CONTROLLING PRODUCTION FIRING: THE FCL LANGUAGE *

Leonard Friedman

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Dr.
Pasadena, GA 91109

Abstract

While implementing a diagnostic expert system, FAITH, we have developed a new language, FCL, for controlling the firing of productions. FCL greatly simplifies the task of the user to direct this firing in a coherent and purposeful way without sacrificing some of the main advantages enjoyed by production systems. One of the interesting features of the language is the use of declarative forms to direct the firing of proper production sequences in a diagnostic expert system. Declarative forms which represent circuit diagrams and problem families control goal-directed processes like tracing through circuits and searching for symptoms to confirm or deny closely related problems. Because we have the FCL facility at our disposal, we have been able to incorporate a larger than usual number of diagnostic strategies in FAITH, with the assurance that they will be employed under the right circumstances.

Introduction

Ever since the advent of production systems, one of the key issues in using them has been the difficulty experienced in controlling the firing of sequences of productions, Davis and King offer an extended discussion of the subject which is still relevant [Davis 77], They mention many control methods, like ordering rules, tags, and meta-rules. With the growing popularity of expert systems organized around thousands of production rules, the issue of control has become even more pressing. Recently both Davis and Genesereth have introduced diagnostic production systems that employ still other control strategies [Davis 82, Genesereth 82], Genesereth has used the "linear input strategy", and the "unit preference strategy" for controlling a resolution-based diagnostician, terms defined in [Nilsson 80], All of these methods have been more or less ad hoc, and the need for a more systematic and orderly way to specify firings has remained.

During the course of implementing an expert system diagnostician, FAITH, we have developed a more orderly method, a language we are calling the the FAITH Control Language or FCL, One element in the operation of FCL is the use of extended declarative forms which represent circuit diagrams and system block diagrams, to enable the reasoning engine to choose rules and direct the correct instantiation of variables embedded in rule statements,**

We shall call these extended declarative forms relational maps. In FCL, the use of declarative knowledge for control purposes is combined with a number of other devices to permit the design of coherent strategies that direct whole sequences of production firings. Having this power, we are able to employ many diverse diagnostic strategies in a single system and are assured that they will be used in controlled circumstances.

Other control elements of the FCL include:

the use of predicate keywords in literals to identify subsets of rules related to a given firing sequence. An example would be "OutputFault", used to characterise rules pertaining to upstream tracing through a circuit,

special forms, "PreferredFor"s to assert constraints and direct rule choice. They are embedded in the rules themselves, A constraint might be that the subsystem under suspicion should be identified as an adder for the rule to apply,

special forms, "Modes", which name firing sequences directed in an orderly fashion by a "Tracing" is an example of a Mode,

the use of typed predicates called Given Predicates, which signify that a well formed formula (wff) in a rule is to be taken as true if it has a ground instance counterpart embedded somewhere in the relational maps. For design facts like "Part A is connected to Part B", Connected would be a Given Predicate, Rules which cannot be consistently instantiated when they contain such statements are rejected.

*The research described in this paper was performed by the Jet Propulsion Laboratory, California Institute of Technology, under contract with the National Aeronautics and Space Administration,

**This form of control is the equivalent of the linear input strategy used by Genesereth, He employs circuit diagrams "to avoid drawing unwarranted conclusions". Circuit diagrams serve this purpose in FCL, but have another important control function explained below.

The elements mentioned above can be put in one-to-one correspondence vith some of the elements of a procedural language. A Mode maps into a function. A PreferredFor vith constraints maps into a function call. Keywords identify the scope of a Mode by indicating to the reasoning engine which rules are to be considered when in that Mode. The relational maps provide a context in an execution sequence that spans across individual rule firings, thus controlling which production is to fire next. This is analogous to the way ordinary programming languages execute a sequence of statements. The process of composing a specific relational map and associated rules related to its use is the equivalent of coding a function in other languages.

