To the Graduate Council:

I am submitting herewith a thesis written by Uday Kumar Gupta entitled "A Knowledge Based Hybrid Approach to an Expert System for Power System Control." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Seung C. Lee, Major Professor

We have read this thesis
and recommend its acceptance:

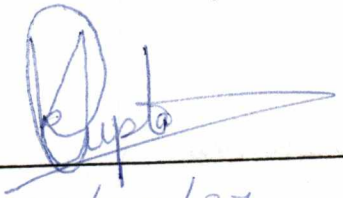
Accepted for the Council:

Vice Provost
and Dean of the Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature 

Date 10/23/87

A KNOWLEDGE BASED HYBRID APPROACH TO AN EXPERT SYSTEM FOR POWER SYSTEM CONTROL

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Uday Kumar Gupta

December 1987

DEDICATION

This thesis is dedicated to my parents, Sarla and Sachidanand Gupta. They have unselfishly provided me with many educational opportunities as well as given me the timely encouragements necessary to complete this task. Their constant love has made many things possible and I wish to present this as a token of my love and appreciation.

ACKNOWLEDGMENTS

The author wishes to extend his deepest gratitute to Dr. Seung C. Lee who has provided invaluable guidance throughout the development of this thesis. His patience and concern during this time are greatly appreciated.

The author wishes to extend sincere thanks to Professors Moonis Ali and Kenneth K. Kimble, for their invaluable time in going through this work, and giving necessary suggestions.

ABSTRACT

Many of the engineering problems require a knowledge based hybrid approach. The term hybrid here indicates the approach in which the knowledge based symbolic processing programs are supported by the algorithmic analysis programs. The knowledge based hybrid approach is particularly necessary in the case of power system control problems where the expert knowledge needs to be supported by analyses such as the power flow analysis and the sensitivity analysis.

In narrow problem domains, expert systems can provide high performance, equalling or even exceeding those of individual human experts. This thesis describes a knowledge based hybrid approach in building an expert system, to bring a power system from an alert or an emergency state to a normal or the best possible state under given constraints.

A prototype of the expert system with machine learning capability has also been implemented on Symbolics 3670 Lisp Machine, which demonstrates a promising performance. The prototype is referred as EPSY, abbreviation for ``Expert System for Power System Control,'' throughout in this thesis. The power system configuration chosen for implementaion is popularly known as ``New England 39 Bus System.'' It is demonstrated that by integrating symbolic processing program with the power system analysis programs, the expert system exhibits a superior performance than can be obtained by using just the symbolic processing programs.

Two power system analysis programs, the power flow analysis and the sensitivity analysis programs have been used. The inference and control for the problem solving system and the machine learning subsystem of this expert system uses the

blackboard architecture, which has been found particularly suitable for our problem.

An important feature of this expert system is that while attempting to correct bus voltage violations, the expert system learns from its past failures during its problem solving process. An attempt to correct a bus voltage violation that fails will help the expert system to learn the reasons that lead to the failure, so that the expert system can correct its strategy in its next attempt.

All aspects of such an expert system have been covered with particular emphasis on the machine learning. Finally the recommended future research and the scope of such a design approach have been mentioned.

TABLE OF CONTENTS

CHAPTER	PAGE
1. STATE OF THE ART IN EXPERT SYSTEMS	1
1.1. An Overview Of Expert Systems	1
1.2. Expert Systems For Power System Control	2
1.3. Future Outlook	4
2. INTRODUCTION	6
2.1. Objective of EPSY	6
2.2. Need for EPSY	6
3. THE EXPERT SYSTEM (EPSY) ARCHITECTURE	9
3.1. The User Interface	10
3.2. The Knowledge Base	13
4. THE BLACKBOARD BASED INFERENCE AND CONTROL STRATEGY	21
4.1. The Blackboard Architecture in EPSY	21
4.2. The Knowledge Application Strategy	30
4.3. Heuristic Evaluation Function	34
5. THE POWER SYSTEM ANALYSIS MODULES IN EPSY	36
5.1. Power Flow Analysis	37
5.2. Violation Simulation	39
5.3. Sensitivity Analysis	40
6. MACHINE LEARNING IN EPSY	46
6.1. Background Of Machine Learning	46
6.2. Need For Machine Learning in Expert Systems	47
6.3. Machine Learning Subsystem in EPSY	47
6.4. Final Remarks on the Machine Learning in EPSY	55

7. CONCLUSION 57
7.1. Further Research 57
7.2. Scope of this Expert System 58
7.3. Concluding Remarks 60
BIBLIOGRAPHY 62
APPENDICES 65
APPENDIX A. EPSY PROGRAM 66
APPENDIX B. EPSY DEMONSTRATION 112
VITA 123

LIST OF FIGURES

FIGURE		PAGE
1.	CONFIGURATION OF ``NEW ENGLAND 39 BUS SYSTEM''	8
2.	EXPERT SYSTEM (EPSY) ARCHITECTURE	11
3.	INFERENCE AND CONTROL STRATEGY ORGANIZATION	23
4.	BASIC FORM OF ORGANIZATION FOR MACHINE LEARNING BY OBSERVATIONS	48
5.	ELABORATION ON MACHINE LEARNING BY OBSERVATIONS	49

CHAPTER 1

STATE OF THE ART IN EXPERT SYSTEMS

1.1. An Overview of Expert Systems

In recent years, research in the field of Artificial Intelligence (AI) has had many important successes. Among the most significant of these has been the development of powerful new computer systems known as``expert'' or ``knowledge based'' systems. These programs are designed to represent and apply knowledge of specific areas of expertise to solve problems. For example, collaborative efforts by human experts and system developers have resulted in systems which diagnose diseases, configure computer systems, and prospect for minerals at performance levels equal to or surpassing human expertise. The potential power of systems which can replicate expensive or rare human knowledge has led to a worldwide effort to extend and apply this technology.

Over time, the knowledge engineering field will have an impact on all areas of human activity where knowledge provides the power for solving important problems. We can foresee two beneficial effects.The first and most obvious will be the development of knowledge systems that replicate and autonomously apply human expertise. For these systems, knowledge engineering will provide the technology for converting human knowledge into industrial power. The second benefit may be less obvious. As an inevitable side effect, knowledge engineering will catalyze a global effort to collect, codify, exchange, and exploit applicble forms of human knowledge. In this way, knowledge engineering will accelerate the development, clarification,

and expansion of human knowledge itself.

Expert systems differ in important ways from both conventional data processing systems and systems developed in other branches of AI. In contrast to traditional data processing systems, AI applications generally involve several distinguishing features, which include symbolic representation, symbolic inference, and heuristic search. Expert systems differ from the broad class of AI tasks in several aspects. First, they perform difficult tasks at expert levels of performance. Second, they emphasize domain-specific problem-solving strategies over the more general ``weak methods'' of AI. Third, they employ self-knowledge to reason about their own inference processes and provide explainations or justifications for conclusions reached. And, last, they solve problems that generally fall into one of the following categories: interpretation, prediction, diagnosis, debugging, design, planning, monitoring, repair, instruction, and control.

1.2. Expert Systems for Power System Control

Some attempts have been made in the building of expert systems or knowledge based systems for power system control or restoration. The paper titled ``Development of a knowledge based system for power system restoration,'' by T. Sakaguchi and K.Matsuomoto, of Mitsubishi Electric corporation, Japan published in IEEE Transactions on Power Apparatus and Systems, Vol. PAS-102, No. 2, Feb. 1983, is probably one of the initial and important attempts in using Artificial Intelligence techniques in the field of power system restoration and control. This paper by Sakaguchi and Matsuomoto gives its summary as follows: ``A knowledge based system is developed for automatic control of a power system in the restorative state. System operators's heuristics seem to play a central role during the course

of system restoration. So efforts are made to represent those heuristics by a set of knowledge the goal of of which is to restore the power supply without overloading any transmission line. While the overload check is a numerical computation, reasoning in the knowledge based system is a symbolic computation. A method to link these two different types of computation is proposed. The knowledge based system is expected to serve managing a large scale power system in the restorative state.''

``Intelligent knowledge based programs have the potential for great benefit to real-time operations and training, especially if they are designed to be readily integrated into the problem solving environments of energy control centers.''[1] Efforts made by Carnegie Mellon University in investigating the areas of power system operations where artificial intelligence tools like OPS5, KEE-II, ART etc. can be used, resulted in the set of problems that are feasible and have high payoff, some of them are as follows:

(1) **Alarm Processing:** When many alarms go off simultaneously, identify the primary alarms.

(2) **Fault Diagnosis:** When a disturbance occurs, identify what it is.

(3) **Contingency Selection and Security Analysis:** Identify the likely contingencies and assess the effects they would have.

(4) **Normal Control:** Operate the system so as to maintain a good balance between cost and security.

(5) **Preventive Control:** Restore security to an adequate level when it has been found to have slipped.

(6) **Emergency Control:** When the load or operating constraints of a system

[1] ``Artificial Intelligence Technologies for Power System Operations,'' prepared by Carnegie-Mellon University for Electric Power Research Institute. Final report, January 1986.

have been violated, take action to minimize further violations and, if possible, eliminate existing violations.

(7) **Restorative Control:** When loads are not being supplied and after the system has reached a steady state, take action to restore supply.

(8) **Operator Training:** Intelligent simulators and tutors for operator training.

At this time not many expert systems are available to solve the problems in the domain of power systems. There is a great need for expert systems in solving various problems in the area of power systems, where unfortunately not much of Artificial Intelligence research has been done up till now.

1.3. Future Outlook

``Clearly expert knowledge provides a rich source of opportunities for intelligent systems. Workers in this field speculate that the most significant byproduct of expert systems work will be the codification of knowledge. The technology of expert systems provides an opportunity for feedback and knowledge evaluation. This will stimulate development of new knowledge, especially knowledge related to performing tasks expertly. In many areas of human endeavour, this will signal a transition from a personal to a public basis for thought. This transition will revolutionize human culture.

Expert systems are expected to become quite adept at rationalization, at examining and analyzing their own chains of reasoning during problem-solving. Future systems will do more than just display the rules that produced their conclusions. To explain their operation, they will reconstruct rational lines of argument from fundamental principles of the domain, tailored to fit the perceived audience. This

self-knowledge will also provide the basis for various types of self-modification capabilities, including rule correction, knowledge base reorganization, and system reconfiguration.

The next generation of expert systems can be expected to fare well in the commercial marketplace. Numerous applications readily suggest themselves in which a computerized expert can play some important role. The field should be expected to develop rapidly in areas of great commercial interest, such as energy, and minerals, home entertainment, office automation, and military systems.''[2]

[2] ``Building Expert Systems,'' edited by Frederick Hayes-Roth, Donald A. Waterman, Douglas B. Lenat.

CHAPTER 2

INTRODUCTION

2.1. Objective of EPSY

The purpose of this expert system is to bring a power system from an alert or an emergency state to a normal state. This objective is achieved by making use of both the heuristic knowledge and the sensitivity information during the process of expert decision making just like the power system experts would.

The decision making is needed mainly to find out those generators whose generation should be changed and the extent to which they should be changed within the constraints of the power system being controlled, such that the power system is brought to the best possible overall state. The capability of machine learning which is an imporatant feature of this expert system has been attempted and implemented in the prototype expert system.

2.2. Need for EPSY

A power system under alert or emergency state should be brought to normal or best possible state as soon as possible. Time is critical because many of the power system components, under violated conditions, may not be able to survive for an extensive time. Also, there exists a good chance of aggravation of the state with time by adding more severe violations to the power system. This cascading effect makes it more difficult to bring the power system back to normal state again. Even for the most experienced human expert, it is very difficult to select appropriate

control measures among various alternatives while considering all the constraints.

A power system has usually associated with it an extensive amount of domain knowledge and numerical state information which increases as the size of the power system increases. Especially, the information relevant to a specific operating status is far too much to be remembered by a power system control expert even for a fairly small size power system. For this reason, a knowledge based hybrid approach is used for developing this expert system to make the best use of all the domain knowledge, the heuristic knowledge, and the large amount of numerical information which are critical in supporting the process of expert decision making for controlling a power system.

In order to bring out the salient features of this expert system design approach, the power system configuration of the ``New England 39 Bus System'' is used for illustration, as shown in Figure.1. This power system has 39 buses, nine of which are voltage buses, one is a swing bus, and the rest are load buses. There are overall ten generators, one of which is a swing generator connected to the swing bus.

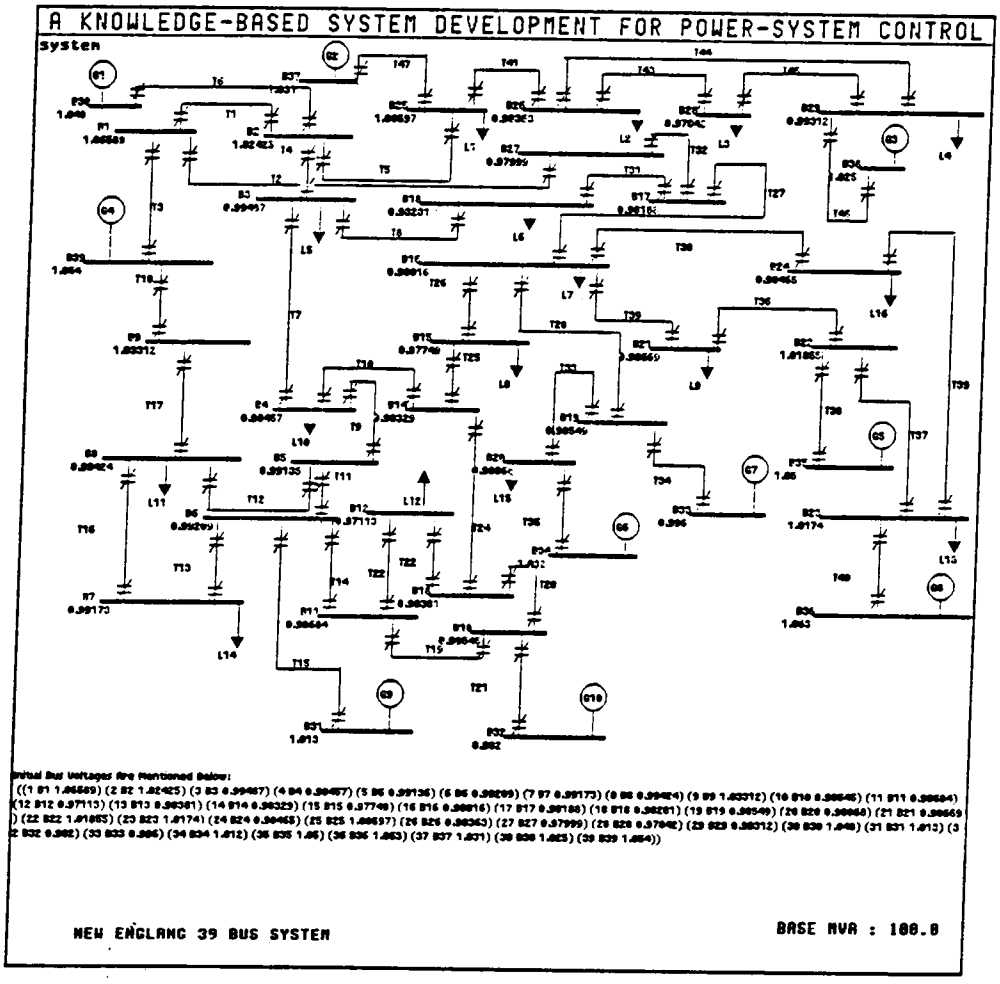


FIGURE. 1. CONFIGURATION OF "NEW ENGLAND 39 BUS SYSTEM"

CHAPTER 3

THE EXPERT SYSTEM (EPSY) ARCHITECTURE

The overall design and architecture of the expert system is directed towards problem solving and machine learning using heuristic and domain knowledge related to the power system, with an emphasis on the way the expert system can convey itself to and interact with the user.

Generally, the approach or the problem solving strategy a system takes to solve a problem also reflects its organization, level of intelligence, efficiency, speed, and flexibility in terms of future additions or modifications to the system architecture. In another way, the approach may decide the overall architecture and performance of the system to a large extent. For this reason it is extremely important to take into account all relevant factors related to the problem solving, area of application and the performance standards the system is expected to meet. For example, a system solving a complex problem in a narrow domain may take an artificial intelligence approach, by having an architecture similar to an expert system. An expert system may have components like user interface, knowledge base and an inference and control mechanism. Each of these components may take an approach to best suit its expected performance.

In case of EPSY, a number of factors related to the performance have been considered before deciding the approach to be taken for the overall system design and the design of the components of the system. These factors are the system speed, accuracy of results, flexibility in modification or addition to the overall system architecture, user friendliness, complexity of organization and learning capability

of the system. The inference and control strategy for the problem solving and machine-learning is based on the blackboard architecture.

This expert system is comprised of the following four major components:

1. The User Interface.
2. The Knowledge base.
3. The Blackboard Based Inference And Control Strategy.
4. The Power System Analysis Modules in EPSY.

Only the first two of the above mentioned components are described in this chapter. The remaining components are described in later chapters (Chapters 4, 5, 6). Please refer Figure.2. for the architecture of EPSY.

3.1. The User Interface

This acts as a front end processor of the EPSY, and is comprised of the following four components:

3.1.1. Input Handler.

3.1.2. Graphics Interface.

3.1.3. Natural Language interface.

3.1.4. Output Handler.

Brief description and need for the above mentioned components are given next.

3.1.1. Input Handler

EPSY accepts the input from a user through the keyboard and the mouse.

The user types in the information on various attributes of the power system components using keyboard, as requested by the system on the Dialogue window.

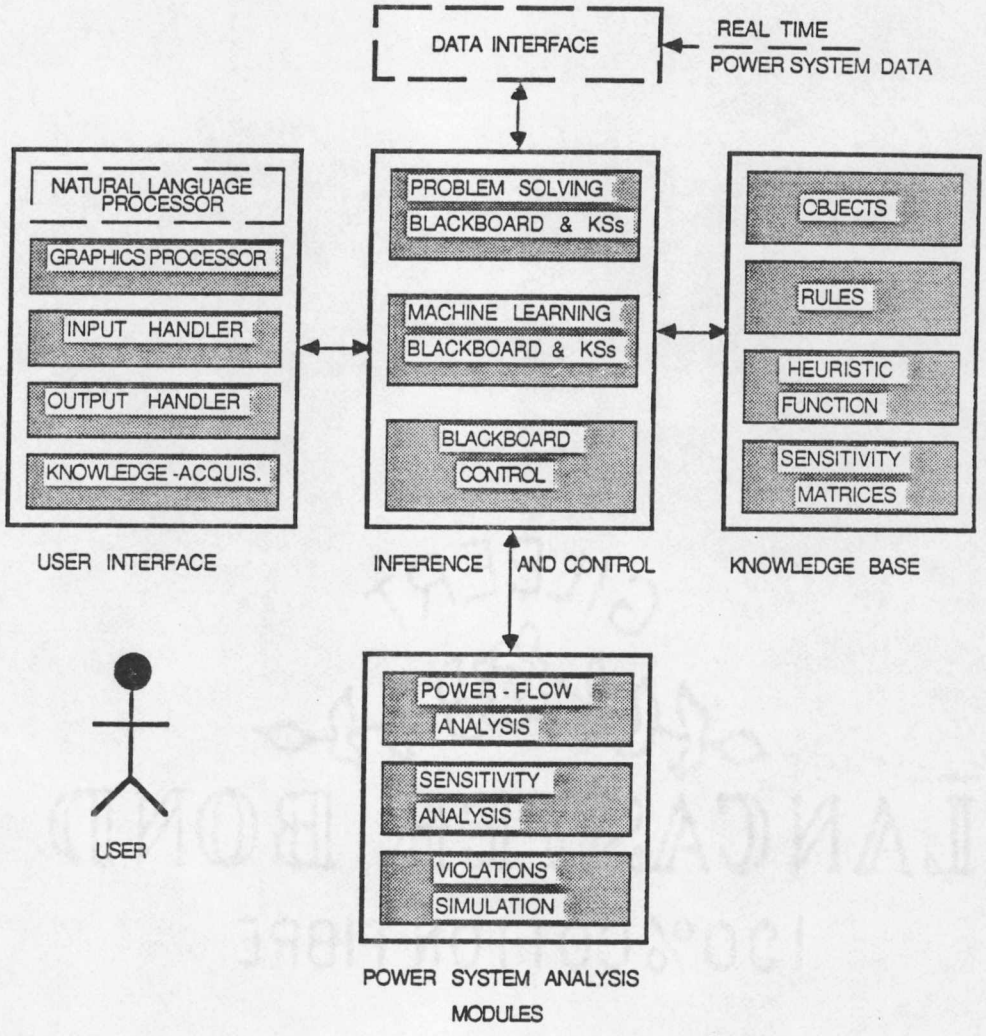


FIGURE. 2. ARCHITECTURE OF - AN EXPERT SYSTEM FOR POWER SYSTEM CONTROL. (EPSY)

Mouse is used to select mouse sensitive items or to pop up the menus to select the menu items. All the system functions to handle the input from a user come under the input handler.

3.1.2. Graphics Interface

In EPSY the graphics interface is used for the following functions:

(1) Display the following mouse sensitive screen windows:
   (i) Power system drawing or display window.
   (ii) Dialogue or Message window.
   (iii) Disturbance Selection window.

(2) Draw a new power system or retrieve an old power system from a specified file, on the system display window.

(3) Allow modification or deletion of any component or its attributes.

(4) Draw or erase screen grid. The grid helps a user to draw a power system with more accuracy and ease on the screen.

(5) Facilitate input or output through window graphics.

3.1.3. Natural Language Interface

This component, although not implemented presently on EPSY is expected to allow the use of high level language in a non-rigid format to interact with the power system. This will make EPSY more user friendly.

3.1.4. Output Handler

This component takes care of any information or message that the system needs to display on the screen. If implemented with the natural language interface, it may call the natural language interface for translation of a particular information into a language that the user understands before sending it to graphics processor.

3.2. The Knowledge Base

Domain Knowledge in EPSY is represented using the following knowledge representation schemes:

3.2.1 Object Oriented Programming.

3.2.2 Production Rules.

3.2.3 Sensitivity Information Matrix.

3.2.1. The Object Oriented Programming

A number of knowledge representation techniques such as objected oriented programming (based on flavors), frames, semantic nets etc. have been considered before selecting the scheme for representing the domain knowledge on a power system. In EPSY domain knowledge has been represented in form of objects using object oriented programming due to the following reasons:

(1) Object oriented programming uses a message passing mechanism, hence it results in easy and fast access to any slot attribute value of an object.

(2) Attributes can be added, modified, redefined or deleted externally by simply adding codes, without modifying the old codes. This is very important for an expert system which is in the process of evolution and is likely to be revised a

number of times.

(3) Default values to certain slot attribute value can be assigned externally.

(4) For a slot attribute, whose value depends on other slot values of the same object, methods to compute the attribute value can be defined and attached externally with those slot attributes using ``defmethod'' facility. Whenever this slot attribute is referenced, its value is obtained by automatic execution of the defined methods using values of other slots. It should be noted that the latest values of relevant slot attributes are used during execution of defmethods.

(5) Programs that use modular programming constructs, as with object oriented programming, facilitate writing programs that are easy to read and understand. Such programs are more reliable and maintainable.

On Symbolics 3670 Lisp Machine, the object oriented programming is based on the flavor system. The object types are defined using flavors. Some of the flavors are built on top of other flavors and each flavor has certain characteristics . Thus each component is considered as a particular instance of a flavor corresponding to the particular component type, since component types are defined as flavors using ``defflavor facility'' with certain attribute slots. Please refer Symbolics Lisp Machine manuals for further information.

3.2.2. The Production Rules

A production rule provides a clear and tansparent form of representing a condition and its corresponding action. The action part executes if the condition part is found true.

Each rule has a left hand side (LHS) the condition part, termed ``situation''

or ``antecedent'' and usually indicated by the keyword ``IF'' in an English description, which describes the data configuration for which the rule is applicable. This side of the rule is usually a Boolean combination of clauses. The types of Boolean operators vary with the production system language. Many languages allow only the operators ``and'' and ``not.''

A rule also has a right hand side (RHS), the action part, termed ``consequent'' and usually indicated by the keyword ``THEN'' in an English description which gives instructions for changing the data configuration. This side of the rule is generally a list of modifications to be made into data memory when the rule fires. These actions usually add, modify or delete elements in data memory, but they may also perform such external operations as writing a message to, or reading data from an input or output device.

The set of rules is stored in the system memory and that part of memory is called the production memory. In case of a large or complex power system like the one considered by EPSY, heuristic knowledge is required in decision making, in order to bring a power system from an alert state or an emergency state back to a normal state. These heuristics fall under the groups described next.

3.2.2.1. Power System State Assessment Heuristics

These heuristics help in deciding as to what regions of the state of a power system should be considered alert state regions , emergency state regions, normal state regions or extreme state region. This depends on the particular type, design, location, operating conditions and environment of the power system. Even the past history of the power system may decide the criterion for defining its present violations. Hence, heuristic knowledge is needed to define these regions of the state

of a power system and to evaluate the system. Also, it may define the regions within which the parameters of the power system components should lie, in order to be considered in normal state.

These heuristics in form of rules are used by the knowledge sources explained later in this chapter.

3.2.2.2. Top Priority Violation Defining Heuristics

These heuristics help in deciding which violation needs more attention over others and should be corrected first before all other violations. These heuristics may also depend on the nature or the design of power system and other theoretical and practical considerations. For example, violations like generator overloadings may be given a higher priority over the bus voltage violations.

3.2.2.3. Control Generator Selection Heuristics

These heuristics help in deciding what all generators can be possibly used and how much their generation may be changed in order to bring the power system in normal state. These heuristics are supported by the sensitivity information obtained as a result of the power system analysis programs (described in Chapter V).

3.2.2.4. State and System Evaluation Heuristics

These heuristics help in deciding whether the current state of the power system has improved since the latest control measures were executed. They also help in evaluating the overall state of the power system and to find if there is any potential for further improvement in the overall state of the power system. Apart from several

heuristic rules, a ``Heuristic Function'' (described in Chapter IV.) is needed to evaluate the overall state of the power system. It takes into account the precontrol state, postcontrol state and all the related information in these two states of the power system.

3.2.2.5. Other Heuristics

Heuristics are also needed to decide how much of the violation is to be corrected, and to make other decisions while the inference and control strategy plans the next set of control measures.

Most of the heuristic knowledge in EPSY is stored in form of production rules under the following groups (Refer APPENDIX A. for example rules):

(1) **Component Violation Rules:** These rules store the heuristic knowledge that helps in deciding what parameter values can be considered to be in a normal state, an alert state or an abnormal state regions. These rules vary with the design considerations and theoretical and practical constraints related to that particular power system.

(2) **System Evaluation Rules:** They store the heuristic knowledge that allows decision making about the overall state of the power system, given the nature of contingencies and power system components involved.

(3) **Priority Rules:** These rules contain the heuristic knowledge that allows in deciding that among all the violations of the system, which one should be considered to be the most severe. This most severe violation is called the top priority violation and is attempted for correction before all other violations.

(4) **Planning Rules:** These rules decide as to how much of the violation should be

corrected in a single attempt. Also, they decide which generator to use in order to bring the power system violation under control. The system then decides whether to change the strategy or change the extent of correction applied on the power system violation being corrected presently. It also checks what all control measures are available and plans the best possible strategy under the given conditions and situation. Thus these rules plan the specific control details, taking into account the top priority violation, the available control measures, and the effect of previous set of control measures.

(5) **State Computation and Evaluation Rules:** These rules execute the strategy planned, and recommend as to what action needs to be taken next. The recommendation is passed in form of a message to the blackboard. The recommendation is the suggested action that the module itself is unable to handle but wants the flow control rules to decide which module to trigger next to carry out the recommended action.

(6) **System Control Rules:** These rules are used as top level overall system control rules that trigger the modules that are capable of doing the recommended action. Whenever a module completes its part of the action it passes a message to the blackboard control in form of a recommendation as to the action that it itself cannot handle but suggests to be carried out . The control rules then decide the module to pass control to for the specific action, based on the state of the power system, and the available module or modules that can carry out the particular recommended action. After completing its part of action, the module returns control to the system control rules for decision making on the next action recommended.

The reason for using rules is that they are more transparent to the user, allow maintainability, ease of modifications and additions in future.

3.2.3. The Sensitivity Information Matrix

The numerical knowledge in EPSY is stored in the form of a two dimensional array or matrix. Two such matrices are used. These are called ``sensitivity matrices'' because each of them give sensitivity information on the parameters of a bus with respect to the parameters of another bus of the power system.

The sensitivity information is obtained as a result of the Power Flow analysis of a power system followed by its sensitivity analysis. The sensitivity information holds good as long as the power system configuration has not been changed. Whenever the configuration of the power system changes, such as any transmission line is added or disconnected from the power system, power flow analysis followed by sensitivity analysis needs to be performed and new sensitivity matrices obtained. One of the sensitivity matrices called JH inverse contains information on the relationship between the change of real power on one bus to the change of voltage-angle on all other buses of the power system. The other sensitivity matrix called JL inverse contains information on the relationship between the change of reactive power on one bus to the change of voltage on all other buses of the power system.

The above mentioned sensitivity matrices form the basis of any numerical knowledge that goes into decision making as to what all generators should be used to control various parameters of the power system, and to compute the new state of the power system after any such control measure is applied.The reason for using a two dimensional array to store this numerical information is that an element belonging to any particular row and column can be accessed directly, and also it

is fairly easy to scan a particular row or column of the matrix. These features are extensively needed and used when numerical information is manipulated.

CHAPTER 4

THE BLACKBOARD BASED INFERENCE AND CONTROL STRATEGY

4.1. The Blackboard Architecture in EPSY

In EPSY, the inference and control strategy for the problem solving system and the machine learning subsystem uses the blackboard approach due to the following reasons:

(1) ``Blackboard approach is useful for complex, ill structured problems.''[1] The problem of controlling a power system is rather ill structured and complex for large power systems.

(2) In case of EPSY, there appeared to be a need for an evolutionary solution. Blackboard approach is considered good for an evolutionary solution to a problem.

(3) For incremental and exploratory system development, which means that at each development phase the expert system might have to be redesigned and rebuilt, blackboard approach has been found very favourable.

(4) A Blackboard architecture will allow for a multi expert system architecture in the future, when more problem solvers might be interfaced with the blackboard.

(5) Blackboard approach allows the postponement of design decisions. Also, problems having large solution spaces and diverse knowledge sources are considered the best candidates for blackboard approach. In EPSY, if in future more problem solvers are added to diversify its capabilities, the blackboard approach is

[1] H. Penny Nii, ``Blackboard Systems,'' THE AI Magazine August, 1986.

expected to suit very well, requiring little or no modifications, but additions.

(6) ``The blackboard approach provides freedom from message passing constraints. The message passing paradigm requires a recipient of a message as well as a sender. Often the recipient is not known, or the recipient may have been deleted. In the blackboard approach the message sending module places the information on the blackboard, and the developer of the module is freed from worrying about other modules.''[2]

(7) The utility of blackboard model in the area of machine learning has been explored by us with very interesting and good results.

EPSY consists of two blackboard systems. One system, which solves a problem, consists of the standard blackboard configuration called problem solving blackboard. The second system learns from the behavior of the first system and consists of a blackboard containing observations and inferences. This blackboard is called a machine learning blackboard (machine learning blackboard is explained in Chapter 6). Its knowledge sources concentrate on learning. The blackboard control is common for the problem solving and the machine learning blackboards. Please refer Figure.3. for the blackboard architecture in EPSY.

The problem solving blackboard architecture essentially consists of the following three major components or structures:

4.1.1. The problem solving Blackboard Structure.

4.1.2. The Problem solving Knowledge Sources.

4.1.3. The Blackboard Control.

---

[2] H. Penny Nii, ``Blackboard Systems, ``THE AI Magazine August, 1986.

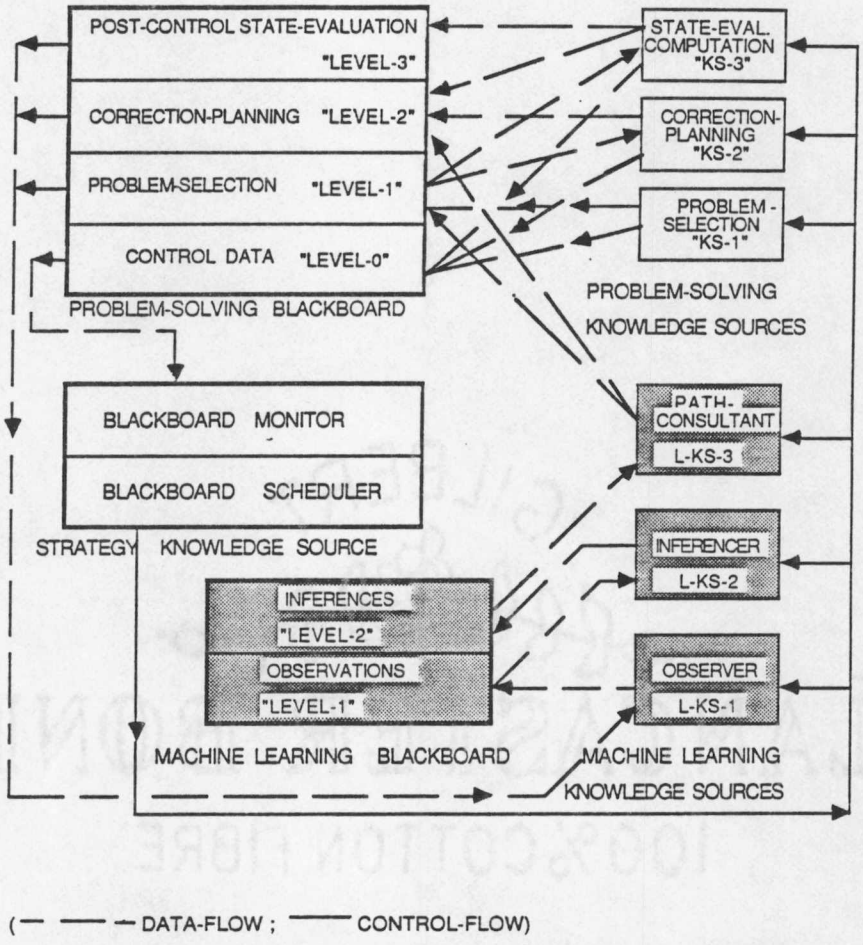


FIGURE. 3. INFERENCE AND CONTROL STRATEGY : OVERALL BLACKBOARD ORGANIZATION FOR PROBLEM-SOLVING AND MACHINE-LEARNING

4.1.1. The Problem Solving Blackboard Structure

In EPSY the blackboard is partitioned into three levels of analysis corresponding to the intermediate levels of the decoding process. These levels form a hierarchy in which the solution space elements on each level is described loosely as forming an abstraction of information on its adjacent lower level. The three levels of hierarchy is comprised of, from the lowest to the highest level: Problem Selection, Correction Planning, postcontrol State Evaluation.

