LEVEL	CONJUNCTS	PURPOSE OF PLAN
Situation Board (all fleets at successive time intervals)	Tracks of individual fleets	To integrate movements of fleets at successive time intervals
Tracks of fleets	tracks of individual vessels	To group vessels tracks into functional fleet tracks
Vessel tracks (location of individual vessels at successive time intervals)	Locations of individual vessels at an instant in time	To integrate location of individual vessels in time, into continuous tracks
Individual vessels located at an instant in time	Sources of perceptual evidence and intelligence reports	To integrate sources of perceptual evidence an reports in support of a vessel hypothesis
....	... .	... .

Table 5.12 - Alternative model (model 2) for HASP problem using blackboard

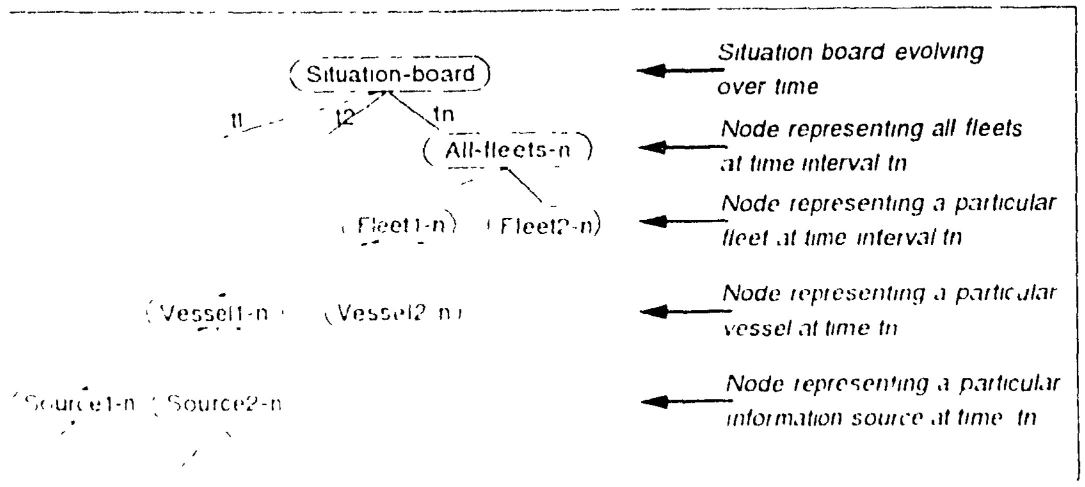

Situation-board
t1
t2
tn
All-fleets-n
Fleet1-n
Fleet2-n
Vessel1-n
Vessel2-n
Source1-n
Source2-n
Situation board evolving over time
Node representing all fleets at time interval tn
Node representing a particular fleet at time interval tn
Node representing a particular vessel at time tn
Node representing a particular information source at time tn

Figure 5.13 Solution elements using model 1

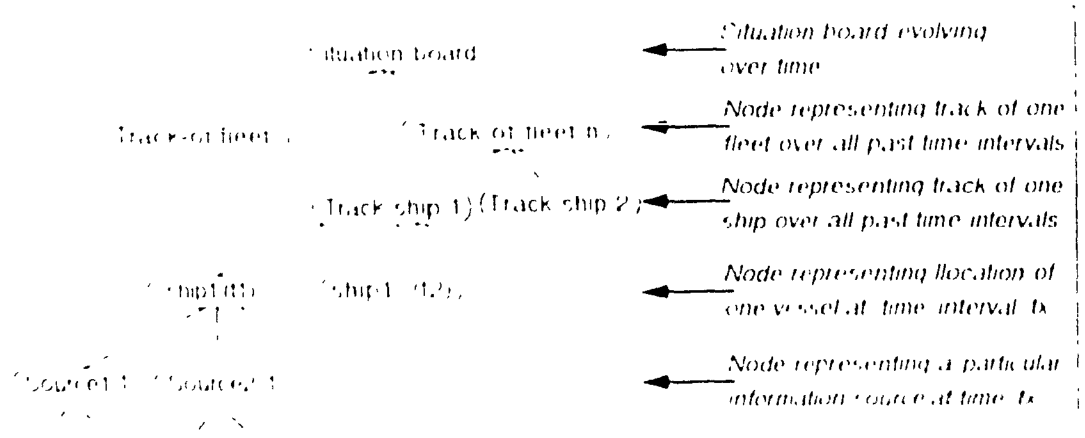

Situation board
Track-of fleet 1
Track of fleet n
Track ship 1
Track ship 2
Ship1 (t1)
Ship1 (t2)
Source1
Source2
Situation board evolving over time
Node representing track of one fleet over all past time intervals
Node representing track of one ship over all past time intervals
Node representing llocation of one vessel at time interval tx
Node representing a particular information source at time tx

Figure 5.14 Solution elements using model 2

5.6. Characteristics and features of the system and its extensions

5.6.1. Synopsis of the method proposed and analysis of the computational advantage it offers

The blackboard provides a formalism for representing a problem

The user is required to formulate an explicit specification for the solution in terms of a generalized description of the data structures and the necessary constraints imposed on them. This information is always known, since it is a detailed expression of the goal and its identification is a prerequisite for all machine assisted problem solving. In the HASP/SIAP example the solution elements are described by the definition of prototype objects:

SITUATION-BOARD
FLEET
VESSEL
SOURCE
HARMONIC-GROUP
LINE

the relations between them and the constraints on their values

In the speech example we had:

SENTENCE
CONSTITUENT
WORD
PPF
PAC

The system can only solve *occurrences* of the problem defined by simultaneous instances of prototype objects. In the speech example, the system attempts to solve an occurrence of the problem by interpreting an occurrence of instances of PACs. In the

HASP example the system solves the problem it is given by interpreting an occurrence of instances of LINEs and possibly instances of SOURCEs.

The simultaneity of occurrences is represented by a conjunction, possibly ordered of instances of initial data

A solution involves the generation of instances of prototypical objects as functions of the initial data, which gives rise to an AND graph

The criterion for accepting a solution need not necessarily specify ground instances of every prototype Some attributes on objects may assume variable values or value ranges, and thereby retain a measure of generality In the HASP application for example, it may not be necessary to pin point the location of a vessel at every interval in time It may be acceptable to specify a range of values for a location within a particular time interval

Complexity arises because of uncertainty associated with the mapping functions, which generate competing alternatives The solution as a result takes the form of an AND/OR graph instead of an AND graph

Complexity can be reduced by relaxing the requirements of acceptability For example if it is necessary to identify the class of a vessel and the analysis of signals is ambiguous, the system would need to generate as many hypotheses (instances) of the vessel object as there are types If this constraint can be relaxed totally, then only hypothesis with a variable value for the class attribute, need be generated

The AND arcs relate the input object to the output in the mapping functions

Redundancy of knowledge means that hypotheses, represented as instances of an object, may be generated (or modified) by alternative inputs When the uncertainty associated with alternative knowledge sources has different consequences, redundancy can be used to reduce complexity. The set of acceptable hypotheses is the intersection of those produced by alternative knowledge sources. The task may be viewed as a constraint satisfaction problem in which each object must satisfy the constraints represented through all incident links. In the HASP example, a location hypothesis for a ship at a particular interval, may be eliminated because it cannot be integrated into a track, or perhaps the track to which the solution element belongs is incompatible with the track of the fleet.