Due to the very nature of production systems, there is no explicit expression of a conditional. The effect is obtained by composing the right set of rules to fire in the given domain contingencies. Although elements corresponding to a procedural language are present, we emphasize that we are not sacrificing the advantage that production systems provide, flexibility and the avoidance of FIXED sequences of executed statements. The production firings are still driven by the problem itself.

For a relational map to provide the context in a firing sequence, a certain style of rule composition must be adopted by the user. When this style is adopted, the system will either fire rules in the desired order, or fire the same rule many times with correctly altered instantiations. One aspect of the user style is to embed in a rule one or more Given Predicate formulas whose syntax corresponds exactly to selected statements in the relational map.

During execution, the reasoning engine can use partially instantiated Given Predicate formulas within the rule to find the match with a specific statement in the relational map. The engine completes the remaining instantiations from the matched knowledge. For example, if the relational map has defined a set of predecessor-successor relationships between objects in a signal path, then a rule under consideration which has instantiated a particular predecessor will find the correct successor.

In explaining the above rather abstract statements, we shall use examples from FAITH's operation. This will serve to make clear why we have introduced the various devices, and their functions. The knowledge bases from several domains will be introduced, and FAITH's application to the diagnosis of ailing spacecraft will be included. To make these examples easier to follow, we give a brief overview of FAITH* The concepts underlying FCL are general and we will discuss briefly their possible extension to the control of a planner.

The Basic FAITH Operation Cycle

FAITH employs predicate logic in its reasoning engine, and alternates between two phases of a basic diagnostic cycle. These phases are Explanation (Backward Chaining), and Confirmation or Denial (Forward Chaining). During explanation we are searching for consequent matches of production rules with the literal to be explained. Rules whose consequents match are used to establish a subset of the antecedents as hypotheses. Only literals whose predicates are untyped are chosen. Examples of untyped predicates are "Fault", "OutputFault", etc. The rule is chosen for further consideration if and only if the remaining literals in the antecedent which are not constraints can be matched with literals taken to be true. Once a set of possible hypotheses has been determined, the confirmation phase is entered.

During confirmation/denial we examine each of the selected hypotheses in turn, seeking to confirm one. We now search among the rules for antecedent matches to the untyped predicates that are the hypotheses to be confirmed. Any rule may be used which chains forward to specify one or more consequent predicates with the type "measurable". Such predicates are linked to the antecedent hypothesis because they tend to confirm it if true or within defined bounds, and deny it if false or outside defined bounds. These consequents constitute predictions about what should be measured if the antecedent is true. Their truth can only be established by comparing the prediction with an actual measurement, which has to be supplied by an outside agency.*

If a hypothesis is denied, the next one on the stack of hypotheses is examined until either all hypotheses are exhausted, or one is confirmed. If confirmation occurs, say at level-of-explanation 1, we reenter the cycle, seeking to explain the confirmed hypothesis. Confirmation may occur for one of the hypotheses at level 1 while others on the stack remain unexamined. If this is the case, we may go through several more cycles of explanation and confirmation at levels 2, 3, etc. Still later we may encounter a contradiction between predicted and measured test variables, that fires an inference contradicting the hypothesis at level 1. FAITH then backtracks, and activates the next unexamined hypothesis on the level 1 stack.

This is an oversimplified description, omitting complications such as measurements not being available, the hypothesis to be confirmed is itself a measurable, terminating the cycle, etc. Nevertheless, it does explain the main features of FAITH's operation.

♦During the development phase of FAITH, the outside agency has been the programmer. FAITH's initial application is to the troubleshooting of spacecraft by monitoring the telemetry stream transmitted to earth. This telemetry stream contains the test measurements from a wide variety of subsystems. We have developed a module, the EXECUTION MONITOR, to read the telemetry data stream, detect errors or out-of-bounds measurements, and automatically supply requested measurements to FAITH.