4.1.1.1. The Problem Selection Level

This level consists of the blackboard elements giving information on the violations in the power system and the most severe violation among those violations. A frame called complaint blackboard has been defined to form a node with level, violations, most severe violation and system status as its slots.

4.1.1.2. The Correction Planning Level

This level consists of the blackboard elements giving information on the step size and the control generator chosen for correcting the most severe violation. A frame called planner blackboard has been defined to form a node with level, step size and control gen now as its slots.

4.1.1.3. The Postcontrol State Evaluation Level

This level consists of the blackboard elements that give information on the results of evaluating the postcontrol state of the power system. A frame called

evaluation blackboard has been defined to form a node with level, new violations and new system status as its slots.

In addition to the blackboard, EPSY has other globally accessible information generated directly or indirectly by the knowledge sources. This global information is used primarily by the strategy knowledge source. The control data or control variables are described below:

(1) **Recommendation:** This contains the latest recommendation by a knowledge source or module that should be carried out, which the module or the knowledge source itself was unable to do but wants the blackboard control to find and trigger the knowledge source that could possibly carry out the recommendation.

(2) **Control Level:** This informs the depth of control that is being attempted by the system. If an attempt for correction fails, then the system goes a depth more and makes a corrective step smaller than before. The maximum depth can be defined by user.

(3) **Attempted Control:** This variable informs if the attempt to correct the most severe violation has begun or not.

(4) **Attempted Components List:** This contains a list of all components that have been attempted for correction.

(5) **System Status:** This variable contains the present status of the overall power system.

(6) **Available Control Generators List:** This contains a list of all control generators that are available for further loading in order to correct the power system violations.

4.1.2. The Problem Solving Knowledge Sources

``A knowledge source (KS) is a mega chunk of knowledge consisting of a labeled set of rules that are a priori to belong together. How the rules within a KS is organized and what rules are included in a KS depend on the intended role of the KS in the overall problem solving plan. For example, a KS may be organized to represent models, in which case all rules are grouped around some objects or concepts, much like schemata or frames. Or, a KS may be organized around events in which case all rules which, for example, process input data, are grouped together.

In order to solve problems within the framework of the Blackboard model, KSs must place their inferences on the Blackboard. In other words, KSs must generate hypotheses using rules that:

(1) add or modify hypothesis elements(s), or

(2) add or modify relationships between elements.''[3]

Each knowledge source has two major components a condition part (often referred to as a precondition) and an action part. The condition component prescribes the situations in which the knowledge sources may contribute to the problem solving activity, and the action component specifies what the contribution is and how to integrate it into the current situation. When executed, the condition part searches the blackboard for hypotheses that are of interest to its corresponding action part; all the relevant hypotheses found during the search are passed on to the action part. Upon activation, the action part processes all the hypotheses passed to it. The task of knowledge sources range from defining violations, to planning strategy to computing new state and evaluating it.

[3] AGE by H.Penny Nii and Nelleke Aiello, IEE Proceedings '82, p.647.

In EPSY, apart from a strategy knowledge source, three knowledge sources are used and are discussed next.

4.1.2.1. Problem-Selection KS

This module does the following functions:

(1) **Find violations:** Finds out all the power system violations that have already occured or are expected to occur, and writes them on the blackboard. It uses Componenet Violation rules during this function.

(2) **Evaluate power system:** Evaluates the overall state of the power system to find out if the system is in alert, emergency, or normal state, and writes it on the black board. It uses system evaluation rules during this function.

(3) **Decide top priority violation:** Decides the power system violation that must be attempted for correction first, among all the other power system violations, and writes it on the blackboard. It uses violation priority rules during this function.

(4) **Recommendation:** Recommends the next action that the system should take.

4.1.2.2. Correction-Planning KS

This module is responsible for the following functions:

(1) **Decide step size:** Step size gives the extent of correction that is going to applied to the power system violation currently being dealt. It writes this information on the blackboard. This depends upon the control level and the effect of previous set of control measures. It uses a heuristic function to decide the step size.

(2) **Select control generator:** It selects the most appropriate generator out of

the available generators, to bring the currently dealt power system violation under control. It writes this information on the blackboard. The selection of a particular generator depends on the sensitivity information in relation to the bus or component whose violation is being attempted for correction.

(3) **Recommendation:** It recommends the action that it thinks should be taken next, because it itself is unable to handle that particular action. This is written by this module on the blackboard and it is left at the descretion of the strategy knowledge source to decide as to which module it should call upon that is capable of handling the recommended action.

4.1.2.3. State-Computation-Evaluation KS

This module is responsible for the following functions:

(1) **Execution of the strategy:** The strategy chalked out by the Problem-Selection KS and Correction-Planning KS is executed by this module. It tries to bring under control the power system violation selected for control by the Problem-Selection KS, using the step size and control generator decided by the Correction-Planning KS.

(2) **Computation of the New State of the Power System:** It performs numerical computation on parameters of various Power System Components using the sensitivity matrices information and depending upon the strategy details in the previous function explained above.

(3) **Postcontrol State Evaluation and Recommendation:** After new state is obtained, this knowledge source evaluates the state of the power system and compares it with the previous state to decide if any improvement has

resulted after the latest set of control measures. Based on this evaluation and comparison with the previous state, it recommends the next move. The recommendation is based upon the use of a heuristic function. The heuristic function makes use of the heuristic knowledge of the power system expert. This heuristic knowledge defines the criterion of measuring the performance of the latest set of control measures executed to correct the top violation of the power system. Since this criterion of evaluation depends on the type of the power system, past history, theoretical and practical constraints, the heuristic function may vary from one power system to another. The heuristic function may measure the performance of the latest set of control measures by taking into account the result of the following:

(i) Comparison of the total number of alert and abnormal state violations in the precontrol state of the power system, with those in the postcontrol state. This gives the overall improvement in the state of the power system.

(ii) Comparison of the top violation in the precontrol state of the power system with that in the postcontrol state. This gives the improvement in the top violation which was being attempted for correction.

4.1.3. The Blackboard Control

In EPSY the blackboard control module is written in the same form as the domain knowledge sources, that is, as a set of rules. The control knowledge source decides which categories of events to process next, based on the information on the blackboard control frame consisting of recommendation and related information.

The strategy knowledge source, which forms the control component of the blackboard, consists of a blackboard monitor and a blackboard scheduler. The

blackboard monitor keeps an account of each change made to the blackboard, its primitive change type, and any new hypotheses. The blackboard scheduler schedules the triggering of various knowledge sources depending on the action recommended by the most recently executed knowledge source. It uses the flow control rules to decide which module is capable of doing the recommended action, and that module is triggered accordingly.

The knowledge source or module that is triggered does the action or function it is capable of, and then writes the recommendation on the blackboard control frame and returns the control to the blackboard control knowledge source. The blackboard scheduler again goes through the flow control rules and the knowledge source or module whose precondition matches with the recommended action is triggered. As a result the knowledge source or module does the recommended action and then passes the control back to the strategy knowledge source. In this way various modules or knowledge sources are triggered, and the expert system moves in the direction of the goal state.

4.2. The Knowledge Application Strategy

This expert system has two sets of knowledge sources - one set is part of the problem solving system and called problem solving knowledge sources (PSKSs), while the other set is part of the machine learning subsystem and called machine learning knowledge sources (MLKSs). The PSKSs aim on the problem solving by using a strategy of forward chaining and backtracking to bring the power system in the best possible state. The MLKSs concentrate on the learning by utilizing the observations made during the problem solving process to make certain inferences that would help the PSKSs to make better decisions.

Once the expert system is called, power flow analysis and sensitivity analysis is performed. The information resulting this way is stored in the object knowledge base and sensitivity matrices respectively. From this point onwards, the PSKSs get triggered one by one depending on whose precondition is met. The MLKSs are triggered after the execution of each of the PSKSs. Some or all of the machine learning knowledge sources may trigger one by one, depending on which PSKS was triggered prior to them.

The whole process of problem solving with machine learning can be summarised into the following steps:

(1) The PSKS called Problem-Selection KS is triggered. This knowledge source finds out all the violations in the power system. Also, it finds out the violation that must be attended first for correction. All this information is written on the ``Problem Selection'' level of the problem solving blackboard (PSBB). The next action recommended by this knowledge source is written on the ``Control Data Level'' of the problem solving blackboard. The control is then returned to the strategy knowledge source.

Next, the machine learning subsystem is called. The MLKS ``Path Consultant'' is triggered. It uses any inferences available on the ``Abstraction'' level of the machine learning blackboard (MLBB) to make any correction in the decision of the Problem-Selection KS. Any changes so made are written on the ``Problem Selection'' level. Next, the MLKS ``Observer'' is triggered. It makes the observations resulting after the knowledge sources ``Problem-Selection KS and Path-Consultant KS are executed. The observations are written on the ``Observation'' level of the MLBB. The control is then returned to the strategy knowledge source.

(2) The PSKS called Correction-Planning KS is triggered. It decides the percent-

age of the highest priority violation that should be attempted for correction. Also, it decides the generator that should be used to affect the correction of the violation. All these decisions are written on the ``Correction Planning'' level of the PSBB. The action recommended next by the Problem-Selction KS is written on the ``Control Data'' level of the PSBB. The control is then returned to the strategy knowledge source.

Next, the machine learning subsystem is called. The two MLKSs mentioned in the step (1) above are triggered in the same order, except that this time in the context of the Correction-Planning KS. After the execution of the two MLKSs the control is returned to the strategy knowledge source.

(3) The PSKS called State-Computation-Evaluation KS is triggered. It computes the new state of the power system. Also, it evaluates the postcontrol state of the power system using a heuristic function, explained later in this chapter. The heuristic function also concludes if any further improvement of a particular violation or overall state of power system is possible or not, with the available generators of the power system. Any new violations and such other information found during the heuristic evaluation is written on the Evaluation level of the PSBB. On evaluation, the heuristic function recommends the control knowledge source if the system should backtrack to the previous state, or remain in the new state. Backtracking is required only if the postcontrol state of the power system is worser than the precontrol state. The next action recommended by the knowledge source is written on the ``Control Data'' level of the PSBB. If the power system is found in the normal state or the best possible state, the heuristic function recommends the strategy knowledge source to return control from the expert module. Otherwise, the MLKSs Observer and Inferencer are executed in the order mentioned. The MLKS Inferencer makes certain relevant

inferences and writes them on the ``Abstraction'' level of the MLBB. The observations made by the Observer KS are written on the ``Observations'' level of the MLBB. The reason MLKS Inferencer is called in the context of the PSKS State-Computation-Evaluation KS is that, observations on a complete round of attempt are needed to make an accurate and sensible inferences. The heuristic evaluation is performed only at the end of each round of attempt for correcting the top-priority violation. Hence, once the heuristic function is called, the MLKS Observer makes observations and the complete round of latest observations is used by the Inferencer to make inferences. After this, the control is returned to the strategy knowledge source.

(4) The system goes to step(1) if the violation presently being attempted has been corrected to the best possible extent, but other violations remain to be corrceted. The system goes to step(2) if the violation presently being attempted has not been corrected to the best possible extent, although the overall state of the power system may be better than before.

In EPSY, the machine learning subsystem learns the following things from the failures of the problem solving system:

(1) The range of correction that should be attempted next time.

(2) Whether the control generator being used presently is appropriate at all for correcting the violation being attempted presently i.e. whether the next most sensitive control generator should be chosen next time.

(3) What is the possibility that the violation being attempted can be corrected at all under the given constraints of the power system particularly the control generators.

To allow for backtracking, the object knowledge base frames have certain extra

slots that contain the previous state values of the power system. During backtracking the information contained in the previous state slots is copied on to the current state slots.

4.3. Heuristic Evaluation Function

In case of EPSY, the ``Heuristic Evaluation Function'' (HEF) plays a very important role in the expert decision making, and can greatly influence the efficiency and effectiveness of this expert system. The HEF sets the criterion for evaluating the overall state of the power system in the light of various violations of different nature and severity in the postcontrol state of the power system and the similar information in the precontrol state of the power system. Since this criterion depends on the type of power system, the past history of the power system and other theoretical and practical constraints that vary from one power system to another, hence the HEF would also vary from one power system to another. For this reason it should be defined taking into account the specific power system under consideration.

For example, the heuristic evaluation function may evaluate and recommend based on the following criterion:

(1) Effect of the latest control measures on the top violation of the power system that was being corrected. This could be categorized as good, bad or same, as the case may be. This defines improvement in a specific violation.

(2) Effect of the latest set of control measures on the overall state of the power system. This is obtained by comparing the number of violations in the alert and abnormal condition in the precontrol state of the power system with those in the postcontrol state of the power system. The result could again be categorized as good, bad or same. This defines the potential of violations in

future and the overall improvement in the state of the power system. Based on the combination of the above two results, different recommendations could be given.

For example, if the total number of violations increase after applying a particular set of control measures then the particular set of control measures be discarded and the power system state should be pushed back to the previous state. From there a different strategy may be used. Or, it may have heuristics suggesting that if the total number of ``abnormal state'' violations have decreased, then the overall state of the power system should be considered to be better than the previous state, even if the number of ``alert state'' violations have increased.

CHAPTER 5

THE POWER SYSTEM ANALYSIS MODULES IN EPSY

A power system has a number of components like generators, buses, transmission lines etc. Loads, in form of various electrical devices consuming the electrical power from a power system are connected directly or indirectly to certain buses of the power-system called load buses. Other buses to which the generators are connected are called the voltage buses.

Load buses are to be maintained at a certain voltage which should always lie within a certain predefined voltage region, in order to be considered normal. Hence, whenever the voltage of a load bus tends to go outside this region, this bus voltage needs to be brought back within the normal region. This is done by changing the voltage of that voltage bus which is most likely to have the most desirable affect on the load bus whose voltage is being corrected. The decision to chose a particular voltage bus is made, based on the sensitivity matrix information. The voltage bus thus chosen should also be the one causing no new violations to the power system. This kind of correction can be possible only if fairly accurate information is available on the relationships such as between the change of voltage of a bus on the change of voltage on another bus. The parameters of all the components of the power system need to be known before attempting to correct any of its violations or before performing the sensitivity analysis (explained later in this chapter).

Two power system analysis programs are used to obtain the complete information on all the parameters of the power system and the relationships between any

two power system buses. One of these programs finds out all the other parameters of the power system components based on the given parameters of the power system components. This program is called the power flow analysis program. The other one is called the sensitivity analysis program, and it finds the relationship that informs - how the change in parameters of a bus would affect the parameters of another bus. One more program called the violation simulation program is used to simulate the violations of power system components. Given the type of the power system violation a user wants to simulate, it computes the new parameters of the power system components which would reflect their values under the simulated violations.

The expert system uses three power system analysis modules:

5.1. Power Flow Analysis.

5.2. Violation Simulation.

5.3. Sensitivity Analysis.

5.1. Power Flow Analysis

The initial parameters such as voltage and angle of all the buses of the power system, and other relevant initial parameters such as real and reactive power of all other components of the power system can be obtained by performing power flow analysis. The power flow analysis is based on the given voltage and angle of various voltage buses, and certain other relevant parameter information on the other components of the power system.

The steady state performance of a power system can be expressed with a set of non-linear, coupled algebraic equations as follows:

$$P_i = P_g i - P_d i = Real(V_i \sum_j Y_{ij} \quad V_j)$$
$$= |V_i| \sum_j |Y_{ij}||V_j| cos(\delta_j - \delta_i + \gamma_{ij}) \qquad (1.1)$$

$$Q_i = Q_{gi} - Q_{di} = -Im(V_i \sum_j Y_{ij} \quad V_j)$$
$$= -|V_i| \sum_j |Y_{ij}||V_j| sin(\delta_j - \delta_i + \gamma_{ij}) \qquad (1.2)$$

$$i, j = 1, 2.....n$$

where

P_i, Q_i : net real and reactive power injections into bus i.

P_{gi}, Q_{gi} : net real and reactive power generations at bus i.

P_{di}, Q_{di} : net real and reactive power demands at bus i.

$|V_i| \angle \delta_i$: complex voltage on bus i.

$|Y_{ij}| \angle \gamma_{ij}$: i, j element in bus admittance matrix.

From eqs. (1.1) and (1.2) bus mismatch equations are formulated as:

$$F_{pi} = P_{gi} - P_{di} - |V_i| \sum_j |Y_{ij}||V_j| cos(\delta_j - \delta_i + \gamma_{ij}) = 0 \qquad (1.3)$$

$$F_{qi} = Q_{gi} - Q_{di} + |V_i| \sum_j |Y_{ij}||V_j| sin(\delta_j - \delta_i + \gamma_{ij}) = 0 \qquad (1.4)$$

where

F_{pi}, F_{qi} : real and reactive power mismatches at bus i.

Using vector notation at the solution point,

$$F(X,U,D) = \begin{pmatrix} F_p \\ F_q \end{pmatrix} = 0 \tag{1.5}$$

where

$F(X,U,D)$	: $(2n-2)$ x 1 full mismatch vector.
F_p	: real power mismatch vector
F_q	: reactive power mismatch vector.
X	: dependent state vector
U	: independent control vector.
D	: diturbance vector.
n	: number of total buses.

Several methods such as Fast Decoupled (FD) method can be used to solve eq. (1.5).

5.2. Violation Simulation

This module takes the new value of any component parameters and solves power flow analysis reflecting the power system parameter changes corresponding to the simulated violation.

In case of violations due to tripping of a transmission line, the sensitivity analysis is also needed to be done again due to the change in the system configuration caused by tripping the transmission lines.

Following types of violations are considered:

(1) Generation drop.

(2) Generation change.

(3) T/L trip.

(4) Load drop.

(5) Load change.

5.3. Sensitivity Analysis

The information on the relationships such as the change in the voltage or angle of one bus on the voltage or angle of another bus can be obtained by performing sensitivity analysis on the given components' parameters of the power system. The sensitivity information on a particular configuration of a power system holds good as long as the power system configuration is not changed, such as, by disconnecting or adding transmission lines to the power system.

Sensitivity analysis in case of a power system is defined as method that gives information as to how much the parameters V and δ of a bus, are sensitive to the change in parameters here P and Q of another bus, where the symbols V, δ, P, Q represent the bus voltage, angle, real power, and reactive power respectively. This information is important and plays a critical role in deciding as to what all generators should be chosen in order to correct the voltage of the violated buses.

Among the various sensitivity analysis methods, the methods which utilize the jacobian matrix have the advantage of utilizing the Jacobian and its inverse which are directly computed in solving the previous power flow. Researchers have proposed approximate sensitivity analysis techniques which utilize the inverse of the full size Jacobian, and some of them even require couple of iterations. These methods can give more accurate results than the approximate FD method, but they require the information about the inverse of the full Jacobian.

5.3.1 FD Algorithm for Finding the Sensitivity Matrices

Power Flow mismatch at the solution point due to the state change:

$$\begin{pmatrix} F_p \\ F_q \end{pmatrix} = \begin{pmatrix} \Delta \mathrm{P} \\ \Delta \mathrm{Q} \end{pmatrix} = \begin{pmatrix} (\partial F_p/\partial X_1)(\partial F_p/\partial X_2) \\ (\partial F_q/\partial X_1)(\partial F_q/\partial X_2) \end{pmatrix} \begin{pmatrix} \Delta X_1 \\ \Delta X_2 \end{pmatrix} \approx \begin{pmatrix} H & K \\ J & L \end{pmatrix} \begin{pmatrix} \Delta X_1 \\ \Delta X_2 \end{pmatrix}$$

where

F_p : real-power-mismatch.

F_q : reactive power mismatch.

$$X_1 = \begin{pmatrix} \delta_2 \\ \delta_3 \\ \delta_4 \\ . \\ . \\ \delta_{\mathrm{n}} \end{pmatrix}$$

$$X_2 = \begin{pmatrix} Q_{g2} \\ Q_{g3} \\ . \\ . \\ Q_{gg} \\ V_{g+1} \\ . \\ . \\ V_n \end{pmatrix}$$

where

n : total number of buses

g : total number of generators

$$\Delta P = H\Delta X_1 + K\Delta X_2 \tag{1.6}$$

$$\Delta Q = J\Delta X_1 + L\Delta X_2 \tag{1.7}$$

where

$$H = \begin{pmatrix} (\partial P_2/\partial \delta_2)(\partial P_2/\partial \delta_3).....(\partial P_2/\partial \delta_n) \\ (\partial P_3/\partial \delta_2)(\partial P_3/\partial \delta_3).....(\partial P_3/\partial \delta_n) \\ \cdot \\ \cdot \\ \cdot \\ (\partial P_n/\partial \delta_2)(\partial P_n/\partial \delta_3).....(\partial P_n/\partial \delta_n) \end{pmatrix}$$

$$L = \begin{pmatrix} (\partial Q_2/\partial V_2)(\partial Q_2/\partial V_3).....(\partial Q_2/\partial V_4) \\ (\partial Q_3/\partial V_2)(\partial Q_3/\partial V_3).....(\partial Q_3/\partial V_4) \\ \cdot \\ \cdot \\ \cdot \\ (\partial Q_n/\partial V_2)(\partial Q_n/\partial V_3).....(\partial Q_n/\partial V_4) \end{pmatrix}$$

In most cases, matrices K and J are close to zeros and can be neglected. Neglecting K and J, from eqs. (1.6) and (1.7) we have:

$$\Delta P \approx H \Delta X_1 \tag{1.8}$$

$$\Delta Q \approx J \Delta X_2 \tag{1.9}$$

Therefore,

$$\Delta X_1 \approx H^{-1} \Delta P \tag{1.10}$$

$$\Delta X_2 \approx L^{-1} \Delta Q \tag{1.11}$$

Matrices H^{-1} and J^{-1} are called sensitivity matrices, given as:

$$H^{-1} = \begin{pmatrix} (\partial\delta_2/\partial P_2)(\partial\delta_2/\partial P_3).....(\partial\delta_2/\partial P_n) \\ (\partial\delta_3/\partial P_2)(\partial\delta_3/\partial P_3).....(\partial\delta_3/\partial P_n) \\ \cdot \\ \cdot \\ \cdot \\ (\partial\delta_n/\partial P_2)(\partial\delta_n/\partial P_3).....(\partial\delta_n/\partial P_n) \end{pmatrix}$$

$$L^{-1} = \begin{pmatrix} (\partial V_2/\partial Q_2)(\partial V_2/\partial Q_3).....(\partial V_2/\partial Q_n) \\ (\partial V_3/\partial Q_2)(\partial V_3/\partial Q_3).....(\partial V_3/\partial Q_n) \\ \cdot \\ \cdot \\ \cdot \\ (\partial V_n/\partial Q_2)(\partial V_n/\partial Q_3).....(\partial V_n/\partial Q_n) \end{pmatrix}$$

From eq.(1.10) we obtain the relationship between $\Delta\delta_i$ and ΔP_k. The sensitivity ratio $\partial\delta i/\partial P_k$ is obtained from the sensitivity-matrix H^{-1}, i.e.,

$$\delta_i = (\Delta\delta_i/\Delta P_k)\Delta P_k \tag{1.12}$$

From eq.(1.11) we obtain the relationship between ΔV_i and ΔQ_k. The sensitivity-ratio $\partial V_i/\partial Q_k$ is obtained from the sensitivity-matrix L^{-1}.

$$\Delta V_i = (\partial V_i/\partial Q_k)\Delta Q_k \tag{1.13}$$

Just to demonstrate how the sensitivty-matrices can be used for the computation of a new-state and to support a certain decision, the case of voltage-violation of a load-bus is considered and is mentioned next.

5.3.2. Case of Voltage Violation of a Load Bus

Considering that the voltage of a load-bus i is violated and needs to be changed by a value ΔV_i, then from equation (1.13) we can find out the value of ΔQ_k,

the required increase in reactive power of the voltage-bus k to which the load-bus i is most sensitive to. The most sensitive voltage-bus k is obtained from the sensitivity matrix by taking the i^{th} row of the sensitivity matrix and picking up the column containing the highest absolute value of the sensitivity ratio. In case of bus angle violation sensitivity-matrix H^{-1} is used, whereas for bus voltage violations sensitivity-matrix L^{-1} is used.

Since the sensitivity ratio corresponding to the row i and column k is known, and also the value of desirable change in the voltage of load bus i is known, hence using equation(1.13) we obtain the value of ΔQ_k that should be added or subtracted to increase or decrease the reactiv-power of the generator connected to the voltage-bus k. Now, in order to affect the reactive power of this generator the voltage of the generator connected to voltage-bus k needs to be changed by a value which can be obtained using equation (1.13). The above mentioned computation can be explained as follows:

Let V_i, Q_k, and $(\partial V_i/\partial Q_k)$ represent the desirable change in the voltage of load-bus i, change in the reactive power of voltage bus k, and the highest sensitivity ratio corresponding to the bus k respectively. Then using equation (1.13), we obtain:

$$\Delta Q_k = \Delta V_i/(\partial V_i/\partial Q_k) \tag{1.14}$$

Now, let ΔV_k and $(\partial V_k/\partial Q_k)$ be the required change in the voltage of the bus k to change its reactive-power by ΔQ_k. Then using equation (1.13), we obtain:

$$\Delta V_k = (\partial V_k/\partial Q_k)\Delta Q_k \tag{1.15}$$

Thus to affect the desirable voltage-change in the load-bus i the voltage of the voltage-bus k should be changed by ΔV_k as computed by equation 1.15. Now, the

previous voltages of all the other buses in the power system would change by a value that can be computed using eq. (1.13). For example, for bus-1 we obtain:

$$\Delta V_1 = (\partial V_1 / \partial Q_k) \Delta Q_k$$

Similarly for bus-2 we obtain:

$$\Delta V_2 = (\partial V_2 / \partial Q_k) \Delta Q_k$$

Similarly we can find the voltage changes for all other buses of the power system. The new bus voltage in each case is given by adding in case of higher voltage violation, or subtracting in case of lower voltage violation, the voltage change ΔV_k so obtained to the previous voltage of each bus. Once the new bus voltage for each bus of the power system is known, the power flowing through various transmission lines can be easily computed.

CHAPTER 6

MACHINE LEARNING IN EPSY

6.1. Background of Machine Learning

``Machine learning is any change to a behaving system that alters its long term performance. The changes of major interest are those that improve performance, any other change is either irrelevant or pathological.''[1] In another way, learning can be defined as any change in a system that allows it to perform better the second time on repetition of the same task or on another task drawn from the same population.

Present day computer systems normally do not have a machine learning capability, and so cannot truly learn to perform a task through examples or by analogy to a similar, previously solved task. Nor can they improve their performance significantly on the basis of the past failures, or acquire new abilities by observing and imitating experts. Hence, Machine learning research strives to open the possibility of instructing computers in new ways like learning to perform a task through examples or analogy to a similar, previously-solved task. Machine learning promises to ease the burden of, hand-programming the growing volumes of increasingly complex information into the computers of tomorrow.

[1] Brownston, Kant, Farell, Martin ``Programming Expert Systems in OPS5,'' Chapter 8, Knowledge Acquisition, Learning and Explanation, p333.

6.2. Need for Machine Learning in Expert Systems

From the perspective of current expert system technology, learning systems are needed when all the possible situations that a system will encounter are not known beforehand. By adding a learning component or subsystem to a system, the user can extend the system's capabilities through interaction with it, rather than by the process of reprogramming.

In most of the real time expert systems the efficiency and speed of the expert system is a major factor in judging its performance. Therefore it becomes indespensable to have a schema that would, like an intelligent onlooker record observations during the problem solving process and make inferences based on that. These inferences can then be used to improve the decisions made by the problem solving system. This would improve the overall performance of the expert system.

6.3. Machine Learning Subsystem in EPSY

In EPSY machine learning is implemented using a blackboard model. The learning subsystem uses observations and past experience. Please refer Figure.4. and Figure.5. The learning is incremental, acquiring a piece of knowledge at a time, and continues to grow by making use of the past inferences and present observations.

The machine learning subsystem of EPSY is made up of the following three components:

6.3.1. The Machine Learning Blackboard.

6.3.2. The Machine Learning Knowledge Sources.

6.3.3. The Machine Learning Blackboard Control.

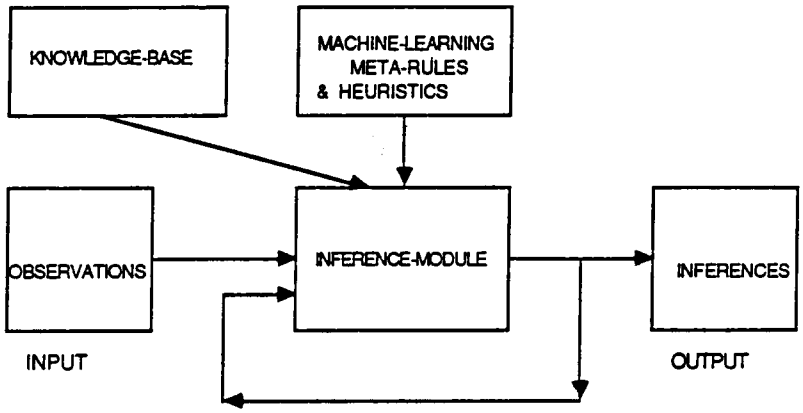


FIGURE.4. BASIC FORM OF ORGANIZATION FOR MACHINE LEARNING BY OBSERVATIONS.

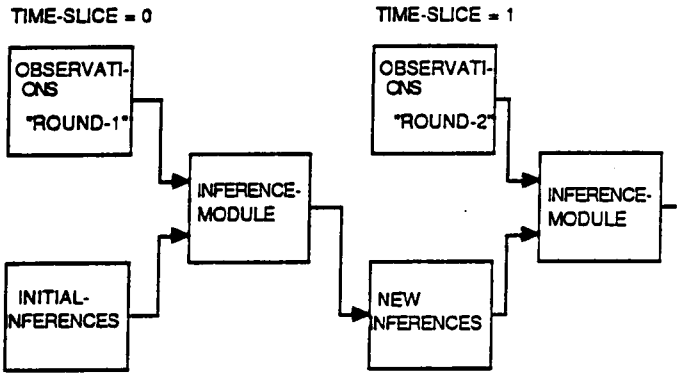


FIGURE.5. ELABORATION ON - MACHINE-LEARNING BY OBSERVATIONS

6.3.1. The Machine Learning Blackboard

This blackboard consists of two levels. The lower level consists of all the hypothesis, information or observations (passive) taken during the process of problem-solving. The upper level consists of the hypothesis, abtraction or inferences based upon the lower level information, learning-heuristics and any previous inferences in the upper level.

6.3.2. The Machine Learning Knowledge Sources

In EPSY following three knowledege sources are used:

6.3.2.1. Observation-Extraction

This part acts like an onlooker trying to observe the search process that the system is going through. It records all the relevant and vital information that could be used to make intelligent inferences. This part itself does not make inferences but just concentrates on gathering all the information that would be used by the Inference-Module, which is another component of the machine learning in EPSY and is described later in this chapter.

Specific attributes are observed and recorded depending upon the current problem solving context. This in turn depends on the knowledge source currently being active. The observations are made at the end of each problem solving knowledge source execution since certain attribute values may have been changed. The attributes and their values are saved in an observation frame having the following slots:

(1) **Time-Slice:** This slot gives the n^{th} round or sequence of the observations

being made. Thus it gives an idea of time or which observation was made after which and so on. This will avoid any confusion when the inferences are being made.

(2) **Context:** This slot gives the information such as which knowledge source or context is active when the observation is being made.

(3) **Object:** This slot contains the name of the component, whose violation is being attempted for correction at the instance the observation is being made.

(4) **Object-Type:** This slot tells the type of the power system component noted in the Object slot above.

(5) **Object-Parameters:** The relevant parameters or attribute names are saved in form of a list against this slot. These parameters or attribute names vary depending on the context.

(6) **Parameter-Values:** The corresponding values of the parameters or attributes mentioned in the Object-Parameters slot are saved in a form of a list in the same order with the parameter or attribute names in the Object-Parameters slot.

6.3.2.2. Inferencer

The primary aim of this module which is a part of the machine learning process is to make inferences along certain lines preconceived by the person or expert designing it. The inferences made are recorded on an abstraction frame explained later in this section. Inferences are made by taking into account the latest set of observations and the abstraction frame. The inference module consists of a set of inferential operator. Each inferential-operator operates on certain attributes or information from the observation frame and abstraction frame to give output in form

of inferences. The abstraction frame is then updated with this inference output.

