STEPS TOWARD AN ARTIFICIALLY INTELLIGENT CAI SYSTEM

AN INTERIM REPORT ON RESEARCH IN PROGRESS

John Seely Brown
Alan G. Bell
Frank Zdybel

July 1973

UCI ICS Technical Report ~~#25~~ 36

Acknowledgements

The work described in this paper was supported by the Air Force Human Resources Laboratory (Lowry AFB) under contract F41609-73-C-0006. Grateful acknowledgement is also made to Catherine Hausmann and Richard Burton for their assistance with the natural language aspects of this research.

TABLE OF CONTENTS

PART I: Overview

Introduction

During the last few years, several researchers in artificial intelligence (AI) have been investigating how best to direct the technological advances in AI toward the goal of producing more flexible CAI systems [1,2,3,4]. It seems almost axiomatic that the more a CAI system "understands" about its subject area, the more effective a teaching system it will be. Even more important is the fact that systems which possess such understanding can provide a qualitatively new level of interaction with the student. With a CAI system that not only embodies a thorough knowledge base of its subject but that also incorporates powerful inferencing or deductive capabilities, the student can pose his own problems or conjectures (theretofore unknown to the system) and receive intelligent responses.

In particular, if the student is confronted with a situation to be analyzed, he may happen to form hypotheses which are correct and yet were not anticipated by the author of the CAI lesson. Likewise, the student might create an involved hypothesis which is partially correct. If the system can "understand" his hypothesis, it can then decompose it, isolate the logically correct components and then generate a comprehensible explanation of what is wrong with the remainder. To be able to provide these services without having to have all possibilities pre-stored in its data files is a principal goal for any artificially intelligent CAI system.

Electronic Trouble Shooting

To explore and concretize the above mentioned goal, we have begun the construction of a CAI system for teaching the art of electronic trouble shooting. Our CAI system is to present (graphically) to the student a circuit schematic of a moderately complicated transistorized instrument and a partial specification of the symptoms displayed by this circuit for some particular but unknown fault. The student's task is to interact with the system and try to discover the actual identity of the fault. He is permitted to examine the external behavior of the instrument by requesting that certain measurements be made of its output under various settings of the controls. He can also request that certain tests be made at any internal point of the circuit. Before the system discloses the answers to these tests, however, it requires that the student specify precisely why he needs to know the information. In other words, the student must articulate his hypothesis or set of hypotheses, and relate how the requested test will apply to them (*). If he has just one hypothesis, then it alone is analysed by the system in order to determine if it is logically consistent with the currently known symptoms. If it is indeed consistent, then the result of his requested test is deduced and given to him. If his hypothesis is inconsistent with those symptoms which are known, or if it should have been rejected in considering the results of a prior test, then he is so informed and given an explanation of what he has done wrong. Finally, if he has several hypotheses, then each is checked for consistency and then a check is made to see if the test will, in fact, split his hypothesis space.

What features must such a system have? First of all, it must have a natural language processor which is capable of transforming statements expressed in English into a set of specifications couched in some internal language. Secondly, it must have a powerful inferencing mechanism which is not only capable of deducing the result of tests concocted by

(*) We feel that the objection that accomplished trouble shooters often cannot articulate their reasoning is irrelevant. During the training phase, students should be made to justify their reasoning.

the student, but also of determining whether a hypothesis is consistent with a given set of symptoms. Thirdly, it must have an answer generator which is able to accept information from the inferencer and compose an answer -- in English -- as to any detected logical discrepancies.

In the following sections we will describe, in fairly general terms, our approach to the inferencing mechanisms which comprise the core intelligence of this system. We will similarly discuss some of the natural language mechanisms that we are currently using. Following these two general descriptions, we will delve into some technical descriptions of these two major subsystems. Whenever possible, we will choose the IP-28 Heathkit regulated power supply to be the example of our discussions (see Appendix 3 for a brief technical description of this instrument).