The FAITH Control Language (FCL)

Control of a Simple Firing Sequence: Traversing a Tree Structure

We shall consider a system requiring diagnosis whose top level organization can be represented as a tree of subsystems. A typical block diagram is shown in Figure 1, This is a simplified representation of the organization of parts of the VOYAGER spacecraft and ground system* Suppose the "presenting symptom", say "No Telemetry", causes the error detection mechanism of the EXECUTION MONITOR to trigger. This symptom has a multiplicity of possible causes scattered through the entire VOYAGER system. A human troubleshooter would want to systematically traverse the tree, top down, eliminating the largest and most likely blocks or subsystems, until he has found one that, from test measurements, indicates evidence of possible malfunction. How do we instruct FAITH to accomplish the same thing?

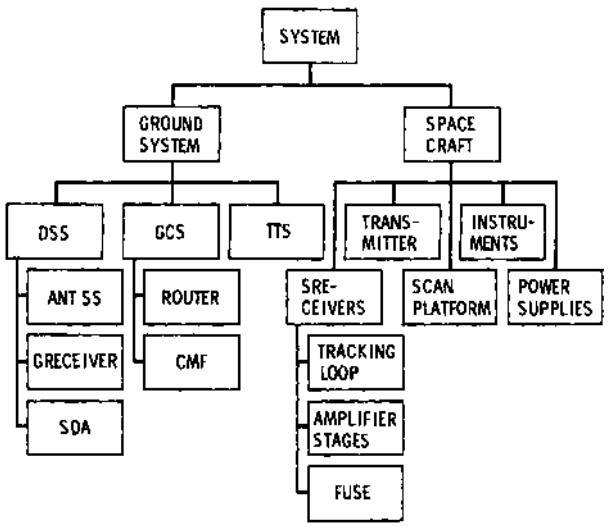

Figure 1: VOYAGER System Block Diagram

Recall that the presence of a Given Predicate within a rule literal indicates that a match is to be found in a relational map. Let us declare a predicate, "Contains", as a Given Predicate, where (Contains A B C) means "A contains B and C". Next we represent the tree structure of Figure 1 with statements such as those of Figure 2, forming one relational map.

The statements in Figure 2 are not in the form of literals needed for unification with those literals contained in the production rules being processed by the reasoning engine. In the initial version of FAITH we have written a translator which creates selected subsets of proper literals on demand from the statements in the relational map. For example, the first statement in Figure 2 is translated into two literals, (Contains System GroundSystem) and (Contains System Spacecraft).

```
(Contains System GroundSystem Spacecraft)
 (Contains GroundSystem DSS1 GCS1 TTSI)
  (Contains DSS1 AntSSI GReceiverl SDA1 88A1 TPA1
                                        CMF1 DI81)
 (Contains Spacecraft SReceiverl SReceiver2
  STransmitter1 Instruments ScanPlatform1
                                        PowerSupplies)
  (Contains SReceiverl TrackingLoopl
   AmplifierStages1 RPC1 Limiter5 Fuse27)
```

Figure 2: A Relational Map of a Tree Structure: VOYAGER System Blocks

Each of the predicates that requires expansion has been declared as an expansion type and the type is placed on its property list. From this information, the translator can select the correct translation parse. Later versions will have a more sophisticated marker device in each expression so that expansion types need not be used.

Suppose then, ve write a rule of the form shown in Figure 3. Note the correspondence of the Contains literal in the rule to the syntax of the expressions translated from the relational map. Assume that we can start the rule firing properly at the top of the tree. Then, once it got started, FAITH would process each subsystem at the VOYAGER top level (shown in Figure 2) in the order in which they were listed, declaring a Fault in each until one of these hypotheses was confirmed as described in Basic Operation. Assuming one hypothesis was confirmed, say (Fault GroundSystem), this in turn would be asserted as requiring an explanation by backward chaining, and the next level of subsystems directly attached to or contained in the Ground System would be successively declared at fault, because only those subsystems are present in the Contains ground instances with the term GroundSystem in the second position.

```
(<=SwitchToFocus Inference
 ((Contains ?System ?Subsystem)
  (Fault ?Subsystem))
 --->
 ((Fault ?System))

 (PreferredFor ((Fault ?system)
                (COND ((MEMBER ?System FocusList)
                       (SETQ Mode 'Focus)))))

 [Terms beginning with "?" are variables.]
```