During the first phase of problem solving the solution required of the system is less constrained. For a given level of knowledge, less constrained solution requirements reduce complexity The purpose is create, if necessary, under specified objects (containing attribute values that are under specified), as quickly as possible and use them to eliminate hypotheses that would otherwise overwhelm the system

The planning process is is geared to maximize the use of constraints discovered during the first phase of processing Since a solution consists of an AND graph, the planners task is to select the disjuncts that can be combined into conjunctions The plan elements being under specified objects, there is an inevitable amount of search within the space specified by the object. The combinatorial complexity is nevertheless considerably reduced, because the combinable sub-search spaces represented by compatible disjuncts are much smaller.

Meta-knowledge provides additional levels of complexity reduction through two mechanisms:

By providing the user with a vehicle for expressing additional constraints in terms of partially specified objects and thereby eliminating some plans (combinations of disjuncts), from further consideration.

by providing a much weaker, but nonetheless useful, means of evaluating (heuristically) competing plans. The evaluation of plans is a sophisticated form of scheduling since it does not simply order immediately executable instantiations, but instead orders whole programs of actions based on immediate task as well as their dependencies. A plan in fact constitutes a (partial) program for solving the entire problem, through the scheduling of entire sub trees of the AND/OR graph.

5.6.2. Generality and applicability of the method

If the domain dependent elements of this system are removed, what remains consists of:

1) A specification for describing domain objects.

2) A specification for defining a generic solution which consists of AND graph of instances of prototypical objects. The organization of the graph is based on the hierarchical order of the data objects.

3) A specification or algorithm for directing the search for the solution. The algorithm which is sensitive to data and therefore to some extent data driven, selects sets of disjuncts which partially specify solutions.

4) A vehicle for expressing various levels of combinatorial constraints founded upon incidental knowledge of the properties of the domain.

5) Very weak guidelines for partitioning the search space.

Compliance with the first two points restricts applications that may be represented on this blackboard. The effectiveness of this architecture is dependent on finding a suitable criterion for partitioning the search space. Ideally partitions should be based on attribute values that are most critical in the selection of conjuncts The efficiency of the search is enhanced considerably by meta-knowledge. This is due in part, because of the power of plan selection as a scheduling mechanism, and in part because meta-knowledge can have a direct consequence on plan refinement or elimination, which excludes much of the search space from further consideration These properties are only relevant for knowledge intensive applications

In this respect our speech example was particularly poor, while the HASP example appears at the contrary, to be very promising

5.6.3. Extensions and further work

Alternative hierarchy schemes express different views of the problem Each perspective affords a different basis for reasoning about constraints In practice domain knowledge may exist in multiple forms An architecture provides a much more powerful problem solving model, if it allows multiple representations and merges all the constraints As the data objects at the top and the bottom levels of the hierarchies represented in figures 5 13 and 5 14 are identical, the difference must reside in the distribution of attributes between the individual abstract objects and their conjunctions.

Since different abstraction schemes reflect different representations of the same fundamental problem Constraints pertaining to plans discovered in one representation scheme must be equally applicable to another. It would therefore be very advantageous to generate all the alternative representations for which constraint meta-knowledge is known, formulate equivalent plans, to enable the application of all constraints. The resulting plans when merged would result in a much smaller and consequently more effective search.

An aspect of processing which has been mentioned but was particularly well developed concerns the allocation of processing resources and the scheduling of processing tasks The

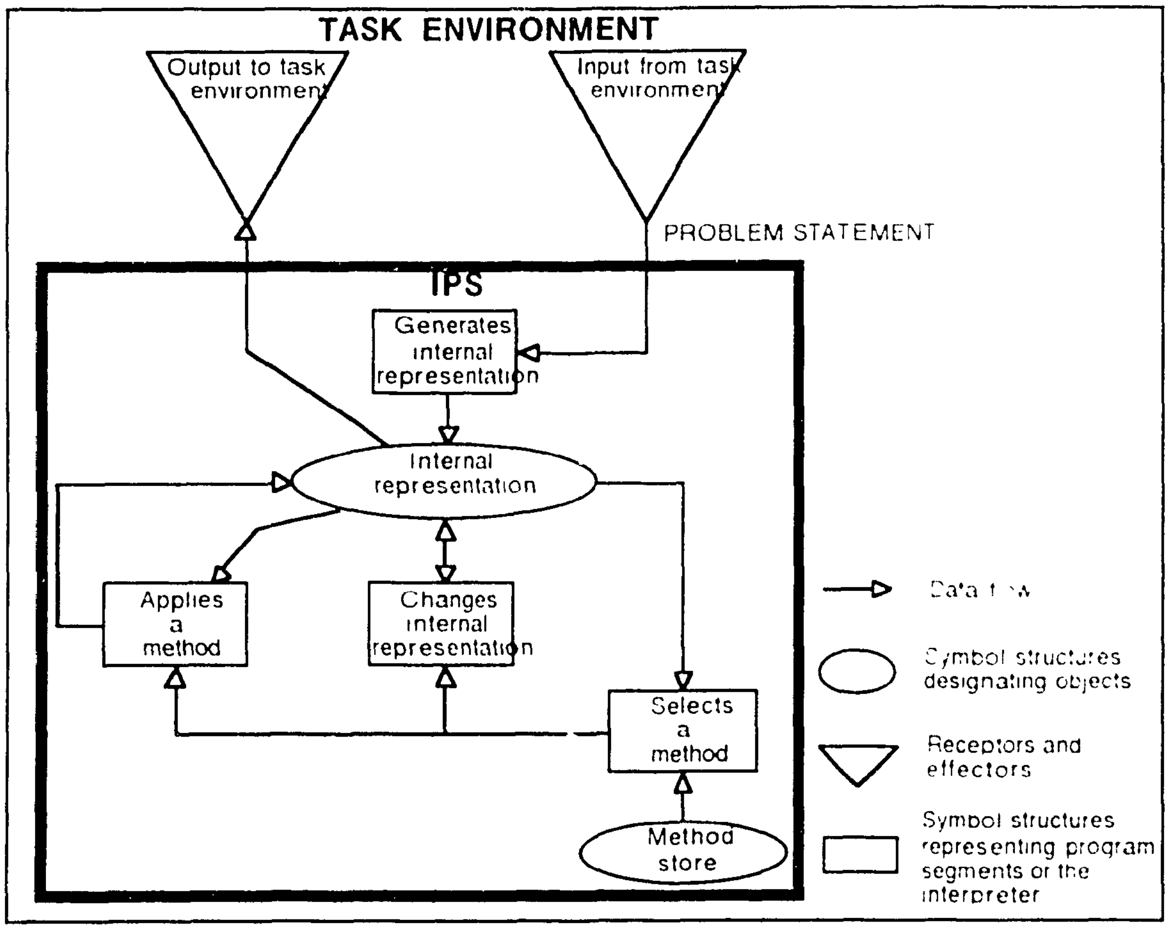
HASP application with its continuous (as opposed to batch) real time treatment of signal highlights the need for reasoning about process schedules Should for instance the system devote an arbitrary amount of resources to the generation of a particular solution element, or should it suspend temporarily that process in the hope that clarification or reinforcement of hypotheses can result from new signal sources? The answer to the question depends on the lag that can be tolerated in the overall solution and whether an initial coarse solution is acceptable if followed by a more precise revision In any case this level of planning involves reasoning about computational time and processing resources, quality and timeliness of solutions, and expectations about the evolution of the problem The architecture supports much more development in theses aspects of problem solving Proper testing and evaluation of such techniques requires a more demanding and sophisticated application

6. Appendices

6.1. Appendix 1

6.1.1. The Information Processing System:

The fundamental postulate of Newell and Simon's theory is that Human Problem Solving behavior can be modelled bv an IPS [SIM&NEW72].