The abstraction frame defines the basic guidelines for machine learning. These guidelines have been decided taking into account the ways by which an improvement in the system performance can be affected. The abstraction frame has the following slots:

(1) **Time-Slice:** This is used to give an idea of the number of times the abstraction frame has been updated.

(2) **Object:** This slot gives the latest most severe violation that should be attempted for correction.

(3) **Occured-Violations:** The violations that have been attempted for correction or being attempted presently are given by this slot. It also gives a number which is indicative of a certain instance of time, when the violation was first recorded as being the most severe violation.

(4) **Frequent-Violations:** The violations that have occured more than once and are found to be the most severe violation during all those times are recorded in this slot.

(5) **Incorrectible-Violations:** The violations that meet the conditions of occuring frequently, whose correction is not possible without causing more violations and for whom no more effective control-generators are available, are considered as incorrectible. The incorrectible violations are recorded at this slot. Whenever the learning system finds out that the problem solving system is attempting to correct an incorrectible violation, it brings that violation to the notice of the user. That violation is not attempted for correction by the expert system, unless specifically asked to do so by the user.

(6) **Infered-Step-Size:** The extent of correction that should be attempted by the expert system is infered and saved against this slot. The inference is based

on the most severe violation in the postcontrol state that occurs during the attempt. An example of how the system infers the ``Infered step size,'' along with the relevant scenario is given below.

Scenario : Let,

B1 : Load bus whose voltage violation is being corrected.

B12 : New most severe postcontrol voltage violation bus.

B32 : Voltage bus being used to correct violation of bus B1.

The selection of buses B1, B12 and B32 is based on the heuristic rules used by the problem solving knowledge sources.

For load-bus B1: Let,

$$\text{Initial-Voltage } V_1 = 1.06589 \quad pu$$

$$\text{Previous-Control-Step-Size} = -0.04942 \quad pu$$

$$\text{Sensitivity } \partial V_1/\partial Q_{32} = 0.00319$$

For load-bus B12: Let,

$$\text{Voltage-Violation } \Delta V_{12} = -0.37019 \quad pu$$

$$\text{Sensitivity } \partial V_{12}/\partial Q_{32} = 0.03814$$

Now, in order to take care of a decrease in the voltage of bus B12 by the above mentioned value ($-0.37019 \quad pu$), the reactive power of voltage-bus B32 should be increased by the value ΔQ_{32-12} computed below:

$$\Delta Q_{32-12} = \Delta V_{12}/(\partial V_{12}/\partial Q_{32}) = -0.37019/0.03814 = -9.706 \quad Mvar$$

Now, in order to correct the voltage of bus B1, its voltage should be decreased by the value, which is the previously chosen step-size (0.04942 pu) value. Thus, the

voltage of bus B1 can be corrected by decreasing its reactive power by the value ΔQ_{32-1} computed below:

$$\Delta Q_{32-1} = \Delta V_1/(\partial V_1/\partial Q_{32}) = -0.04942/0.00319 = -15.51 \quad Mvar$$

Hence, the reactive power of the voltage bus B32 should be decreased only by a value given by:

$$\Delta Q_{32} = \Delta Q_{32-1} - \Delta Q_{32-12} = -15.51 - (-9.7) = -5.87 \quad Mvar$$

Let ΔX be the infered voltage step-size that we are looking for, so that the voltage violation of bus B1 can be corrected by a step-size of ΔX. Then the value of this ΔX can be obtained as follows:

$$\Delta X = \Delta Q_{32} \times (\partial V_1/\partial Q_{32}) = -5.87 \times 0.00319 = -0.01873 \quad pu$$

Thus the value of infered control voltage step-size would be -0.01873.

(7) **Prediction:** The chances that the present violation can be corrected is given by this slot. The prediction is categorized into good, fair or low based on the factors such as frequency of violation, inferred step size and available control measures. In EPSY three inferential operators are used to make specific inferences and to add references to the abstraction frame.

6.3.2.3. Path Consultant

This knowledge source renders consultation based on the inferences available on the abstraction frame and in effect corrects or modifies the search path of the expert system. The ``Path Consultant'' knowledge source concentrates on specific inferences depending on the context of the problem solving knowledge source active

at that time. Path Consultant modifies the information on the various levels of the problem solving blackboard depending upon the context, before the next knowledge source or module is triggered.

6.3.3. The Machine Learning Blackboard Control

The machine learning blackboard in EPSY has the blackboard control in common with the blackboard control of the main system or the problem solving blackboard. The blackboard scheduler mainly schedules or triggers the learning knowledge sources based on the control data and preconditions of the knowledge sources. The blackboard monitor keeps track of any changes made to the control data by the learning knowledge sources.

6.4. Final Remarks on the Machine Learning in EPSY

Machine learning in EPSY has been found very effective in reducing the possibilities of backtracking. Normally backtracking occurs whenever the postcontrol state of the power system is worser than the precontrol state. In order to backtrack, the system has to restore the previous values of the various parameters of the system. Time is spent in backtracking, making an attempt which may fail or in taking very small control step-size which virtually may not change the power system state. Hence a great saving in time or enhancement of overall system speed is realized by the machine learning as described above. Waste of time resulting from misjudgement or a path that would lead to a failure can be avoided to a great extent. This will significantly increase the utility and reduce the problem solving time of the expert system. The expert system can also bring the frequently occuring violation to the attention of the user. This may help the user in improving the design of the

power system.

CHAPTER 7

CONCLUSION

7.1. Further Research

A new dimension to this expert system can be given by distributed problem solving(DPS). A DPS system has two components: a collection of distinct, independent problem-solvers, and a control-mechanism to promote collaboration among them. This will mean that this expert system will have two parts:

(1) An expandable library of programs, these programs can be written in a variety of languages, and can run on different machines.

(2) The skeleton of a mechanism for integrating these programs, controlling them and managing their information flows.

The programs can be interconnected by means of interfaces to the blackboard, thus once the blackborad has been built the interconnection of N programs requires only the writing of N interfaces. Programs can post and read messages from the blackboard. The blackboard-monitor does the actual writing of messages, maintains the blackboard and draws the attention of programs to messages that might be of interest to them. With the above-mentioned type of architecture, this expert system can be extended to result in a Multi-Expert System.

Another very useful dimension can be added to this expert system by modifying its blackboard architecture to allow concurrent-processing. This will tremendously increase the speed and performance of this expert system, since it will be able

to compute a number of states simultaneously and choose the one with the least number of power system violations.

However, the introduction of concurrent execution into blackboard system requires extension of the architecture with new mechanisms for scheduling knowledge source interactions, and accessing shared data. For further details please refer ``Transactional Blackboards,'' J.Robert Ensor and John D. Gabbe. There is also a need for an intelligent user interface using the natural language processing, so that a user can query the system on its various decisions and actions.

7.2. Scope of this Expert System

In a survey made by Carnegie-Mellon Group for EPRI on the role that AI and expert systems can play to improve the performance of power systems, it is found that for various types of problems in power systems, different degrees of Operating Cost Savings, Capital Cost Savings, Improved Quality of Service, Generalizability, Overall Benefit is obtained. Please refer Table 1. It is found that the degree (on a scale 0 to 4) of overall benfit for Preventive-Control and Emergency-Control are 4 and 3 respectively, and the improved Quality of Service in case of Preventive-control and Emergency-control are 4 and 2 respectively. Hence there is a tremendous scope of this expert system in case of power system control, and as the size of a power system becomes larger this expert system will be more needed, and will be indespensable. Futher the architecture of this expert system demonstrates how a knowledge based hybrid approach could be the answer to most of the similar kind of engineering problems wherein the symbolic knowledge and decision making has to be supported by many analysis programs.

TABLE 1. A SUBJECTIVE ASSESMENT OF OPPORTUNITIES FOR USING ARTIFICIAL INTELLIGENCE TECHNIQUES IN POWER SYSTEM PROBLEMS.

PROBLEM	BENEFITS				
	OPERAT. COST BENEFITS	CAPITAL COST SAVINGS	IMPROVED QUALITY OF SERVICE	GENERALI-ZABILITY	OVERALL BENEFIT
ALARM-PROCESSING	0	0	3	2	3
FAULT DIAGNOSIS	0	0	2	0	2
CONTINGENCY-SELECTION & SECURITY-CONTROL	0	2	3	1	3
ECONOMIC CONTROL	2	1	2	1	2
PREVENTIVE CONTROL	2	2	4	1	4
EMERGENCY CONTROL	0	3	2	0	3
RESTORATIVE CONTROL	0	1	4	0	4
OPERATOR TRAINING	1	0	2	2	2
DEMAND MANAGEMENT	3	3	2	0	3
AUTOMATIC MANUALS	0	0	2	3	3

(SOURCE: INVESTIGATION BY CARNEGIE-MELLON FOR EPRI, '83)

7.3. Concluding Remarks

The main idea behind this expert system was to demonstrate how a knowledge based hybrid approach can be applied in developing an expert system for power system control. An important aspect of this expert system is that it learns from its failures while attempting to correct bus voltage violations. By doing so it reduces its chances of backtracking in the solution finding process, thus making it more intelligent in its approach to correct a bus voltage violation.

It has also been demonstrated that a blackboard architecture can be appropriately used to implement the inference and control strategy of such an expert system. The result is a greater flexibility in adding more knowledge sources or modules to this expert system as it grows. Also, with suitable modifications in the structure of the blackboard, concurrent and distributed processing could be made possible, which will significantly increase the speed and performance of the expert system.

The following are some of the specific features of this expert system:

(1) Both the machine learning and problem solving have been implemented using the blackboard architecture.

(2) Although the expert system application to power systems is in its infant stage, the performance of EPSY is impressive.

(3) This expert system has a hybrid architecture, which means both the numeric and symbolic problem-solvers have been used.

(4) The architecture of this expert system is highly flexible in terms of future growth, and it is easy to modify. The use of blackboard control structure encourages modularity in form of the use of knowledge sources. Its problem domain can be broadened and more problem-solvers can be easily added to this expert system.

(5) In EPSY, the emphasis has been on allowing the user to change the heuristic knowledge and other criterion to suit his needs. For this reason much of the heuristic knowledge is represented in form of production rules as they can be easily modified.

BIBLIOGRAPHY

BIBLIOGRAPHY

T. Sakaguchi, K. Matsumoto, "Development of a knowledge-based system for power-system restoration," IEEE Transactions on Power Apparatus and Systems, Vol. PAS-102, February 1983, pp. 320-329.

S.C. Lee, "A fast and accurate postcontingency state computation technique in power systems," proceedings of IEEE Southeastern conference '87, Tampa, FL., April 5-8 1987.

Brownston, Kant, Farell, Martin, "Programming Expert Systems in OPS5: an Introduction to rule-based programming." [et. al]. - Reading Mass., Addison Wesley, 1985.

"Building Expert Systems," edited by Frederick Hayes-Roth, Donald A. Waterman, Douglas B. Lenat, published by Addison-Wesley Company, Inc., 1983.

"Artificial Intelligence Technologies for Power System Operations," prepared by Carnegie-Mellon University for Electric Power Research Institute, CA.. Principal Investigators S. N. Talukdar, E. Cardozo. Final report, January 1986.

Elaine Rich, "Artificial Intelligence," 1983, McGraw-Hill, Inc.

H. Penny Nii, "Blackboard Systems, Blackboard Application Systems, Blackboard Systems from a Knowledge-Engineering Perspective," AI Magazine, August '86, pp. 82-106.

"Machine Learning - An Artificial Intelligence Approach," Ryszard S. Michalski, Jaime G. Carbonell, Tom M. Mitchell, 1983, Morgan Kaufmann Publishers, Inc.

"A comprehensive evaluation of expert system tools," J.F. Gilmore, K. Pulaski, C. Howard, Georgia Tech. Research Institute, Proceedings of - The International Society for Optical Engineering, volume 635, Applications of AI - III.

Ensor, J. R. and J. D. Gabbe. 1985. "Transactional Blackboards." Proceedings of the 9^{th} International Joint Conference on Artificial Intelligence. California: William Kaufmann, Inc., 340-344.

Erman. L. D., P. E. London, and S. F. Fickas. 1981. "The Design and an Example Use of HEARSAY-III." Proceedings of the 7^{th} International Joint

Conference on Artificial Intelligence. Los Altos, California: William Kaufmann, Inc., 409-415.

Nii, H. Penny., and N. Aiello. 1979. AGE: A Knowledge-based ``Program for Building Knowledge-based programs.'' Proceedings of the 6^{th} International Joint Conference on Artificial Intelligence. Los Altos, California: William Kaufmann, Inc., 645-655.