Figure 3: Rule for Traversing a Tree

The controlled firing of this rule would continue to any depth, provided there were a Fault confirmed at each level. Ve shall shortly describe ways to terminate this sequence appropriately. Thus we see that we have formed a plan for a complete production firing sequence employing only a single rule. This was accomplished by writing the relational map of Contains statements. Ve call this pattern of firings that traverses a tree structure of subsystems "focusing'* and refer to it as the Focus Mode. Mote that we have taken pains to make the user represention a form equivalent to that ordinarily used by humans to describe system organization. Thus it is particularly easy for

humans to supply it.

We turn our attention now to how ve initially steer the firing to the top of the tree. The mechanism used is the "PreferredFor" with constraints*** The constraint applied here is list membership, with the objects we know are going to be encountered in diagnosing the system declared in advance in the knowledge base as members of an appropriate Mode list •

A PreferredFor is present in Figure 3, with the constraint that the instantiation of the variable "?System" be a member of the Focus list. Since GroundSystem is on this list, this rule will fire if (Fault GroundSystem) is declared. An idiosyncratic rule to do this is easily supplied and relates the initially presented symptom (No Telemetry) with two hypotheses (Fault GroundSystem) and (Fault Spacecraft)* Note that the Focus mode will be invoked whenever an object at any level that is a member of the Focus list is asserted to be at fault.

The method of terminating a mode is to declare (Fault <object>). All PreferredFors start with this literal or a similar keyword • The convention enables the user to select the next mode by determining the object's list membership* This has the effect of throwing open the choice of Mode, rather than the choice of all production rules* By sacrificing the complete freedom to choose any rule after each firing, we have gained a great deal of control and speed*

Transitions Between Firing Patterns: Mode Calls

Focusing is a diagnostic pattern particularly appropriate for top level subsystems that have relatively little interaction with each other* Eventually, we will reach a system with internal signal flows and chains of dependency among its subsystems* Faults may best be localized in such a system by tracing* In FAITH, tracing includes a number of Modes in which the internal structure of subsystems is ignored, with only the input/output relations described by a transfer function* The path to be followed is that of causal dependency* Normally this is defined by design signal paths, and those paths are initially assumed by humans to still hold in a malfunction* Ve make the same assumptions in FAITH* Of course, we are not limited to such design paths, and are presently implementing thermal causal pathways involved in failures of a VOYAGER instrument*

Ve call the UpstreamTrace Mode by using the PreferredFor mechanism with the constraint that the suspected system be on the UpstreamTrace list. Figure 4 shows the rule that accomplishes this* Note that what has happened in our consideration of various systems during firing of the Focus rule is that eventually one of them which is confirmed at fault is a member of a different Mode list* When this happens Focus mode is terminated and the different Mode entered*

```
(<=SvitchToTrace Inference
 ((OutputFault ?Subsystem)
  (Ports ?System (OutputPort ?Port))
  (Connected (Out (?Pin ?Subsystem))
             (OutputPort ?Port)))

 ((Fault ?System))

 (PreferredFor
   ((Fault ?System)
    (COND ((MEMBER ?System UpstreamTraceList)
           (SETQ Mode 'UpTrace))))))
```

Figure A: Selecting a Trace Mode

Here we have introduced a new predicate. OutputFault* OutputFault is a keyword that enables FAITH to restrict rules that are candidates for firing to those related to tracing in the upstream direction, against the signal flow. All rules related to upstream tracing contain the keyword predicate OutputFault*

Control of Simple Tracing; A Linear Chain

If we add declarations in a new relational map defining the input and output ports, and the signal flow, we can control the firing of simple upstream or downsteam tracing which halts whenever a fault is found with input ok and bad output. InputOR serves for scoping downstream tracing. Ve introduce a Given predicate. (Connected A B). which is defined by the property that a signal value appearing at A will be shared by B, This can represent either electrical wires or higher level signals passing between subsystems.