TASK ENVIRONMENT
Output to task environment
Input from task environment
PROBLEM STATEMENT
IPS
Generates internal representation
Internal representation
Applies a method
Changes internal representation
Selects a method
Method store
Data flow
Symbol structures designating objects
Receptors and effectors
Symbol structures representing program segments or the interpreter

Figure 6.1 Functional Organization of the Human IPS

Such an IPS interacts with the environment through receptors and effectors The main components of the IPS are the processor and the memory The fundamental entity within these components is the symbol token. Instances of the elementarv set of symbol tokens combine through relations to form symbol structures Svmbol structures function as designators when they reference objects, they function as programs when they form elementary information processes which may be executed by the interpreter, to generate, modify or delete object designators

Organizationally the IPS (Fig 1), consists of a program (or a set of EPI's) capable of translating the problem statement into an internal representation which defines the problem space Another program selects the most appropriate method(s) for processing, an interpreter applies the method(s) The resulting transformations which occur in the temporary memory (storing the internal representation) may prompt segments of programs to modify the internal representation or change the methods, as properties of the problem are discovered If the solution is found the IPS returns it to the task environment, the goal driving it having been satisfied

Observation that were noted pertain both to performance of the components of the IPS (these are analogous to specifications of hardware) and to the function of the IPS (analogous to the software organization)

As was noted earlier the human IPS is organized out of two components memory and processors

6.1.1.1. Memory

The memory elements involved in the problem solving process fall into three distinct categories

An associative Long Term Memory (LTM) within which the learning process takes place Such a process involves generation of new symbolic object designators or chunks , to map stimulus patterns to networks of constituent designators Symbols in LTM can designate complex compiled processes as well as elementary ones LTM is associative. Writing to LTM, which occurs during learning, takes an order of magnitude longer than reading (1.5 s. for associative retrieval including recognition versus 5 to 10 s. to fixate a new symbol pattern) There appears to be no limit to the capacity of LTM.

Short Term Memory (STM), is very transient decaying with time (unless refreshed with access), and is characteristically small (containing some 5 to 7 symbols) Because both reading and writing to STM is relatively fast, elementary processes use it as store for intermediate results

External Memory (EM), is available through sensory perception, in the form of read only visual displays such as game boards, or read/write in the form of paper and pencil. The performance of IPSs changes dramatically when deprived of EM Functionally EM supplements STM both in capacity and duration, when consisting of visual displays and it can also supplement LTM

6.1.1.2. Processors

The most striking characteristic of human information processing (perceptual problems aside), is that it executes serially. The time required to perform multiple instances of the same task increases linearly with the number of instances There appears to be a set of basic processes (including read, write and symbol comparison, and replacement), referred to as elementary processes, which are themselves composed of symbol structures stored in memory. The speed of execution of elementary processes (40 ms) suggests that most writing must be done in STM Simon and Newell suggest that almost all input/output to elementary processes is done through STM except for specific reading and writing to LTM or EM The size of STM suggests that most elementary processes involve no more than two symbols A reasonable set of processes would include test-and-comparison and creation of symbols, reading, writing, designating and storing symbol structures

The third component of the processor is the interpreter which integrates the operation of elementary processes The interpreter is essentially a neutral device, the behavior of the IPS being entirely determined by the elementary processes themselves

6.1.2. Program organization of the human IPS

While the behavior of the human information processing system is entirely determined by the symbol structures encoding its knowledge, an understanding of the organization of these structures requires an assumption to be made about the program model Simon and Newell's theory of human problem solving is based on the assumption that the information processing knowledge is represented as productions To support this crucial statement they note the following

Productions provide a homogeneous scheme for encoding program behavior, as opposed to standard control flow, which depends on the content of actions and their sequential organization to specify the program

Individual productions, being independent of one another, can be built up incrementally so that the system's behavior can be developed progressively This feature is essential to allow for learning, which is typically incremental in nature

Each production represents some meaningful and complete component of the problem solving method Since it is context independent it may contribute to problem solving spontaneously without requiring the construction of a complex program

As LTM is the repository of problem solving methods, productions would have to reside in that memory. But the authors go further in suggesting that productions are a reasonable model for the function of LTM itself.

Productions work on a pool of symbols resident in dynamic working memory. The counter part of this in the IPS, is STM and EM. A control flow model on the other hand would require that data be written in LTM (the location of methods in the form of procedures); the write times required for such operations are incompatible with observations of the execution of elementary processes.

The production model accommodates a combination of stimulus bound activity (originated from EM) and stimulus independent activity (originating in STM), both of which are observed experimentally. The Control Flow model on the other hand only accommodates stimulus independent activity.

As the left hand side of productions perform content-based addressing of dynamic memory (through the matching process), no explicit elementary processes are required to retrieve data form STM. Retrieving information from LTM is the act of selecting an instantiated production.

6.1.3. The Task Environment and Internal Representation

The task environment of an IPS encompasses the physical objects comprising the elements (the environment) of the problem to be solved, the problem itself, all its constraints and the goals to be satisfied. The task environment is thus the statement of the problem in the language that is put to the solver. As only some of the properties of the environment's objects may be pertinent to the problem solving processes, an internal representation of the external environment is generated, which maps the relevant properties of the objects which are manipulated, and emphasizes relations which might be implicit in the environment, but are useful to the processing.

6.1.4. The Problem Space

The locus of knowledge about the task environment that the IPS utilizes for the solution and within which all problem solving activity takes place, is called the problem space.

6.1.4.1. Components of the Problem Space

The set U of symbol structures representing the initial data, the goal data and intermediate transformations

The set of operators or information processes generating the intermediate and goals states

The total knowledge available to the IPS is

- Local information generated and used within a single state
- Path information to the current state
- Access information to other states reached.
- Information about possible transformation to the current state and other states reached.
- Information pertaining to the properties of the set of elements U.

The values that the symbols U can assume under the transformations constitute the search space.

6.1.5. The Problem Solving Process

Given that a concise description of the task environment and unambiguous definitions of the goals are available, the first step in problem solving involves the generation of the problem space or internal representation of the problem.

6.1.5.1. Steps in constructing a problem space

- Devising a representation for the knowledge states.

- Determining the complete set of operators which will generate all allowable states.

- Factoring.

  This involves the recognition of properties in the problem space that makes seemingly different states equivalent from the processing point of view Factoring may bring about a revision of the set of operators.

- Devising sets of heuristics.

1) functions for evaluating the distance between two states, usually a goal state and a candidate node from which the problem solver is considering expanding the search; the arguments to such functions are therefore both states.

2) functions for selecting the most applicable operator at a particular node.

The determination of the problem space occurs in the first few instants of problem solving. If the task is similar to a problem that has already been tackled then it will evoke a problem space already stored in LTM. If on the other hand the task appears to be completely new, then chances are that the task instructions themselves, and the EM displays (eg. game boards, displays etc...), will be in terms of a particular problem space.

These two classes of heuristics assist the IPS in two decisions it must make at every step while it remains within the particular problem space, namely:

- The selection of the node from which to proceed

- The selection of the operator to apply at such a node.

The two types heuristics are not entirely equivalent. If the IPS has available discriminating state evaluation functions, but no operator selecting heuristics, it can tentatively apply all operators to the best node at each cycle and evaluate the new states, the resulting behavior being a best-first search. If on the other hand the system has available very powerful operator selection mechanisms, but no state evaluation functions, its behavior appears to be almost algorithmic

6.1.6. Interpreted, Planned and Compiled Behavior

Operator selection heuristics amount to control flow information. A problem solver which is saturated with control flow knowledge throughout the search path of the solution, no longer performs a tentative search By reading the state, such a system is able to determine the correct action to perform at the next step. It is still not able to prescribe the sequence of actions that will take it from the initial state to the goal state. The system requires an evaluation of the current state or path to determine the appropriate transition

If heuristics are an integral part of the problem space (they are not developed during the course of problem solving), it is feasible to compile the behavior of the problem solver (with conditional tests on the data) such that the whole sequence of possible instructions is known a priori. Under such circumstance the problem solver would appear to exhibit programmed behavior.