APPENDICES

APPENDIX A

EPSY PROGRAM

```
;;; -*- Mode: LISP; Syntax: Zetalisp; Package: USER; Base: 10; -*-

;**************MOUSE SENSITIVE WINDOW GRAPHICS  FOR A POWER SYSTEM*****************

(defvar alist-alpha nil)
(defvar fault-list nil)

(defflavor new ()
           (tv:centered-label-mixin
            tv:borders-mixin tv:top-box-label-mixin
            tv:basic-mouse-sensitive-items
            tv:window))

;global variables

(setq busx1 0 busy1 0 genx1 0 geny1 0 ldx 0 ldy 0 trlx 0 trly 0)

; This function draws a bus-bar

(setq buslist nil bus-name nil bus-number nil bus-voltage nil bus-angle nil bus-type nil)

(defun function4 ()
  (prog(button x1 y1 x2 y2)
       (send sensitive-window :expose)
       (setq button 0 x1 0 y1 0 x2 0 y2 0)
    label1
       (setq button tv:mouse-buttons)
       (cond ((equal button nil)(go label1))
             ((equal button 1 )(setq x1 tv:mouse-x)(setq y1 tv:mouse-y))
             ((equal button 2 )(setq x2 tv:mouse-x)(setq y2 tv:mouse-y))
             ((and (equal button 4 )(not (equal x1 0)))
              (draw-lines x1 y1 x2 y2 scal1)
              (setq busx1 x1)(setq busy1 y1)(start 'bus)
              (cond ((equal wish "yes")
                     (setq buslist (uniq (cons (list (list x1 y1 x2 y2)
                                                     (list bus-name bus-number bus-voltage
                                                           bus-angle bus-type)) buslist)))
                     )))
             ((greaterp tv:mouse-x 800)(return)))
       (go label1)))

;This function draws a transmission line or a connection vertical or horizontal

(setq l nil translist nil trl-name nil trl-number nil trl-admitt-serial nil
      trl-admitt-parallel nil trl-connect-bus1 nil trl-connect-bus2 nil trl-capacity
      nil trl-cb1-status nil trl-cb1-num nil trl-cb2-status nil trl-cb2-num nil)

(defun function5 ()
  (prog(button x1 y1 x2 y2)
       (send sensitive-window :expose)
       (setq x1 0 y1 0 x2 0 y2 0 l nil)
    label2
       (setq button tv:mouse-buttons)
       (cond ((equal button nil)(go label2))
             ((and (equal button 1 )(not (equal tv:mouse-x nil)))
              (setq x1 tv:mouse-x)(setq y1 tv:mouse-y)(setq l (cons (list x1 y1) l)))
             ((and (equal button 2 )(not (equal tv:mouse-x nil)))
              (setq x2 tv:mouse-x)(setq y2 tv:mouse-y)(setq l (cons (list x2 y2) l)))
             ((and (equal button 4 )(not (equal l nil)))
              (drawline (reverse l))(setq trlx x1 trly y1)(start 'trln)
              (cond ((equal wish "yes")
                     (setq translist (uniq (cons (list l
                                                       (list trl-name trl-number
                                                             trl-admitt-serial
                                                             trl-admitt-parallel
                                                             trl-connect-bus1
                                                             trl-connect-bus2
                                                             trl-tap-position
                                                             trl-cb1-status
                                                             trl-cb1-num
                                                             trl-cb2-status
                                                             trl-cb2-num))
                                                 translist)))))
             (setq l nil))
```

```
            ((greaterp tv:mouse-x 800)(return)))
        (go label2)))

(defun drawline (lstpt)
  (prog(lst num first last)
       (setq lst lstpt)(setq num (length lst))
    label1
       (cond ((equal num 1)(return))
             (t(setq xy1 (car lst) xy2 (cadr lst) lst (cdr lst))
               (setq x1 (car xy1) y1 (cadr xy1) x2 (car xy2) y2 (cadr xy2))
               (cond ((>(abs (- x1 x2))(abs (- y1 y2)))
                      (send sensitive-window :draw-line
                            (- x1 3)(- y1 23)  (- x2 3) (- y1 23)  tv:alu-xor))
                     (t(send sensitive-window :draw-line
                             (- x1 3) (- y1  23) (- x1 3)(- y2 23) tv:alu-xor)))
               (setq num (- num 1))(go label1)))))

; This function draws a generator

(setq genlist nil gen-name nil gen-number nil gen-rating nil gen-current-mw nil
      gen-current-mva nil gen-current-mvar nil connected-to-bus nil)

(defun function6 ()
  (send sensitive-window :expose)
  (prog (button x1 y1 x2 y2)
       (setq x1 0 y1 0 x2 0 y2 0)
    label3
       (setq button tv:mouse-buttons)
       (cond ((equal button nil)(go label3))
             ((equal button 1)(setq x1 tv:mouse-x)(setq y1 tv:mouse-y))
             ((equal button 2)(setq x2 tv:mouse-x)(setq y2 tv:mouse-y))
             ((and (equal button 4)(not (equal x1 0)))(draw-circle x1 y1)
              (cond ((> (abs (- x1 x2))(abs (- y1 y2)))
                     (send sensitive-window :draw-line
                           (+ genx1 scal2) geny1 (- x2 3) geny1 tv:alu-xor))
                    (t(send sensitive-window :draw-line
                            genx1 (+ geny1 scal2) genx1 (- y2 23) tv:alu-xor)))
              (start 'gen)
              (cond ((equal wish "yes")
                     (setq genlist (uniq (cons (list (list x1 y1 x2 y2)
                                                     (list gen-name gen-number
                                                           gen-rating
                                                           gen-current-mw
                                                           gen-current-mva
                                                           gen-current-mvar
                                                           connected-to-bus))
                                               genlist))))))
             ((greaterp tv:mouse-x 800)(return)))
       (go label3)))

; This function draws an open ckt-breaker

(defun function7()
  (send sensitive-window :expose)
  (prog (button x1 y1 x2 y2)
       (setq x1 0 y1 0 x2 0 y2 0)
    label4
       (setq button tv:mouse-buttons)
       (cond ((equal button nil)(go label4))
             ((equal button 1)(setq x1 tv:mouse-x)(setq y1 tv:mouse-y))
             ((equal button 2)(setq x2 tv:mouse-x)(setq y2 tv:mouse-y))
             ((equal button 4)(draw-cb-open x1 y1 x2 y2))
             ((greaterp tv:mouse-x 800)(return)))
       (go label4)))

; This function draws a closed circuit breaker

(defun function07()
  (send sensitive-window :expose)
  (prog (button x1 y1 x2 y2)
       (setq x1 0 y1 0 x2 0 y2 0)
    label4
       (setq button tv:mouse-buttons)
       (cond ((equal button nil)(go label4))
             ((equal button 1)(setq x1 tv:mouse-x)(setq y1 tv:mouse-y))
             ((equal button 2)(setq x2 tv:mouse-x)(setq y2 tv:mouse-y))
```

```
                ((equal button 4)(draw-cb-close x1 y1 x2 y2))
                ((greaterp tv:mouse-x 800)(return)))
          (go label4)))

  ;this function draws a load

  (setq loadlist nil load-name nil load-number nil load-size-mw nil
        load-size-mvar nil connected-to-bus nil)

  (defun function8()
    (send sensitive-window :expose)
    (prog (button x1 y1 x2 y2)
          (setq x1 0 y1 0 x2 0 y2 0)
       label8
          (setq button  tv:mouse-buttons)
          (cond ((equal button nil)(go label8))
                ((equal button 1)(setq x1 tv:mouse-x)(setq y1 tv:mouse-y))
                ((equal button 2)(setq x2 tv:mouse-x)(setq y2 tv:mouse-y))
                ((and (equal button 4)(not (equal x1 0)))
                 (draw-load x1 y1 x2 y2)(setq ldx x2)(setq ldy y2)(start 'load)
                 (cond ((equal wish "yes")
                        (setq loadlist (uniq (cons (list (list x1 y1 x2 y2)
                                                         (list load-name
                                                               load-number
                                                               load-size-mw
                                                               load-size-mvar
                                                               connected-to-bus))
                                                   loadlist))))))
                ((greaterp tv:mouse-x 800)(return)))
          (go label8)))

 ; This function draws an electrical load

 (defun draw-load(x1 y1 x2 y2)
   (setq xx1 (- x1 3) yy1 (- y1 25) xx2 (- x2 3) yy2 (- y2 25))
   (cond ((> (abs (- x1 x2))(abs (- y1 y2)))
          (send sensitive-window :draw-line xx1 yy1 xx2 yy1 tv:alu-xor)

          (cond ((> x2 x1) (send sensitive-window :draw-triangle
                                 xx2 (- yy1 5) (+ xx2 10) yy1 xx2  (+ yy1 5)
                                 tv:alu-xor))
                (t(send sensitive-window :draw-triangle
                        xx2 (- yy1 5) (- xx2 10)yy1 xx2 (+ yy1 5)
                        tv:alu-xor))))
         (t(send sensitive-window :draw-line xx1 yy1 xx1 yy2 tv:alu-xor)

           (cond ((> y2 y1)(send sensitive-window :draw-triangle
                                 (- xx1 5) yy2  xx1 (+ yy2 10)(+ xx1 5)yy2
                                 tv:alu-xor))
                 (t(send sensitive-window :draw-triangle
                         (- xx1 5)yy2 xx1 (- yy2 10)(+ xx1 5) yy2
                         tv:alu-xor))))))


 ; This function draws a transformer

 (defun function9()
   (send sensitive-window :expose)
   (prog (button x1 y1 x2 y2)
         (setq x1 0 y1 0 x2 0 y2 0)
      label9
         (setq button tv:mouse-buttons)
         (cond ((equal button nil)(go label9))
               ((equal button 1)(setq x1 tv:mouse-x)(setq y1 tv:mouse-y))
               ((equal button 2)(setq x2 tv:mouse-x)(setq y2 tv:mouse-y))
               ((equal button 4)(draw-transformer x1 y1 x2 y2))
               ((greaterp tv:mouse-x 800)(return)))
         (go label9)))


 ; This function draws an open circuit breaker

 (defun draw-cb-open(x1 y1 x2 y2 name)
   (setq xcb1 (- x1 3) xcb2 (- x2 3) ycb1 (- y1 23) ycb2 (- y2 23))
   (cond ((and (> (abs (- xcb1 xcb2))(abs (- ycb1 ycb2)))(>(- xcb2 xcb1) 0))
          (setq xcb1 (+ xcb1 13))
```

```
          (send sensitive-window :draw-line   xcb1 ycb1 (+ xcb1 5) ycb1
                tv:alu-xor)
          (send sensitive-window :draw-line xcb1 (- ycb1 5) xcb1 (+ ycb1 5)
                tv:alu-xor)
          (send sensitive-window :draw-line
                (+ xcb1 5) (- ycb1 5) (+ xcb1 5) (+ ycb1 5) tv:alu-xor)
          (send sensitive-window :draw-string name xcb1  (- ycb1 5)))

         ((and (> (abs (- xcb1 xcb2))(abs (- ycb1 ycb2)))(<(- xcb2 xcb1) 0))
          (setq xcb1 (- xcb1 13))
          (send sensitive-window :draw-line   xcb1 ycb2 (+ xcb1 5) ycb2
                tv:alu-xor)
          (send sensitive-window :draw-line xcb1 (- ycb2 5) xcb1 (+ ycb2 5)
                tv:alu-xor)
          (send sensitive-window :draw-line
                (+ xcb1 5) (- ycb2 5) (+ xcb1 5) (+ ycb2 5) tv:alu-xor)
          (send sensitive-window :draw-string name  (- xcb1 15)(- ycb2 5)))

         ((and (<(abs (- xcb1 xcb2))(abs (- ycb1 ycb2)))(>(- ycb2 ycb1)0))
          (setq ycb1 (+ ycb1 13))
          (cond ((equal sw 1)(setq xcb1 xcb2)))
          (send sensitive-window :draw-line xcb1 ycb1 xcb1 (+ ycb1 5)
                tv:alu-xor)
          (send sensitive-window :draw-line
                (- xcb1 5) ycb1 (+ xcb1 5) ycb1 tv:alu-xor)
          (send sensitive-window :draw-line
                (- xcb1 5) (+ ycb1 5) (+ xcb1 5)(+ ycb1 5) tv:alu-xor)
          (send sensitive-window :draw-string name (+ xcb1 4) (+ ycb1 15)))

         ((and (<(abs (- xcb1 xcb2))(abs (- ycb1 ycb2)))(<(- ycb2 ycb1)0))
          (setq ycb1 (- ycb1 13))
          (send sensitive-window :draw-line xcb2 ycb1 xcb2 (+ ycb1 5)
                tv:alu-xor)
          (send sensitive-window :draw-line
                (- xcb2 5) ycb1 (+ xcb2 5) ycb1 tv:alu-xor)
          (send sensitive-window :draw-line
                (- xcb2 5) (+ ycb1 5) (+ xcb2 5)(+ ycb1 5) tv:alu-xor)
          (send sensitive-window :draw-string name (+ xcb2 4) (- ycb1 3)))))

;This function draws a closed circuit-breaker

(setq sw 0)

(defun draw-cb-close(x1 y1 x2 y2 name)
  (setq xcb1 (- x1 3) ycb1 (- y1 23) xcb2 (- x2 3) ycb2 (- y2 23))
  (cond ((and (> (abs (- xcb1 xcb2))(abs (- ycb1 ycb2)))(> (- xcb2 xcb1) 0))
          (setq xcb1 (+ xcb1 13))
          (send sensitive-window :draw-line xcb1 ycb1 (+ xcb1 5) ycb1
                tv:alu-ior)
          (send sensitive-window :draw-line xcb1 (- ycb1 5) xcb1 (+ ycb1 5)
                tv:alu-ior)
          (send sensitive-window :draw-line
                (+ xcb1 5) (- ycb1 5) (+ xcb1 5) (+ ycb1 5) tv:alu-ior)
          (send sensitive-window :draw-line
                (- xcb1 5) (+ ycb1 5) (+ xcb1 10) (- ycb1 5) tv:alu-ior))

         ((and (> (abs (- xcb1 xcb2))(abs (- ycb1 ycb2)))(< (- xcb2 xcb1) 0))
          (setq xcb1 (- xcb1 13))
          (send sensitive-window :draw-line xcb1 ycb2 (+ xcb1 5) ycb2
                tv:alu-ior)
          (send sensitive-window :draw-line xcb1 (- ycb2 5) xcb1 (+ ycb2 5)
                tv:alu-ior)
          (send sensitive-window :draw-line
                (+ xcb1 5) (- ycb2 5) (+ xcb1 5) (+ ycb2 5) tv:alu-ior)
          (send sensitive-window :draw-line
                (- xcb1 5) (+ ycb2 5) (+ xcb1 10) (- ycb2 5) tv:alu-ior))

         ((and (<(abs (- xcb1 xcb2))(abs (- ycb1 ycb2)))(>(- ycb2 ycb1)0))
          (setq ycb1 (+ ycb1 13))(cond ((equal sw 1)(setq xcb1 xcb2)))
          (send sensitive-window :draw-line (- xcb1 5) ycb1 (+ xcb1 5) ycb1 tv:alu-ior)
          (send sensitive-window :draw-line (- xcb1 5) (+ ycb1 5) (+ xcb1 5)(+ ycb1 5)
                tv:alu-ior)
          (send sensitive-window :draw-line
                (- xcb1 5) (+ ycb1 10) (+ xcb1 5) (- ycb1 5) tv:alu-ior))

         ((and (<(abs (- xcb1 xcb2))(abs (- ycb1 ycb2)))(<(- ycb2 ycb1)0))
```

```
            (setq ycb1 (- ycb1 13))
            (send sensitive-window :draw-line (- xcb2 5) ycb1 (+ xcb2 5) ycb1 tv:alu-ior)
            (send sensitive-window :draw-line
                  (- xcb2 5) (+ ycb1 5) (+ xcb2 5)(+ ycb1 5) tv:alu-ior)
            (send sensitive-window :draw-line
                  (- xcb2 5) (+ ycb1 10) (+ xcb2 5) (- ycb1 5) tv:alu-ior))))

  ;This function draws a  circle

  (defun draw-circle(x1 y1)
    (setq xxx1 (- x1 3))(setq yyy1 (- y1 25))(send sensitive-window :draw-circle
                                                  xxx1 yyy1 scal2 tv:alu-xor)
    (setq genx1 xxx1 geny1 yyy1))

  ; This function draws either vertical or horizontal line

  (defun draw-lines (x1 y1 x2 y2 thick)
    (setq height (isqrt (+ (* (- x1 x2)(- x1 x2)) (* (- y1 y2)(- y1 y2)))))
    (cond ((> (abs (- x1 x2))(abs (- y1 y2)))
          (setq y1 (- y1 24)) (setq x1 (- x1 3))
          (send sensitive-window :draw-rectangle height thick x1 y1
                tv:alu-xor))
         (t (setq y1 (- y1 26))(setq x1 (- x1 3))
            (send sensitive-window :draw-rectangle
                  thick height x1 y1 tv:alu-xor))))

  ;This function draws a transformer when called by function 9

  (defun draw-transformer(x1 y1 x2 y2)
    (setq xx1 (- x1 3) yy1 (- y1 25) xx2 (- x2 3) yy2 (- y2 25))
    (cond ((> (abs (- x1 x2))(abs (- y1 y2)))
          (dln xx1 (- yy1 10))(dln xx1 (- yy1 5))(dln xx1 yy1)(dln xx1 (+ yy1 5))
          (send sensitive-window :draw-lines tv:alu-xor
                xx2 (- yy1 10) (- xx2 5) (- yy1 7) xx2 (- yy1 5)
                (- xx2 5)(- yy1 3) xx2 yy1
                (- xx2 5)(+ yy1 3) xx2 (+ yy1 5) (- xx2 5)(+ yy1 7) xx2 (+ yy1 10)))
         (t(send sensitive-window :draw-lines tv:alu-xor
                 (- xx1 10) yy1 (- xx1 7)(+ yy1 5)(- xx1 5) yy1 (- xx1 3)(+ yy1 5)
                 xx1 yy1
                 (+ xx1 3) (+ yy1 5) (+ xx1 5) yy1 (+ xx1 7) (+ yy1 5) (+ xx1 10) yy1)
           (send sensitive-window :draw-lines tv:alu-xor (- xx1 10) yy2
                 (- xx1 7)(- yy2 5)(- xx1 5) yy2 (- xx1 3)(- yy2 5) xx1 yy2
                 (+ xx1 3) (- yy2 5)(+ xx1 5) yy2 (+ xx1 7)(- yy2 5)(+ xx1 10) yy2))))

  ; This function draws a right arrow head
  (defun dln(x0 y0)
    (send sensitive-window :draw-line x0 y0 (+ x0 5) (+ y0 3) tv:alu-xor)
    (send sensitive-window :draw-line (+ x0 5)(+ y0 3) x0 (+ y0 5)))


  ;Functions below modify the attributes of components

  (defun modify()
    (prog(choice)
         (setq z (tv:make-window 'tv:momentary-menu
                                 ':label "New Selection "
                                 ':item-list '("Modify Attributes"
                                               "Delete a Component")))
         (setq choice (send z ':choose))
         (cond ((equal choice "Modify Attributes")(attr-modify))
               (t(delete-comp)(return)))))

 (defun attr-modify()
   (setq z1 (tv:make-window 'tv:momentary-menu
                            ':label "Attribute Modification "
                            ':item-list '("generator" "bus" "load"
                                          "transmission line")))
   (setq choice1 (send z1 ':choose))
   (cond ((equal choice1 "generator")(gen-mod))
         ((equal choice1 "bus")(bus-mod))
         ((equal choice1 "load")(load-mod))
         ((equal choice1 "transmission line")(trans-mod))))

; This function deletes a component

(defun delete-comp()
```

```
(prog(choice2)
     (setq z2 (tv:make-window 'tv:momentary-menu
                              ':label "Component Deletion"
                              ':item-list '("generator" "bus" "load"
                                            "transmission line")))
     (setq choice2 (send z2 ':choose))
     (cond ((equal choice2 "generator")(gen-del)(return))
           ((equal choice2 "bus")(bus-del)(return))
           ((equal choice2 "load")(load-del)(return))
           ((equal choice2 "transmission line"(trans-del)(return)))))))
```

```
(tv:add-typeout-item-type alist-alpha
                          :new-type "EXIT              " (exit))
(tv:add-typeout-item-type alist-alpha
                          :new-type "fault-menu        " (fault-menu))
(tv:add-typeout-item-type alist-alpha
                          :new-type "SYSTEM EVALUATION"
                          (init-system-evaluation))
(tv:add-typeout-item-type alist-alpha
                          :new-type "MODIFY            " (modify))
(tv:add-typeout-item-type alist-alpha
                          :new-type "ERASE SCREEN-GRID"
                          (grid 800 550))
(tv:add-typeout-item-type alist-alpha
                          :new-type "UPDATE SCREEN     " (update))
(tv:add-typeout-item-type alist-alpha
                          :new-type "RETRIEVE          " (retrieve))
(tv:add-typeout-item-type alist-alpha
                          :new-type "SAVE              " (save))
(tv:add-typeout-item-type alist-alpha
                          :new-type "TRANSMISSION-LINE" (function5))
(tv:add-typeout-item-type alist-alpha
                          :new-type "GENERATOR         " (function6))
(tv:add-typeout-item-type alist-alpha
                          :new-type "CKT-BREAKER       " (function07))
(tv:add-typeout-item-type alist-alpha
                          :new-type "LOAD              "(function8))
(tv:add-typeout-item-type alist-alpha
                          :new-type "TRANSFORMER       "(function9))
(tv:add-typeout-item-type alist-alpha
                          :new-type "BUS               " (function4))
(tv:add-typeout-item-type fault-list
                          :new-type "6. EXIT             " (exit))
(tv:add-typeout-item-type fault-list
                          :new-type "5. LOAD CHANGE      " (blink-load))
(tv:add-typeout-item-type fault-list
                          :new-type "4. LOAD DROP        " (blink-load))
(tv:add-typeout-item-type fault-list
                          :new-type "3. T//L TRIP        "
                          (trip-trln2 (trip-trln1)))
(tv:add-typeout-item-type fault-list
                          :new-type "2. GENERATION CHANGE" (blink-load))
(tv:add-typeout-item-type fault-list
```

```
                              :new-type "1. GENERATION DROP   " (show-generator))

; ******** MOUSE SENSITIVE WINDOW FLAVOR IS DEFINED BELOW ********

(defvar sensitive-window
        (tv:make-window
          'new
          ':borders 2
          ':top 0
          ':bottom 736
          ':right 750
          ':width 750
          ':blinker-p nil
          ':label '(:string
                    "A KNOWLEDGE-BASED SYSTEM DEVELOPMENT FOR POWER-SYSTEM CONTROL"
                    :font fonts:bigfnt)
          ':item-type-alist alist-alpha
          ':font-map (list fonts:HL7)))

; Fault selection window is defined using flavor below

(defvar fault-selection-window
        (tv:make-window
          'new
          ':borders 2
          ':top 600
          ':bottom 736
          ':right 1100
          ':width 350
          ':expose-p t
          ':label '(:string "POWER-SYSTEM DISTURBANCE-SELECTION"
                            :font fonts:cptfontcb)
          ':item-type-alist fault-list))

; This function defines fault-selection window menu

(defun fault-menu()
  (tv:window-call (fault-selection-window :deactivate)
    (send fault-selection-window ':expose)
    (send fault-selection-window :set-font-map  (list fonts:cptfontcb))
    (send fault-selection-window ':item ':new-type "select")
    (loop as blip = (send fault-selection-window ':any-tyi)
          until (equal (cadr blip) '(exit))
          do (eval (cadr blip)))) )

;***************** THIS IS TOP-MOST  LEVEL FUNCTION  ****************
(setq scal1 nil scal2 nil scale 0)

(defun set-up ()
  (tv:window-call (interactive-window :expose)
    (send interactive-window :line-out "enter the component-scaling factor")
    (setq scale (read interactive-window))
    (cond ((equal scale 1)(setq scal1 5 scal2 20))
          (t(setq scal1 3 scal2 10))))
  (tv:window-call (sensitive-window :deactivate)
    (send sensitive-window ':expose)(grid 800 550 )
    (send sensitive-window :set-font-map  (list fonts:cptfontcb))
    (send sensitive-window ':item ':new-type "system")
    (loop as blip = (send sensitive-window ':any-tyi)
          until (equal (cadr blip) '(exit))
          do (eval (cadr blip)))))

; This function retrieves a power system from a file.

(defun retrieve-set-up ()
  (grid 800 550)
  (send sensitive-window ':item ':new-type "system"))

; ******** THIS FLAVOR DEFINES A DIALOGUE/MESSAGE  WINDOW ********

(defvar interactive-window
        (tv:make-window
          'tv:window
          ':borders 2
          ':top 0
          ':bottom 600
```

```
          ':right 1100
          ':width 350
          ':expose-p t
          ':label '(:string "MESSAGE//DIALOGUE WINDOW"
                            :font fonts:cptfontcb)))

(setq wish nil)

(defun start(typ)
  (prog()
       (tv:window-call (interactive-window :expose)
         (send interactive-window :home-cursor)
         (send interactive-window :refresh)
         (send interactive-window :line-out 'want-to-enter-attributes)
         (setq wish (readline interactive-window))
         (cond ((equal wish "no")(return))
               ((equal typ 'ocb)
                (send interactive-window :line-out 'enter-the-left-component)
                (setq left-component (readline interactive-window))
                (setq status 'open)(return))
               ((equal typ 'ccb)
                (send interactive-window :line-out 'enter-the-left-component)
                (setq left-component (readline interactive-window))
                (setq status 'close))
               ((equal typ 'bus)
                (send interactive-window :refresh)
                (format interactive-window
                        "~% bus-name= ~A ~% number= ~A ~% voltage(kv)= ~A ~% angle(degree)= ~A
                 ~% type(S V L)= ~A ~%"
                        (setq bus-name (read interactive-window))
                        (setq bus-number (read interactive-window))
                        (setq bus-voltage (read interactive-window))
                        (setq bus-angle (read interactive-window))
                        (setq bus-type (read interactive-window))
                        (send sensitive-window :draw-string
                              bus-name (- busx1 16)(- busy1 23))))
              ((equal typ 'gen)
               (send interactive-window :refresh)
               (format interactive-window
                       "~% generator-name= ~A ~% number= ~A ~% rated-capacity(mva)= ~A
                        ~% current-output-mw= ~A ~% current-output-mva= ~A
                        ~% current-output-mvar= ~A ~% connected-to-bus no.= ~A ~%"
                       (setq gen-name (read interactive-window))
                       (setq gen-number (read interactive-window))
                       (setq gen-rating (read interactive-window))
                       (setq gen-current-mw (read interactive-window))
                       (setq gen-current-mva (read interactive-window))
                       (setq gen-current-mvar (read interactive-window))
                       (setq connected-to-bus (read interactive-window))
                       (send sensitive-window :draw-string
                             gen-name (- genx1 7)  (+ geny1 4))))
             ((equal typ 'load)
              (send interactive-window :refresh)
              (format interactive-window
                      "~%load-name= ~a ~%number= ~a ~%size-mw= ~a ~% size-mvar= ~a
                       ~%connected-to-bus-no.= ~a ~%"
                      (setq load-name (read interactive-window))
                      (setq load-number (read interactive-window))
                      (setq load-size-mw (read interactive-window))
                      (setq load-size-mvar (read interactive-window))
                      (setq connected-to-bus (read interactive-window))
                      (send sensitive-window :draw-string load-name (- ldx 16)(- ldy 8))))
             ((equal typ 'trln)
              (send interactive-window :refresh)
              (format interactive-window
                      "~%transmn-line-name= ~a ~%number= ~a ~%admitt-serial-pu= ~a
                       ~%admitt-parall-pu(1//2)= ~a ~%connected-bus-1= ~a
                       ~%connected-bus-2= ~a ~%tap-position= ~a ~%CB-1-status= ~a
                       ~%CB-1-num= ~a ~%CB-2-status= ~a ~%CB-2-num= ~a ~%"
                      (setq trl-name  (read interactive-window))
                      (setq trl-number  (read interactive-window))
                      (setq trl-admitt-serial  (read interactive-window))
                      (setq trl-admitt-parallel  (read interactive-window))
                      (setq trl-connect-bus1  (read interactive-window))
                      (setq trl-connect-bus2  (read interactive-window))
                      (setq trl-tap-position  (read interactive-window))
```

```
                    (setq trl-cb1-status  (read interactive-window))
                    (setq trl-cb1-num  (read interactive-window))
                    (setq trl-cb2-status  (read interactive-window))
                    (setq trl-cb2-num  (read interactive-window)))
              (setq l (reverse (setq l (uniq l))))
              (setq xy4 (car l))
              (setq xy3 (cadr l))
              (setq revl (reverse l))
              (setq xy1 (car revl) xy2 (cadr revl))
              (cond ((equal trl-cb1-status 'open)(setq sw 0)
                     (draw-cb-open (car xy1) (cadr xy1) (car xy2) (cadr xy2)
                                   trl-cb1-num))
                    (t(setq sw 0)(draw-cb-close (car xy1) (cadr xy1)
                                                (car xy2)(cadr xy2) trl-cb1-num)))
              (cond ((equal trl-cb2-status 'open)
                     (setq sw 1)
                     (draw-cb-open (car xy4)(cadr xy4)(car xy3)(cadr xy3)
                                   trl-cb2-num))
                    (t(setq sw 1)(draw-cb-close (car xy4)(cadr xy4)
                                                (car xy3)(cadr xy3) trl-cb2-num)))
              (naming trl-name))))))


(setq stringposlist nil)

(defun naming(name)
  (prog()
   label
      (cond((equal tv:mouse-buttons 1)
            (send sensitive-window :draw-string name
                  (- tv:mouse-x 3) (- tv:mouse-y 23))
            (setq stringposlist (cons (list trl-name (- tv:mouse-x 3)
                                            (- tv:mouse-y 23)) stringposlist))
            (return))
           (t(go label)))))

(defun draw-stringposlist (l)
  (prog(nam mous-x mous-y)
   label
      (cond ((null l)(return))
            (t(setq strgl (car l) nam (car strgl) mous-x (cadr strgl)
                    mous-y(caddr strgl) l(cdr l))
              (send sensitive-window :draw-string nam mous-x mous-y )
              (go label)))))

; This function draws a graphics screen grid

(defun grid(x y)
  (prog(px py)
       (setq px x py y)
   label1
      (send sensitive-window :draw-point px py tv:alu-xor)
      (setq px (- px scal2))
      (cond ((< px scal2)(setq px x)(go label2))
            (t(go label1)))
   label2
      (setq py (- py scal2))
      (cond ((< py scal2)(return))
            (t(go label1)))))

;This function writes output messages in an file

(setq file-name nil)

(defun save()
  (tv:window-call (interactive-window :expose)
    (send interactive-window :home-cursor)
    (send interactive-window :refresh)
    (send interactive-window :line-out "enter file-name to save info")
    (send interactive-window :set-cursorpos 5 15)
    (with-open-file (str (setq file-name (readline interactive-window))
                         ':direction ':output
                         ':characters t)
      (print   genlist str)(print loadlist str)(print buslist str)
      (print translist str)(print stringposlist str)
      (print file-name str))))
```

```
;This function RETRIEVES  info from a given file and draws the power system

(defun retrieve()
  (setq genlist nil buslist nil loadlist nil translist nil)
  (tv:window-call (interactive-window :expose)
    (send interactive-window :line-out "enter scaling factor")
    (setq scale (read interactive-window))
    (cond ((equal scale 1)(setq scal1 5 scal2 20))
          (t(setq scal1 3 scal2 10)))
    (send interactive-window :home-cursor)
    (send interactive-window :refresh)
    (send interactive-window :line-out  "enter file-name to retrieve info")
    (send interactive-window :set-cursorpos 5 15)
    (with-open-file (str (readline interactive-window)
                         ':direction ':input
                         ':characters t)
      (setq genlist (read  str ) loadlist (read  str ) buslist (read  str )
            translist (read  str ) stringposlist (read str)
            file-name (read str))))
  (create-object-knowledge-base buslist genlist loadlist translist)
  (tv:window-call (sensitive-window :expose)
    (send sensitive-window :home-cursor)
    (send sensitive-window :refresh)
    (retrieve-set-up)
    (send sensitive-window :draw-string   file-name 50 690)
    (send sensitive-window :draw-string   "BASE MVA : 100.0 " 600 680)
    (send sensitive-window :set-font-map (list fonts:hl7))
    (draw-genlist genlist)
    (draw-buslist buslist)
    (draw-loadlist loadlist)
    (draw-translist translist)
    (draw-stringposlist stringposlist)))

; This function updates the information on the screen

(defun update()
  (create-object-knowledge-base buslist genlist loadlist translist)
  (tv:window-call (sensitive-window :expose)
    (send sensitive-window :home-cursor)
    (send sensitive-window :refresh)
    (retrieve-set-up)
    (draw-genlist genlist)
    (draw-buslist buslist)
    (draw-loadlist loadlist)
    (draw-translist translist)
    (draw-stringposlist stringposlist)))

;This function draws a generator from the genlist

(defun draw-genlist(inlist)
  (prog (nlist compos compname)
        (setq nlist inlist)
    label
       (cond ((null nlist)(return))
             (t(setq compos (caar nlist))
               (draw-circle (car compos)(cadr compos))
               (cond ((> (abs (- (car compos)(caddr compos)))
                         (abs (- (cadr compos)(cadddr compos))))
                      (send sensitive-window :draw-line
                            (+ genx1 scal2) geny1
                            (- (caddr compos) 3) geny1))
                     (t(send sensitive-window :draw-line
                             genx1 (+ geny1 scal2)
                             genx1 (- (cadddr compos) 23))))
               (setq compname (caadar nlist))
               (send sensitive-window :draw-string
                     compname (- genx1 7) (+ geny1 4))
               (setq nlist (cdr nlist))(go label)))))


;This function draws bus from the buslist

(defun draw-buslist(inlist)
  (prog (nlist compos compname bus-volt-list volt n)
        (setq nlist inlist bus-volt-list nil)
    label
```

```
          (cond ((null nlist)
                 (setq final-list (sortcar  bus-volt-list #'<))
                 (send sensitive-window :set-cursorpos  10 550)
                 (format sensitive-window
                         "~4%Initial Bus Voltages Are Mentioned Below: ~4 ~a"
                         final-list)
                 (return))
                (t(setq compos (caar nlist))(draw-lines
                                             (car compos)(cadr compos)
                                             (caddr compos)(cadddr compos)
                                             scall)
                  (setq bus-name (caadar nlist) busx1 (car compos)
                        busy1 (cadr compos))
                  (send sensitive-window :draw-string
                        bus-name (- busx1 16) (- busy1 23))
                  (send sensitive-window :set-cursorpos  (- busx1 27)(- busy1 20))
                  (setq volt (funcall (eval bus-name) :voltage-pu))
                  (setq n (funcall (eval bus-name) :number))
                  (format sensitive-window "~a" volt )
                  (setq bus-volt-list (cons (list n bus-name volt) bus-volt-list))
                  (setq nlist (cdr nlist))
                  (go label)))))

;This function draws load from the loadlist

(defun draw-loadlist(inlist)
  (prog(nlist compos compname)
       (setq nlist inlist)
    label
       (cond ((null nlist)(return))
             (t(setq compos (caar nlist))
               (draw-load (car compos)(cadr compos)
                          (caddr compos)(cadddr compos))
               (setq compname (caadar nlist))
               (send sensitive-window :draw-string
                     compname (- (caddr compos) 16)(- (cadddr compos) 8))
               (setq nlist (cdr nlist))(go label)))))


;This function draws a transmission line from translist.

(setq xy1 nil xy2 nil xy3 nil xy4 nil nlist nil)

(defun draw-translist(inlist)
  (prog(compos compname nlist  revlis nth)
       (setq nlist inlist)
    label
       (cond ((null nlist)(return))
             (t(setq compos (caar nlist))
               (setq compattr (cadar nlist) nth (cddddr compattr))
               (drawline (reverse compos))
               (setq trlx (car compos) trly (cadr compos))
               (setq trl-cb1-status  (fourth nth) trl-cb1-num
                     (fifth nth) trl-cb2-status (sixth nth) trl-cb2-num
                     (seventh nth) trl-name (car compattr))
               (setq xy4 (car compos) xy3 (cadr compos)
                     revlis (reverse compos) xy1 (car revlis)
                     xy2 (cadr revlis))
               (cond ((equal trl-cb1-status 'open)(setq sw 0)
                      (draw-cb-open (car xy1) (cadr xy1)
                                    (car xy2) (cadr xy2) trl-cb1-num))
                     (t(setq sw 0)(draw-cb-close (car xy1) (cadr xy1)
                                                 (car xy2)(cadr xy2)
                                                 trl-cb1-num)))
               (cond ((equal trl-cb2-status 'open)
                      (setq sw 1)
                      (draw-cb-open (car xy4)(cadr xy4)
                                    (car xy3)(cadr xy3) trl-cb2-num))
                     (t(setq sw 1)(draw-cb-close (car xy4)(cadr xy4)
                                                 (car xy3)(cadr xy3)
                                                 trl-cb2-num)))
               (setq nlist (cdr nlist))(go label)))))

;This function gets rid of repetitions of atoms inside a list.

(defun uniq (ls)
```

```
    (prog (newls)
          (setq newls nil)
          (for (i in ls)
               (do (or (member i newls)
                       (setq newls (cons i newls)) )))
          (return newls)))

  ;;******* COMPONENT DELETION  AND ATTRIBUTE MODIFICATION FUNCTIONS  ********

  (defun bus-del()
    (prog(positn)
         (tv:window-call (interactive-window :expose)
           (send interactive-window :home-cursor)
           (send interactive-window :refresh)
           (send interactive-window :line-out "enter bus-name")
           (setq comp (read interactive-window))
           (setq comp-lis (find-comp comp buslist))(setq positn (car comp-lis))
           (blink-bus (car comp-lis))(tv:window-call (sensitive-window :expose)
                                       (draw-lines (car positn)(cadr positn)
                                                   (caddr positn)(cadddr positn)scall))
           (setq buslist (delete comp-lis buslist))(return))))

  (defun gen-del()
    (prog()
         (tv:window-call (interactive-window :expose)
           (send interactive-window :home-cursor)
           (send interactive-window :refresh)
           (send interactive-window :line-out "enter generator-name")
           (setq comp (read interactive-window))
           (setq comp-lis (find-comp comp genlist))(blink-gen-2 (car comp-lis))
           (setq genlist (delete comp-lis genlist))(return))))

  (defun trans-del()
    (prog()
         (tv:window-call (interactive-window :expose)
           (send interactive-window :home-cursor)
           (send interactive-window :refresh)
           (send interactive-window :line-out "enter transmission-line-name")
           (setq comp (read interactive-window))
           (setq comp-lis (find-comp comp translist))(blink-trans-2 (car comp-lis))
           (setq translist (delete comp-lis translist))
           (setq comp-lis (assoc comp stringposlist))
           (setq stringposlist (delete comp-lis stringposlist))(return))))


  (defun erase (com )
    (prog(lcomp complis)
         (setq lcomp stringposlist)
     label
        (cond     ((equal com (caar lcomp))
                   (setq complis (car lcomp))
                   (delete complis stringposlist)(return))
                  (t (setq lcomp (cdr lcomp))(go label)))))

  (defun load-del()
    (prog()
         (tv:window-call (interactive-window :expose)
           (send interactive-window :home-cursor)
           (send interactive-window :refresh)
           (send interactive-window :line-out "enter load-name")
           (setq comp (read interactive-window))
           (setq comp-lis (find-comp comp loadlist))
           (blink-load-2 (car comp-lis))
           (setq loadlist (delete comp-lis loadlist))(return))))

  (defun bus-mod()
    (prog()
         (tv:window-call (interactive-window :expose)
           (send interactive-window :home-cursor)
           (send interactive-window :refresh)
           (send interactive-window :line-out "Enter bus-name")
           (setq comp (read interactive-window))
           (modfy comp buslist 'bus)(return))))

  (defun gen-mod()
    (prog()
```

```
        (tv:window-call (interactive-window :expose)
          (send interactive-window :home-cursor)
          (send interactive-window :refresh)
          (send interactive-window :line-out "Enter generator-name")
          (setq comp (read interactive-window))
          (modfy comp genlist 'gen)(return))))

(defun trans-mod()
  (prog()
        (tv:window-call (interactive-window :expose)
          (send interactive-window :home-cursor)
          (send interactive-window :refresh)
          (send interactive-window :line-out "Enter transmission-line-name")
          (setq comp (read interactive-window))
          (modfy comp translist 'trans)(return))))

(defun load-mod()
  (prog()
        (tv:window-call (interactive-window :expose)
          (send interactive-window :home-cursor)
          (send interactive-window :refresh)
          (send interactive-window :line-out "Enter load-name")
          (setq comp (read interactive-window))
          (modfy comp loadlist  'load)(return))))


(defun find-comp (tcomp tlist)

  (prog(ncomp comlist)
       (setq comlist tlist)
    label
       (setq ncomp (car comlist))
       (cond ((equal tcomp (caadr ncomp))(return ncomp))
             (t(setq comlist (cdr comlist))(go label)))))

(setq ncomp nil)

(defun modfy (tcomp tlist typ)

  (prog( comlist postn lst)
       (setq comlist tlist)
    label
       (setq ncomp (car comlist) postn (car ncomp) lst (car ncomp))
       (cond ((equal tcomp (caadr ncomp))
              (cond ((equal typ 'bus)(blink-bus (car ncomp))
                     (dsp-mod-bus ))
                    ((equal typ 'gen)(blink-gen (car ncomp))(dsp-mod-gen ))
                    ((equal typ 'trans)(blink-trans (car ncomp))(dsp-mod-trans ))
                    ((equal typ 'load )(blink-load (car ncomp))(dsp-mod-load )))
              (return))
             (t(setq comlist (cdr comlist))(go label)))))

(defun dsp-mod-bus ()
  (prog()
       (setq comp-sublst (cadr ncomp))
       (tv:window-call (interactive-window :expose)
         (send interactive-window :home-cursor)
         (send interactive-window :refresh)
         (format interactive-window "~%  Attributes-of-the-Component
              ~%bus-name = ~a ~%number    = ~a ~%voltage(kv)  = ~a
              ~%angle(degrees)     = ~a ~%type      = ~a~%"
                 (first  comp-sublst)(second comp-sublst)(third  comp-sublst)
                 (fourth comp-sublst)(fifth  comp-sublst))
         (send interactive-window :line-out
               "Enter New-attribute or - symbol for no-modification")
         (format interactive-window
                 "~%bus-name= ~a ~%number= ~a ~%voltage(pu)= ~a
                  ~%angle(degrees)= ~a ~%type= ~a ~%"
         (setq e1 (read interactive-window ))
         (setq e2 (read interactive-window ))
         (setq e3 (read interactive-window ))
         (setq e4 (read interactive-window ))
         (setq e5 (read interactive-window )))
         (cond ((not (equal e1 '-))(rplaca comp-sublst e1)))
         (cond ((not (equal e2 '-))(rplaca (cdr comp-sublst) e2 )))
         (cond ((not (equal e3 '-))(rplaca (cddr comp-sublst) e3)))
```

```
          (cond ((not (equal e4 '-))(rplaca (cdddr comp-sublst) e4)))
          (cond ((not (equal e5 '-))(rplaca (cddddr comp-sublst) e5)))
          (return))))

(defun dsp-mod-load ()
  (prog()
       (setq comp-sublst (cadr ncomp))
       (tv:window-call (interactive-window :expose)
         (send interactive-window :home-cursor)
         (send interactive-window :refresh)
         (format interactive-window "~%  Attributes-of-the-Component
              ~%load-name = ~a ~%number   = ~a ~%size-mw  = ~a
              ~%size-mvar   = ~a ~%connected-to-bus-num  = ~a~%"
                 (first  comp-sublst)(second comp-sublst)(third  comp-sublst)
                 (fourth comp-sublst)(fifth  comp-sublst))
         (send interactive-window :line-out
               "Enter New-attribute or - symbol for no-modification")
         (format interactive-window "~%load-name= ~a ~%number = ~a ~%size-mw = ~a
                                     ~%size-mvar= ~a ~%connect-bus = ~a ~%"
                 (setq e1 (read interactive-window ))
                 (setq e2 (read interactive-window ))
                 (setq e3 (read interactive-window ))
                 (setq e4 (read interactive-window ))
                 (setq e5 (read interactive-window )))
         (cond ((not (equal e1 '-))(rplaca comp-sublst e1)))
         (cond ((not (equal e2 '-))(rplaca (cdr comp-sublst) e2 )))
         (cond ((not (equal e3 '-))(rplaca (cddr comp-sublst) e3)))
         (cond ((not (equal e4 '-))(rplaca (cdddr comp-sublst) e4)))
         (cond ((not (equal e5 '-))(rplaca (cddddr comp-sublst) e5)))
         (return))))

(defun dsp-mod-gen ()
  (prog()
       (setq comp-sublst (cadr ncomp))
       (tv:window-call (interactive-window :expose)
         (send interactive-window :home-cursor)
         (send interactive-window :refresh)
         (format interactive-window "~%  Attributes-of-the-Component
              ~%generator-name = ~a ~%number   = ~a ~%rated-capacity(mw)  = ~a
              ~%current-capacity-mw = ~a ~%current-output-mw = ~a
              ~%current-output-mvar = ~a ~%connected-to-bus-num = ~a ~%"
                 (first  comp-sublst)(second comp-sublst)(third  comp-sublst)
                 (fourth comp-sublst)(fifth  comp-sublst)(sixth comp-sublst)
                 (seventh comp-sublst))
         (send interactive-window :line-out
               "Enter New-attribute or - symbol for no-modification")
         (format interactive-window
                 "~%gen-name= ~a ~%number= ~a ~%rated-capacity(mw)= ~a
                  ~%current-capacity-mw= ~a ~%current-output-m = ~a
                  ~%current-output-mvar= ~a ~%connected-bus= ~a ~%"
                 (setq e1 (read interactive-window ))(setq e2 (read interactive-window ))
                 (setq e3 (read interactive-window ))(setq e4 (read interactive-window ))
                 (setq e5 (read interactive-window ))(setq e6 (read interactive-window ))
                 (setq e7 (read interactive-window )))
         (cond ((not (equal e1 '-))(rplaca comp-sublst e1)))
         (cond ((not (equal e2 '-))(rplaca (cdr comp-sublst) e2 )))
         (cond ((not (equal e3 '-))(rplaca (cddr comp-sublst) e3)))
         (cond ((not (equal e4 '-))(rplaca (cdddr comp-sublst) e4)))
         (cond ((not (equal e5 '-))(rplaca (cddddr comp-sublst) e5)))
         (cond ((not (equal e6 '-))(rplaca (nthcdr 5 comp-sublst) e6)))
         (cond ((not (equal e7 '-))(rplaca (nthcdr 6 comp-sublst) e7)))
         (return))))


(defun dsp-mod-trans ()
  (prog()
       (setq comp-sublst (cadr ncomp))
       (tv:window-call (interactive-window :expose)
         (send interactive-window :home-cursor)
         (send interactive-window :refresh)
         (format interactive-window "~%  ATTRIBUTES OF THE COMPONENT /"T//L/"
              ~%T//L name = ~a ~%T//L number   = ~a ~%resistance(pu) = ~a
              ~%reactance(pu) = ~a ~%susceptance(pu)=~a ~%connected-bus-1= ~a
              ~%connected-bus-2 = ~a ~%tap-position= ~a ~%CB-1-status = ~a
              ~%CB-1-num = ~a ~%CB-2-status = ~a ~% CB-2-num = ~a ~%"
                 (first  comp-sublst)(second comp-sublst)(third  comp-sublst)
```

```
                (fourth comp-sublst)(nth 11 comp-sublst)(fifth  comp-sublst)
                (sixth comp-sublst))
          (seventh comp-sublst)(nth 7 comp-sublst)(nth 8 comp-sublst)
          (nth 9 comp-sublst)(nth 10 comp-sublst))
        (format interactive-window
                "~%ENTER NEW-ATTRIBUTES ~%ENTER /"-/" TO SKIP AN ATTRIBUTE~%
                  ~%T//L  name= ~a ~%T//L  number= ~a ~%resistance(pu)= ~a
                  ~%reactance(pu)= ~a ~%susceptance(pu)= ~a ~%connected-bus-1= ~a
                  ~%connected-bus-2= ~a ~%tap-position= ~a ~%CB-1-status= ~a
                  ~%CB-1-num= ~a ~%CB-2-status= ~a ~%CB-2-num= ~a ~%Capacity= ~a"
               (setq e1 (read interactive-window ))(setq e2 (read interactive-window ))
               (setq e3 (read interactive-window ))(setq e4 (read interactive-window ))
               (setq e12 (read interactive-window))(setq e5 (read interactive-window ))
               (setq e6 (read interactive-window ))(setq e7 (read interactive-window ))
               (setq e8 (read interactive-window ))(setq e9 (read interactive-window ))
               (setq e10 (read interactive-window))(setq e11 (read interactive-window))
               (cond ((not (equal e1 '-))(rplaca comp-sublst e1)))
               (cond ((not (equal e2 '-))(rplaca (cdr comp-sublst) e2 )))
               (cond ((not (equal e3 '-))(rplaca (cddr comp-sublst) e3)))
               (cond ((not (equal e4 '-))(rplaca (cdddr comp-sublst) e4)))
               (cond ((not (equal e5 '-))(rplaca (cddddr comp-sublst) e5)))
               (cond ((not (equal e6 '-))(rplaca (nthcdr 5 comp-sublst) e6)))
               (cond ((not (equal e7 '-))(rplaca (nthcdr 6 comp-sublst) e7)))
               (cond ((not (equal e8 '-))(rplaca (nthcdr 7 comp-sublst) e8)))
               (cond ((not (equal e9 '-))(rplaca (nthcdr 8 comp-sublst) e9)))
               (cond ((not (equal e10 '-))(rplaca (nthcdr 9 comp-sublst) e10)))
               (cond ((not (equal e11 '-))(rplaca (nthcdr 10 comp-sublst) e11)))
               (cond ((not (equal e12 '-))(rplaca (nthcdr 11 comp-sublst) e12)))
               (return))))


(defun blink-bus(posl)
  (tv:window-call (sensitive-window :expose)
    (prog(n)
         (setq n 0)
     label
        (setq n (+ n 1))
        (draw-lines (car posl)(cadr posl)(caddr posl)(cadddr posl) scal1)
        (cond ((and (evenp n) (greaterp tv:mouse-x 800)) (return))
              (t(go label))))))
(defun blink-bus-2(posl)
  (tv:window-call (sensitive-window :expose)
    (prog(n)
         (setq n 0)
     label
        (setq n (+ n 1))
        (draw-lines (car posl)(cadr posl)(caddr posl)(cadddr posl) scal1)
        (cond ((and (oddp n) (greaterp tv:mouse-x 800)) (return))
              (t(go label))))))

(defun blink-gen(posl)
  (tv:window-call (sensitive-window :expose)
    (prog(n)
         (setq n 0)
     label
        (setq n (+ n 1))
        (cond ((> (abs (- (car posl)(caddr posl)))
                  (abs (- (cadr posl)(cadddr posl))))
               (draw-circle (car posl)(cadr posl))
               (send sensitive-window :draw-line
                     (+ genx1 scal2) geny1 (- (caddr posl) 3) geny1
                     tv:alu-xor))
              (t(draw-circle (car posl)(cadr posl))
                (send sensitive-window :draw-line
                      genx1 (+ geny1 scal2)  genx1 (- (cadddr posl) 23)
                      tv:alu-xor)))
        (cond ((and (evenp n)(greaterp tv:mouse-x 800)) (return))
              (t(go label))))))

(defun blink-gen-2(posl)
  (tv:window-call (sensitive-window :expose)
    (prog(n)
         (setq n 0)
     label
        (setq n (+ n 1))
```

```
          (cond ((> (abs (- (car posl)(caddr posl)))
                    (abs (- (cadr posl)(cadddr posl))))
                 (draw-circle (car posl)(cadr posl))
                 (send sensitive-window :draw-line
                       (+ genxl scal2) genyl (- (caddr posl) 3) genyl
                       tv:alu-xor))
                (t(draw-circle (car posl)(cadr posl))
                  (send sensitive-window :draw-line
                        genxl (+ genyl scal2)  genxl (- (cadddr posl) 23)
                        tv:alu-xor)))
          (cond ((and (oddp n)(greaterp tv:mouse-x 800)) (return))
                (t(go label))))))

(defun blink-load(posl)
  (tv:window-call (sensitive-window :expose)
    (prog(n)
         (setq n 0)
     label
         (setq n (+ n 1))
         (draw-load  (car posl)(cadr posl)(caddr posl)(cadddr posl) )
         (cond ((and (evenp n)(greaterp tv:mouse-x 800)) (return))
               (t(go label))))))

(defun blink-load-2(posl)
  (tv:window-call (sensitive-window :expose)
    (prog(n)
         (setq n 0)
     label
         (setq n (+ n 1))
         (draw-load  (car posl)(cadr posl)(caddr posl)(cadddr posl) )
         (cond ((and (oddp n)(greaterp tv:mouse-x 800)) (return))
               (t(go label))))))

(defun blink-trans(posl)
  (tv:window-call (sensitive-window :expose)
    (prog(n)
         (setq n 0)
     label
         (setq n (+ n 1))
         (drawline posl)
         (cond ((and (evenp n)(greaterp tv:mouse-x 800)) (return))
               (t(go label))))))

(defun blink-trans-2(posl)
  (tv:window-call (sensitive-window :expose)
    (prog(n)
         (setq n 0)
     label
         (setq n (+ n 1))
         (drawline (reverse posl))
         (cond ((and (oddp n)(greaterp tv:mouse-x 800)) (return))
               (t(go label))))))
```

```
;;; -*- Mode: LISP; Syntax: Zetalisp; Package: USER; Base: 10; -*-
;;*************** FLAVORS FOR VARIOUS POWER SYSTEM COMPONENTS OR OBJECTS***************

(defvar *default-component-status* 'normal)
(defvar *default-nature-of-violation* 'none)

(defflavor basic-component(comp-type
                           position
                           number
                           (COMP-PERC-VIOL 0)
                           (COMP-VIOL  'none)
                           (COMPONENT-STATUS     *default-component-status*)
                           (NATURE-OF-VIOLATION *default-nature-of-violation* ))
          ()
  :gettable-instance-variables
  :settable-instance-variables
  :initable-instance-variables)

; Flavor for a bus

(defflavor component-bus(INIT-VOLT-PU
                         IMPROVED-VOLT-PU
                         voltage-pu
                         voltage-pu-up-limit
                         voltage-pu-low-limit
                         angle-degrees
                         type)
          (basic-component)
  :gettable-instance-variables
  :settable-instance-variables
  :initable-instance-variables)

; Defmethods for the component bus

(defmethod (component-bus :COMP-PERC-VIOL) ()
  (cond ((equal type 'L)
         (cond  ((or (equal nature-of-violation 'bus-volt-pu-up-limit-violated)
                     (equal nature-of-violation 'bus-voltage-may-exceed))
                 (* 100.0 (// (- voltage-pu voltage-pu-up-limit)
                              voltage-pu-up-limit)))
                ((or (equal nature-of-violation 'bus-volt-pu-low-limit-violated)
                     (equal nature-of-violation 'bus-voltage-may-fall))
                 (* 100.0 (// (- voltage-pu-low-limit voltage-pu)
                              voltage-pu-low-limit)))
                (t 0.0)))))


(defmethod (component-bus :COMP-VIOL) ()
  (cond ((equal type 'L)
         (cond  ((or (equal nature-of-violation 'bus-volt-pu-up-limit-violated)
                     (equal nature-of-violation 'bus-voltage-may-exceed))
                 (- voltage-pu voltage-pu-up-limit))
                ((or (equal nature-of-violation 'bus-volt-pu-low-limit-violated)
                     (equal nature-of-violation 'bus-voltage-may-fall))
                 (- voltage-pu-low-limit voltage-pu))
                (t 0.0)))))

; Flavor for generator

(defflavor component-generator  (rated-capacity-mw
                                 current-capacity-mw
                                 INIT-OUTPUT-MW
                                 IMPROVED-OUTPUT-MW
                                 current-output-mw
                                 INIT-OUTPUT-MVAR
                                 IMPROVED-OUTPUT-MVAR
                                 current-output-mvar
                                 connected-bus
                                 minimum-set-output-mw)
          (basic-component)
  :gettable-instance-variables
  :settable-instance-variables
  :initable-instance-variables)
```

```
;Defmethods for generator

(defmethod (component-generator :COMP-PERC-VIOL) ()
  (cond ((or (equal nature-of-violation 'generator-overloaded)
             (equal nature-of-violation 'generator-heavilyloaded))
         (* 100.0 (//(- current-output-mw current-capacity-mw)
                     current-capacity-mw)))
        ((equal nature-of-violation 'generator-underutilized)
         (* 100.0 (//(- minimum-set-output-mw current-output-mw)
                     minimum-set-output-mw)))))

; Flavor for transmission-line

(defflavor component-transmission-line(admitt-serial-pu
                                       admitt-parall-pu-half
                                       susceptance
                                       connected-bus
                                       ampacity
                                       INIT-PWRFLO-MW-BUS-1-2
                                       IMPROVED-PWRFLO-MW-BUS-1-2
                                       pwrflo-mw-bus-1-2
                                       INIT-PWRFLO-MW-BUS-2-1
                                       IMPROVED-PWRFLO-MW-BUS-2-1
                                       pwrflo-mw-bus-2-1
                                       INIT-PWRFLO-MVAR-BUS-1-2
                                       IMPROVED-PWRFLO-MVAR-BUS-1-2
                                       pwrflo-mvar-bus-1-2
                                       INIT-PWRFLO-MVAR-BUS-2-1
                                       IMPROVED-PWRFLO-MVAR-BUS-2-1
                                       pwrflo-mvar-bus-2-1
                                       cb-status cb-nums
                                       TRLN-CURRENT-FLOW)
          (basic-component)
  :gettable-instance-variables
  :settable-instance-variables
  :initable-instance-variables)

; defmethods for transmission-line

(defmethod (component-transmission-line :COMPONENT-STATUS) ()
  (cond ((equal cb-status '(open open)) 'TRIPPED)))

(defmethod (component-transmission-line :COMP-PERC-VIOL) ()
  (cond ((or (equal nature-of-violation 't//l-overloaded)
             (equal nature-of-violation 't//l-heavilyloaded))
         (* 100.0 (//(- trln-current-flow ampacity) ampacity)))))


(defflavor component-load (size-mw
                           size-mvar
                           connected-bus)
          (basic-component)
  :gettable-instance-variables
  :settable-instance-variables
  :initable-instance-variables)

(defmethod (component-transmission-line :COMP-PERC-VIOL) ()
  (cond ((equal nature-of-violation 'none)0.)
        (t 0.)))
```

```
;;;******************** OBJECT KNOWLEDGE BASE CREATION ********************

(defun create-object-knowledge-base (buslist genlist loadlist translist)
  (create-gen-kb genlist)
  (create-load-kb loadlist)
  (create-trans-kb translist)(create-bus-kb buslist))

(defun create-bus-kb (in-list)
  (setq bus-obj nil)
  (prog(l pos-list attr-list obj)
       (setq l in-list)
    label
       (cond ((null l)(return))
             ((setq pos-list (caar l) attr-list (cadar l)
                    obj (car attr-list) BUS-OBJ (CONS OBJ BUS-OBJ))
              (set obj (make-instance
                         'component-bus
                         :comp-type           'bus
                         :position            pos-list
                         :number              (second attr-list)
                         :INIT-VOLT-PU        (third attr-list)
                         :IMPROVED-VOLT-PU    (cons (third attr-list)nil)
                         :voltage-pu          (third attr-list)
                         :voltage-pu-up-limit  1.0
                         :voltage-pu-low-limit 0.75
                         :angle-degrees       (fourth attr-list)
                         :type                (fifth attr-list)))
              (setq l (cdr l))(go label)))))

(defun create-gen-kb (in-list)
  (setq gen-obj nil)
  (prog(l pos-list attr-list obj)
       (setq l in-list)
    label
       (cond ((null l)(return))
             ((setq pos-list (caar l) attr-list (cadar l)
                    obj (car attr-list) GEN-OBJ (CONS OBJ GEN-OBJ))
              (set obj (make-instance
                         'component-generator
                         :comp-type          'generator
                         :position            pos-list
                         :number              (second attr-list)
                         :rated-capacity-mw   (third attr-list)
                         :current-capacity-mw (fourth attr-list)
                         :INIT-OUTPUT-MW      (fifth attr-list)
                         :IMPROVED-OUTPUT-MW  (cons (fifth attr-list)nil)
                         :current-output-mw   (fifth attr-list)
                         :INIT-OUTPUT-MVAR    (sixth attr-list)
                         :IMPROVED-OUTPUT-MVAR(cons (sixth attr-list)nil)
                         :current-output-mvar (sixth attr-list)
                         :connected-bus       (seventh attr-list)
                         :minimum-set-output-mw 50.0))
              (setq l (cdr l))(go label)))))

(defun create-trans-kb (in-list)
  (setq trln-obj nil)
  (prog(l pos-list attr-list obj)
       (setq l in-list)
    label