Figure 5 shows the Deep Space Station or DSS, which receives transmissions from the spacecraft. The DSS may be represented by a group of subsystems starting with the Antenna Subsystem, feeding into the Ground Receiver which feeds in turn into a subsystem designated the SDA. Signal flow may be considered good or bad depending on whether the synchronization of the telemetry stream is locked or unlocked.

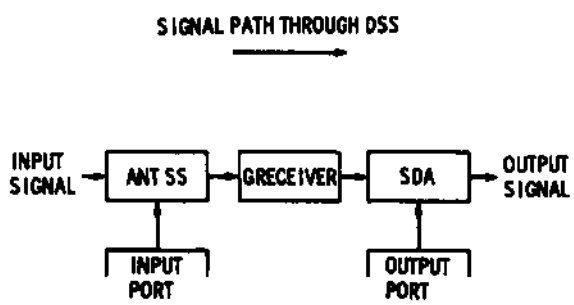

Figure 5: The Deep 8pace 8tation (DSS)

The station may be represented in a relational

**PreferredFor without constraints was used in the planner. DEVISER, to choose among several actions that might accomplish the same goal [Vere 83].

```
(Ports DSS1 (InputPort D1) (OutputPort D2))
(Ports Ant881 (InputPort A1) (OutputPort D2))
(Ports GReceiverl (InputPort Gl) (OutputPort G2))
(Ports SDA1 (InputPort Sl)(OutputPort S2))
(Connected (InputPort Dl) (In (Al AntSSl)))
(Connected (Out (A2 AntSS)) (In (Gl GReceiverl)))
(Connected (Out (G2 GReceiverl)) (In (S1 SDA1)))
(Connected (Out (S2 SDAl)) (OutputPort D2))
```

Figure 6: Relational Map of DSS

map as in Figure 6.

Two of the rules that are used in the tracing of this linear chain are shown in Figure 7 and Figure 8.

```
(<=TraceUpstream Inference
 ((OutputFault ?Predecessor)
  (Connected (Out (?PinP ?Predecessor))
             (In (?PinS ?Successor))))
 --->
 ((OutputFault ?Successor)))
```

Figure 7: Rule that Propagates a Faulty Signal Upstream

```
(<=Up8treamTerminatorl Inference
              ((InputOK ?Subsystem)
               (Fault TSubsystem))
              --->
              ((OutputFault ?Subsystem)))
```

Figure 8: Rule that Detects a Possible Fault

Four rules are required for backward chaining to trace through the DSS in the upstream direction using the connection information of Figure 6.

The Potential Problem Mode

The diagnostic goal of the firing sequences described to this point is to localize the malfunction to the smallest subsystem possible. Once this has been accomplished, we may be able to characterize the problem within that unit more precisely than simply "Fault". A great deal of diagnostic knowledge exists about various units such as receivers and transmitters, and this can be incorporated in the Problem knowledge base as another tree structure. Such knowledge is taught in medical school as disease families. Figure 9 shows a simple representation of this kind of knowledge.

"PotentialProblem" is a Given Predicate, and "Problem" an untyped predicate subject to confirmation. If we introduce the rule shown in Figure 10, we insure making the proper transition when the appropriate systems are encountered.