For a human problem solver such a programmed method could be learned and executed without any understanding (after all, it is possible to solve quadratic equations using the formula without knowing why it works). This suggests that the mechanism of understanding requires evaluation at some grain size below that of the task to be understood.

If a measure of control flow knowledge is available in the problem space but does not span the entire path of the solution, the system can (with advantage) by evaluating the current state plan sequences of steps in advance. The planning can involve degrees of uncertainty and it can be underspecified. The detail will be discussed in chapter two.

The difference in problem solving modes is therefore due not only to the amount of knowledge available about the task environment but in the way it is used. When problem solving actions are decided during the course of the problem solving process (and is therefore based on the evaluation of knowledge states, whether this is done with hindsight through the application of state evaluation functions, or with foresight through evaluation of state generators), the systems appear to be functioning heuristically.

6.2. Appendix 2

6.2.1. The precursor of planning systems GPS (Means End Analysis)

The method used by General Problem Solver (GPS [NEWELL&SHAW60, ERNST&NEWELL69]) which became known as Means End Analysis (MEA) was the first to take into consideration the cumulative effect of a succession of operations before actually applying them. GPS was innovative in a number of respects:

It supported goal driven searching which being a more humanly natural way to solve problems made it easier to find and express heuristics. Decomposing a problem into a number of sub-goals also provides a choice upon which to focus and direct the search process while traversing regions of uncertainty. In many cases goal generation is entirely mechanical and can be automated.

It was capable of discriminating between operators prior to their application. The economy of computation is not the real advantage; being able to anticipate the effect of an action however, opens the way for planning a number of transformations and evaluating their cumulative outcome.

By explicitly representing the differences that an operator reduces, the criterion for the selection of transformations could be state based. If the relative importance of these differences is goal dependent, then the ordering scheme can be selected accordingly.

Means end analysis suffers from two major drawbacks, which make the method very weak in the majority of applications.

Means end analysis makes an implicit assumption that local reductions in goal/current-state differences can ensure global progress. While MEA bases its decisions on dynamic evaluations, the window spanned by such evaluations does not extend beyond the current state. The heuristics used to select the most important difference reductions must be correct throughout the path. This property cannot always be guaranteed for most domains.

The second major failing relates to conjunctive goals. The goals of most problems are specified as conjunctions of constraints. MEA however, only considers one goal at a time; it is unable to identify those actions which reverse or are incompatible with previously achieved goals. The success of the method is therefore dependent on the availability of a heuristic which orders the sub-goals such that the achievement of one does not eliminate the progress achieved by another Mutually interacting sub-goals cannot be processed at all .

6.2.2. Planning a program of actions for robots in a blocks world

Planning systems have been primarily developed to enable robots to function in blocks world type environments. Given a description of the world, usually in predicate logic form, a final state and a modest repertoire of actions, the system builds up a program of transformations, which if performed in the correct order, achieves the necessary goals. The reason that planning has focused on this rather artificial problem, is that one can only plan (and reason about) actions as far as one can predict their outcome. With a robot working in a blocks world, one can assume to know everything about operators Under such circumstances a completed plan is equivalent to a solution

6.2.2.1. The STRIPS system

STRIPS [NIL&FIK71] was the one of the first robot control systems. The problem solving method adopted is a modified form of MEA. Whereas the original version of GPS did not keep a pending goal list, but relied on the recursive call to the search function to implement chronological backtracking, STRIPS maintains an explicit *stack* of goals and identifies conjunctions. The solution is developed explicitly in terms of goals/sub-goals, in other words by step addition, in the manner of modern planners. STRIPS has some minimal capacity for identifying interactions in compound goals [NILS80]. After all the conjuncts have been processed, the actual state is compared with the compound goal Differences due to one sub-goal "undoing" the work of another, cause the system to reprocess that component STRIPS uses First Order predicate Logic (FOL) to describe states Operators consist of Preconditions, and Postconditions (Add and Delete lists) When an operator is applied, the new state contains the same predicates as its predecessor after adding those in the Add list and removing those in the Delete list All variables are bound when the rule is applied This solution to the frame problem is adopted by all domain independent planners

One of the major breakthroughs achieved by subsequent planners has been the capacity to solve the problem illustrated in figure 6.2, sometimes called the Sussman anomaly

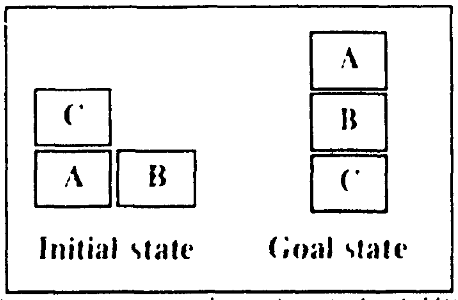


Figure 6.2 Graphical representation of intial state for STRIPS anomaly problem

INITIAL STATE (ON C A) (CLEAR B)

GOAL: (AND (ON A B) (ON B C))

A linear planner starting with the first sub-goal (ON A B), would achieve (after clearing A) the state illustrated by figure 6.3

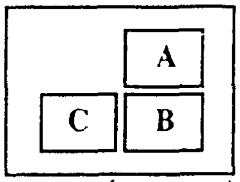


Figure 6 3 - Graphical representation of intermediate state for STRIPS anomaly problem

If the second sub-goal (ON B C) were attempted first, the state illustrated in figure 6.4 would result

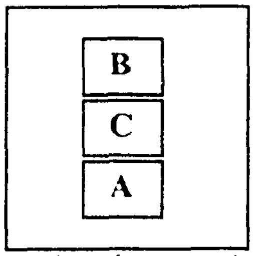


Figure 6 4 - Graphical representation of intermediate state for STRIPS anomaly problem

Both options make the overall goal more difficult to achieve. Linear planners such as STRIPS which will invoke a rule that satisfies the top sub-goal of the stack make precipitous goal ordering decisions that require backtracking

6.2.2.1.1. Limitations of STRIPS

The failure of MEA in the example above can be attributed to two aspects of representation used in the STRIPS system:

The description of the world (in terms of predicates) available to the system is incomplete. Two additional predicates are needed to describe the goal state: (ON

C TABLE), (CLEAR A) The completed description of the main goal would enable STRIPS to solve the problem, possibly with some backtracking

Another property of the problem unknown to the system, could reduce the amount of backtracking even further. Since piling blocks is a stack operation, the lower blocks in the goal should be processed first The goal predicate pertaining to the lowest block (ON C TABLE) should be the highest priority goal, followed by (ON B C) then (ON A B) With this complete information STRIPS (as well as GPS) could solve the Sussman anomaly without incurring a search

The example illustrates the difficulty encountered in all problem solving systems, namely that the effectiveness of methods, is intimately related to the knowledge representation schemes used, and to the knowledge available for problem solving

6.2.2.2. Conjunctive Planning systems

Most modern domain independent planners use some form of FOL representation with the STRIPS assumption that successive states are identical save for the predicates in the ADD and DELETE lists. This allows the operators to be selected according to the predicates whose truth values need to be altered Predicate logic also has the advantage of being processable efficiently with a pattern matcher The incremental improvements made in planning systems have gone a long way to resolve some of the vexing limitations described earlier

As was illustrated above, errors in plans produced by FOL based systems are due to decisions taken with incomplete descriptions of the world For such planners to succeed they must retain the flexibility of pursuing the planning activity to the end (that is back to the initial condition) and retain the option of modifying (or patching) plans when conflicts occur These conflicts occur because decisions taken regarding the selection of steps (operators) and their ordering are made in an environment of incomplete knowledge about the problem As the solution is pursued further these errors can be detected by the system and can be corrected, with relatively minor modifications. By prematurely invoking Forward Rules, STRIPS and GPS commit themselves to an irrevocable and incorrect sequence of steps

After STRIPS, a number of conjunctive planning systems were developed. The earliest systems, HACKER [SUSS75], WARPLAN [WARR74] and INTERPLAN [TATE75] were linear systems During the process of plan construction, as a new step is added to achieve a particular sub-goal (conjunct), a decision on its ordering, with respect to the steps resolving other sub-goals is made immediately. This decision may be revised at a later stage.