Inferencing Mechanisms

The fundamental component of our inferencing mechanism is a simulation program which incorporates a model of the particular instrument under consideration. The basic theory of how this simulator performs inferences is straight-forward. Consider a hypothesis of the form:

If X, then Y

where X is a proposition about some circuit component and Y is a proposition about a symptom or a set of symptoms. An example of such a hypothesis is:

If C2 is shorted, then there will be no output voltage

where X is the proposition "C2 is shorted" and Y is the proposition "there will be no output voltage".

The simulator computes the logical consequences of the proposition X by <u>changing</u> the internal circuit model so as to make X true (e.g. by replacing C2 with a .1 ohm resistor) executing a simulation of this modified model and summarizing the results of the simulation run. This summary will contain descriptions of all the logical consequences of "X". If the ultimate task is to determine the truth of the hypothesis "if X then Y", then the proposition "Y" must be evaluated over the above mentioned summary.

The above paradigm glosses over numerous difficulties which are worth mentioning. The first of these concerns the boundary or input conditions on which to run the simulation. If the task is to describe all the logical consequences of a hypothetical modification, then this can be achieved through simulation if and only if the simulation can be run over all possible settings of the instrument's controls, and over all possible input and output conditions. While for most practical situations there are only a finite number of cases worth considering, this number can still be quite large. It is clearly desirable to have an additional inferencing mechanism which can determine what the worthwile cases are for any particular hypothesis. This additional mechanism must embody electronic knowledge of a different sort than is represented by the simulator. In particular, it is of a more axiomatic nature.

A second problem encountered in using a simulator as the chief inferencing mechanism is one of speed. Anyone familiar with simulation techniques is aware of the awesome amount of computation involved. However, certain types of questions require drastically less computational effort to answer through simulation than do others. For example, if a question can be answered through using just a DC simulation of a circuit, then one can expect reasonable response times. Likewise, certain AC analyses can be performed quickly, but deductions requiring transient analysis are extremely time consuming. This observation suggests two things. First, some preliminary analysis of a hypothesis is in order, if for no other reason than to determine what modes of simulation may be excluded. Second, if transient analysis is absolutely mandatory for the simulation run, then perhaps some other techniques (e.g. axiomatic) should be used to deduce the desired information, or at least some additional analysis should be performed so as to minimize the amount of transient analysis required.

The third problem concerns the inappropriateness of using a simulation inferencer for certain kinds of questions. In particular, a person knowledgeable about electronics can often qualitatively determine the effects of a hypothetical fault or modification to a circuit. His behavior in such cases is more analogous to common sense reasoning than to detailed quantitative calculation of the sort underlying a simulation. Ideally, we would like to be able to capture this kind of human reasoning, not only for the sake of efficiency, but also because it brings us closer to being able to model how the student is thinking about the circuit.

A State Model

Although the major emphasis of our research has been directed toward building a general inferencing scheme for deducing the consequences of a hypothetical fault, we have thoroughly analyzed the IP-28 instrument for the purpose of building a special purpose table-driven inferencer. The table-driven system not only acts as a back-up for the more interesting general purpose inferencer (for handling questions beyond its present scope) but also provides information as to the boundary conditions required for simulating various hypothetical faults. In addition, the table-driven system contains information as to the propagation of faults. That is, it specifies how one fault might cause another fault. Much of this information was actually derived from the simulation system, and hence functions as a summarizer for typical faults (a complete specification of the information used by this model is contained in Appendix 6).

Our final system will combine this simple table-driven scheme with the more ambitious general purpose one. In this way, answers to often asked questions can be more efficiently handled. For the unusual question or hypothesis, the general purpose inferencer can be invoked. Of course, we would expect the designer of a particular lesson to utilize the general purpose scheme in building the pre-stored tables for a new circuit or instrument.

Natural Language Processing

One aspect of an intelligent appearing CAI system is its ability to converse with the user in a comfortable or habitable subset of English. The need for such a capability is crucial in our system because we expect the student to justify his requests by stating his hypotheses. We are trying, however, to avoid doing predominantly linguistic research and are instead endeavoring to make maximal use of Woods' Augmented State Transition Network Parser [5]. Core size limitations have forced us to modify his parser so as to make it considerably smaller. We have also extended his grammar to cover more of the syntactic constructions which occur frequently in this domain. Finally, we have had to develop our own dictionary for electronic jargon.