"ValueOf" causes the function "ClassMember" to be EVAL'ed. ClassMember is an inverse indexing function that retrieves the class name such as "Receiver" when given a specific instance such as "GReceiverl". With an additional rule similar to

```
(PotentialProblem Receiver AmplifierStageProblems
  Oscillations TrackingLoopFailures Shorts
  PowerSupplyFailures)

(PotentialProblem
  Transmitter AmplifierStageProblems
  SpacecraftMispointed Shorts PowerSupplyFailures)

(PotentialProblem
  Shorts ShortsWithFuse ShortsWithLimiter/Fuse)
```

Figure 9: Potential Problems of Typical Units, and a Problem Family

```
(<=ProblemClassificationl Inference
 ((Problem ?Class Problem)
  (ValueOf ?Class (ClassMember ?System))
  (PotentialProblem ?Class ?Problem))
 --->
 ((Fault ?Sy8tem))

 (PreferredFor
   ((Fault ?Syetem)
    (MEMBER ?System PotentialProblemList))))
```

Figure 10: Transition to Problem Characterization

the one introduced in Figure 3 we can now traverse the potential problem tree, and in many instances can pin down the nature of the problem.

Control of Complex Tracing

We present a non-VOYAGER problem as a last example of the power of this method of controlling production system firing. The problem posed is a digital subsystem discussed in [Davis, et al 82], which also uses design knowledge extensively, but applied to generate tests rather than to explicitly control production firing. It is easy to see that if a subsystem of this type were present in a VOYAGER system, FAITH might declare it at fault by the methods of focussing or tracing. We now consider the control of production firings in the attempt to localize the problem to a component within the subsystem.

Davis employed procedural methods to represent the design and simulate the workings of the complex circuit of multipliers and adders shown in Figure 11.

We represent this circuit diagram as shown in Figure 12. A rule for representing the transfer function of an adder is given in Figure 13. An assumption we make is that only the input ports and output ports are available for measurement, so that indirect inferencing is required to determine possible candidate faulty components. The assumptions that there is only a single fault, and no intermittent faults are also being made in this case.

To perform this diagnosis in FAITH, we introduced some seventeen rules and additional firing modes. One of these additional modes is simulation, introduced by a PreferredFor. With rules that define transfer functions for the adders and

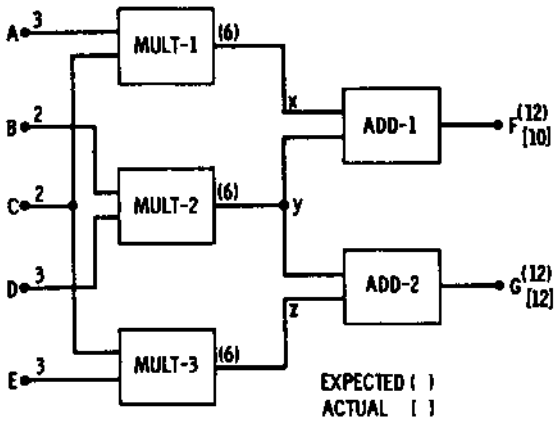

Figure 11: A Digital Circuit

```
(Contains Digil  Multl Mult2 Mult3 Add1 Add2)
(Multiplier Multl Mul2 Mult3)
(Adder Addl Add2)
(Ports Digil
      (inputPort A B C D E) (OutputPort F G))
(Connected (InputPort A) (In (1 Multl)))
(Connected (InputPort B) (In (1 Mult2)))
(Connected (InputPort C) (In (2 Multl))
                         (In (1 Mult3)))
(Connected (InputPort D) (In (2 Mult2)))
(Connected (InputPort E) (In (2 Mult3)))
(Connected (Out (1 Multl) (In (1 Addl)))
(Connected (Out (1 Mult2) (In (2 Addl)
                          (In (1 Add2)))
(Connected (Out (1 Mult3) (In (2 Add2)))
(Connected (Out (1 Addl)) (Outport F))
(Connected (Out (1 Add2)) (Outport G))
```

Figure 12: FAITH representation of Digil circuit diagram

```
(Addition=> Event
 ((OK ?a)
  (Adder ?a)
  (MeasuredValue (1 ?a) ?x)
  (MeasuredValue (2 ?a) ?y)
  (ValueOf ?z (plus ?x ?y)))
 ——>
 ((Out (1 ?a) ?z)))
```