       (cond ((null l)(return))
             ((setq pos-list (caar l) attr-list (cadar l)
                    obj (car attr-list) trln-obj (cons obj trln-obj))
              (set obj (make-instance
                         'component-transmission-line
                         :comp-type                          'T//LN
                         :position                           pos-list
                         :number                             (second attr-list)
                         :admitt-serial-pu                   (third attr-list)
                         :admitt-parall-pu-half              (fourth attr-list)
                         :susceptance                        (nth 11  attr-list)
                         :ampacity                           120.0       ;(nth 12 attr-list)
                         :INIT-PWRFLO-MW-BUS-1-2             (nth 13 attr-list)
                         :pwrflo-mw-bus-1-2                  (nth 13 attr-list)
                         :IMPROVED-PWRFLO-MW-BUS-1-2         (cons (nth 13 attr-list) nil)
                         :INIT-PWRFLO-MW-BUS-2-1             (nth 15 attr-list)
```

```
                        :IMPROVED-PWRFLO-MW-BUS-2-1       (cons (nth 15 attr-list) nil)
                        :pwrflo-mw-bus-2-1                (nth 15 attr-list)
                        :INIT-PWRFLO-MVAR-BUS-1-2         (nth 14 attr-list)
                        :IMPROVED-PWRFLO-MVAR-BUS-1-2     (cons (nth 14 attr-list) nil)
                        :pwrflo-mvar-bus-1-2              (nth 14 attr-list)
                        :INIT-PWRFLO-MVAR-BUS-2-1          (nth 16 attr-list)
                        :IMPROVED-PWRFLO-MVAR-BUS-2-1     (cons (nth 16 attr-list) nil)
                        :pwrflo-mvar-bus-2-1              (nth 16 attr-list)
                        :connected-bus  (list (fifth attr-list)(sixth attr-list))
                        :cb-status  (list (nth 7 attr-list)(nth 9 attr-list))
                        :cb-nums (list (nth 8 attr-list)(nth 10 attr-list))
                        :TRLN-CURRENT-FLOW 100.0))
              (setq l (cdr l))(go label)))))

(defun create-load-kb (in-list)
  (setq load-obj nil)
  (prog(l pos-list attr-list obj)
       (setq l in-list)
    label
       (cond ((null l)(return))
             ((setq pos-list (caar l) attr-list (cadar l)
                    obj (car attr-list) load-obj (cons obj load-obj))
              (set obj (make-instance
                         'component-load
                         :comp-type              'load
                         :position               pos-list
                         :number                 (second attr-list)
                         :size-mw                (third attr-list)
                         :size-mvar              (fourth attr-list)
                         :connected-bus          (fifth attr-list)))
              (setq l (cdr l))(go label)))))
```

```
;;*****************************************************************
;;  MACHINE LEARNING SUBSYSTEM  FRAMES  AND  MODULES
;;*****************************************************************

;;*****************      OBSERVATION FRAME *************************

(defflavor  OBSERVATION ((time-slice 0)
                         context
                         obj
                         obj-type
                         (parm-obj nil)
                         (val-obj nil))
            ()
  :gettable-instance-variables
  :settable-instance-variables
  :initable-instance-variables)

;;*****************      ABSTRACTION/INFERENCE  FRAME *****************

(defflavor ABSTRACTION ((time-slice 0)
                        obj
                        (occured-violations nil)
                        (frequent-violations nil)
                        (incorrectible-violations nil)
                        (infered-step-size nil)
                        (prediction nil))
           ()
  :gettable-instance-variables
  :settable-instance-variables
  :initable-instance-variables)

(setq AB (make-instance 'ABSTRACTION))

;;  ****************   BLACKBOARD STRUCTURE LEVELS     ****************

(DEFFLAVOR COMPLAINT-BLKBOARD((VIOLATIONS NIL)
                              (MOST-SEVERE-VIOLATION 'NONE)
                              (SYSTEM-STATUS 'NORMAL)
                              (LEVEL 'PROBLEM-SELECT))
           ()
  :GETTABLE-INSTANCE-VARIABLES
  :SETTABLE-INSTANCE-VARIABLES
  :INITABLE-INSTANCE-VARIABLES)

(DEFFLAVOR PLANNER-BLKBOARD((STEP-SIZE 0)
                            (CONTROL-GEN-NOW NIL)
                            (LEVEL 'PROBLEM-PLAN))
           ()
  :GETTABLE-INSTANCE-VARIABLES
  :SETTABLE-INSTANCE-VARIABLES
  :INITABLE-INSTANCE-VARIABLES)

(DEFFLAVOR EXEC-EVAL-BLKBOARD((VIOLATIONS NIL)
                              (MOST-SEVERE-VIOLATION NIL)
                              (SYSTEM-STATUS 'NORMAL)
                              (LEVEL 'EVALUATION))
           ()
  :GETTABLE-INSTANCE-VARIABLES
  :SETTABLE-INSTANCE-VARIABLES
  :INITABLE-INSTANCE-VARIABLES)

(DEFUN INITIALIZE()
  (SETQ       N1 (MAKE-INSTANCE 'COMPLAINT-BLKBOARD)
              N2 (MAKE-INSTANCE 'PLANNER-BLKBOARD)
              N3 (MAKE-INSTANCE 'EXEC-EVAL-BLKBOARD)))

;; ***************** GLOBAL-VARIABLES (initial values) ****************

(SETQ V-BUS-GEN '(G1 G2 G2 G5 G6 G7 G8 G9 G10)
      SWING-GEN 'G4
      SWING-BUS 'B39
      CONTROL-GENERATORS NIL
      SYSTEM-STATUS 'NORMAL
      RECOMMENDATION 'DO-POWER-FLOW-ANALYSIS
      CONTROL-LEVEL 2
```

```
        ATTEMPT 'NEW
        ATTEMPTED-CONTROL 'FALSE
        ATTEMPTED-COMPS NIL
        MACHINE-LEARNING NIL)


  ;;***FUNCTION TO GET OBSERVATIONS DURING THE PROBLEM-SOLVING BY THE EXPERT SYSTEM***

  (setq obsvn-list nil time-slice 0)

  (defun EXTRACT-OBSERVATION (context)
    (prog(V)
         (COND ((EQUAL RECOMMENDATION 'OVER)(RETURN)))
         (setq V (SEND N1 :MOST-SEVERE-VIOLATION))
         (cond ((equal context 'KS1)
                (setq frame 'OB1)
                (set frame (make-instance
                            'OBSERVATION
                            :parm-obj (list 'voltage-pu 'viol-type 'perc-viol 'status)
                            :val-obj (list
                                      (funcall (eval V) :voltage-pu)
                                      (funcall (eval V) :nature-of-violation)
                                      (funcall (eval V) :comp-perc-viol)
                                      (funcall (eval V) :component-status)))))
               ((equal context 'KS2)
                (setq frame 'OB2)
                (set frame (make-instance 'OBSERVATION
                                          :parm-obj (list 'step-size 'control-gen)
                                          :val-obj (list
                                                    (SEND N2 :STEP-SIZE)
                                                    (SEND N2 :CONTROL-GEN-NOW)))))
               ((equal context 'KS3)
                (setq frame 'OB3)
                (set frame (make-instance 'OBSERVATION
                                          :parm-obj (list 'new-voltage-pu 'mvar-control-gen)
                                          :val-obj (list
                                                    new-voltage-pu
                                                    (funcall (eval
                                                              (cadr
                                                                (SEND N2 :CONTROL-GEN-NOW)))
                                                             :current-output-mvar)))))
               ((equal context 'H-EVAL)
                (setq frame 'OB4)
                (set frame (make-instance 'OBSERVATION
                                          :parm-obj (list 'affect 'new-viols 'prediction)
                                          :val-obj (list
                                                    effect-flag
                                                    (SEND N3 :VIOLATIONS)
                                                    future-flag)))))
         (funcall (eval frame)      :set-time-slice time-slice)
         (funcall (eval frame)      :set-context context)
         (funcall (eval frame)      :set-obj V)
         (funcall (eval frame)      :set-obj-type (funcall (eval  V) :comp-type))
         (cond ((equal context 'H-EVAL)(:= time-slice (+ 1 time-slice))))
         (setq obsvn-list (cons frame obsvn-list))))


 (defun describe-observations()
   (mapcar 'describe (reverse obsvn-list)))


 ;;;;**************** INFERNTIAL OPERATORS FOR MACHINE-LEARNING PROCESS ***************

 (defun INFERENCE-ENGINE ()
   (INFERENTIAL-OPERATOR-1)
   (INFERENTIAL-OPERATOR-2)
   (INFERENTIAL-OPERATOR-3))

 ;; Function below infers: OCCURED-VIOLATIONS, FREQUENT-VIOLATIONS, INCORRECTIBLE-VIOLATIONS.

 (defun INFERENTIAL-OPERATOR-1()
   (prog(x y lv z)
        (setq x (funcall OB1 :obj) y (funcall AB :occured-violations)lv
              (funcall OB1 :time-slice))
        (funcall AB :set-obj x)
        (setq z (list  x lv))
```

```
        (cond ((and (assoc (car z) y)(not(equal (cadr (assoc (car z) y)) lv)))
               (funcall AB :set-frequent-violations (cons
                                                     x
                                                     (funcall AB :frequent-violations))))
              ((not(member z y))(funcall AB :set-occured-violations (cons (list x lv)y))))))


;; Function below infers: CONTROL STEP-SIZE  based on the most severe violation in the
;; postcontrol state of the power system.

(setq top-viol-status  nil)


(defun INFERENTIAL-OPERATOR-2()

  ;; i be the top-viol  being corrected
  ;; j be the  new top-viol that occured as a result of correcting viol i
  ;; k be the voltage bus to which the control generator is connected to.

  (funcall AB :set-infered-step-size
           (prog(i j k i-num j-num k-num new-viols-list dVi-by-dQk dVj-by-dQk
                 dQk-i dQk-j dVi dVj dQ-final infered-step-size)
                (setq i                (funcall OB1 :obj)
                      i-num            (funcall (eval i) :number)
                      k-num            (caddr (cadr (funcall OB2 :val-obj)))
                      new-viols-list   (cadr (funcall OB4 :val-obj))
                      dVi              (car (funcall OB2 :val-obj)))

                ;;dVi above is the step-size at this stage

                (cond ((list new-viols-list)
                       (priority-finder new-viols-list priority-rules)
                       (setq j           top-priority-comp
                             j-num       (funcall (eval j) :number)
                             dVi-by-dQk  (abs (aref sensitivity-matrix-jl i-num k-num))
                             dVj-by-dQk  (abs (aref sensitivity-matrix-jl j-num k-num))
                             dVj         (abs (funcall (eval j) :comp-viol)))
                       (setq top-viol-status (funcall (eval j) :component-status))
                       (setq dQk-i       (// dVi  dVi-by-dQk)
                             ;; dQk-i  gives the reactive power of k to be changed
                             ;; to affect voltage of bus i

                             dQk-j       (// dVj  dVj-by-dQk)
                             ;; dQk-j  gives the reactive power of k to be changed
                             ;; to affect voltage of bus j

                             dQ-final    (- dQk-i dQk-j)
                             ;; dQ-final gives the net reactive power change desirable
                             ;; to infer the step-size for i

                             infered-step-size  (* dQ-final dVi-by-dQk))
                       (setq top-priority-comp i)
                       (return (cons (list i j infered-step-size)
                                     (funcall AB :infered-step-size)))))))))


;; This operator infers:  CORRECTIBILITY of the most severe violation in question
;; based on the past and present experience and the present control measures
;; available with the power-system

(defun INFERENTIAL-OPERATOR-3()
  (prog(obj wf1 wf2 wf3 wf4 1 wf x)
       (setq obj (funcall AB :obj) wf1 3 wf2 3 wf3 3 wf4 3
             1  (cadr (funcall OB4 :val-obj)))
       (cond ((member obj (funcall AB :frequent-violations))
              (setq wf1 1))
             ((> 75 (* 100 (// (caddr(car(funcall AB :infered-step-size)))
                               (car (funcall OB2 :val-obj)))))
              (setq wf2 1))
             ((or (equal top-viol-status  'abnormal)
                  (equal top-priority-comp 'abnormal))
              (setq wf3 1))
             ((or (null (cdr control-generators))
                  (zerop (caadr control-generators)))
              (setq wf4 1)))
       (setq wf (+ wf1 wf2 wf3 wf4))
```

```
        (cond ((> wf 10)(setq x 'good))
              ((> wf 8 )(setq x 'fair))
              ((> wf 5 )(setq x 'low))
              (t(setq x 'incorrectible)
                (funcall AB :set-incorrectible-violations
                         (cons obj
                               (funcall AB :incorrectible-violations)))))
        (funcall AB :set-prediction (cons x (funcall AB :prediction)))))


;; PATH-CONSULTANT FUNCTION
;; improves decisions made by the  problem-solving  knowledge-sources

(DEFUN PATH-CONSULTANT (CONTEXT)
  (PROG(X SS ISS VAR)
       (SETQ X (SEND N1 :MOST-SEVERE-VIOLATION)
             TOP-PRIORITY-COMP X
             SS (SEND N2 :STEP-SIZE))
       (COND ((EQUAL CONTEXT 'KS1)
              (COND ((AND (MEMBER X (FUNCALL AB :INCORRECTIBLE-VIOLATIONS))
                          (MEMBER X (FUNCALL AB :FREQUENT-VIOLATIONS)))
                     (TRY-NEXT-VIOLATION X))
                    ((MEMBER X (FUNCALL AB :FREQUENT-VIOLATIONS))
                     (INFORM-AND-ASK-USER))))
             ((EQUAL CONTEXT 'KS2)
              (COND((NULL (SEND AB :INFERED-STEP-SIZE))(RETURN))
                   (T(SETQ ISS (CADDR (CAR (SEND AB :INFERED-STEP-SIZE))))
                     (COND((EQUAL (CAAR (SEND AB :INFERED-STEP-SIZE)) X)
                           (COND((> ISS SS) (SEND N2 :SET-STEP-SIZE
                                                     (// (+ ISS SS) 2)))
                                ((< ISS SS) (SEND N2 :SET-STEP-SIZE
                                                     ISS)))))))))))
```

```
;;****************          RULES  KNOWLEDGE BASE          ****************

;;; COMPONENT-VIOLATIONS DEFINING RULES

(SETQ COMP-VIOLATION-RULES
      '(

        ;;;*****GENERATOR RULES ********
        (VRULE1  IF
                 (AND (EQUAL COMP-TYPE 'GENERATOR)
                      (OR(> CURRENT-OUTPUT-MW     CURRENT-CAPACITY-MW)
                         (EQUAL CURRENT-OUTPUT-MW     CURRENT-CAPACITY-MW)))

                 THEN
                 ((SETQ  COMPONENT-STATUS 'ABNORMAL)
                  (SETQ NATURE-OF-VIOLATION 'GENERATOR-OVERLOADED)))

        (VRULE2  IF
                 (AND (EQUAL COMP-TYPE 'GENERATOR)
                      (> CURRENT-OUTPUT-MW   (* .85 CURRENT-CAPACITY-MW))
                      (< CURRENT-OUTPUT-MW     CURRENT-CAPACITY-MW))

                 THEN
                 ((SETQ COMPONENT-STATUS 'ALERT)
                  (SETQ NATURE-OF-VIOLATION 'GENERATOR-HEAVILYLOADED)))

        (VRULE3  IF
                 (AND (EQUAL COMP-TYPE 'GENERATOR)
                      (< CURRENT-OUTPUT-MW     MINIMUM-SET-OUTPUT-MW))

                 THEN
                 ((SETQ COMPONENT-STATUS 'ALERT)
                  (SETQ NATURE-OF-VIOLATION 'GENERATOR-UNDERUTILIZED)))


       ;;;;********BUS-RULES********
       (VRULE4  IF
                (AND (EQUAL COMP-TYPE 'BUS)
                     (EQUAL TYPE 'L)
                     (OR (> VOLTAGE-PU (* 1.1 VOLT-PU-UP-LIMIT))
                         (EQUAL VOLTAGE-PU (* 1.1 VOLT-PU-UP-LIMIT))))

                THEN
                ((SETQ COMPONENT-STATUS 'ABNORMAL)
                 (SETQ NATURE-OF-VIOLATION 'BUS-VOLT-PU-UP-LIMIT-VIOLATED)))

       (VRULE5  IF
                (AND (EQUAL COMP-TYPE 'BUS)
                     (EQUAL TYPE 'L)
                     (> VOLTAGE-PU (* 1.05 VOLT-PU-UP-LIMIT))
                     (< VOLTAGE-PU (* 1.1 VOLT-PU-UP-LIMIT)))

                THEN
                ((SETQ COMPONENT-STATUS 'ALERT)
                 (SETQ NATURE-OF-VIOLATION 'BUS-VOLTAGE-MAY-EXCEED)))


       (VRULE6  IF
                (AND (EQUAL COMP-TYPE 'BUS)
                     (EQUAL TYPE 'L)
                     (OR(< VOLTAGE-PU (* 0.90 VOLT-PU-LOW-LIMIT))
                        (EQUAL VOLTAGE-PU (* 0.90 VOLT-PU-LOW-LIMIT))))

                THEN
                ((SETQ COMPONENT-STATUS 'ABNORMAL)
                 (SETQ NATURE-OF-VIOLATION 'BUS-VOLT-PU-LOW-LIMIT-VIOLATED)))


       (VRULE7  IF
                (AND (EQUAL COMP-TYPE 'BUS)
                     (EQUAL TYPE 'L)
                     (< VOLTAGE-PU (* 0.95 VOLT-PU-LOW-LIMIT))
                     (> VOLTAGE-PU (* 0.90 VOLT-PU-LOW-LIMIT)))
```

```
          THEN
          ((SETQ COMPONENT-STATUS 'ALERT)
           (SETQ NATURE-OF-VIOLATION 'BUS-VOLTAGE-MAY-FALL)))


;;;********T//L RULES********
(VRULE8   IF
          (AND (EQUAL COMP-TYPE 'T//LN)
               (OR (> TRLN-CURRENT-FLOW AMPACITY)
                   (EQUAL TRLN-CURRENT-FLOW AMPACITY)))

          THEN
          ((SETQ COMPONENT-STATUS 'ABNORMAL)
           (SETQ NATURE-OF-VIOLATION 'T//L-OVERLOADED)))


(VRULE9   IF
          (AND (EQUAL COMP-TYPE 'T//LN)
               (> TRLN-CURRENT-FLOW (* .85 AMPACITY))
               (< TRLN-CURRENT-FLOW  AMPACITY))

          THEN
          ((SETQ COMPONENT-STATUS 'ALERT)
           (SETQ NATURE-OF-VIOLATION 'T//L-HEAVILYLOADED)))

))
```

```
;;*************************************************************************
;;THIS FUNCTION DOES INITIAL-SYSTEM-EVALUATION . IT USES FUNCTION SYSTEM-EVALUATOR.
;;IF SYSTEM IS IN ALERT OR EMERGENCY STATE IT ASKS USER IF HE WANTS EXPERT-CONTROL
;;TO BRING POWER-SYSTEM IN NORMAL-STATE. IF YES THEN IT TRIGGERS FUNCTION EXPERT.
;;*************************************************************************

(DEFUN INIT-SYSTEM-EVALUATION()
  (INITIALIZE)
  (PROG(reply)
       (TV:WINDOW-CALL (INTERACTIVE-WINDOW :EXPOSE)
         (SEND INTERACTIVE-WINDOW :HOME-CURSOR)
         (SEND INTERACTIVE-WINDOW :REFRESH)
         (COND ((OR(NULL BUS-OBJ)(NULL GEN-OBJ)(NULL TRLN-OBJ))
                (SEND INTERACTIVE-WINDOW :LINE-OUT
                      "PLEASE LOAD POWER-SYSTEM BEFORE EVALUATION")
                (RETURN))
               (T(SYSTEM-EVALUATION)
                 (SEND N1 :SET-SYSTEM-STATUS SYSTEM-STATUS)
                 (send INTERACTIVE-WINDOW  :line-out
                       "POWER SYSTEM ON EVALUATION IS FOUND IN FOLLOWING STATE:")
                 (SEND INTERACTIVE-WINDOW :LINE-OUT (SEND N1 :SYSTEM-STATUS))
                 (find-tripped-lines trln-obj)
                 (FUNCALL N1 :SET-VIOLATIONS COMPLAINT-BLKBOARD)
                 (cond ((not(null tripped-lines ))
                        (SEND INTERACTIVE-WINDOW :LINE-OUT
                              "   THESE T//L ARE IN TRIPPED STATE ")
                        (format INTERACTIVE-WINDOW  "~% ~A ~% "  TRIPPED-LINES)))
                 (COND ((EQUAL (SEND N1 :VIOLATIONS) NIL)
                        (FORMAT INTERACTIVE-WINDOW
                                "~% HENCE NO EXPERT CONTROL IS NEEDED ~%"))
                       (T(SEND INTERACTIVE-WINDOW :LINE-OUT
                               "PARAMETERS OF FOLLOWING COMPONENTS ARE VIOLATED")
                         (FORMAT INTERACTIVE-WINDOW "~%  ~A ~% "
                                 (SEND N1 :VIOLATIONS))
                         (DISPLAY-VIOLATIONS (SEND N1 :VIOLATIONS))
                         (SEND INTERACTIVE-WINDOW :LINE-OUT
                               "WANT TO CORRECT THE VIOLATIONS ? ")
                         (SETQ reply (read interactive-window))
                         (cond((MEMBER REPLY '( t T y Y YES YEAH))
                               (report "***EXPERT-MODULE IS TRIGGERED***")
                               (SEND INTERACTIVE-WINDOW :LINE-OUT
                                     "EXPERT-SYSTEM IS ATTEMPTING TO")
                               (SEND INTERACTIVE-WINDOW :LINE-OUT
                                     "CORRECT THE VIOLATIONS")
                               (REPORT "***WANT MACHINE-LEARNING CAPABILITY*** ?")
                               (setq reply (read interactive-window))
                               (cond ((member reply '(t T y Y yes YEAH))
                                      (SETQ MACHINE-LEARNING 'T))
                                     (T(SETQ MACHINE-LEARNING NIL)))
                               (EXPERT FLOW-CONTROL-RULES)))))))))

;;*************************************************************
;;THIS IS TOP SYSTEM-EVALUATION FUNCTION
;;USES FUNCTIONS  VIOLATIONS-FINDER and SYSTEM-STATUS-EVAL.
;;*************************************************************

(DEFUN SYSTEM-EVALUATION ()
  (SETQ COMPLAINT-BLKBOARD NIL)
  (FIND-FINAL-VIOLATIONS BUS-OBJ GEN-OBJ TRLN-OBJ)
  (SYSTEM-STATUS-EVAL  SYSTEM-EVAL-RULES)
  (COND ((NOT(NULL COMPLAINT-BLKBOARD))
         (SETQ RECOMMENDATION 'DEFINE-PROBLEM-COMPONENT))
        (T(SETQ RECOMMENDATION 'OVER))))

;***** FUNCTION BELOW  FINDS ALL THE POWER SYSTEM VIOLATIONS *****
;Note that arguments have been passed as variables
;and thus need to be evaluated as shown in various functions below
;bus-obj gen-obj trln-obj are lists containing objects or component names
;for bus, generator, t/l respectively.

(DEFUN VIOLATIONS-FINDER(BUS-OBJ GEN-OBJ TRLN-OBJ)
  (SETQ COMPLAINT-BLKBOARD NIL )
  (PROG(BS GN TR)
       (SETQ BS BUS-OBJ GN GEN-OBJ TR TRLN-OBJ)
   LABEL1
```

```
        (COND ((AND (NULL BS)(NULL GN)(NULL TR))(RETURN))
              ((NOT (NULL BS))
               (SETQ COMP-NAME (CAR BS))
               (COMP-VIOLATION-CHECKS COMP-VIOLATION-RULES COMP-NAME)
               (SETQ BS (CDR BS)))
              ((NOT (NULL GN))
               (SETQ COMP-NAME (CAR GN))
               (COMP-VIOLATION-CHECKS COMP-VIOLATION-RULES COMP-NAME)
               (SETQ GN (CDR GN)))
              ((NOT (NULL TR))
               (SETQ COMP-NAME (CAR TR))
               (COMP-VIOLATION-CHECKS COMP-VIOLATION-RULES COMP-NAME)
               (SETQ TR (CDR TR))))
        (GO LABEL1)))


(DEFUN COMP-VIOLATION-CHECKS(COMP-VIOLATION-RULES COMP-NAME )
                                                ;comp-name is passed as variable
  (FILL-SLOTS COMP-NAME)
  (PROG(L)
       (SETQ L COMP-VIOLATION-RULES)
    LABEL
       (SETQ COMPONENT-STATUS 'NORMAL NATURE-OF-VIOLATION 'NONE)
       (COND ((NULL L)(RETURN))
             ((TEST-RULE (CAR L))
              (FUNCALL (EVAL COMP-NAME) :SET-COMPONENT-STATUS COMPONENT-STATUS)
              (FUNCALL (EVAL COMP-NAME) :SET-NATURE-OF-VIOLATION  NATURE-OF-VIOLATION)
              (SETQ COMPLAINT-BLKBOARD (CONS COMP-NAME COMPLAINT-BLKBOARD))))
       (SETQ L (CDR L))
       (GO LABEL)))

(DEFUN FIND-FINAL-VIOLATIONS(BUS-OBJ GEN-OBJ TRLN-OBJ)
  (SETQ COMPLAINT-BLKBOARD NIL )
  (PROG(BS GN TR)
       (SETQ BS BUS-OBJ GN GEN-OBJ TR TRLN-OBJ)
    LABEL1
       (COND ((AND (NULL BS)(NULL GN)(NULL TR))(RETURN))
             ((NOT (NULL BS))
              (SETQ COMP-NAME (CAR BS))
              (FUNCALL (EVAL COMP-NAME) :SET-COMPONENT-STATUS 'NORMAL )
              (FUNCALL (EVAL COMP-NAME) :SET-NATURE-OF-VIOLATION 'NONE )
              (COMP-VIOLATION-CHECKS COMP-VIOLATION-RULES COMP-NAME)
              (SETQ BS (CDR BS)))
             ((NOT (NULL GN))
              (SETQ COMP-NAME (CAR GN))
              (FUNCALL (EVAL COMP-NAME) :SET-COMPONENT-STATUS 'NORMAL)
              (FUNCALL (EVAL COMP-NAME) :SET-NATURE-OF-VIOLATION  'NONE)
              (COMP-VIOLATION-CHECKS COMP-VIOLATION-RULES COMP-NAME)
              (SETQ GN (CDR GN)))
             ((NOT (NULL TR))
              (SETQ COMP-NAME (CAR TR))
              (FUNCALL (EVAL COMP-NAME) :SET-COMPONENT-STATUS 'NORMAL)
              (FUNCALL (EVAL COMP-NAME) :SET-NATURE-OF-VIOLATION 'NONE)
              (COMP-VIOLATION-CHECKS COMP-VIOLATION-RULES COMP-NAME)
              (SETQ TR (CDR TR))
             ))
      (GO LABEL1)))

;;;*********************************************************
;;;VIOLATION  SELECTION  FUNCTIONS
;;;*********************************************************
;;This function along with trip-trln2 is used to change the
;;status of a transmission line as tripped and open ckt-breakers

(defun TRIP-TRLN1 ()
  (tv:window-call (interactive-window :expose)
    (prog(l rd)
      loop1
         (send interactive-window :home-cursor)
         (send interactive-window :refresh)
         (send interactive-window :line-out "  Please enter a valid T//L name")
         (setq rd (read interactive-window))
         (cond ((member rd trln-obj)(setq l (cons rd l)))
               (t (go loop1)))
         (send interactive-window :line-out "ANY MORE ?")
         (cond ((member (read interactive-window) '(yes Y))(go loop1))
```

```
                (t(return 1))))))

(defun TRIP-TRLN2 (ls)
  (for (i in ls)
       (do (funcall (eval i) :set-cb-status '(open open)))))

;;Function to find all tripped t//l lines and display their names
;;on interactive window

(DEFUN FIND-TRIPPED-LINES(ls)
  (setq tripped-lines nil)
  (for (i in ls)
       (do (cond ((equal (funcall (eval i) :component-status )'tripped)
                  (setq tripped-lines (cons i tripped-lines)))))))

;Function below tests each rule and triggers then part of applicable rules.

(DEFUN TEST-RULE(RULE)
(COND ((EVAL(CADDR RULE))(MAPCAR 'EVAL (CAR (CDDDDR  RULE)) ))))

; Function below gives  variables  the slot-values from the knowledge base

(DEFUN FILL-SLOTS ( COMP-NAME)
  (SETQ COMP-TYPE (FUNCALL (EVAL COMP-NAME) :COMP-TYPE))
  (COND((EQUAL  COMP-TYPE 'GENERATOR)
        (SETQ CURRENT-OUTPUT-MW (FUNCALL (EVAL COMP-NAME) :CURRENT-OUTPUT-MW)
              CURRENT-CAPACITY-MW (FUNCALL (EVAL COMP-NAME) :CURRENT-CAPACITY-MW)
              MINIMUM-SET-OUTPUT-MW (FUNCALL (EVAL COMP-NAME) :MINIMUM-SET-OUTPUT-MW)))
       ((EQUAL  COMP-TYPE 'BUS)
        (SETQ TYPE (FUNCALL (EVAL COMP-NAME) :TYPE)
              VOLTAGE-PU (FUNCALL (EVAL COMP-NAME) :VOLTAGE-PU)
              VOLT-PU-UP-LIMIT (FUNCALL (EVAL COMP-NAME) :VOLTAGE-PU-UP-LIMIT)
              VOLT-PU-LOW-LIMIT (FUNCALL (EVAL COMP-NAME) :VOLTAGE-PU-LOW-LIMIT)))
       ((EQUAL COMP-TYPE 'T//LN)
        (SETQ TRLN-CURRENT-FLOW (FUNCALL (EVAL COMP-NAME) :TRLN-CURRENT-FLOW)
              AMPACITY  (FUNCALL (EVAL COMP-NAME) :AMPACITY)))))
```

```
;********** SYSTEM EVALUATION RULES ***************

(setq SYSTEM-EVAL-RULES
      '(
        (ERULE1 IF
                (NULL COMPLAINT-BLKBOARD)

                THEN
                ((SETQ SYSTEM-STATUS 'NORMAL-STATE)))

        (ERULE2 IF
                (AND (NOT (NULL COMPLAINT-BLKBOARD))
                     (ANY-COMP-IN  COMPLAINT-BLKBOARD 'ABNORMAL))

                THEN
                ((SETQ SYSTEM-STATUS 'EMERGENCY-STATE)))

        (ERULE3 IF
                (AND (NOT (NULL COMPLAINT-BLKBOARD))
                     (NOT(ANY-COMP-IN  COMPLAINT-BLKBOARD 'ABNORMAL))
                     (ANY-COMP-IN  COMPLAINT-BLKBOARD 'ALERT))

                THEN
                ((SETQ SYSTEM-STATUS 'ALERT-STATE)))))


;;Function below evaluates the system and informs whether system is in
;; normal  alert or  emergency state. It uses  system evaluation rules
;;and list of components written on complaint-blackboard.

(DEFUN SYSTEM-STATUS-EVAL (RULES )
  (SETQ SYSTEM-STATUS NIL)
  (PROG(L)
       (SETQ L RULES)
    LABEL1
       (COND((NULL L)(RETURN))
            (T(TEST-RULE  (CAR L))(SETQ L (CDR L))(GO LABEL1)))))

;; This function finds if any component in complaint-blackboard is in the given state.

(defun ANY-COMP-IN (blkboard state)
  (prog (l)
        (setq l blkboard )
    label
       (cond ((and(null l))(return nil))
             ((equal (funcall (eval (car l)) :component-status) STATE)(return t)))
       (setq l (cdr l))(go label)))
```

```
;;************** VIOLATIONS  PRIORITY RULES  ****************
;;These rules define a policy to decide the violation to be attended first.

(setq PRIORITY-RULES
      '(

        (PRULE1         IF
                        (EQUAL GIVEN-COMP-STAT 'NORMAL)

                        THEN
                        ((SETQ COMP-PRIORITY 'NIL SW 'ON)))

        (PRULE2         IF
                        (AND (EQUAL GIVEN-COMP-STAT 'ABNORMAL)
                             (EQUAL ASSUM-COMP-STAT 'ALERT))

                        THEN
                        ((SETQ COMP-PRIORITY 'TOP)
                         (SETQ ASSUM-COMP GIVEN-COMP)))

        (PRULE3         IF
                        (AND (EQUAL GIVEN-COMP-STAT ASSUM-COMP-STAT)
                             (EQUAL GIVEN-COMP-TYPE 'GENERATOR)
                             (OR (EQUAL ASSUM-COMP-TYPE 'BUS)
                                 (EQUAL ASSUM-COMP-TYPE 'T//LN)))

                        THEN
                        ((SETQ COMP-PRIORITY 'TOP)
                         (SETQ ASSUM-COMP GIVEN-COMP)))

        (PRULE4         IF
                        (AND (EQUAL GIVEN-COMP-STAT ASSUM-COMP-STAT)
                             (EQUAL GIVEN-COMP-TYPE 'BUS)
                             (EQUAL ASSUM-COMP-TYPE 'T//LN))

                        THEN
                        ((SETQ COMP-PRIORITY 'TOP)
                         (SETQ ASSUM-COMP GIVEN-COMP)))

        (PRULE5         IF
                        (AND (EQUAL GIVEN-COMP-STAT ASSUM-COMP-STAT)
                             (EQUAL GIVEN-COMP-TYPE  ASSUM-COMP-TYPE)
                             (> GIVEN-COMP-PERC-VIOL ASSUM-COMP-PERC-VIOL))

                        THEN
                        ((SETQ COMP-PRIORITY 'TOP)
                         (SETQ ASSUM-COMP GIVEN-COMP)))))
```

```
;; top level function to find highest priority violation USING PRIORITY RULES

(DEFUN PRIORITY-FINDER (BLKBOARD RULES)
  (PROG ()
        (SETQ BL BLKBOARD TOP-PRIORITY-COMP NIL)
        (REPORT " THESE COMPONENTS OF POWER-SYSTEM ARE VIOLATED :")
        (MAPCAR '(LAMBDA(X)(REPORT X)) COMPLAINT-BLKBOARD)
        (COND ((NOT(NULL BL))
               (SETQ ASSUM-COMP (CAR BL)
                     ASSUM-COMP-STAT (FUNCALL (EVAL ASSUM-COMP) :COMPONENT-STATUS)
                     ASSUM-COMP-TYPE (FUNCALL (EVAL ASSUM-COMP) :COMP-TYPE)
                     ASSUM-COMP-PERC-VIOL (FUNCALL (EVAL ASSUM-COMP) :COMP-PERC-VIOL))))
     LABEL1
        (SETQ SW 'OFF  1 RULES)
        (COND ((NULL BL)
               (SETQ TOP-PRICRITY-COMP ASSUM-COMP)
               (REPORT "TOP-PRIORITY-COMP IS  : ")
               (REPORT  TOP-PRIORITY-COMP)
               (RETURN))
              ((NOT(NULL BL))
               (SETQ GIVEN-COMP (CAR BL)
                     BL (CDR BL)
                     GIVEN-COMP-STAT (FUNCALL (EVAL GIVEN-COMP) :COMPONENT-STATUS)
                     GIVEN-COMP-TYPE (FUNCALL (EVAL GIVEN-COMP) :COMP-TYPE)
                     GIVEN-COMP-PERC-VIOL (FUNCALL (EVAL GIVEN-COMP) :COMP-PERC-VIOL))))
     LABEL2
        (TEST-RULE (CAR 1))
        (COND ((AND (NULL 1)(EQUAL SW 'OFF))(GO LABEL1))
              ((EQUAL SW 'OFF)(SETQ L (CDR L))(GO LABEL2))
              ((EQUAL SW 'ON)(SETQ
                              ASSUM-COMP-STAT (FUNCALL (EVAL ASSUM-COMP) :COMPONENT-STATUS)
                              ASSUM-COMP-TYPE (FUNCALL (EVAL ASSUM-COMP) :COMP-TYPE)
                              ASSUM-COMP-PERC-VIOL (FUNCALL (EVAL ASSUM-COMP) :COMP-PERC-VIOL))
               (GO LABEL1)))))

;;*****************************************************
;;         DOMAIN      EXPERT
;;*****************************************************
;;THIS FUNCTION IS TOP-LEVEL FUNCTION
;;ACTING AS DOMAIN EXPERT USING FLOW-CONTROL-RULES

(DEFUN EXPERT (RULES)
  (TV:WINDOW-CALL (INTERACTIVE-WINDOW :EXPOSE)
    (WITH-OPEN-FILE (STR "REPORT"
                         ':DIRECTION
                         ':OUTPUT
                         ':CHARACTERS T)
      (SETQ  RECOMMENDATION 'DC-POWER-FLOW-ANALYSIS ATTEMPT
             'NEW ATTEMPTED-COMPS NIL)
      (prog()
        LABEL1
           (MAPCAR '(LAMBDA (X)(TEST-RULE X)) RULES)
           (COND ((EQUAL RECOMMENDATION 'OVER)(RETURN))
                 (T(GO LABEL1)))))))

;;*****************************************************
;;    MODULES OR KSs  CALLED BY THE FLOW CONTROL RULES
;;*****************************************************

(DEFUN DEFINE-TOP-VIOLATION()
  (SETQ L (SEND N1 :VIOLATIONS))
  (PRIORITY-FINDER L PRIORITY-RULES)
  (SEND N1 :SET-MOST-SEVERE-VIOLATION TOP-PRIORITY-COMP)
  (SETQ CONTROL-LEVEL 2)
  (SETQ RECOMMENDATION 'PLAN-STRATEGY))

(DEFUN DEFINE-NEW-TOP-VIOL()
  (SETQ COMPLAINT-BLKBOARD (SEND N1 :VIOLATIONS))
  (PROG(L)
       (SETQ L (COPY COMPLAINT-BLKBOARD))
       (MAPCAR '(LAMBDA (X)
                  (COND ((MEMBER X ATTEMPTED-COMPS)
                         (REMOVE X  COMPLAINT-BLKBOARD))))
               COMPLAINT-BLKBOARD)
       (COND ((NULL COMPLAINT-BLKBOARD)
```

```
                (SETQ COMPLAINT-BLKBOARD  L  RECOMMENDATION 'OVER))
               (T(PRIORITY-FINDER L PRIORITY-RULES)
                 (SETQ COMPLAINT-BLKBOARD L CONTROL-LEVEL 2
                       RECOMMENDATION 'PLAN-STRATEGY)))
         (SEND N1 :SET-MOST-SEVERE-VIOLATION TOP-PRIORITY-COMP)))


 (DEFUN PLAN-STRATEGY ()
   (SETQ TOP-PRIORITY-COMP (SEND N1 :MOST-SEVERE-VIOLATION))
   (cond ((EQUAL ATTEMPT 'NEW)
          (SETQ USED-GENS NIL CONTROL-LEVEL 2)
          (SETQ CONTROL-GENERATORS (FIND-SORT-GEN  TOP-PRIORITY-COMP ))
 ;find-sort-gen finds all the generators that can be used for
 ;controlling the voltage of the top-violation

          (SEND N2 :SET-CONTROL-GEN-NOW (CAR CONTROL-GENERATORS))
          (SETQ CONTROL-GENERATORS (CDR CONTROL-GENERATORS))))
   (SETQ EXTENT-OF-VIOLATION (FUNCALL (EVAL TOP-PRIORITY-COMP) :COMP-VIOL))
   (SETQ RECOMMENDATION NIL)
   (SETQ CONTROL-GEN-NOW (SEND N2 :CONTROL-GEN-NOW))
   (MAPCAR '(LAMBDA (RULE) (TEST-RULE RULE))  PLANNING-RULES)
   (SEND N2 :SET-STEP-SIZE STEP-SIZE)
   (SEND N2 :SET-CONTROL-GEN-NOW CONTROL-GEN-NOW)
   (SETQ ATTEMPT 'OLD)
   (COND ((NULL RECOMMENDATION)(SETQ RECOMMENDATION 'EXECUTE-STRATEGY)))
   (cond ((ZEROP (CAR (SEND N2 :CONTROL-GEN-NOW)))
          (SETQ CONTROL-GENERATORS NIL)))
   (cond ((null control-generators)
          (SETQ CONTROL-GEN-NOW (SEND N2 :CONTROL-GEN-NOW))
          (MAPCAR '(LAMBDA (RULE) (TEST-RULE RULE))  PLANNING-RULES))))


(DEFUN  SELECT-NEXT-MOST-SENS-GEN ()
   (SETQ USED-GENS (APPEND CONTROL-GEN-NOW USED-GENS))
   (SEND N2 :SET-CONTROL-GEN-NOW (CAR CONTROL-GENERATORS ))
   (SETQ CONTROL-GENERATORS  (CDR CONTROL-GENERATORS ))
   (cond((zerop (SEND N2 :car control-gen-now))
         (setq control-generators nil))))

(DEFUN REPORT(X)
   (FORMAT INTERACTIVE-WINDOW "~% ~a~%" X))


(DEFUN SUGGEST-GENERATION-CHANGES()
   (REPORT "GENERATION OF THE FOLLOWING GENERATORS NEEDS TO BE CHANGED")
   (MAPCAR '(LAMBDA(X)
              (COND ((NOT (EQUAL (FUNCALL (EVAL X) :INIT-OUTPUT-MVAR)
                                 (CAR (FUNCALL (EVAL X) :IMPROVED-OUTPUT-MVAR))))
                     (FORMAT INTERACTIVE-WINDOW
                             "~% GENERATOR = ~a ~% INITIAL-MVAR = ~a ~% RECOMMENDED-MVAR = ~a~%"
                             X
                             (FUNCALL (EVAL X)       :INIT-OUTPUT-MVAR)
                             (CAR (FUNCALL (EVAL X) :IMPROVED-OUTPUT-MVAR))))))
           GEN-OBJ))
```

```
;;*********************************************************************
;; KNOWLEDGE SOURCE SCHEDULING RULER
;;*********************************************************************

(SETQ FLOW-CONTROL-RULES
      '(
        (XRULE1          IF
                         (OR  (EQUAL RECOMMENDATION 'NEEDS-NO-FURTHER-ACTION)
                              (EQUAL RECOMMENDATION 'START-CONTROL)
                              (EQUAL RECOMMENDATION 'EVALUATE-SYSTEM))
                         THEN
                         ((REPORT "CONTROL-RULE-1 TRUE BY RECOMMENDATION: EVALUATE-SYSTEM.")
                          (REPORT "MODULE-TRIGGERED: SYSTEM-EVALUATION.")
                          (SYSTEM-EVALUATION)
                          (SEND N3 :SET-SYSTEM-STATUS SYSTEM-STATUS)))

        (XRULE2          IF
                         (EQUAL RECOMMENDATION 'DEFINE-PROBLEM-COMPONENT)
                         THEN
                         ((SETQ ATTEMPT 'NEW)
                          (REPORT "CONTROL-RULE-2 TRUE BY RECOMMENDATION: DEFINE-PROBLEM-COMPONENT.")
                          (REPORT "MODULE-TRIGGERED: DEFINE-TOP-VIOLATION.")
                          (DEFINE-TOP-VIOLATION )
                          (SEND N1 :SET-MOST-SEVERE-VIOLATION TOP-PRIORITY-COMP)
                          (COND  (MACHINE-LEARNING
                                  (PATH-CONSULTANT 'KS1)
                                  (EXTRACT-OBSERVATION 'KS1)))))

        (XRULE3          IF
                         (EQUAL RECOMMENDATION 'REDEFINE-PROBLEM-COMPONENT)
                         THEN
                         ((SETQ ATTEMPT 'NEW)
                          (REPORT "CONTROL-RULE-3 TRUE BY RECOMMENDATION:REDEFINE-PROBLEM-COMP.")
                          (REPORT "MODULE-TRIGGERED:DEFINE-NEW-TOP-VIOL.")
                          (DEFINE-NEW-TOP-VIOL)
                          (SEND N1 :SET-MOST-SEVERE-VIOLATION TOP-PRIORITY-COMP)
                          (COND(MACHINE-LEARNING
                                (PATH-CONSULTANT 'KS1)
                                (EXTRACT-OBSERVATION 'KS1)))))

        (XRULE4          IF
                         (OR  (EQUAL RECOMMENDATION 'PLAN-STRATEGY)
                              (EQUAL RECOMMENDATION 'REPLAN-STRATEGY))

                         THEN
                         ((REPORT "CONTROL-RULE-4 TRUE BY RECOMMENDATION:PLAN-STRATEGY.")
                          (REPORT "MODULE-TRIGGERED:PLAN-STRATEGY.")
                          (PLAN-STRATEGY)
                          (COND(MACHINE-LEARNING
                                (PATH-CONSULTANT 'KS2)
                                (EXTRACT-OBSERVATION 'KS2)))))

        (XRULE5          IF
                         (EQUAL RECOMMENDATION 'EXECUTE-STRATEGY)

                         THEN
                         ((REPORT "CONTROL-RULE-5 TRUE BY RECOMMENDATION:EXECUTE-STRATEGY.")
                          (REPORT "MODULE-TRIGGERED:EXECUTE-STRATEGY.")
                          (EXECUTE-STRATEGY)
                          (COND(MACHINE-LEARNING
                                (EXTRACT-OBSERVATION 'KS3)))))

        (XRULE6          IF
                         (EQUAL RECOMMENDATION 'GIVE-EXPLAINATION)

                         THEN
                         ((EXPLAIN-ACTION )))

        (XRULE7          IF
                         (EQUAL RECOMMENDATION 'DO-POWER-FLOW-ANALYSIS)

                         THEN
                         ((REPORT "CONTROL-RULE-7 TRUE BY RECOMMENDATION:DO-POWER-FLOW-ANALYSIS.")
                          (REPORT "MODULES-TRIGGERED:POWER-FLOW-ANALYSIS and SENSITIVITY-ANALYSIS.")
                          ;(SOLVE-POWER-FLOW-ANALYSIS)
```

```
                    (REPORT "SENSITIVITY-MATRIX CREATION IS REQUIRED.")
                    (CREATE-SENSITIVITY-MATRIX)))

(XRULE8             IF
                    (EQUAL RECOMMENDATION 'OVER)

                    THEN
                    ((FIND-FINAL-VIOLATIONS BUS-OBJ GEN-OBJ TRLN-OBJ)
                     (REPORT "CONTROL-RULE-8 TRUE BY RECOMMENDATION:OVER.")
                     (REPORT "FUNCTION-TRIGGERED:SUGGEST-GENERATION-CHANGES.")
                     (SUGGEST-GENERATION-CHANGES)
                     (REPORT "***EXPERT MODULE RETURNS***")))

))
```

```
;;********************  VIOLATION CORRECTION-PLANNING RULES ************************

(setq PLANNING-RULES
      '(
        (PLRULE1        IF
                        (NULL CONTROL-GENERATORS)

                        THEN
                        ((SETQ ATTEMPTED-COMPS (CONS TOP-PRIORITY-COMP ATTEMPTED-COMPS)
                               RECOMMENDATION  'EVALUATE-SYSTEM)
                         (REPORT "RULE-TRIGGERED:PLANNING-RULE-1;NO MORE AVAILABLE GENERATORS")
                         (REPORT "EVALUATING POWER SYSTEM; CONTROL-LEVEL WAS :")
                         (REPORT CONTROL-LEVEL)))

        (PLRULE2        IF
                        (< CONTROL-LEVEL 5)

                        THEN
                        ((:= STEP-SIZE (* EXTENT-OF-VIOLATION
                                          (// (- 100.0 (* 25.0 (- CONTROL-LEVEL 1))) 100.0)))
                         (FORMAT INTERACTIVE-WINDOW
                                 "~% PLANNING-RULE-2 IS TRIGGERED ~% CONTROL-LEVEL = ~A
                          ~% TOTAL-VIOLATION NOW = ~A
                          ~% PERCENT OF TOTAL-VIOLATION ATTEMPTING = ~A
                          ~% VIOLATION ATTEMPTING  FOR CORRECTION = ~A~%"
                                 CONTROL-LEVEL
                                 EXTENT-OF-VIOLATION
                                 (- 100.0 (* 25.0 (- CONTROL-LEVEL 1))) STEP-SIZE)))

        (PLRULE3        IF
                        (= CONTROL-LEVEL 5)

                        THEN
                        ((:= STEP-SIZE (* 0.15 EXTENT-OF-VIOLATION))
                         (FORMAT INTERACTIVE-WINDOW
                                 "~% PLANNING-RULE-3 IS TRIGGERED ~% CONTROL-LEVEL = ~A
                          ~% TOTAL-VIOLATION NOW = ~A
                          ~% PERCENT OF TOTAL-VIOLATION ATTEMPTING = 15
                          ~% VIOLATION ATTEMPTING  FOR CORRECTION = ~A~%"
                                 CONTROL-LEVEL
                                 EXTENT-OF-VIOLATION
                                 STEP-SIZE)))

        (PLRULE4        IF
                        (> CONTROL-LEVEL 5)

                        THEN
                        ((:= CONTROL-LEVEL 2)
                         (FORMAT INTERACTIVE-WINDOW
                                 "~% PLANNING-RULE-4 IS TRIGGERED, VOLTAGE-VIOLATION OF BUS ~A
                          ~% CANNOT BE FURTHER CORRECTED BY GENERATOR ~A
                          ~%HENCE LOOKING FOR OTHER POSSIBLE GENERATORS."
                                TOP-PRIORITY-COMP
                                (CADR CONTROL-GEN-NOW))
                         (SELECT-NEXT-MOST-SENS-GEN)
                         (COND ((NULL CONTROL-GENERATORS)
                                (REPORT "NO MORE GENERATORS AVAILABLE FOR CONTROL"))
                               (T(REPORT " NEXT CONTROL GENERATOR TO BE USED IS :")
                                 (REPORT CONTROL-GEN-NOW)))))
      ))
```

```
;;*************************************************************
;;STATE-COMPUTATION  AND  POSTCONTROL STATE EVALUATION RULES.
;;*************************************************************
(setq  STRATEGY-EXECUTION-RULES
       '(
         (EXRULE1        IF
                         (OR   (EQUAL VIOLATION 'BUS-VOLT-PU-UP-LIMIT-VIOLATED)
                               (EQUAL VIOLATION 'BUS-VOLTAGE-MAY-EXCEED))

                         THEN
                         ((REPORT "RULE-TRUE:EXRULE1 - DUE TO VIOLATION:BUS-OVER-VOLTAGE")
                          (REPORT
                            "MODULES-TRIGGERED:DCR-BUS-VOLT-COMPUTE-NEW-STATE & HEURISTIC-EVAL-FUNC.")
                          (DCR-BUS-VOLT-COMPUTE-NEW-STATE)
                          (HEURISTICALLY-EVAL-NEW-STATE-AND-RECOMMEND)
                          (SETQ ATTEMPTED-CONTROL 'TRUE)))

         (EXRULE2        IF
                         (OR   (EQUAL VIOLATION 'BUS-VOLT-PU-LOW-LIMIT-VIOLATED)
                               (EQUAL VIOLATION 'BUS-VOLTAGE-MAY-FALL))

                         THEN
                         ((REPORT "RULE-TRUE:EXRULE2 - DUE TO VIOLATION:BUS-UNDER-VOLTAGE")
                          (REPORT
                            "MODULES-TRIGGERED:INR-BUS-VOLT-COMPUTE-NEW-STATE & HEURISTIC-EVAL-FUNC.")
                          (INR-BUS-VOLT-COMPUTE-NEW-STATE)
                          (HEURISTICALLY-EVAL-NEW-STATE-AND-RECOMMEND)
                          (SETQ ATTEMPTED-CONTROL 'TRUE)))

         (EXRULE3        IF
                         (OR   (EQUAL VIOLATION 'GENERATOR-OVERLOADED)
                               (EQUAL VIOLATION 'GENERATOR-HEAVILYLOADED))

                         THEN
                         ((DCR-GENERATION-COMPUTE-NEW-STATE )
                          (HEURISTICALLY-EVAL-NEW-STATE-AND-RECOMMEND)
                          (SETQ ATTEMPTED-CONTROL 'TRUE)
                          (REPORT "EXRULE3;VIOLATION = GENERATOR IS OVERLOADED ")))

         ))
```

```
(DEFUN EXECUTE-STRATEGY()
  (PROG(V)
       (SETQ V (SEND N1 :MOST-SEVERE-VIOLATION))
       (SETQ VIOLATION (SEND (EVAL V) :NATURE-OF-VIOLATION))
       (MAPCAR '(LAMBDA (X)(TEST-RULE X)) STRATEGY-EXECUTION-RULES)))

;;This function creates a two dimensional array or matrix of sensitivity

(DEFUN CREATE-SENSITIVITY-MATRIX ()
  (prog(size s1 s2)
       (SETQ SIZE (LENGTH BUS-OBJ) S1 0 S2 0 RECOMMENDATION 'START-CONTROL)
       (tv:window-call (interactive-window :expose)
         (send interactive-window :line-out "ENTER SENSITIVITY-INFO FILE NAME ")
         (WITH-OPEN-FILE (STR (SETQ SENSITIVITY-FILE (READLINE interactive-window))
                              ':DIRECTION
                              ':INPUT
                              ':CHARACTERS T)
           (SETQ SENSITIVITY-MATRIX-JH (MAKE-ARRAY (LIST SIZE SIZE) :TYPE 'ART-Q))
           (SETQ SENSITIVITY-MATRIX-JL (MAKE-ARRAY (LIST SIZE SIZE) :TYPE 'ART-Q))

;;; create sensitivity-matrix JH-inverse ( delta-angle / delta-P )

           (LOOP FOR S1 FROM 1 TO (- SIZE 1) DO
             (LOOP FOR S2 FROM 1 TO(- SIZE 1) DO
               (ASET (READ STR) SENSITIVITY-MATRIX-JH S1 S2)))

;;; create sensitivity-matrix JL-inverse ( delta-V / delta-Q )

           (LOOP FOR S1 FROM 1 TO (- SIZE 1)DO
             (LOOP FOR S2 FROM 1 TO (- SIZE 1)  DO
               (ASET (READ STR) SENSITIVITY-MATRIX-JL S1 S2)))
           (REPORT "COMPLETED CREATION OF SENSITIVITY-MATRICES JH-INVERSE, JL-INVERSE")
           ))))


;; ********************************************************************
;; FUNCTION THAT FINDS AND ARRANGES IN ORDER OF PRIORITY THE AVAILABLE
;; GENERATORS THAT COULD BE USED FOR CORRECTING THE GIVEN VIOLATION.
;;********************************************************************

(SETQ V-BUS-GEN '(G1 G2 G3 G5 G6 G7 G8 G9 G10))
(SETQ SWING-GEN 'G4 SWING-BUS 'B39)


(DEFUN FIND-SORT-GEN (COMP)
  (setq ROW (funcall (eval COMP) :NUMBER))
  (SORTCAR  (MAPCAR '(LAMBDA (X)
                       (LIST
                         (ABS(AREF SENSITIVITY-MATRIX-JL
                                   ROW
                                   (SETQ COLUMN
                                         (funcall (EVAL (funcall (EVAL X) :CONNECTED-BUS))
                                                  :NUMBER))))
                         X
                         COLUMN
                         ))
                    V-BUS-GEN)
            #'>))
```

```
;;*****************************************************************************
;; FUNCTIONS CALLED BY STATE COMPUTATION AND POSTCONTROL STATE EVALUATION RULES
;;*****************************************************************************

(DEFUN DCR-BUS-VOLT-COMPUTE-NEW-STATE()
  (SETQ DIRECTION 'DCR)
  (CHANGE-BUS-VOLT-COMPUTE-NEW-STATE))

(DEFUN INR-BUS-VOLT-COMPUTE-NEW-STATE()
  (SETQ DIRECTION 'INR)
  (CHANGE-BUS-VOLT-COMPUTE-NEW-STATE))

(DEFUN CHANGE-BUS-VOLT-COMPUTE-NEW-STATE()
  (PROG(DELTA-Vi-BUS
        DELTA-Vk-GEN
        DVi-BY-DQk
        DVk-BY-DQk
        OLD-Vi-BUS
        OLD-Vk-GEN
        POS VAR
        NEW-Vi-BUS
        NEW-Vk-GEN
        OLD-Qk-GEN)
       (SETQ G (FUNCALL N2 :CONTROL-GEN-NOW)
             V (FUNCALL N1 :MOST-SEVERE-VIOLATION))

       ;;;;  First compute various new parameters
       (setq N nil DELTA-Qk-GEN nil)

       (setq B (FUNCALL (EVAL (CADR G)) :CONNECTED-BUS))
;B gives the voltage bus name to which this most sensitive generator is connected to

(:= OLD-Vk-GEN   (FUNCALL (EVAL B) :VOLTAGE-PU))

(setq DELTA-Vi-BUS (SEND N2 :STEP-SIZE)
;Extent of violation correction to be done in a single shot

      DVi-BY-DQk   (CAR G)
;find sensitivity relation between the bus-voltage and gen-reactive-power

      N            (CADDR G)
;gives the column  of the generator being used in sensitivity matrix

      DVk-BY-dQk   (ABS (AREF SENSITIVITY-MATRIX-JL N N)))
;gives sensitivity ratio (D-volt/D-q) of generator being used now

(:= DELTA-Qk-GEN (// DELTA-Vi-BUS DVi-BY-DQk))
; Calculate how much of reactive-power of most sensitive generator should be
; decreased to affect the decrease in the voltage of the bus under correction

(:= DELTA-Vk-GEN (* DVk-BY-DQk DELTA-Qk-GEN))
;Calculate how much of the voltage of the generator should be decreased to affect
;decrease in its reactive power which in turn will decrease the voltage of the
;bus being corrected

(:= OLD-Vi-BUS  (FUNCALL (EVAL V) :VOLTAGE-PU)) ; call up older voltage value of

(SETQ OLD-Qk-GEN (FUNCALL (EVAL (CADR G)) :CURRENT-OUTPUT-MVAR))
(COND ((EQUAL DIRECTION 'DCR)
       (FUNCALL (EVAL (CADR G)) :SET-CURRENT-OUTPUT-MVAR (- OLD-Qk-GEN DELTA-Qk-GEN))
       (:= NEW-Vi-BUS  (- OLD-Vi-BUS  DELTA-Vi-BUS))
       (:= NEW-Vk-BUS  (- OLD-Vk-GEN  DELTA-Vk-GEN))
       (format interactive-window
               "~% VOLTAGE OF VIOLATED BUS ~a IS BEING DECREMENTED BY ~a"
               (SEND N1 :MOST-SEVERE-VIOLATION) delta-vi-bus)
       (report "OTHER DETAILS RELATED TO THIS LOAD BUS ARE MENTIONED BELOW:")
       (format interactive-window
               "~%GENERATOR BEING USED FOR CORRECTION OF THE VIOLATION =  ~a
                ~%OTHER PROBABLE GENERATORS(SENSITIVITY-INFO, GENERATOR-NAME) ARE : ~% ~a
                ~% THE PREVIOUS BUS-VOLTAGE WAS = ~a ~% THE POST-CONTROL VOLTAGE = ~a ~%"
               G
               CONTROL-GENERATORS
               OLD-Vi-BUS
               NEW-Vi-BUS))
      ((EQUAL DIRECTION 'INR)
```

```
        (FUNCALL (EVAL (CADR G)) :SET-CURRENT-OUTPUT-MVAR (+ OLD-Qk-GEN DELTA-Qk-GEN))
        (:= NEW-Vi-BUS  (+ OLD-Vi-BUS  DELTA-Vi-BUS))
        (:= NEW-Vk-BUS  (+ OLD-Vk-GEN  DELTA-Vk-GEN))
        (format interactive-window
                "~& VOLTAGE OF VIOLATED BUS ~a IS BEING INCREMENTED BY ~a"
                V delta-vi-bus)
        (report "OTHER DETAILS RELATED TO THIS LOAD BUS ARE MENTIONED BELOW:")
        (format interactive-window
                "~&GENERATOR BEING USED FOR CORRECTION OF THE VIOLATION =  ~a
                 ~&OTHER PROBABLE GENERATORS(SENSITIVITY-INFO, GENERATOR-NAME) ARE ~& ~a
                 ~&THE PREVIOUS BUS-VOLTAGE WAS = ~a
                                ~& THE POST-CONTROL VOLTAGE = ~a~&"

                G
                CONTROL-GENERATORS
                OLD-Vi-BUS
                NEW-Vi-BUS)

        (SETQ POS (FUNCALL (EVAL V) :POSITION))
        (SEND SENSITIVE-WINDOW :SET-CURSORPOS (- (CAR POS) 27)(- (CADR POS) 20))
        (SEND SENSITIVE-WINDOW :DELETE-CHAR 12 ':CHARACTER)
        (SEND SENSITIVE-WINDOW :INSERT-CHAR 12 ':CHARACTER)

        (SEND SENSITIVE-WINDOW :SET-CURSORPOS (- (CAR POS) 27)(- (CADR POS) 20))
        (FUNCALL (EVAL V) :SET-VOLTAGE-PU NEW-BUS-VOLT)
        (FORMAT SENSITIVE-WINDOW "~A" (FUNCALL (EVAL V) :VOLTAGE-PU))))

;; Update in the knowledge base the two new bus voltages  computed above
(REPORT "NEW-STATE OF POWER-SYSTEM IS COMPUTED")
(SETQ VAR (SEND N1  :MOST-SEVERE-VIOLATION))
(FUNCALL (EVAL VAR) :SET-VOLTAGE-PU NEW-Vi-BUS)
(FUNCALL (EVAL B)   :SET-VOLTAGE-PU NEW-Vk-BUS)

;; Compute and update in the knowledge-base  voltages of all other buses
;;due to the above mentioned change in voltage of the top-priority bus
;;and control generator bus, along the whole nth column  of jl-inverse matrix.

(REPORT "THE GENERATOR BEING USED FOR CONTROL AND ITS DELTA-Qk VALUE ARE:")
(report (CADR G))
(REPORT DELTA-Qk-GEN)

(MAPCAR '(LAMBDA (X )
           (PROG(ROW DELTA-V-Q-RATIO DELTA-BUS-VOLT NEW-BUS-VOLT POS V)
                (SETQ V (FUNCALL N1 :MOST-SEVERE-VIOLATION))

                (COND ((NOT (MEMBER X (LIST B V SWING-BUS)))
                       (SETQ ROW (FUNCALL (EVAL X) :NUMBER))
                       (SETQ DELTA-V-Q-RATIO (ABS (AREF SENSITIVITY-MATRIX-JL ROW N)))
                       (:= DELTA-BUS-VOLT (* DELTA-V-Q-RATIO DELTA-Qk-GEN))
                       (COND ((EQUAL DIRECTION 'DCR)
                              (:= NEW-BUS-VOLT
                                  (- (FUNCALL (EVAL X) :VOLTAGE-PU) DELTA-BUS-VOLT)))
                             ((EQUAL DIRECTION 'INR)
                              (:= NEW-BUS-VOLT
                                  (+ (FUNCALL (EVAL X) :VOLTAGE-PU) DELTA-BUS-VOLT))))
                       (SETQ POS (FUNCALL (EVAL X) :POSITION))
                       (SEND SENSITIVE-WINDOW :SET-CURSORPOS (- (CAR POS) 27)(- (CADR POS) 20))

                       (SEND SENSITIVE-WINDOW :DELETE-CHAR 12 ':CHARACTER)

                       (SEND SENSITIVE-WINDOW :INSERT-CHAR 12 ':CHARACTER)
                       (SEND SENSITIVE-WINDOW :SET-CURSORPOS (- (CAR POS) 27)(- (CADR POS) 20))

                       (FUNCALL (EVAL X) :SET-VOLTAGE-PU NEW-BUS-VOLT)
                       (FORMAT SENSITIVE-WINDOW "~A" (FUNCALL (EVAL X) :VOLTAGE-PU))))))
        BUS-OBJ)
))
```

```
(defun display-violations(L)
  (cond ((not(null L))
         (report "PLEASE ACKNOWLEDGE VIOLATIONS, BY MOVING CURSOR TO THE RIGHT.")
         (blink-violations L))))

(defun blink-violations(L)
  (tv:window-call (sensitive-window :expose)
    (prog(n)
         (setq n 0)
      label1
         (cond ((AND (greaterp tv:mouse-x 800)(evenp n))(return))
               (t(setq n (+ n 1))
                 (mapcar '(lambda(x)
                            (prog(pos n)
                                 (setq pos (funcall (eval x) :position))
                                 (draw-lines (car pos)(cadr pos)
                                             (caddr pos)(cadddr pos) scall)))
                         L)
                 (go label1))))))
```

```
;; ***************** HEURISTIC EVALUATION FUNCTION ****************

(DEFUN HEURISTICALLY-EVAL-NEW-STATE-AND-RECOMMEND()
  (PROG(OVERALL-IMPROV-FLAG
;Above flag indicates  improvement in the overall state of  power-system.

        VIOL
        NEW-VOLT
        IMP-VOLT
        OLD-VIOLS
        NEW-VIOLS VAR)

       (setq EFFECT-FLAG  nil
;Above flag indicates effect of  latest set of control measures on the top-violation.

             FUTURE-FLAG  nil)
;Above flag indicates  potential of improvement of the overall-state or of the top-violation comp.

; find out the value for EFFECT-FLAG
(report "HEURISTIC EVALUATION OF POSTCONTROL STATE OF POWER SYSTEM IS IN PROGRESS...")

(SETQ VAR           (SEND N1  :MOST-SEVERE-VIOLATION)
      VIOL          (SEND (EVAL VAR) :NATURE-OF-VIOLATION)
      NEW-VOLT      (SEND (EVAL VAR) :VOLTAGE-PU)
      IMP-VOLT (CAR (SEND (EVAL VAR) :IMPROVED-VOLT-PU))
      NEW-VOLTAGE-PU NEW-VOLT)

(COND ((MEMBER  VIOL  '(BUS-VOLT-PU-UP-LIMIT-VIOLATED BUS-VOLTAGE-MAY-EXCEED))
       (COND ((< NEW-VOLT  IMP-VOLT)
              (SETQ EFFECT-FLAG 'GOOD))
             ((= NEW-VOLT IMP-VOLT)
              (SETQ EFFECT-FLAG 'SAME))
             (T(SETQ EFFECT-FLAG 'BAD))))
      ((MEMBER  VIOL  '(BUS-VOLT-PU-LOW-LIMIT-VIOLATED BUS-VOLTAGE-MAY-FALL))
       (COND ((> NEW-VOLT  IMP-VOLT)
              (SETQ EFFECT-FLAG 'GOOD))
             ((= NEW-VOLT IMP-VOLT)
              (SETQ EFFECT-FLAG 'SAME))
             (T(SETQ EFFECT-FLAG 'BAD)))))
(REPORT "PREVIOUS BUS-VOLTAGE = ")
(REPORT imp-volt)
(REPORT "BUT, POST-CONTROL BUS-VOLTAGE = ")
(REPORT new-volt)
(report "HENCE, EFFECT OF LATEST SET OF CONTROL MEASURES IS : ")
(REPORT EFFECT-FLAG)

;find out the value for OVERALL-IMPROV-FLAG

;       AB1 :  abnormal components in the previous state
;       AB2 :  abnormal components in the present state
;       AL1 :  alert components in the previous state
;       AL2 :  alert components in the present state

(report "HEURISTIC EVALUATION OF POSTCONTROL STATE OF POWER SYSTEM IS IN PROGRESS...")
(SETQ AB1 NIL AB2 NIL AL1 NIL AL2 NIL)
(MAPCAR '(LAMBDA(X)
           (COND ((EQUAL (FUNCALL (EVAL X) :COMPONENT-STATUS)
                         'ABNORMAL)
                  (SETQ AB1 (CONS X AB1)))
                 (T(SETQ AL1 (CONS X AL1)))))
        (FUNCALL N1 :VIOLATIONS))

;; Functions below finds postcontrol state violations.
(FIND-FINAL-VIOLATIONS BUS-OBJ GEN-OBJ TRLN-OBJ)
(FUNCALL N3 :SET-VIOLATIONS COMPLAINT-BLKBOARD)
(SETQ OLD-VIOLS (FUNCALL N1 :VIOLATIONS)
      NEW-VIOLS (FUNCALL N3 :VIOLATIONS))
(MAPCAR '(LAMBDA(X) (COND ((EQUAL (FUNCALL (EVAL X) :COMPONENT-STATUS)
                                  'ABNORMAL)
                           (SETQ AB2 (CONS X AB2)))
                          (T(SETQ AL2 (CONS X AL2)))))
        NEW-VIOLS)
(cond ((< (LENGTH OLD-VIOLS) (LENGTH NEW-VIOLS)) (setq OVERALL-IMPROV-FLAG 'BAD ))
      ((= (LENGTH OLD-VIOLS) (LENGTH NEW-VIOLS)) (setq OVERALL-IMPROV-FLAG 'SAME))
      (t(setq OVERALL-IMPROV-FLAG  'GOOD)))
```

```
   (format interactive-window
           "~% ALERT COMPONENTS IN PREVIOUS-STATE = ~A
            ~% ABNORMAL COMPONENTS IN PREVIOUS-STATE ~a
            ~% ALERT COMPONENTS IN POST-CONTROL STATE ~A
            ~% ABNORMAL COMPONENTS IN POST-CONTROL STATE ~a ~%"
           AL1 AB1 AL2 AB2)
   (display-violations NEW-VIOLS)
   (REPORT "OVERALL IMPROVEMENT IN POSTCONTROL STATE OF POWER SYSTEM IS:")
   (REPORT OVERALL-IMPROV-FLAG)
   (report "HEURISTIC EVALUATION OF POSTCONTROL STATE OF POWER SYSTEM IS IN PROGRESS...")

   (COND ((NULL NEW-VIOLS)
          (SETQ FUTURE-FLAG 'REST))
         ((< (LENGTH AB1)(LENGTH AB2))
          (SETQ FUTURE-FLAG 'BAD))
         ((> (LENGTH AB1)(LENGTH AB2))
          (SETQ FUTURE-FLAG 'GOOD))
         (T(COND ((= (LENGTH AL1)(LENGTH AL2))(SETQ FUTURE-FLAG 'SAME))
                 ((< (LENGTH AL1)(LENGTH AL2))(SETQ FUTURE-FLAG 'BAD))
                 ((> (LENGTH AL1)(LENGTH AL2))(SETQ FUTURE-FLAG 'GOOD)))))

   (format interactive-window "~% EFFECT ON TOP-VIOLATION IS: ~A
                               ~% EFFECT ON OVERALL-STATE IS: ~A
                               ~% HENCE POTENTIAL OF FUTURE VIOLATIONS IS:"
           effect-flag overall-improv-flag )

   (cond ((equal future-flag 'bad)(report "HIGHER THAN BEFORE"))
         ((equal future-flag 'good)(report "LOWER THAN BEFORE"))
         ((equal future-flag 'rest)(report "NONE"))
         (t(report FUTURE-FLAG)))

   ;******** MACHINE LEARNING SUBSYSTEM IS CALLED********
   (COND (MACHINE-LEARNING
          (extract-observation 'H-EVAL)
          (INFERENCE-ENGINE)))
  ;*****************************************************

   (COND ((EQUAL FUTURE-FLAG  'REST)
          (report "NO VIOLATIONS IN POSTCONTROL STATE OF POWER-SYSTEM, EXPERT IS SUCCESSFUL")
          (CHANGE-TO-NEW-STATE) ;;CASE-I
          (SETQ RECOMMENDATION 'OVER)(RETURN))
         ((OR
            (AND(MEMBER EFFECT-FLAG '(GOOD SAME))
                (MEMBER OVERALL-IMPROV-FLAG '(GOOD SAME))
                (MEMBER FUTURE-FLAG '(GOOD SAME)))
            (AND(MEMBER EFFECT-FLAG '(SAME BAD))
                (EQUAL OVERALL-IMPROV-FLAG 'GOOD)
                (EQUAL FUTURE-FLAG 'GOOD)))

          (report "IMPROVEMENT IN POSTCONTROL STATE PROMPTS EXPERT TO MOVE IN SAME DIRECTION")
          (CHANGE-TO-NEW-STATE)
          (COND ((> (FUNCALL (FUNCALL N1 :MOST-SEVERE-VIOLATION) :COMP-VIOL) 0)
                 (SELECT-NEXT-MOST-SENS-GEN)
                 (SETQ CONTROL-LEVEL 2 ATTEMPT 'OLD RECOMMENDATION 'PLAN-STRATEGY))
                (T(SETQ ATTEMPTED-COMPS (CONS (FUNCALL N1 :MOST-SEVER-VIOLATION)
                                              ATTEMPTED-COMPS))
                  (SETQ RECOMMENDATION 'NEEDS-NO-FURTHER-ACTION ATTEMPT 'NEW))))

         (T(BACKTRACK-TO-OLD-STATE)                    ;;CASE-II
           (report "WORSER POSTCONTROL STATE PROMPTS EXPERT TO BACKTRACK TO PREVIOUS STATE.")
           (FIND-FINAL-VIOLATIONS BUS-OBJ GEN-OBJ TRLN-OBJ)
           (SETQ CONTROL-LEVEL (+ CONTROL-LEVEL 1)
                 ATTEMPT 'OLD
                 RECOMMENDATION 'REPLAN-STRATEGY)
           (report "LEVEL OF CONTROL IS")
           (report control-level)
           (report "RECOMMENDATION IS: ")
           (report recommendation)))))

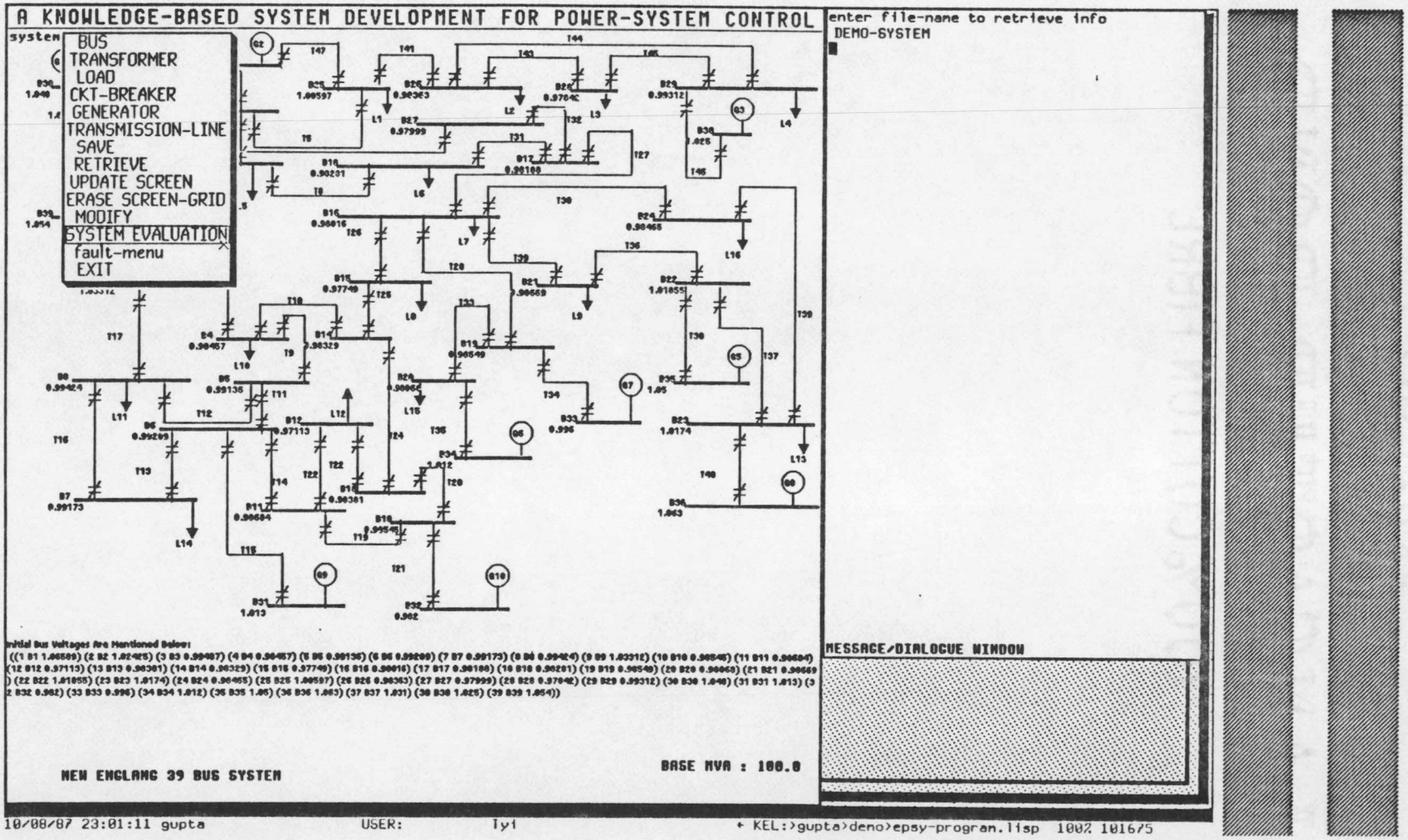

(DEFUN CHANGE-TO-NEW-STATE()
  (MAPCAR '(LAMBDA(X)
             (FUNCALL (EVAL X) :SET-IMPROVED-VOLT-PU
                      (CONS (FUNCALL (EVAL X) :VOLTAGE-PU)
                            (FUNCALL (EVAL X) :IMPROVED-VOLT-PU))))
          BUS-OBJ)
```