Since the Woods parser has been well documented [5,6], we shall encourage the interested reader to consult the literature for the underlying theory and technical details of the parsing scheme. We shall, however, give several annotated examples of the output of this parser (see Appendix 2), in order to provide a feeling for the kinds of structural descriptions it produces. A summary of the modifications made to Woods' Parser appears in Appendix 1.

PART II: Technical Aspects

Handling DC-Type Inferencing

In this section we will describe how we use the simulation program (called SPICE) in conjunction with additional programs to perform DC type inferencing. The basic format for all queries to the inferencer are either of the form:

1) If X, then Y ("X","Y" propositions)

where "Y" might be implicitly specified as the symptom currently under consideration; or of the form:

2) What happens if X.

Of course, all such queries presuppose some particular circuit which has already been defined and identified prior to the query. Appendix 4 provides a simplified flow chart of the processing path, which we will now explain.

After the question is parsed and interpreted, it is passed to the question analyzer. The primary purpose of this block is to decide whether or not DC analysis is pertinent to the query. For example, if the proposition X is "C2 is open" (see schematic in Appendix 3), then the analyzer will reject DC analysis as being irrelevant, since our DC simulation program presupposes that all capacitors are open for the purpose of performing its analyses (such queries will be handled by alternative techniques not yet fully worked out).

If, on the other hand, the query concerned C2 being shorted, the analyser would select the DC inferencing mode. The question analyzer makes this decision on the basis of explicit knowledge contained in a conceptual network. In this latter case, the analyzer would determine that C2 was an instance of the generic entity "Capacitor" and that DC analysis should be invoked whenever this entity is "shorted".

In cases where DC analysis is accepted as a possible inferencing mode, the proposition X is passed to a set of procedures which change SPICE's internal model of the given circuit. These procedures must know how SPICE stores its information about the circuit and also how to change this model in order to make the proposition true. For example, it must know how to alter the topological description of the circuit so as to make a given capacitor appear to be shorted. More ambitiously, it might be expected to interpret what a student means when he says that a given transistor is bad. A transistor can be bad in a multitude of ways. Either the student must specify (under possible prodding by the system) which of these is intended, or else the system must create a set of possible interpretations and try each one, seeing if there are any which would make the proposition "Y" true.

Any proposed modification to the circuit is passed to another set of procedures whose responsibility is to determine a minimal set of conditions for running the simulation. The smarter these procedures are, the faster and more efficient the overall inferencing mechanism will be. At the moment, our approach to this problem is to access the conceptual network for explicit information as to what the boundary or input conditions should be for any particular hypothetical modification. However, much work remains to be done on this problem before a more general solution is found.

After both of the above phases are completed, the simulation model is executed over the specified setup and the results of the simulation are passed to the summarizer. The summarizer surveys the results of a simulation run and determines whether any component is dissipating too much power or is otherwise experiencing an over-taxing situation (voltage too high, etc.). If a circuit component has been subjected to excessive stress, the summarizer may decide to "blow" that component by recursively calling the question analyzer with a query concerning what happens if this new component is blown. This recursive call re-invokes all of the above described activities on the newly modified model. Recursive calls continue until no further excessive stresses are detected. The summarizer then pops back, constructing a complete causal history of what has transpired.

After such a history has been built, it is passed to the answer evaluator which either evaluates the proposition Y over this set of events or else creates a set of coherent descriptions to be passed on to the student. This latter task is, of course, an open ended one. For the moment, however, our approach is to utilize an action model (see Appendix 5) which specifies the communication flow between the functional blocks of the circuit.

The need for this frill is two-fold. First, we need it to decide which statements made by the student are tautological and therefore devoid of real information. For example, if the student is asked to specify the results of shorting C2, he might reply that the voltage drop across it will become zero. While this statement is perfectly true, it does not reveal any real understanding of the consequences. Second, the answer evaluator needs to have some global notion of causality from which to determine how best to synthesize meaningful units of discourse out of the complex answers reported by the summarizer. The action model provides a first step in this direction.