Figure 13: Rule Representing Adder Transfer Function

multipliers (Figure 13), using repeated forward chaining in the simulation mode, and guided by our wiring diagram (Figure 12), we can generate predicted normal outputs from measured inputs. We are thus simulating the normal circuit operation, and can detect errors by measuring different output values than those predicted*

Having accomplished this, still another firing mode is required. This mode propagates back from a detected error, guided by the wiring diagram, to find the possible sources of error among the adders and multipliers. Each time such a source of error is located, an error is hypothesized in that source and propagated forward to all possible output ports. The pattern of errors produced by the error hypothesis is then compared with the actual pattern produced, and, if the patterns match, the hypothesized source is added to the list of candidates for further analysis. (These methods will not work for circuits containing feedback loops.)

All of these complex production rule firing sequences and transitions are successfully controlled by the use of the methods described. The great advantage is that FCL's control mechanisms are quite general, and we can analyze other circuits by altering only the rules that represent the transfer functions of the subsystems. Thus we have made available to the user a file of rules that are always read into FAITH and sufficiently general purpose as to be useful for many different diagnoses. These include all of the modes discussed above.

Altering Relational Maps

Until now we have been considering the relational maps as given and unchangeable fact. The diagnostic performance this produces is often quite inconvenient and clumsy compared with an experienced human troubleshooter. For example, a list of potential problems is processed by the reasoning engine in a fixed order, whereas a humar, would alter the order of consideration based on measured evidence. Changing the order of consideration of problems on the basis of newly confirmed evidence is an important feature of many medical diagnostic systems, such as PIP [Szolovits 78].

To understand how this control feature is implemented in FAITH, we have to describe another aspect of its operation. FAITH permits making forward-chaining inferences on the basis of confirmed test measurements. The rules that are employed for these inferences are segregated from other rules, so that these rules will not fire in normal forward or backward chaining. This makes it possible to infer, from evidence gathered in a particular context, that the most likely cause of a problem is something other than the current line of inquiry.

Using such an inference, we have included a rule in a VOYAGER knowledge base whose effect is to change the order of items in a list to make the most likely cause first, and to abandon the current line of inquiry. The list is one pertaining to potential problems. This can be generalized so that declarative knowledge about symptoms can be consulted using "symptom" predicates rather than Given predicates to instantiate variables in formulas about symptoms*

This suggests the possibility of implementing in FAITH the kind of declarative tree structures used in CASNET [Weiss 78] to relate symptoms to intermediate states (pathophysiological states). In turn, the intermediate states are grouped into a disease category as part of the diagnostic process for the various forms of glaucoma* This is the

inverse of the reasoning process employed in FAITH of hypothesizing a disease cause and then looking for a set of symptoms to confirm it. The inverse, using symptomatic patterns and groupings to suggest a diagnosis is employed as readily by physicians and troubleshooters as is the hypothetical method.

The FCL feature of being able to modify the relational maps greatly extends the power of this diagnostician. For example, we can hypothesize that a short has occurred and determine its location. If that is confirmed, we can alter the original circuit diagram to include a connection that represents the short, and even run a simulation of the changed circuit to determine the state changes it produces with expected inputs. The state changes resulting from the use of the altered circuit can then be simulated to determine its behavior.

Planning

We have just begun to consider the application of FCL to the control of our planners, DEVISER [Vere 83] and SWITCH [Porta 85]. Two examples that we have implemented are the planning of a trip, and a problem in avoiding inefficient backtracking in a VOYAGER setting. The trip planner uses cartographic map information represented as a number of "Connected" statements to define the routes, very much in the manner of the circuit diagrams employed in diagnosis. By contrast, the inefficient backtracking problem requires a somewhat different approach than those we have described, although we still employ additional declarative knowledge to render the backtracking efficient. More extensive investigation is needed to determine whether we shall have to incorporate many new features in FCL and make corresponding changes in the planner s reasoning engine to have the desired degree of control over the planning process.