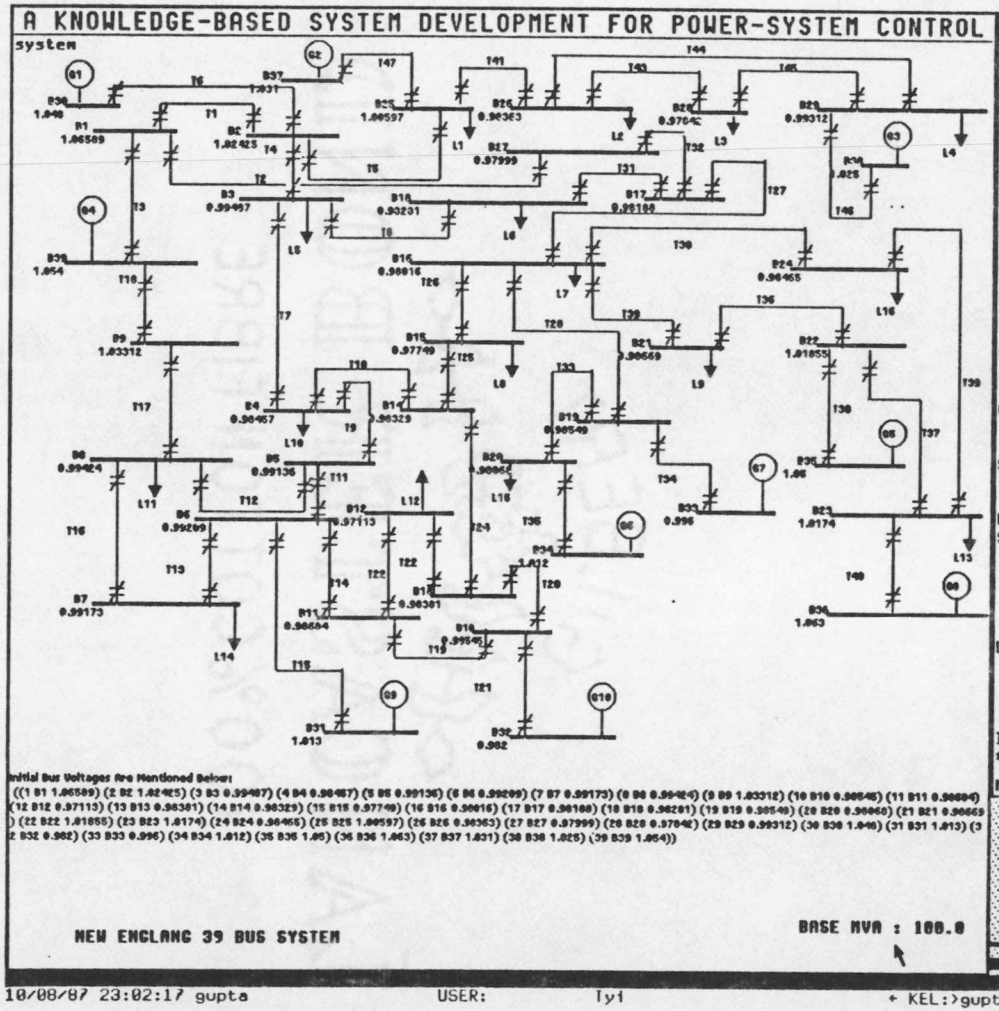
```
  (MAPCAR ' (LAMBDA (X)
              (FUNCALL (EVAL X) :SET-IMPROVED-OUTPUT-MW
                       (CONS (FUNCALL (EVAL X) :CURRENT-OUTPUT-MW)
                             (FUNCALL (EVAL X) :IMPROVED-OUTPUT-MW)))
              (FUNCALL (EVAL X) :SET-IMPROVED-OUTPUT-MVAR
                       (CONS (FUNCALL (EVAL X) :CURRENT-OUTPUT-MVAR)
                             (FUNCALL (EVAL X) :IMPROVED-OUTPUT-MVAR))))
          GEN-OBJ)
  (SEND N1 :SET-VIOLATIONS (FUNCALL N3 :VIOLATIONS))
  (SEND N1 :SET-MOST-SEVERE-VIOLATION (FUNCALL N3 :MOST-SEVERE-VIOLATION))
  (SEND N1 :SET-SYSTEM-STATUS (FUNCALL N3 :SYSTEM-STATUS)))


(DEFUN BACKTRACK-TO-OLD-STATE ()
  (MAPCAR ' (LAMBDA (X)
              (FUNCALL (EVAL X) :SET-VOLTAGE-PU
                       (CAR (FUNCALL (EVAL X) :IMPROVED-VOLT-PU))))
          BUS-OBJ)
  (MAPCAR ' (LAMBDA (X)
              (FUNCALL (EVAL X) :SET-CURRENT-OUTPUT-MW
                       (CAR (FUNCALL (EVAL X) :IMPROVED-OUTPUT-MW)))
              (FUNCALL (EVAL X) :SET-CURRENT-OUTPUT-MVAR
                       (CAR (FUNCALL (EVAL X) :IMPROVED-OUTPUT-MVAR))))
          GEN-OBJ))
```