The Simulator

SPICE [7,8] is the electronic simulation program chosen for use in our system. SPICE was written by Laurence Nagel at the University of California at Berkeley. The original program consists of some 8,000 lines of FORTRAN source code and was originally designed to run on the CDC 6400. Input files for SPICE consist of a description of the topological arrangement of the circuit components and their nominal values, specifications of parameters for whatever models may be used for non-linear components such as BJT's and diodes, and a list of desired output options. Possible circuit components include independent and dependent voltage and current sources, field effect transistors, BJT's and diodes, as well as the usual linear devices. Provision is also made for specifying model devices not already included in SPICE's basic library. The unmodified program performs three different types of circuit analysis: DC operating point, AC response spectrum, and transient behavior.

SPICE's DC operating point analysis includes operating point values for model parameters as well as the voltages at each node of the circuit. Transient analysis consists of a series of DC analyses against a varying input voltage, with charge storage elements taken into account. The AC analysis subprogram simply determines DC operating points whose values are phasors rather than real numbers. AC solutions are expressed separately for the given input signal frequencies rather than as dependent functions of the frequency. Since AC analysis assumes that all nonlinear devices operate linearly sufficiently close to a particular DC operating point, it is of limited use for deducing if a particular modification has caused a component to instantaneously enter a saturation or cu-off state, etc. In addition, since the CPU time required for transient analysis is prohibitive (on the order of ten CPU-seconds for the circuit under consideration), we anticipate using chiefly the DC operating point analysis.

The DC analysis subprogram is divided into three sections: READIN, SETUP, and ITERATION. READIN accepts the description of the circuit and inserts this information into the various internal arrays. Array-stored information includes the nodes to which each element is connected, the value of each element, and the type of each component.

Currently, READIN simplifies the external description (e.g. Re-numbering the nodes so that they are consecutive) and checks for possible specification errors.

SETUP allocates space in the sparse matrix and stores the pointers to it. It first checks each component and determines which locations of the array will be occupied. It then adds whatever locations are required for forming intersections with other locations already allocated. Finally, SETUP stores the appropriate pointers for each component.

The ITERATION section solves the internal representation created by READIN and SETUP. The method of solution is based on Kirchoff's and Ohm's Laws. The basic matrix equation is:

$$E = Y^{-1} I$$

where I is the vector of source currents for the nodes, Y is the matrix of mutual conductances between nodes, and E is the resultant vector of node voltages. The Y matrix is "loaded" by adding the conductance of each component to the two diagonal terms and subtracting it from the off diagonal ones. The I vector is then loaded with the values of the known current sources. After these are loaded for all the circuit components, the Y matrix is inverted by the

Doolittle method [9] and multiplied by the I vector. This obtains the voltage for each node.

The above method would be all that would be required if the circuit contained only elements with linear I-V curves. However, transistors and diodes have exponential curves. These non-linear characteristics are handled by an iterative process. A "best guess" of the voltage drop across a diode, for example, is computed and used to determine a point on the I-V curve. The inverse of the derivative at that point is then determined and used as the value of the effective conductance, thus modifying the Y matrix. The I intercept of this derivative is then determined and used to modify the I vector. The new Y matrix is then inverted and the basic matrix equation is re-solved for the new values. The resulting E vector is then used to replace the previous "best guess" and the above sequence of events is repeated. This process may be thought of as forming a Taylor series approximation to the non-linear I-V curve. After each iteration the newly obtained voltages are compared with those produced by the previous iteration. If the voltages are within 0.1% or 50 microvolts of each other, whichever is greater, then the analysis is complete.

In certain circuits, the iterative process will fail to converge. If 100 iterations are completed without convergence the analysis is stopped. Fortunately this happens primarily with bistable circuits, a situation that should not arise in our case. Also, good convergence may not be achieved due to the rapidly changing nature of the exponential equation. To speed up convergence by limiting overshoot, a limit of 1.4 volts is placed on the amount of change in the voltage across semiconductor junction.