Non-Monotonic Logic: Changing Axioms and Removing Rules

As in any expert system, diagnosis is limited by the assumptions underlying the causal models employed. These underlying assumptions often are not explicitly stated, but are used to support the validity of many of the rules. Thus a particular line of inquiry may lead to a single logical conclusion based on the current axioms, and subsequent testing of the conclusion (for example, by actually replacing a suspected unit) may reveal that the conclusion was wrong. In this case we infer that at least one of the current axioms is wrong.

We are currently altering FAITH so that we can relate given rule sets to their underlying axioms, and make it possible to retract axioms and substitute others. Concomitantly, we will be able to withdraw rules from consideration that were linked to the retracted axioms and bring up for consideration new sets of rules linked to newly substituted axioms. The effect is to redefine the Modes, which depend on the rules available for their execution. This permits us to consider successively less likely causes for the problems encountered which require new rule sets to pursue, while keeping the often contradictory rules segregated. Note that the order of consideration of the less likely causes is susceptible to dynamic reordering on the basis of evidence, as explained earlier.

We are also using meta-rules that depend on some of our axioms. For example, we have a rule that says, "When we assume that a fault is located in a subsystem, prefer rules that mention the subsystems explicitly." These too will become inactive as their supporting axioms are withdrawn.

Discussion

We have presented the presently implemented features of a new control language, FCL, which greatly simplifies the task of the user to direct the firing of productions in a coherent and purposeful way without sacrificing some of the main advantages enjoyed by production systems. Several users, know led gable in AI, who were unacquainted with the internal coding of FAITH, have been able to code extensive VOYAGER relational maps and many rules in a matter of weeks for an actual VOYAGER instrument in much greater detail than the simplified examples used in this paper. FAITH has produced the desired diagnoses and duplicated satisfactorily the performance of the expert. The training the AI users required was minimal. There have even been users with minimal AI knowledge who have successfully used FCL.

Because we have this facility at our disposal, we have been able to incorporate a larger than usual number of diagnostic strategies into FAITH, with the assurance that they will be employed under the right circumstances, and we expect to continue to enlarge the scope of diagnostic techniques available in a quite general manner. Because it possesses many diagnostic strategies, FAITH is approaching operational usefulness in a real engineering environment, and is undergoing testing in that environment. The philosophy underlying FCL is certainly not limited to diagnosis. It may be applied fruitfully to other production systems and we are beginning to do so.

References

Davis, R. and King, J., "An Overview of Production Systems", Machine Intelligence 8, ed. Elcock and Michie, John Wiley, 1977,

Davis, R. Shrobe, H. Hamscher, V. Wieckert, K Shirley M. and Polit, S. 'Diagnosis Based on Description of Structure and Function", Proc. Nat, Conf. on AI, Aug. 18-20, 1982.

Genesereth, M. R., "Diagnosis Using Hierarchical Design Models", Proc. Nat, Conf. on AI, Aug. 18-20, 1982.

Nilsson, N. "Principles of Artificial Intelligence", Tioga Publishing Company, 1980.

Porta, H. "Dynamic Replanning", unpublished JPL Report, to appear Jan. 1985.

Ssolovits, P. and Pauker, S. G. "Categorical and Probabilistic Reasoning in Medical Diagnosis", Artificial Intelligence, Vol. 11, August 1978.

Weiss, S. M. Kulikovski, C. A. Amarel, S. and Safir, A, "A Model-Based Method for Computer-Aided Medical Decision Making", Artificial Intelligence, Vol. 11, August 1978.

Vere, S. A. "Planning in Time: Windows and Durations for Activities and Goals", IEEE Trans, on Pattern Analysis and Machine Intelligence, Vol. PAMI-5, 3, May 1983.