The NOAH system [SACE75] introduced the idea of non-linearity which has been adopted by all its successors NONLIN [TATE75], SIPE [WILK83], TWEAK [CHAP87]. As with linear systems, a plan consists of an ordering of operators selected from a given set which cumulatively make the necessary transformations (assertions and denials) to achieve the predicates specified in the goal description Again as with linear planners, the systems rely on the STRIPS representations. Non-linear planners differ from their predecessors with regards to the ordering of actions. These systems make no initial commitment to the processing order of conjuncts (sub-goals). Only when possible interactions are detected, is the order further specified

A solution being an ordered set of operations, partial relaxation on the ordering specifies a subset of the solution space The process of incrementally specifying ordering constraints is one of progressively eliminating segments of the search space Linear planners on the other hand make specific commitments to solutions, test them, and if necessary, backtrack.

Apart from ordering constraints, planners must make designations constraints on the variables appearing in the selected operators. Again, restrictions on binding values are restrictions on portions of the solution space

The last component of the solution of a planner is the selection of the operators needed to achieve the goal

In searching for solutions a planner starts with an initially empty plan which is compatible with the entire solution space Implicit in every solution is an *initial* state existing chronologically before any transformations and a *final* state which is partially specified in terms of goal predicates As the three classes of constraints are applied, a partial specification of the plan is imposed, which progressively eliminates segments of the search space By defering the application of constraints until an informed decision can be made, planners (and particularly non linear systems) can reduce backtracking resulting from arbitrary and premature decisions A plan is complete when all the possible orderings of actions specified in the plan will transform the initial state into a goal state and the preconditions of each step are satisfied by the state(s) at which the transformation(s) is applied

Using the STRIPS representation the only changes to a state resulting from a transformation are those described by the ADD and DELETE lists To establish a required predicate the planner must attempt to unify it with an element of the ADD list of some operator The variable bindings imposed by unification must be carried over to the preconditions of the selected operator Those preconditions that are not true in the state at which the operator is applied, must be identified as sub goals still to be achieved Any predicate which is required, and is false at the state where it is required, must be established at some point prior to that state, and thus becomes a sub-goal.

Since only the most trivial problems are completely decomposable, some interaction must be expected between sub-goals so the planner must check the validity of predicates after the point of modification.

6.2.2.2.1. Plan construction

A plan in the process of development, is a partially ordered list of operators Where the ordering is not completely specified, "parallel" operations may be ordered arbitrarily. Associated with each transformation are two sets of predicates, those which should be true *before* the transformation, and those which become true *after* it. Before the completion of a plan, there will be predicates which are required, but which have not yet been achieved. The planner must identify these predicates and attempt to achieve them

A system using the STRIPS representation may construct a plan by the application of the following rules:

If a required predicate is not established because it does not exist in the initial state and there is no prior transformation which includes it in its ADD list, then "Step Addition' is the only way to achieve the predicate.

If the predicate exists but contains variables then a commitment must be made to bind it to the appropriate constant

If the required predicate is asserted but also denied by an action parallel (and therefore possibly before) the one that requires it as a prerequisite, then the system must commit itself to a particular order and ensure that predicate denial occurs at a stage in processing after the one that requires it.

A conjunctive parallel planner tackles the Sussman Anomaly as follows:

Two actions are required:

```
PUTON (x y)
PRECONDITIONS    { (ON x z), (CLEAR x), (CLEAR y) }
POSTCONDITIONS  { (ON x y), ~(ON x z), ~(CLEAR y), (CLEAR z) }
```

```
NEWSTACK (x)
PRECONDITIONS    { (ON x z), (CLEAR x) }
POSTCONDITIONS  { (ON x TABLE), ~(ON x z), (CLEAR x) }
```

The second action is needed because PUTON (x y) denies (CLEAR y), when y is bound to the TABLE this is not true. NEWSTACK (x) is similar to PUTON (x TABLE) but omits the denial.

Predicates in *bold and italic* are prerequisites that have not been achieved and which therefore become sub-goals. Variables are represented in lower case.

Figure 6.5 is a graphical representation of the problem statement and the empty plan. Figure 6.2 illustrates the problem in terms of blocks.

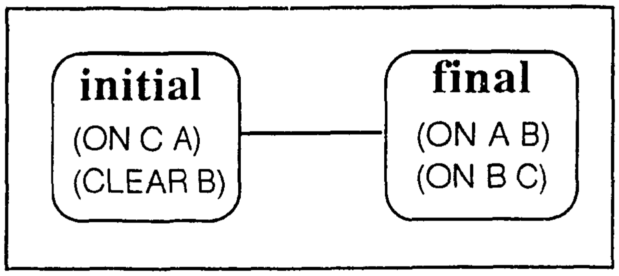


Figure 6.5 - Intial state for STRIPS anomaly problem.

The (minimal) differences specified between the initial and final states expressed in the problem statement is the establishment of the two predicates (ON A B) and (ON B C). Working from the goal state there are two predicates that need to be established and the plan being empty only rule 1 is satisfied. The planner recognizes that the postconditions of the action puton (x y) unify with those two component goals and generates the first pass at the plan binding {A | x}, {B | y} for one of the steps and {B | x}, {C | y} for the other. Since the two goals are conjuncts, and when the plan is empty there is no way of identifying which should be achieved first, the planner specifies the steps as parallel actions, defering a commitment on ordering. As a result this plan is compatible with completions encompassing both orderings. The variable z in the action is not yet bound since the first two applications of rule1 are only concerned with the establishment of the two predicates and not with the origin of the block to be moved. Figure 6.6 shows the plan after two firings of rule 1.