Modifications

The modifications made to SPICE began with the removal of all unnecessary code dealing with AC and transient analysis, and the removal of component models, like those for MOSFET transistors, which are not likely to be used in the circuits being simulated. The size of the matrices was also reduced, since the branch and node capacities of the program were greater than required. This modified program is approximately 3500 lines of FORTRAN code.

The major modification is the introduction of a mechanism to introduce faults and circuit modifications quickly. One method of achieving this would be to modify the external description of the circuit and read it in each time the effect of a fault is to be sought. In the case of a transistor with an open junction, for example, the input statements to SPICE would be changed to show a diode for the good junction and an open circuit for the defective one. The circuit description would then be analyzed in the manner described above. It was felt, however, that this would be too slow.

The method chosen instead was to change the models of all the components so that they take an additional piece of information from a fault array. The number of faults which can be captured in this manner is rather large. For a transistor, it includes any junction being open or shorted, any lead being disconnected, or the beta being wrong. The fault array used contains one location for each element. Coefficients are drawn from it as needed to change the nature of a device. For example an open junction of a transistor would be characterized by an infinite resistance (0 conductance) and a shorted junction would use a resistance of 0.1 ohm.

New Models

In order to increase the speed of SPICE, new circuit elements will be added. Most circuits have a functional nature and can be broken down into a small number of functional blocks. In the circuit under investigation, the external behavior of these blocks is well known. In order to avoid recomputing the internal behavior of normally functioning blocks, equivalent functions for them are being hand-coded into SPICE's FORTRAN source file. In this manner, seven different versions of the circuit will be created. Each version will have one functional block expanded to the component level so that individual components can be modified or inspected. One example of a block in the power supply would be the constant current source. One should note in this case that if the circuit is operating correctly, that the model for the current source does not have to be voltage limited, since the output voltage will never attempt to exceed the supply voltage level. However, a fault could be introduced outside this block which would necessitate an effective voltage limit. Considerations such as this must be taken into account when designing the blocks. The actual implementation of these block models is very similar to that for the transistor models. The same iterative approach is used. The only restriction on what can be modelled is that the derivative of the transfer function must be continuous. The structure of these block models is, of course, very dependent upon the particular circuit in which they are to appear.

BIBLIOGRAPHY

1. Wexler, J. D., Information Networks in Generative Computer Assisted Instruction; IEEE Transactions on Man-Machine Systems Vol. MMS-11, No. 4 (Dec. '70) pp. 181-189

2. Koffman, E., Generative Computer Assisted Instruction: an Application of Artificial Intelligence to CAI; First USA-Japan Computer Conference (1972), Session 2-3-2 pp. 28-35

3. Brown, J. S., Burton, R., and Zdybel, F. A Model Driven Question-Answering System for Mixed Initiative Computer Assisted Instruction; IEEE Transactions on Systems, Man and Cybernetics Vol. SMC-3, No. 3 (May '73) pp. 248-257

4. Koffman, E. and Blount, S., Artificial Intelligence and Automatic Programming in CAI; Third International Joint Conference on Artificial Intelligence (1973), Session 5-3

5. Woods, W. A., "An Experimental Parsing System for Transition Network Grammars"; Natural Language Processing Randall Rustin, Editor, Algorithmics Press Inc., 1973

6. Woods, W. A. and Kaplan, R. M., The Lunar Sciences Natural Language Information System BBN Rpt. # 2265, Bolt Beranek and Newman, Inc. Cambridge, Mass. Sept. '71

7. Nagel, L. W. and Pederson, D. O., SPICE: Simulation Program with Integrated Circuit Emphasis; -emorandum # ERL-M382, Electronics Research Laboratory, College of Engineering, University of California at Berkeley, 12 April, '73

8. Nagel, L. W., Transient Analysis of Large Non-linear Networks; IEEE Journal of Solid State Circuits, Vol. SC-6, No. 4 (Aug. '71) pp. 166-182