APPENDIX B

EPSY DEMONSTRATION


A KNOWLEDGE-BASED SYSTEM DEVELOPMENT FOR POWER-SYSTEM CONTROL
system
BUS
TRANSFORMER
LOAD
CKT-BREAKER
GENERATOR
TRANSMISSION-LINE
SAVE
RETRIEVE
UPDATE SCREEN
ERASE SCREEN-GRID
MODIFY
SYSTEM EVALUATION
fault-menu
EXIT
enter file-name to retrieve info
DEMO-SYSTEM
MESSAGE/DIALOGUE WINDOW
Initial Bus Voltages Are Mentioned Below:
((1 B1 1.06589) (2 B2 1.02455) (3 B3 0.99487) (4 B4 0.99457) (5 B5 0.99136) (6 B6 0.99209) (7 B7 0.99173) (8 B8 0.99424) (9 B9 1.03312) (10 B10 0.99548) (11 B11 0.99684) (12 B12 0.97113) (13 B13 0.98381) (14 B14 0.98329) (15 B15 0.97749) (16 B16 0.98016) (17 B17 0.98188) (18 B18 0.98281) (19 B19 0.98549) (20 B20 0.98468) (21 B21 0.98669) (22 B22 1.01855) (23 B23 1.0174) (24 B24 0.98465) (25 B25 1.00597) (26 B26 0.98363) (27 B27 0.97999) (28 B28 0.97042) (29 B29 0.99312) (30 B30 1.048) (31 B31 1.013) (32 B32 0.982) (33 B33 0.996) (34 B34 1.012) (35 B35 1.05) (36 B36 1.063) (37 B37 1.031) (38 B38 1.025) (39 B39 1.054))
NEW ENGLAND 39 BUS SYSTEM
BASE MVA : 100.0
10/08/87 23:01:11 gupta
USER:
tyi
KEL:>gupta>demo>epsy-program.lisp 1002 101675



POWER SYSTEM ON EVALUATION IS FOUND IN FOLL
OWING STATE:
ALERT-STATE
PARAMETERS OF FOLLOWING COMPONENTS ARE VIOL
ATED

(B1)

PLEASE ACKNOWLEDGE VIOLATIONS, BY MOVING C
URSOR TO THE RIGHT.
WANT TO CORRECT THE VIOLATIONS ?
Y

EXPERT-MODULE IS TRIGGERED
EXPERT-SYSTEM IS ATTEMPTING TO
CORRECT THE VIOLATIONS

WANT MACHINE-LEARNING CAPABILITY ?
YES

CONTROL-RULE-7 TRUE BY RECOMMENDATION:DO-P
OWER-FLOW-ANALYSIS.

MODULES-TRIGGERED:POWER-FLOW-ANALYSIS and
SENSITIVITY-ANALYSIS.

SENSITIVITY-MATRIX CREATION IS REQUIRED.
ENTER SENSITIVITY-INFO FILE NAME
SENSITIVE.MATRIX

COMPLETED CREATION OF SENSITIVITY-MATRICES
JH-INVERSE, JL-INVERSE

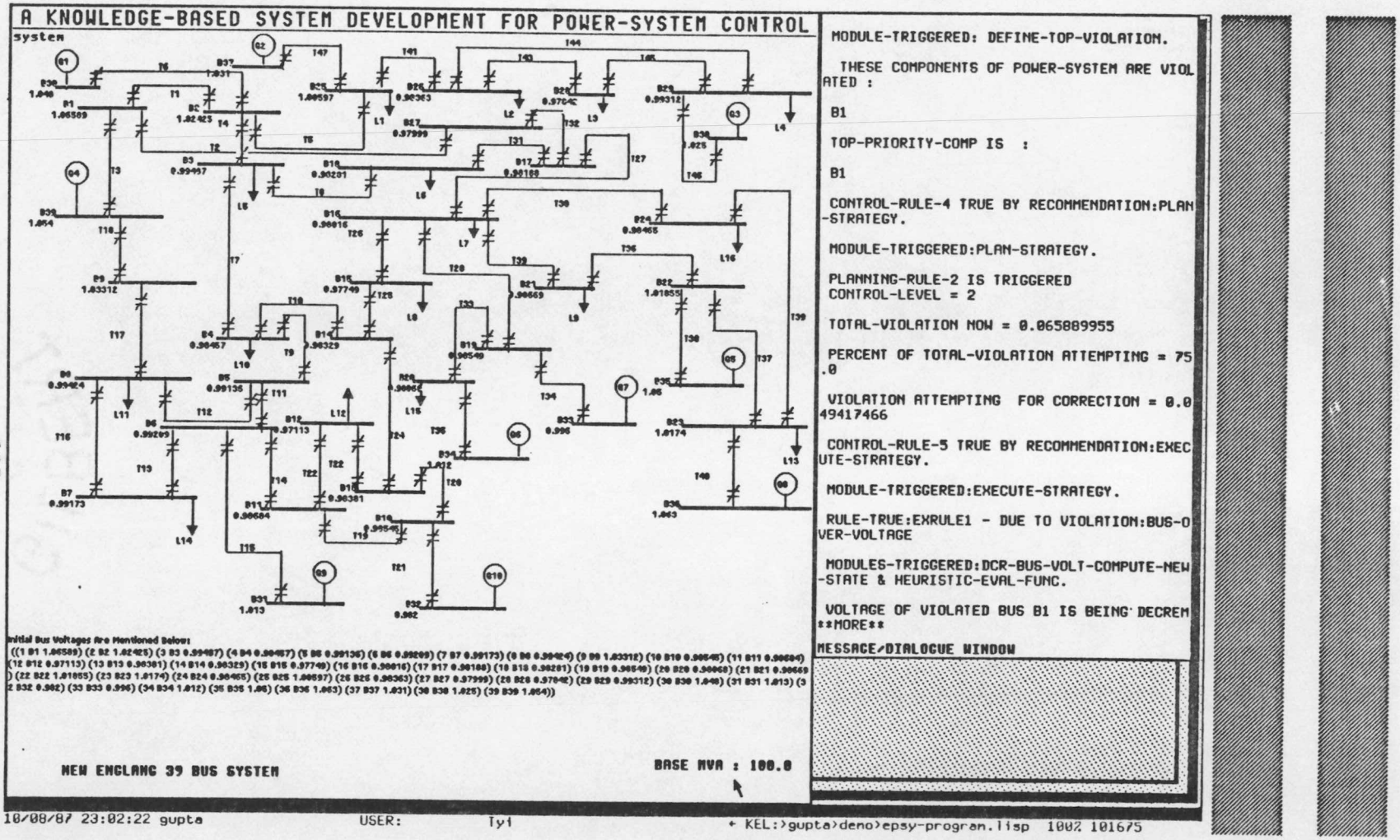
CONTROL-RULE-1 TRUE BY RECOMMENDATION: EVA
LUATE-SYSTEM.

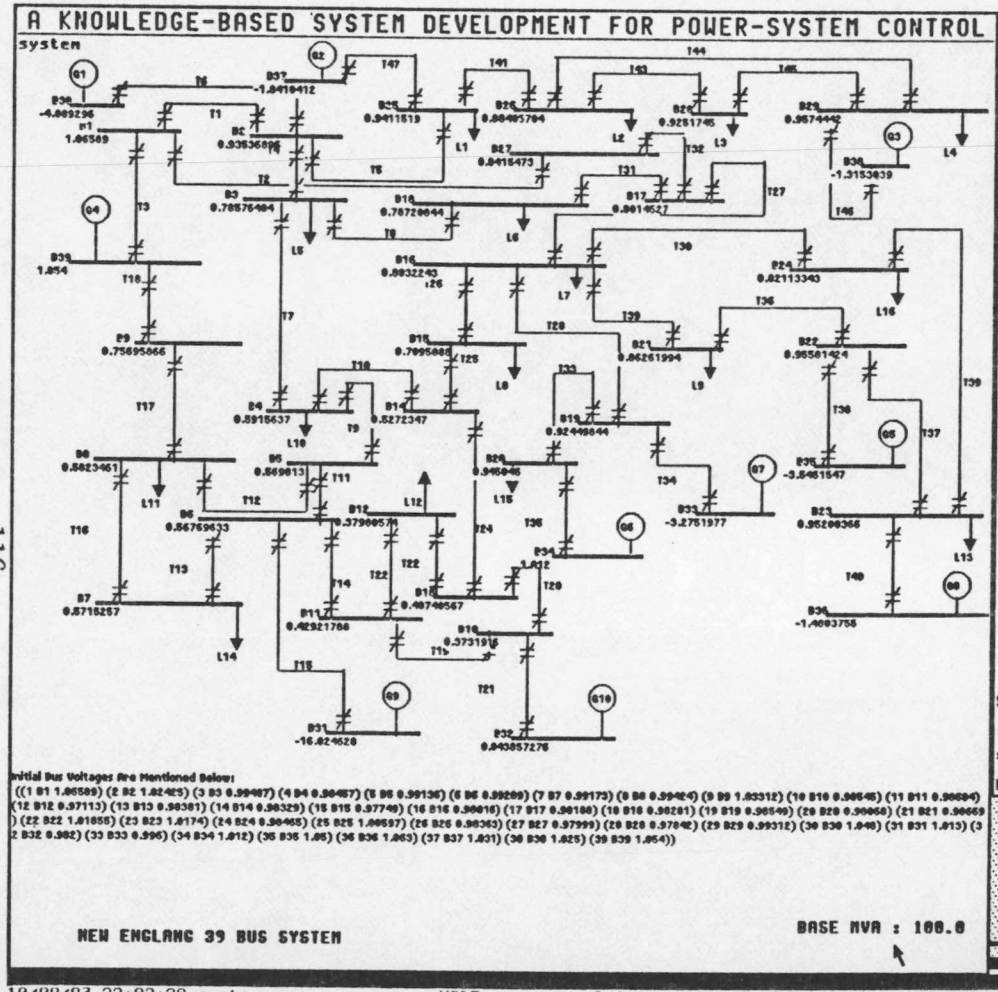
MODULE-TRIGGERED: SYSTEM-EVALUATION.