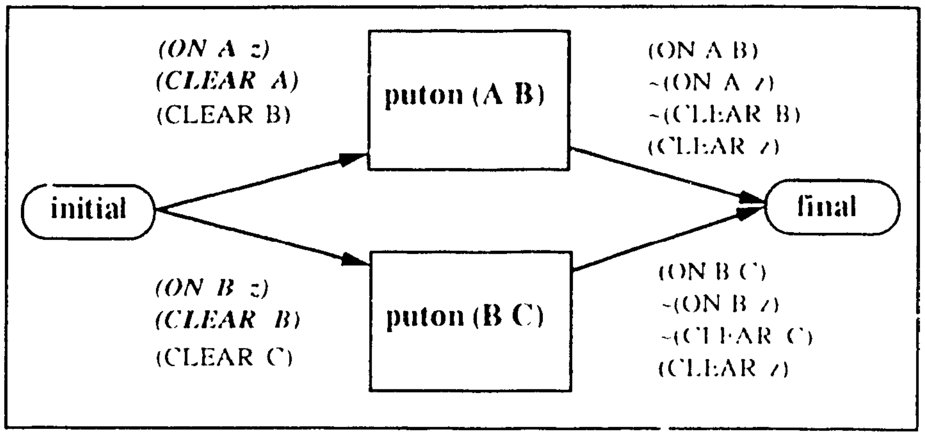


Figure 6 6 - First stage in plan development using least commitment inSTRIPS anomaly problem

Two applications of rule 2 result in variable "z" in the preconditions being bound to the table by unification with the predicates in the initial state (figure 6 7)

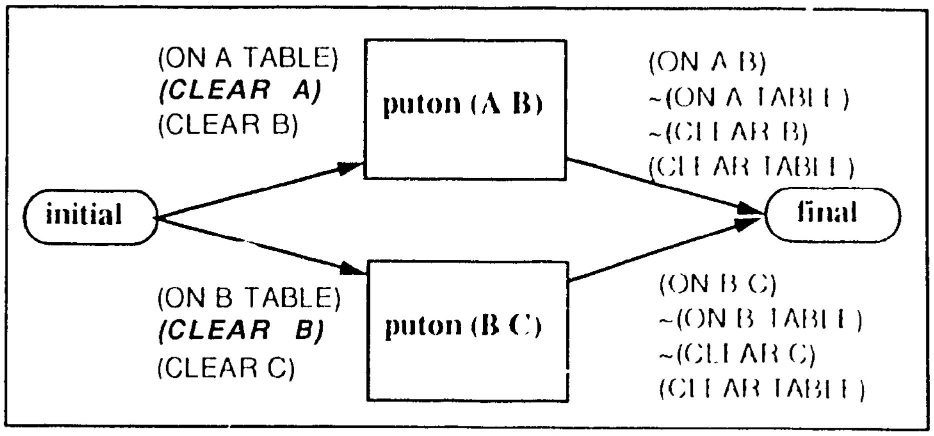


Figure 6.7 - Second stage in plan development using least commitment inSTRIPS anomaly problem

At this stage the planner detects a possible conflict, the precondition (CLEAR B) The action PUTON (A B) denies (CLEAR B), if this action is performed before PUTON (B C) (as the parallelism allows) then the predicate would be denied The system detects this interaction (by comparing the postconditions of the one action with the precondition of another (parallel). A further constraint (on order) must be applied, eliminating a set of possible completions from further consideration. The detection of potential conflict and the correction to the plan is performed by rule 3. The result in shown in figure 6.8

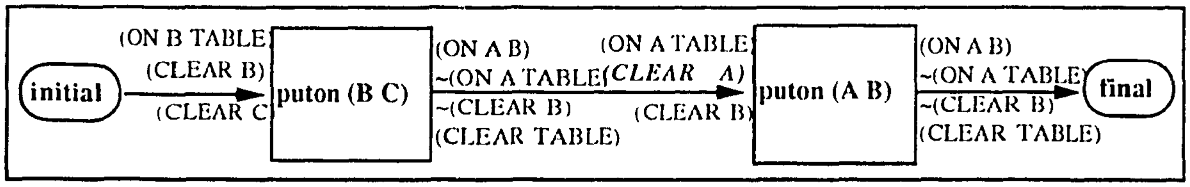


Figure 6.8 - Third stage in plan development using least comitment inSTRIPS anomaly problem.

There remains one precondition to achieve (CLEAR A) which is needed for the second action of the partial plan of figure 6.8. The action that establishes this predicate is NEWSTACK (x). This action is needed "sometime" before PUTON (A B); at this stage the planner is unable to specify the exact position so the location is specified before PUTON (A B) and therefore in parallel to PUTON (B C). The plan step is added by rule 1, to give the partial plan of figure 6 9

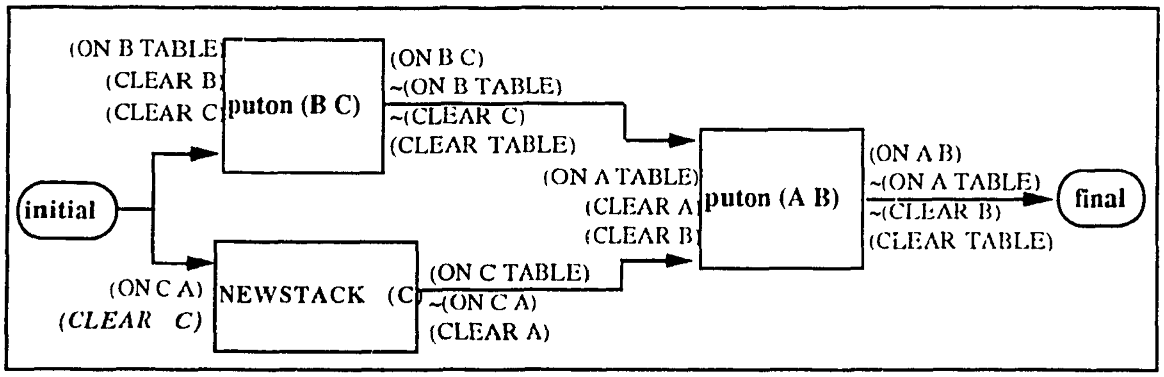


Figure 6.9 - Fourth stage in plan development using least comitment inSTRIPS anomaly problem.

The precondition (CLEAR C) of NEWSTACK cannot be guaranteed because the postcondition of PUTON (B C) denies the predicate; the ordering {puton (b c) , newstack (c)} which denies the required predicate is covered by the parallel representation. Constraining the order to exclude that sequence solves the problem. This action is performed by rule 3. Figure 6.10 shows the completed plan.

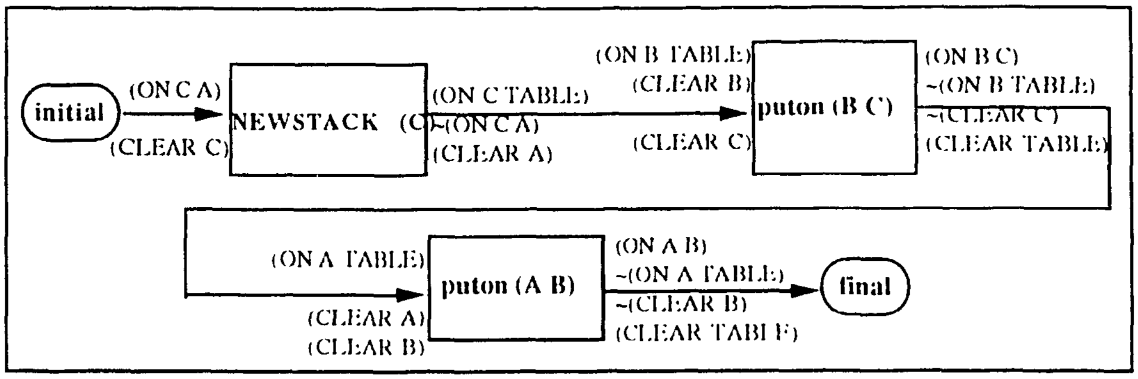


Figure 6.10 - Fith stage in plan development using least comitment inSTRIPS anomaly problem

6.2.3. Characteristics of planners

The distinguishing feature of planners (and of MEA), is that such systems are aware of the changes of state brought about by their operators, and they can reason about sequences of transformations necessary to achieve an overall result In order to possess this capacity, planners must be equipped with knowledge of the changes brought about by an operator (the postconditions), expressed in a form (predicate logic), which is available to the processing mechanism (the pattern matcher).

Another feature of planners (though not of MEA), concerns the span of their reasoning; these systems will devise the necessary sequence of operations, necessary to achieve all sub-goals, before actually executing any transformations By broadening the window of analysis to cover complete paths rather than segments of it, the planner is less likely to fall into heuristic traps where maximizing local progress is counter productive to the persuit of an overall solution.