9. Fox, L., An Introduction to Numerical Linear Algebra Oxford University Press, New York, 1965, pp. 60-65,, 99-102

APPENDIX 1

Modifications to Woods' Parser

In order to explore problems in the area of Natural Language, Woods' parser [5,6] was converted to UCI LISP. The major problem encountered in this conversion was that of reducing the parser to a size small enough to run on a machine with much less core then originally envisioned. This reduction was accomplished by removing the following parts. SYSCONJ and all of the information saved for use by SYSCONJ, including the functions ALT CONJGEN, CONJOIN, CONJSCOPE, CONJ STARTS, LONGBLOCK, POPCONJ, TRAIL, TRAIL1 AND TRAILS were removed, as well as those parts of PARSER and STEP. Removing these items eliminates the ability to parse most occurrances of conjunctions (it is still possible to allow conjoined proper nouns by means of a grammar arc). BACKUP, which uses TRAIL information was removed, as were ASSIST and FORCER, which were deemed unnecessary for the sort of grammar we anticipated developing. SPOP and the Well Formed Sub String Facility (WFSS) were removed, along with a number of the utility routines used in the syntactic portion of Woods' system.

The Random Access Disk I/O routines necessary for maintaining a dictionary on DISK were written and added to

UCI LISP, but were not used in the smaller versions of the parser. The system resulting from all of these reductions requires some 50 K of core when run interpretively (25 K of which is the basic LISP system) and 45 K when run compiled. The files necessary to run the parser are: LOADIT.LSP, INIT, UTL.LSP, PAR.LSP, ARC.LSP, DIC.LSP, MOR.LSP, LEX.LSP, GMF.LSP, VARS.LSP, GMR.LSP and CHANGE.LSP. LOADIT.LSP contains an initialization function which loads the remaining files. INIT and UTL.LSP contain utility routines. PAR.LSP is the body of the parser. ARC.LSP and GMF.LSP contain the functions executed as pre actions. DIC.LSP, MOR.LSP, and LEX.LSP contain the tables and functions used in morphological stripping. VARS.LSP contains the global variable used by the parser, as well as some user interface functions. GMR.LSP is the grammar, and CHANGE.LSP consists of additions made in order to define new words.

In addition to all of the parser functions, before the parser is used a dictionary must be included. The present electronics dictionary resides on ELDEFS.LSP and provides examples of definitions of words. For a further explanation, see Woods [5].

APPENDIX 2

Annotated Parsing Examples

GLOSSARY OF TERMS

S	sentence	NP	noun phrase
DCL	declarative	DET	determiner
IMP	imperitive	ADJ	adjective
Q	question	N	noun
COMPL	complement	NU	number
NEG	negative	SG	singular
		PL	plural
AUX	auxilary	PRO	pronoun
VP	verb phrase		
V	verb	PP	prepositional phrase
TNS	tense	PREP	preposition
PRES	present	ADV	adverb

```
(PARSE @(D3 IS OPEN))


S DCL
  NP DET NIL
     N D3
     NU SG
  AUX TNS PRES
  VP V ADJ OPEN

SHALL I TRY FOR OTHER POSSIBILITIES ? N

NO. OF PARSES: 1
PARSE TIME: 0.83
NO. OF CONSES 472
NIL
*
```

In sentences where there is a copula verb followed by a predicate adjective, the adjective is made the new verb.

```
(PARSE @(C2 IS SHORTED))


S DCL
  NP PRO SOMETHING
  AUX TNS PRES
  VP V SHORT
     NP DET NIL
        N C2
        NU SG

SHALL I TRY FOR OTHER POSSIBILITIES ? N

NO. OF PARSES: 1
PARSE TIME: 0.88
NO. OF CONSES 577
NIL
*
```