CONTROL-RULE-2 TRUE BY RECOMMENDATION: DEF
INE-PROBLEM-COMPONENT.
MORE

MESSAGE/DIALOGUE WINDOW

10/08/87 23:02:17 gupta USER: Tyi * KEL:>gupta>demo>epsy-progran.lisp 1002 101675


A KNOWLEDGE-BASED SYSTEM DEVELOPMENT FOR POWER-SYSTEM CONTROL
system
Initial Bus Voltages Are Mentioned Below:
((1 B1 1.06589) (2 B2 1.02425) (3 B3 0.99487) (4 B4 0.99487) (5 B5 0.99136) (6 B6 0.99209) (7 B7 0.99173) (8 B8 0.99424) (9 B9 1.03312) (10 B10 0.98648) (11 B11 0.98684) (12 B12 0.97113) (13 B13 0.98381) (14 B14 0.98329) (15 B15 0.97748) (16 B16 0.98816) (17 B17 0.98188) (18 B18 0.98281) (19 B19 0.98549) (20 B20 0.98068) (21 B21 0.98669) (22 B22 1.01855) (23 B23 1.0174) (24 B24 0.98465) (25 B25 1.00697) (26 B26 0.98363) (27 B27 0.97999) (28 B28 0.97642) (29 B29 0.99312) (30 B30 1.048) (31 B31 1.013) (32 B32 0.982) (33 B33 0.996) (34 B34 1.012) (35 B35 1.05) (36 B36 1.063) (37 B37 1.031) (38 B38 1.025) (39 B39 1.054))
NEW ENGLAND 39 BUS SYSTEM
BASE MVA : 100.0
MODULE-TRIGGERED: DEFINE-TOP-VIOLATION.
THESE COMPONENTS OF POWER-SYSTEM ARE VIOLATED :
B1
TOP-PRIORITY-COMP IS :
B1
CONTROL-RULE-4 TRUE BY RECOMMENDATION:PLAN-STRATEGY.
MODULE-TRIGGERED:PLAN-STRATEGY.
PLANNING-RULE-2 IS TRIGGERED
CONTROL-LEVEL = 2
TOTAL-VIOLATION NOW = 0.065889955
PERCENT OF TOTAL-VIOLATION ATTEMPTING = 75.0
VIOLATION ATTEMPTING FOR CORRECTION = 0.049417466
CONTROL-RULE-5 TRUE BY RECOMMENDATION:EXECUTE-STRATEGY.
MODULE-TRIGGERED:EXECUTE-STRATEGY.
RULE-TRUE:EXRULE1 - DUE TO VIOLATION:BUS-OVER-VOLTAGE
MODULES-TRIGGERED:DCR-BUS-VOLT-COMPUTE-NEW-STATE & HEURISTIC-EVAL-FUNC.
VOLTAGE OF VIOLATED BUS B1 IS BEING DECREM
MORE
MESSAGE/DIALOGUE WINDOW
10/08/87 23:02:22 gupta
USER:
Tyi
+ KEL:>gupta>demo>epsy-program.lisp 1002 101675



ENTED BY 0.049417466
OTHER DETAILS RELATED TO THIS LOAD BUS ARE MENTIONED BELOW:

GENERATOR BEING USED FOR CORRECTION OF THE VIOLATION = (0.0031873 G6 34)

OTHER PROBABLE GENERATORS(SENSITIVITY-INFO, GENERATOR-NAME) ARE :
((0.0014904 G1 30) (9.783e-4 G9 31) (0.0 G2 37) (0.0 G3 38) (0.0 G5 35) (0.0 G7 33) (0.0 G8 36) (0.0 G10 32))

THE PREVIOUS BUS-VOLTAGE WAS = 1.06589
THE POST-CONTROL VOLTAGE = 1.0164725

NEW-STATE OF POWER-SYSTEM IS COMPUTED

THE GENERATOR BEING USED FOR CONTROL AND ITS DELTA-Qk VALUE ARE:

G6

15.584492

HEURISTIC EVALUATION OF POSTCONTROL STATE OF POWER SYSTEM IS IN PROGRESS...

PREVIOUS BUS-VOLTAGE =

1.06589

BUT, POST-CONTROL BUS-VOLTAGE =

1.0164725

HENCE, EFFECT OF LATEST SET OF CONTROL MEASURES IS :

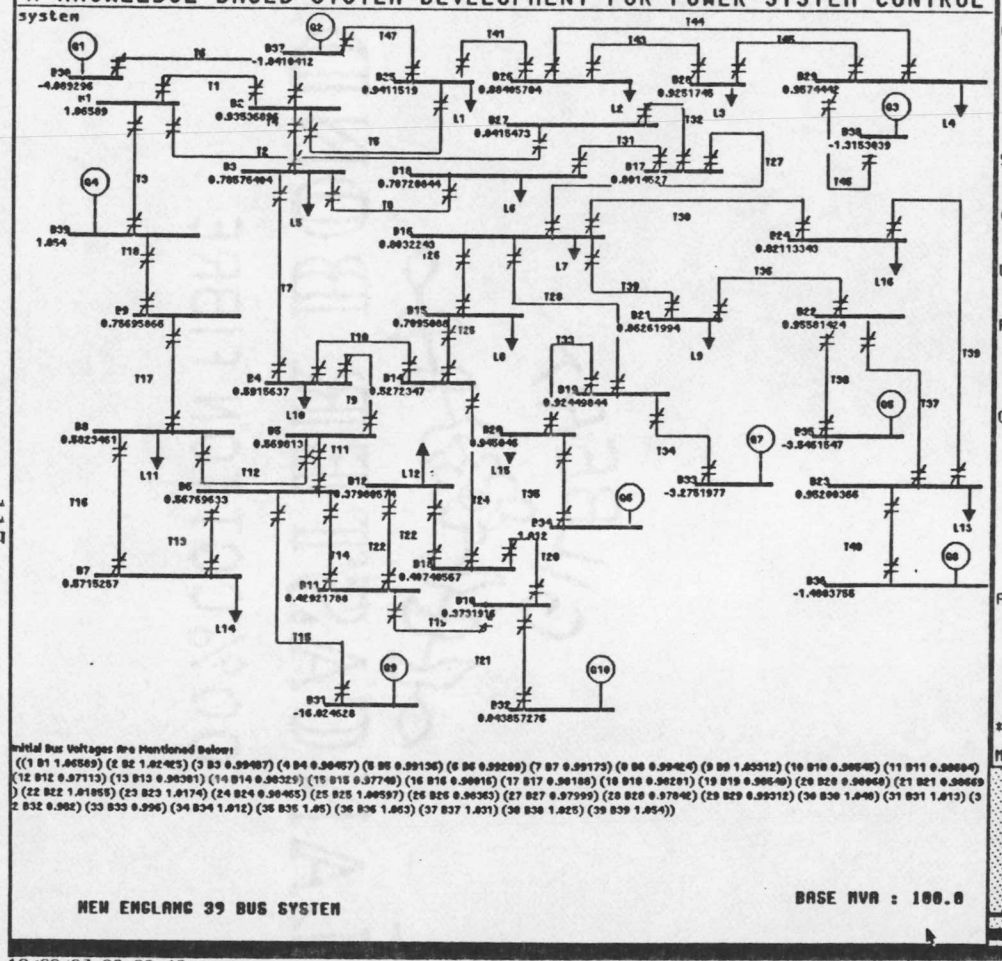
GOOD
MORE

MESSAGE/DIALOGUE WINDOW

Initial Bus Voltages Are Mentioned Below:
((1 B1 1.06589) (2 B2 1.02425) (3 B3 0.99497) (4 B4 0.99457) (5 B5 0.99136) (6 B6 0.99289) (7 B7 0.99173) (8 B8 0.99424) (9 B9 1.03312) (10 B10 0.98548) (11 B11 0.98694) (12 B12 0.97113) (13 B13 0.98381) (14 B14 0.98329) (15 B15 0.97748) (16 B16 0.98016) (17 B17 0.98186) (18 B18 0.98281) (19 B19 0.98548) (20 B20 0.98658) (21 B21 0.98669) (22 B22 1.01858) (23 B23 1.0174) (24 B24 0.98465) (25 B25 1.00597) (26 B26 0.98363) (27 B27 0.97999) (28 B28 0.97842) (29 B29 0.99312) (30 B30 1.048) (31 B31 1.013) (32 B32 0.982) (33 B33 0.996) (34 B34 1.012) (35 B35 1.05) (36 B36 1.063) (37 B37 1.031) (38 B38 1.025) (39 B39 1.054))

10/08/87 23:02:28 gupta USER: Tyi * KEL:>gupta>demo>epsy-program.lisp 1002 101675

A KNOWLEDGE-BASED SYSTEM DEVELOPMENT FOR POWER-SYSTEM CONTROL



HEURISTIC EVALUATION OF POSTCONTROL STATE OF POWER SYSTEM IS IN PROGRESS...

ALERT COMPONENTS IN PREVIOUS-STATE = (B1)

ABNORMAL COMPONENTS IN PREVIOUS-STATE NIL

ALERT COMPONENTS IN POST-CONTROL STATE (B15)

ABNORMAL COMPONENTS IN POST-CONTROL STATE (B12 B7 B5 B13 B6 B11 B8 B4 B10 B14)

PLEASE ACKNOWLEDGE VIOLATIONS, BY MOVING CURSOR TO THE RIGHT.

OVERALL IMPROVEMENT IN POSTCONTROL STATE OF POWER SYSTEM IS:

BAD

HEURISTIC EVALUATION OF POSTCONTROL STATE OF POWER SYSTEM IS IN PROGRESS...

EFFECT ON TOP-VIOLATION IS: GOOD

EFFECT ON OVERALL-STATE IS: BAD

HENCE POTENTIAL OF FUTURE VIOLATIONS IS: HIGHER THAN BEFORE

THESE COMPONENTS OF POWER-SYSTEM ARE VIOLATED :

B14

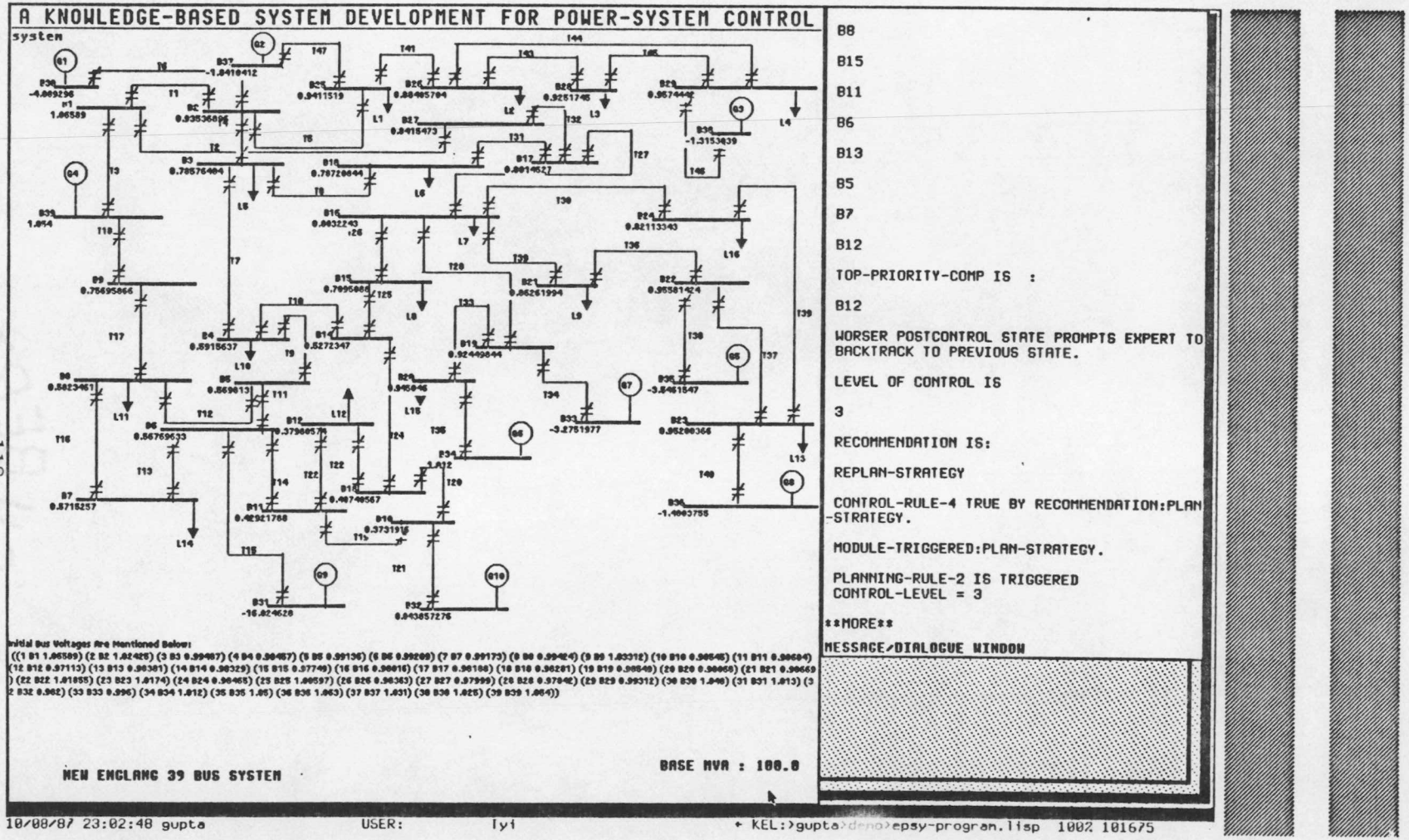
B10

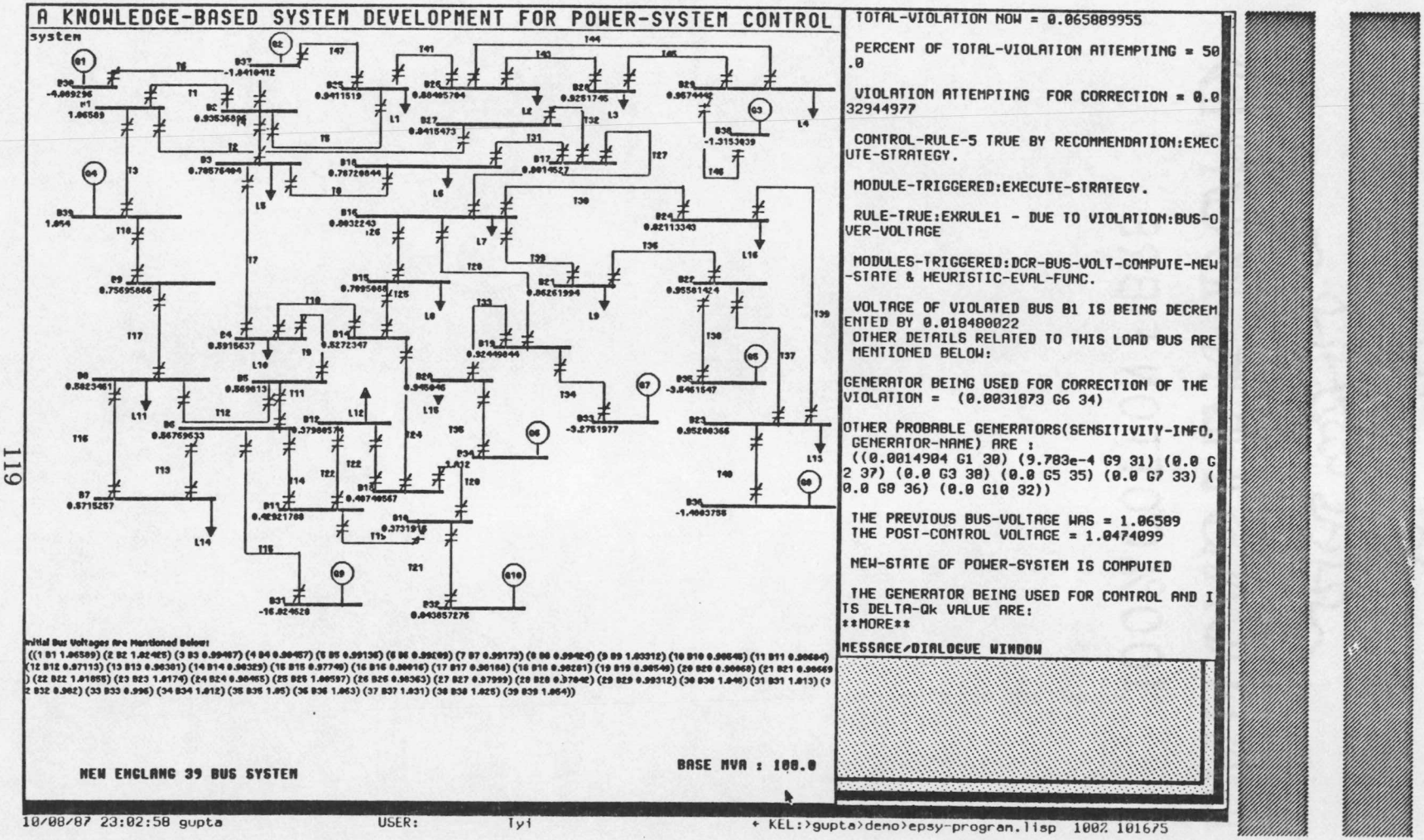
B4

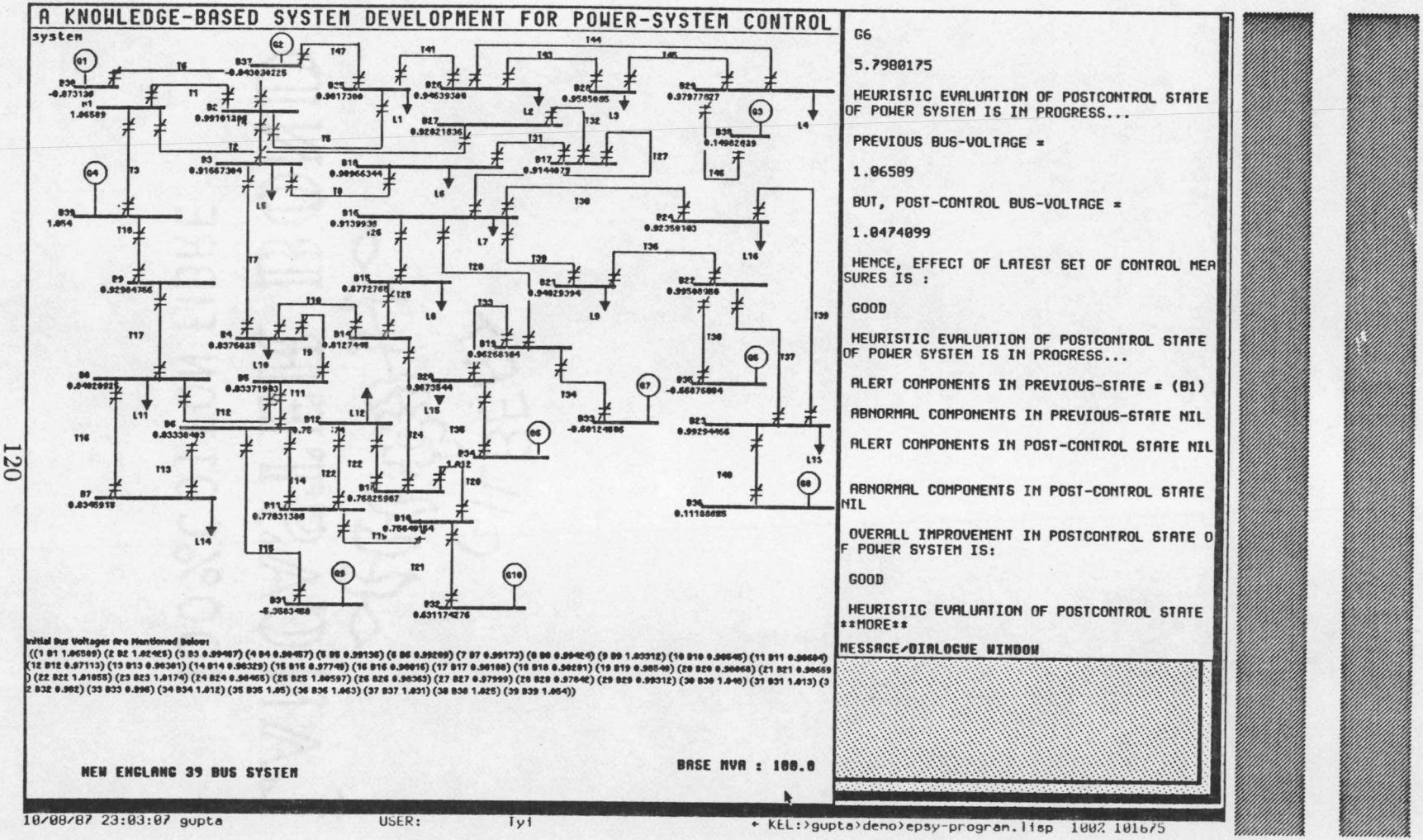
MORE

MESSAGE/DIALOGUE WINDOW

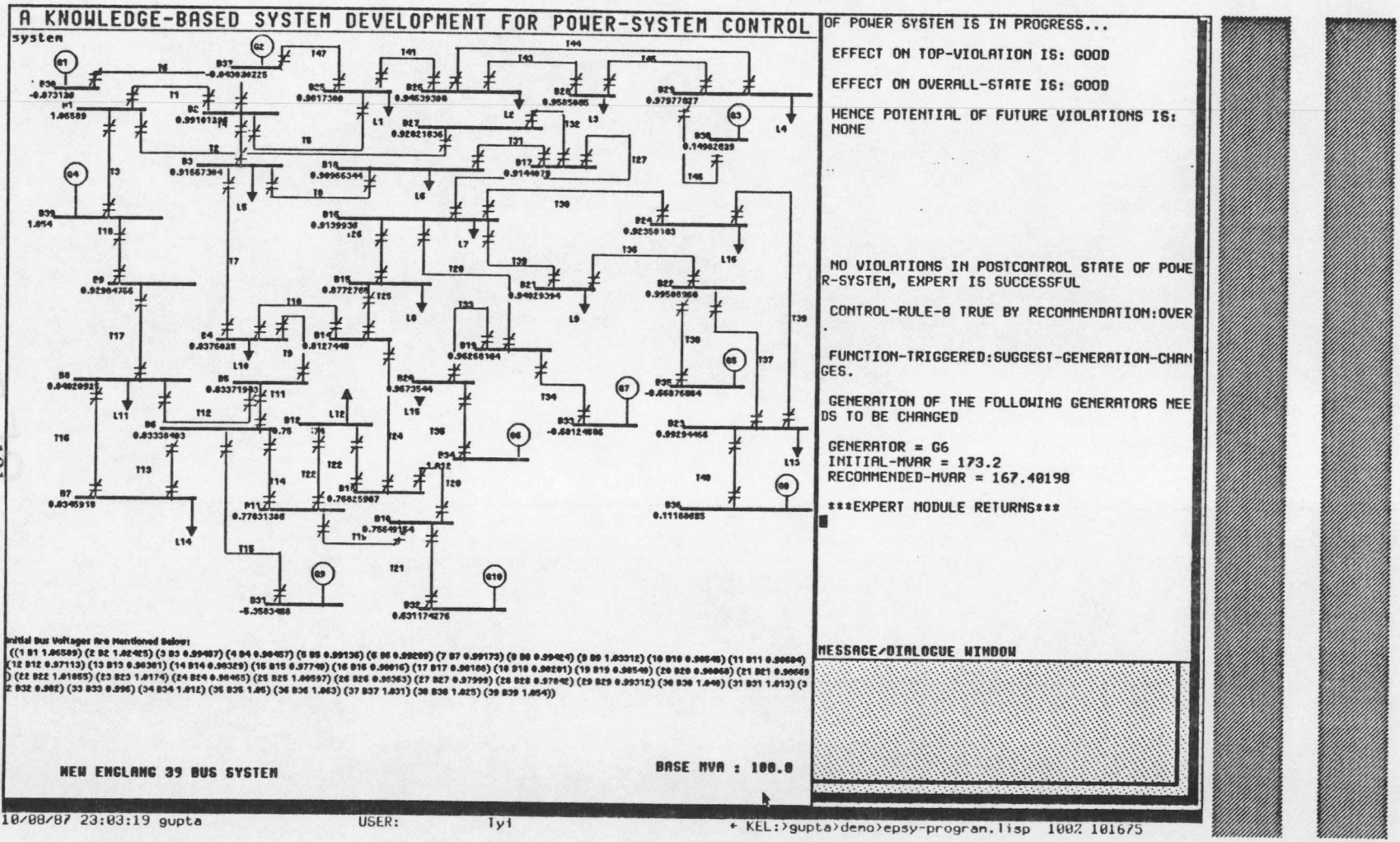
10/08/87 23:02:40 gupta USER: Tyi ← KEL:>gupta>demo>epsy-program.lisp 1002 101675


A KNOWLEDGE-BASED SYSTEM DEVELOPMENT FOR POWER-SYSTEM CONTROL
system
B8
B15
B11
B6
B13
B5
B7
B12
TOP-PRIORITY-COMP IS :
B12
WORSER POSTCONTROL STATE PROMPTS EXPERT TO BACKTRACK TO PREVIOUS STATE.
LEVEL OF CONTROL IS
3
RECOMMENDATION IS:
REPLAN-STRATEGY
CONTROL-RULE-4 TRUE BY RECOMMENDATION:PLAN -STRATEGY.
MODULE-TRIGGERED:PLAN-STRATEGY.
PLANNING-RULE-2 IS TRIGGERED
CONTROL-LEVEL = 3
MORE
MESSAGE/DIALOGUE WINDOW
Initial Bus Voltages Are Mentioned Below:
((1 B1 1.06589) (2 B2 1.02428) (3 B3 0.99487) (4 B4 0.99487) (5 B5 0.99136) (6 B6 0.99269) (7 B7 0.99173) (8 B8 0.99424) (9 B9 1.03312) (10 B10 0.98648) (11 B11 0.98684) (12 B12 0.97113) (13 B13 0.98381) (14 B14 0.98329) (15 B15 0.97749) (16 B16 0.98016) (17 B17 0.98188) (18 B18 0.98281) (19 B19 0.98549) (20 B20 0.98668) (21 B21 0.98669) (22 B22 1.01855) (23 B23 1.0174) (24 B24 0.98468) (25 B25 1.00597) (26 B26 0.98363) (27 B27 0.97999) (28 B28 0.97842) (29 B29 0.99312) (30 B30 1.048) (31 B31 1.013) (32 B32 0.982) (33 B33 0.996) (34 B34 1.012) (35 B35 1.05) (36 B36 1.063) (37 B37 1.031) (38 B38 1.026) (39 B39 1.064))
NEW ENGLAND 39 BUS SYSTEM
BASE MVA : 100.0
10/08/87 23:02:48 gupta
USER:
Iyi
KEL:>gupta>demo>epsy-program.lisp 1002 101675

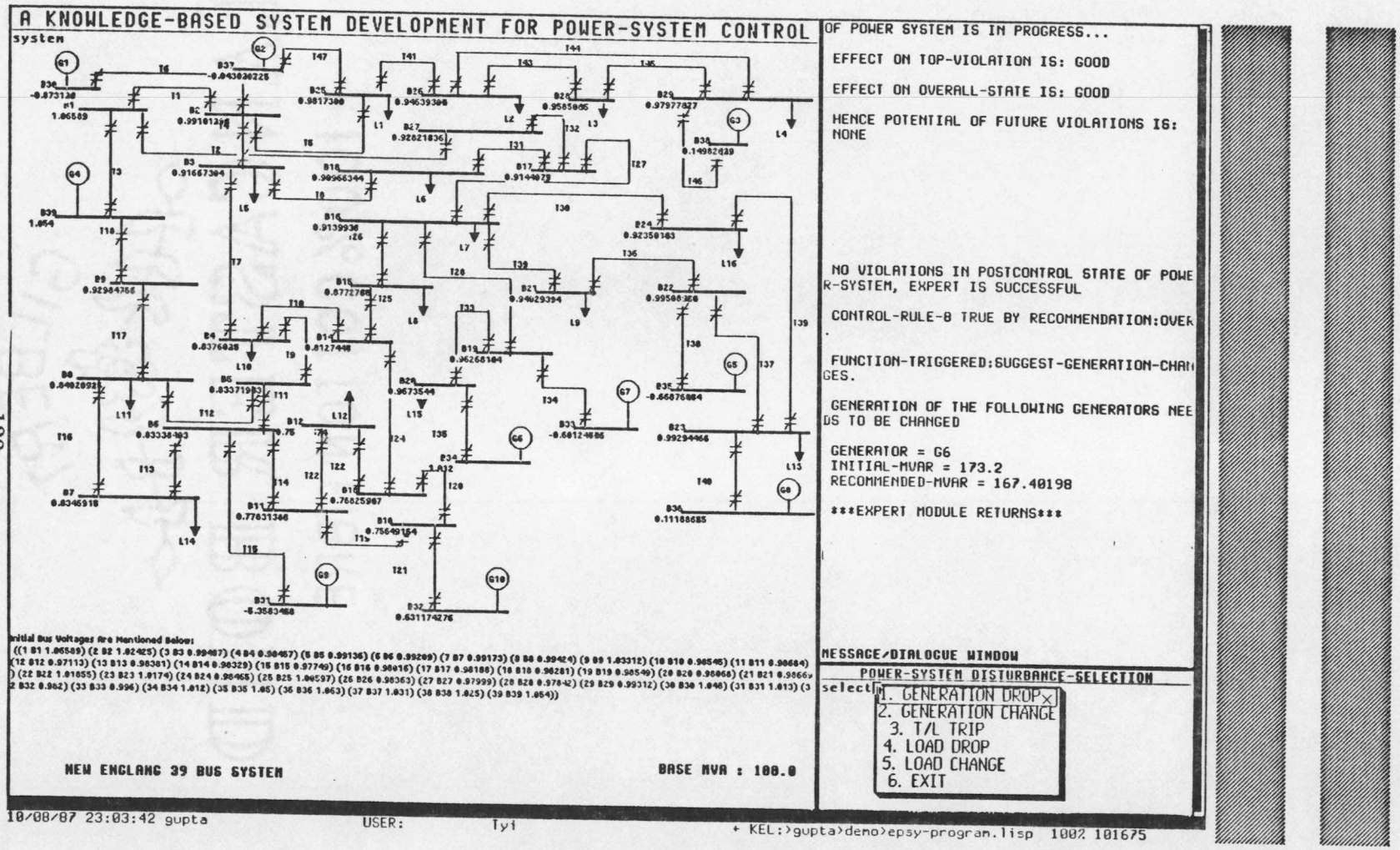

A KNOWLEDGE-BASED SYSTEM DEVELOPMENT FOR POWER-SYSTEM CONTROL
system
NEW ENGLAND 39 BUS SYSTEM
BASE MVA : 100.0
TOTAL-VIOLATION NOW = 0.065089955
PERCENT OF TOTAL-VIOLATION ATTEMPTING = 50.0
VIOLATION ATTEMPTING FOR CORRECTION = 0.032944977
CONTROL-RULE-5 TRUE BY RECOMMENDATION:EXECUTE-STRATEGY.
MODULE-TRIGGERED:EXECUTE-STRATEGY.
RULE-TRUE:EXRULE1 - DUE TO VIOLATION:BUS-OVER-VOLTAGE
MODULES-TRIGGERED:DCR-BUS-VOLT-COMPUTE-NEW-STATE & HEURISTIC-EVAL-FUNC.
VOLTAGE OF VIOLATED BUS B1 IS BEING DECREMENTED BY 0.018480022
OTHER DETAILS RELATED TO THIS LOAD BUS ARE MENTIONED BELOW:
GENERATOR BEING USED FOR CORRECTION OF THE VIOLATION = (0.0031873 G6 34)
OTHER PROBABLE GENERATORS(SENSITIVITY-INFO, GENERATOR-NAME) ARE :
((0.0014904 G1 30) (9.783e-4 G9 31) (0.0 G2 37) (0.0 G3 38) (0.0 G5 35) (0.0 G7 33) (0.0 G8 36) (0.0 G10 32))
THE PREVIOUS BUS-VOLTAGE WAS = 1.06589
THE POST-CONTROL VOLTAGE = 1.0474099
NEW-STATE OF POWER-SYSTEM IS COMPUTED
THE GENERATOR BEING USED FOR CONTROL AND ITS DELTA-Qk VALUE ARE:
MORE
MESSAGE/DIALOGUE WINDOW
10/08/87 23:02:58 gupta
USER:
Tyi
KEL:>gupta>demo>epsy-progran.lisp 1002 101675



10/08/87 23:03:07 gupta USER: Tyi ← KEL:>gupta>demo>epsy-program.lisp 1002 101675



10/08/87 23:03:19 gupta USER: lyi ← KEL:>gupta>demo>epsy-progran.lisp 1002 101675


A KNOWLEDGE-BASED SYSTEM DEVELOPMENT FOR POWER-SYSTEM CONTROL
system
NEW ENGLAND 39 BUS SYSTEM
BASE MVA : 100.0
OF POWER SYSTEM IS IN PROGRESS...
EFFECT ON TOP-VIOLATION IS: GOOD
EFFECT ON OVERALL-STATE IS: GOOD
HENCE POTENTIAL OF FUTURE VIOLATIONS IS: NONE
NO VIOLATIONS IN POSTCONTROL STATE OF POWER-SYSTEM, EXPERT IS SUCCESSFUL
CONTROL-RULE-8 TRUE BY RECOMMENDATION:OVER.
FUNCTION-TRIGGERED:SUGGEST-GENERATION-CHANGES.
GENERATION OF THE FOLLOWING GENERATORS NEEDS TO BE CHANGED
GENERATOR = G6
INITIAL-MVAR = 173.2
RECOMMENDED-MVAR = 167.40198
EXPERT MODULE RETURNS
MESSAGE/DIALOGUE WINDOW
POWER-SYSTEM DISTURBANCE-SELECTION
1. GENERATION DROP
2. GENERATION CHANGE
3. T/L TRIP
4. LOAD DROP
5. LOAD CHANGE
6. EXIT
10/08/87 23:03:42 gupta
USER:
KEL:>gupta>demo>epsy-program.lisp 1002 101675

VITA

Uday Kumar Gupta was born in Liverpool, England on March 8, 1962. He attended public schools in Bhopal, India. He received a Bachelor of Engineering degree with Honors in Electrical and Electronics from the University of Madras, India in July 1984. From July 1984 to August 1985 he worked as a Systems engineer with a major computer manufacturer in Bombay, India.

In the fall of 1985 he accepted a research assistantship in the Computer Science Department of the University of Tennessee Space Institute and begun study toward a Master's degree. This degree is expected in December 1987.