With the blocks world example, executing the plan appears to be a redundant phase since the outcome of each step is certain and known a priori. In real world problems this is not likely to be the case Even for a robot operating in a Blocks World, Sacerdotti discusses plan execution, monitoring and repair to allow for "unexpected" events. Errors in the gearing of the robot's arms and wheels, for instance, could result in incorrect positions and necessitate additional movements; these errors would be discovered as steps are executed and would require "on the fly" modification and re-planning.

In situations where uncertainty about the outcome of transformation arises, planning becomes a little more complex. While as was argued in the example above, there are definite advantages to completing the plan, if the uncertainty is cumulative over steps then the cost effectiveness of completing the plan before the start of execution is in doubt.

6.2.4. Limitations due to problem representation

Most current planning systems use the STRIPS representation. The formalism imposes severe limitations on the types of actions that may be represented:

Because actions are selected on the basis of the predicate changes they bring about (listed in postconditions), such systems cannot cope with actions whose postconditions are functionally dependent on input,

Another limitation which specifically relates to non-linear planners is the problem of synergy among conjuncts [CHAP87]. While these planners can detect interactions in the form of predicate establishments and denials, parallel actions which are individually possible but have a cumulative effect cannot be detected.

The result of actions in real-world problems might not be readily describable by postconditions alone Even using FOL, problems might require deductive engines to evaluate the consequences of each action. This makes the plan space very expensive to search

6.2.5. Extensions to plan representation

6.2.5.1. The SIPE system

The SIPE system [WILK84] extends the representational possibilities of planners considerably and introduces new concepts to facilitate the detection of conflict in parallel actions and to enhance performance

6.2.5.1.1. Hierarchical organization

Operators in SIPE are organized into a conceptual hierarchy which enables plans to be constructed at different levels of abstraction. Reasoning at higher levels allows a reduction in computation, by eliminating the manipulation and processing of considerable and unnecessary lower level data. The hierarchical organization is also a natural (human) way of storing knowledge and therefore makes complex operators easier to express. Finally mechanisms of inheritance through hierarchies are well established and implementations are quite efficient so systems incorporating them can exploit a large repertoire of established techniques.

Instantiating hierarchies of operators generates plans in the form of procedural networks which when fully developed (complete) and debugged (all detectable interactions have been resolved), can be processed at the execution phase of problem solving.

6.2.5.1.2. Constraint posting

SIPE system incorporates a much more sophisticated constraint language for variables. Previous planners could either bind variables to constants or leave them totally unbound. SIPE on the other hand boasts a number of attribute value restrictions in addition to conventional numeric constraints. This means that the system can incrementally constrain the search as far its reasoning permits. Premature and arbitrary commitment to particular values is reduced with a corresponding reduction in backtracking. The posted constraints are propagated quite efficiently as the plan is implemented as a procedural network. The hierarchical data structure facilitates attribute value look-up and thus further enhances processing efficiency.

6.2.5.1.3. Resources

The concept of *resources* is a particularly interesting feature of the system. A resource in SIPE is any object which can only be shared by other instantiated operators, under specific conditions. When a variable is identified as a (finite) resource, any instantiation of that operator blocks other instantiations using the same resource until it is released. Reasoning about resources in SIPE is used to detect possible conflicts between parallel actions at the time the operator is selected. One alternative is to axiomatize the availability of resources in the preconditions and their release in the postconditions; this approach complicates the definition of operators and for some applications, could cause an unmanageable proliferation of them[1] . Early detection of resource allocation

[1] A planner operating in a "blocks world", where different sized blocks are used, would require more than one PUTON operator.

conflicts is particularly useful in hierarchical planners, where considerable effort might be deployed in refining an inadequate plan at lower levels.

6.2.5.1.4. Purpose

SIPE's operators contain a "purpose" attribute which identifies the principal predicate (postcondition) sought by its application. This slot reduces the amount of work performed by the pattern matcher when selecting an operator to establish a predicate. The purpose slot is an expression of the rationale behind the application of a particular operator. When the system corrects parallel plans, the operator's purpose attribute which is inherited down the hierarchy serves to scope the lower level nodes. If the purpose is already achieved then the system can simplify the plan by removing all scoped nodes.

6.2.5.1.5. Deductive operators

SIPE allows the definition of deductive operators. These do not expand or modify the plan directly, but assert and deny predicates describing states. The addition of a deductive engine makes the description of real-world problems easier. It also makes the definitions of planning operators simpler, as they no longer need to incorporate extensive preconditions/postcondtions describing all possible outcomes of an operation. The separation of plan operator from state transformation mechanisms opens the possibility of using much more complex state descriptions which are required in real problems. The drawback is that the operator selection mechanism reasons without full knowledge of the consequences of a transformation and might therefore not make the best choice.

7. References

[BATES75]	M. Bates "Syntactic Analysis in a Speech Understanding System" - Bolt Beranek and Newman Inc. - BBN Report No. 3116 (1985)
[BRAC78]	R. Brachman "A strcutural paradigm for reprsenting knowledge" - Bolt Beranek and Newman Inc. Report No. 3695 (1978)
[CHAP87]	D. Chapman "Planning for conjunctive goals" - *Artificial Intelligence* **32** (1987) 333-377
[DAV&BUC77]	R. Davis, B. Buchanan and E. Shortcliff - "Production Rules as a representation for a knowledge-based Consultation Program" - *Artificial Intelligence* (1977).
[DAV&KIN77]	R. Davis and J. King "Programing Tools for Knowledge Representation" - *Machine Intelligence* **8** - Elcock and Mitchie (Eds.) - Edinburgh University Press (1977).
[DAV&SMI81]	R. Davis and R. Smith "Negotiation as a metaphore for distributed problem solving" - Massachusetts Institute of Technology Artificial Intelligence Laboratory - *AI Memo no. 624* - (May 1981).
[DAVI80]	R. Davis "Reasoning about control" - Artifical Intelligence **15** (1980), 179 - 222
[DeMOR&LAF&MON85]	R. De Mori, L. Laface, Y. Mong " Parallel Algorithms for syllable Recognition in continuous speech" - *IEEE Transaction on Pattern Analysis and Machine Intelligence* - **PAMI-7** No. 1 p 55-69 (1985)

[DeMOR&LAM&GIL87] R. De Mori, L. Lam, M. Gilloux "Learning and Plan Refinement in a Knowledge-based System for automatic Speech Recognition" - "*IEEE Transaction on Pattern Analysis and Machine Intelligence* - **PAMI-9** No. 2 p 239-305 (1987)

[DUR&LES87] E. Durfee and V. Lesser "Incremental Planning to control a time constrained problem solver" - *COINS technical Report 87-07* - (1987)

[ERM&LON&FICK81] L. Erman, P. London, S. Fickas "the design and an example of the use of HEARSAY III" - Proceedings of the 7th, International Joint conference on Artificial Intelligence - William Kaufmann Inc. (1981): 409 - 415

[FEIG63] "Computers and thought", - E. Feigenbaum and J. Feldman (eds) -McGrawhill New York 1963.

[FEIG71] E. Feigenbaum - "On Generality and Problem Solving, a case study involving the DENDRAL program. - *Machine Intelligence* 6, - Elcock and Mitchie (Eds.) - Edinburgh University Press (1971)

[FEIG77] E. Feigenbaum "The art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering" - IJCAI77 (1970).

[FOR&McDE76] "OPS, a Domain-independant Production System Language" - C. Forgy and J. McDermott IJCAI1979 Vol 1...