Passive construction. The parser adds "something" as the subject.

```
(PARSE @(PLEASE TELL ME THE OUTPUT VOLTAGE UNDER THE FOLLOWING SETTINGS
))


S IMP
  NP PRO YOU <-------------------------- Imperative sentence.
  AUX TNS PRESENT                        The parser makes the subject
  VP V TELL                              the implied "you".
     NP DET THE
        ADJ NP N OUTPUT
        N VOLTAGE
        NU SG
        PP PREP UNDER
           NP DET THE
              ADJ FOLLOWING
              N SETTING
              NU PL
     ADV PLEASE
     PP PREP TO <----------------------- Indirect object "me" is written
        NP PRO I <                       in prepositional phrase format.
           NU SG  \--------------------- Objective case pronouns are written
                                         in the subjective case.
SHALL I TRY FOR OTHER POSSIBILITIES ? N

NO. OF PARSES: 1
PARSE TIME: 1.63
NO. OF CONSES 1105
NIL
*
```

```
(PARSE @(TRY TO GET THE MAXIMUM CURRENT THROUGH THAT LOAD))


S IMP
  NP PRO YOU  <-----------------   The implied "you" has been
  AUX TNS PRESENT                   brought down to be the subject
  VP V TRY                          of the embedded sentence.
     COMPL NOM S FOR-TO  <---
                 NP PRO YOU
                 AUX TNS PRESENT    The complement is a sentence.
                 VP V GET           The type of sentence is FOR-TO.
                    NP DET THE      The "for" is implied, that is:
                       ADJ MAXIMUM  "You try for you to get the max-
                       N CURRENT     imum current through that load."
                       NU SG
                       PP PREP THROUGH
                          NP DET THAT
                             N LOAD
                             NU SG

SHALL I TRY FOR OTHER POSSIBILITIES ? N

NO. OF PARSES: 1
PARSE TIME: 2.22
GARBAGE COLLECT TIME: 2.08  <-------   Some time was spent
TOTAL TIME: 4.30                        garbage collecting
NO. OF CONSES 1593
NIL
*
```

```
(PARSE @(SUPPOSE C2 ISN'T SHORTED))
```

"Isn't" is rewritten "is not" before the parse commences.

```
S IMP
  NP PRO YOU  <------------
  AUX TNS PRESENT
  VP V SUPPOSE
     COMPL THAT S DCL
                  NEG  <------------
                  NP PRO SOMETHING
                  AUX TNS PRES
                  VP V SHORT
                     NP DET NIL
                        N C2
                        NU SG

SHALL I TRY FOR OTHER POSSIBILITIES ? N

NO. OF PARSES: 1
PARSE TIME: 2.13
NO. OF CONSES 1476
NIL
*
```

In an imperitive sentence the parser makes the subject the implied "you".

From "not" of "is not"

```
(PARSE @(WHAT HAPPENS IF CAPACITOR C2 BECOMES SHORTED))


S Q
  NP DET WHQ
     N THING  <------------------     "WHQ THING" or "what thing" from the
     NU SG/PL                         definition of the Qword "what".
  AUX TNS PRESENT
  VP V HAPPEN
     COMPL ADV S DCL
                 NP PRO SOMETHING
                 AUX TNS PRES
                 VP V SHORT
                    NP DET NIL
                       ADJ NP N CAPACITOR
                       N C2
                       NU SG

SHALL I TRY FOR OTHER POSSIBILITIES ? N

NO. OF PARSES: 1
PARSE TIME: 1.88
NO. OF CONSES 1086
NIL
*
```

```
(PARSE @(HOW DOES THIS AFFECT THE OUTPUT VOLTAGE))


S Q
  NP PRO THIS
     NU SG
  AUX TNS PRES
  VP V AFFECT
     NP DET THE
        ADJ NP N OUTPUT
        N VOLTAGE
        NU SG
     ADV HOW

SHALL I TRY FOR OTHER POSSIBILITIES ? N

NO. OF PARSES: 1
PARSE TIME: 1.33
NO. OF CONSES 891
NIL
*
```

Note how the handling of "how" differs from the handling of "what" on the previous page. "How" is being used as an adverb. "What" (rewritten WHQ THING) is being used as a noun.