[FORG82] C. Forgy "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem" - *Artifical Intelligence* **19** (1) (1982) p. 17-37

[GALL70] B. Galler and A. Perlis "A view of programming languages" - Addison-Wesley (1970).

[GEORG83] "Strategies in Heuristic Search" - Artificial Intelligence **20** (1983) 393-425

[HAYES-RO85] B. Hayes-Roth - "A Blackboard Architecture for Control" - *Journal of Artificial Intelligence* - 26: 251-321 - 1985

[KNOW88] KnowledgeCraft Technical Reference Manual Vol. 1 Section A (Carnigie Group Inc., Five PPG Place, Pittsburg PA 15222)

[LES&ERM&RED74] V. Lesser, R. Fennell, L Erman, D. Reddy - "Organization of the HEARSAY II Speech Understanding System." - *IEEE Symposium on Speech Recognition, Contributed Papers.* Computer Science Department, Carnegie-Mellon University: IEEE Group on Acoustics, Speech and Signal Processing, (1974) 11-M2-21-M2.

[LES&ERM77] V. Lesser and L. Erman - "The Retrospective view of teh HEARSAY II Architecture." - *Proceedings of the 5th. International Joint Conference on Artificial Intelligence* - Los Altos, California: William Kaufmann Inc. (1977) 790-800

[LIN&BUC80] R. Lindsay, B. Buchanan, E. Feigenbaum - "Applications of Artifical Intelligence to Chemistry: The DENDRAL Project." - McGrawhill NY (1980).

[LIND88] T. Linden "Plan and Goal Representations" - Workshop Report - DARPA Santa Cruz Workshop on Planning - Edt. Swartout - *AI Magazine* - Summer (1988)

[MATH87] L. Mathan "Speaker-independant ASccess to a large lexicon" *Msc Thesis, School of Computer Science, McGill University* (1987)

[McDE&FOR78] J McDermott and C. Forgy - "Production System Conflict Resolution Strategies" - *Pattern-directed inference Systems* - D. A Waterman and F Hayes-Roth (Eds) - Academic Press, 1977.

[McDER82] D. McDermott - "A Temporal Loginc ofr reasoning about Porcesses and Plans" *Cognitive Science* , Vol 6, p. 101-155 (1982)

[McDER85] D. McDermott - "Reasoning about plans", *Formal theories of the Commonsense World*, J. Hobbs eds. p. 269-317 (Ablex Publishing, Norwood, New Jersey) (1985).

[MINS67] M. Minsky "Computation: Finite and Infinit Machines." - Prentice-Hall 1967.

[MORA73] T Moran "The Symbolic Imagery Hypothesis A Production Systems Models" CMU (1973).

[NEW&SHA60]	A. Newell, J. Shaw and H Simons "Report on a general problem-solving program" - *Proceeedings International Conference on Information Processing* - (UNESCO, Paris 1960): 256-264
[NEW&SIM57]	A Newell, J. Shaw and H. Simon - "Empirical explorations of the logic theory machine: A case study in heuristics." - The Rand corporation (1957).
[NEW&SIM65]	A. Newell and H. Simon "AN example of human chess play in the light of chess playing programs." *Progress in biocybernetics* - N. Wiener and J. Schade (eds.) Elsevier (1965).
[NEWE67]	A. Newell "Studies in Problem Solving: Subject 3 on the cryptarithmetic task DONALD + GERALD = ROBERT." - CMU (1967).
[NEWE69]	"Heuristic Progamming: Ill-Structured Problems" - A. Newell - *Progress in Operations Research* - Wiley, New York 1969, pp 361-414
[NEWE70]	A. Newell "Remarks on the relationship between artificial intelligence and cognitive psychology" - *Theoretical Approaches to Non-Numerical Problem Solving* - Part IV - Springer Verlag (1970).
[NEWE73]	"Production Systems: Models of control structures" - A. Newell - *Visual Information Processing* - W. Chase (Eds.) - Academic Press, 1973

[NII&AIE79]	P. Nii and N. Aiello "AGE: A Knowledge-based Program for building Knowledge-based Programs." - *Proceedings of the 6th International Joint Conference on Artificial Intelligence.* - W. Kaufmann 1979.
[NII&FEI82]	P. Nii, E. Feigenbaum, J. Anton and A. Rockmore "Signal-to-Symbol Tranformation: HASP/SIAP Case Study." - *AI Magazine* 3 (2): 23-35
[NIL&FIK71]	N. Nilsson and R. Fikes "STRIPS: A new approach to the application of theorem proving to problem solving". - *Artifical Intelligence* 2 (1971):198-208
[NILL80]	N. Nilsson "Principles of Artificial Intelligence" - Tioga Publishing Co. (1980): 298-302
[PEDN86]	E. Pednault - "Formulating Multiagent, Dynamic-world Problems, in the Classical Planning Framework" - *Reasoning about Actions & Plans, Proceedings of the 1986 Workshop, Timberline, Oregon* - M. Georgeff and A. Lansky eds. (Morgan Kaufmann Publishers, Los Altos, Cal.) (1986).
[REIT58]	"Heuristic Decision Procedures, Open Constraints and the Structure of Ill-Defined Problems" - W. R. Reitman - *Human Judgments and Optimality*- Wiley, New York 1964, pp 282-315.
[ROSE77]	S. Rosenschein "The Produciton System: Architecture and Abstraction" - *Pattern-Directed Inference Systems* - D. Waterman F. Hayes-Roth (Eds.) - Academic Press, 1977.

[RYC&NEW77] M. Rychner and A. Newell - "An instructable production system: Initial design issues" - *Pattern-Directed Inference Systems* - D. Waterman F Hayes-Roth (Eds.) - Academic Press, 1977.

[SACE75] E. Sacerdotti "A structure for plans and behaviour".....

[SELF59] O. Selfridge "Pandemonium: A paradigm for learing" *Proceedings of the symposium on the mechanization of Thought Processes* (1959)· 511-529

[SHOR76] E. Shortcliff "Computer based medical consultation· MYCIN" - American Elsevier (1976)

[SIM&NEW58] "Heuristic Problem Solving: The Next Advance in Operations Research" - H. Simon and A. Newell -*Opns. Res.*, 6, No. 1 1-10 (Jan.-Feb 1958).

[SIM&NEW72] "Human Problem Solving" - A. Newell and H Simon - Prentice-Hall (1972)

[SMIT72] D. Smith et al - Applications of artificial intelligence for chemical inference - *Journal of the American Chemical Society.* **94**. p 5962

[STEF81] M. Stefik "Planning with constraints (MOLGEN: Part1)" - Artifical Intelligence **16** (1981): 111-140

[SUSS75] G. Sussman "A computational model of skill acquisition" - American Elsevier - (1975)

[TATE75] A. Tate "Generating project networks" *Proceedings IJCAI-77*

[TERRY85] A. Terry - " The CRYSALIS Project Hierarchical Control of production systems" - *Tech. Rept HPP-83-19*, Stanford University, Heuristic Programing Project (1983)

[WARR74] D. Warren "Generating conditional plans and programs" *Proceedings AISB Summer Conference, University of Edinburgh* (1976) 344-354

[WATE75] D Waterman "Generalization learning techniques for automating the learning of heuristics" *Artificial Intelligence* - **1** 121 170

[WILK84] D Wilkins "Domain-independant planning Representation and plan generation" *Artificial Intelligence* **22** (3) (1984) 269 301

[WINO75] T Winograd "Frame representations and the procedural / declarative controversy" *Representation and Understanding* Bobrow and Collins (Eds)

[WINO83] T Winograd language as a cognitive process Addison Wesley (1983)