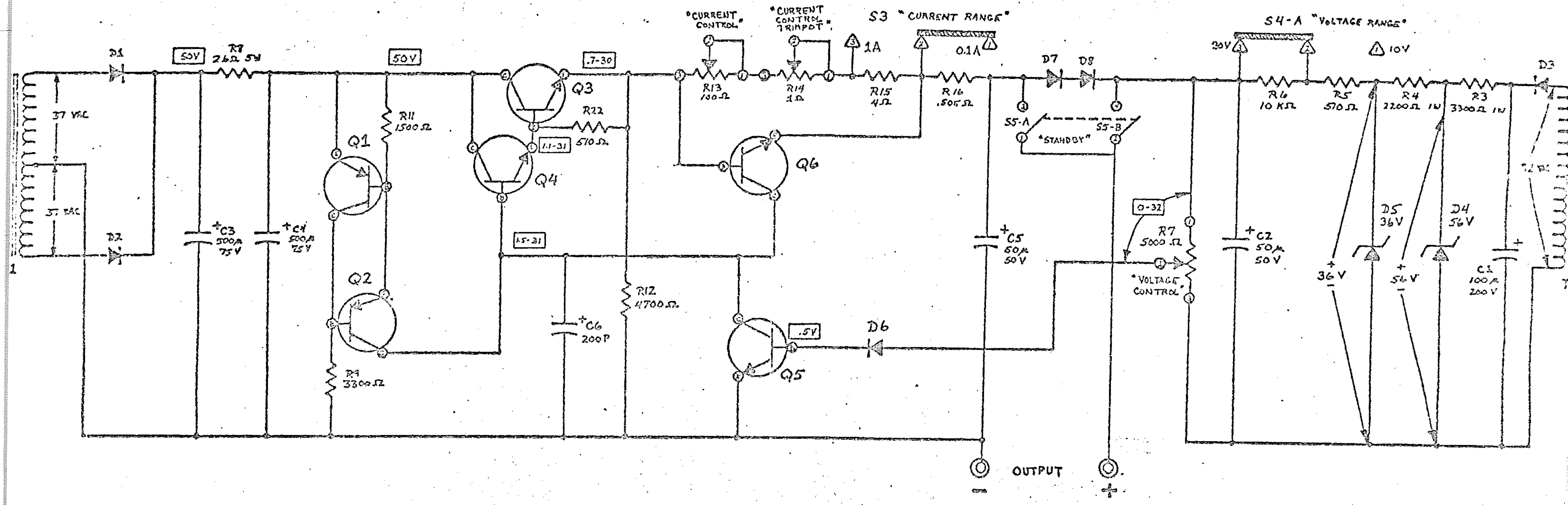
APPENDIX 3

Circuit Description

The internal DC supply voltage level is maintained by means of a full wave rectifier (diodes D1 and D2) connected to the center-tapped winding of the power transformer. A Pi configuration filter (resistor R8 and capacitors C3 and C4) reduces the ripple in the DC supply voltage to manageable proportions (about 4 volts AC remains). Transistors Q1 and Q2 and resistors R9 and R11 form a constant current source. This particular configuration gives good stability over voltage level variations twice as large as may be expected at the output of the filter, as well as assuring temperature stability. The output of the constant current source drives the Darlington pair pass transistor consisting of transistors Q3 and Q4 and resistors R22 and R12. Q3, a power transistor, is capable of handling the desired currents (about 1 Amp), but is of relatively low beta. Q4's beta is very high (nominally 150 - 300); their combination assures both good capacity and quick response. R22 insures that the Darlington will cut off completely in situations where the collector - base leakage current of Q3 becomes significant. R12 is a steady state load which assures that Q3 does not cut off completely, and that minimum output voltage can be reached quickly.

Switching transistors Q5 and Q6 control the amount of current which actually flows into the base of Q4 from the output of the constant current source. Current limiting is achieved via resistor R15 and rheostats R14 (Current Control) and R13 (Trimpot). The voltage across this series resistance determines how much current Q6 will draw away from the base of the Darlington pair.

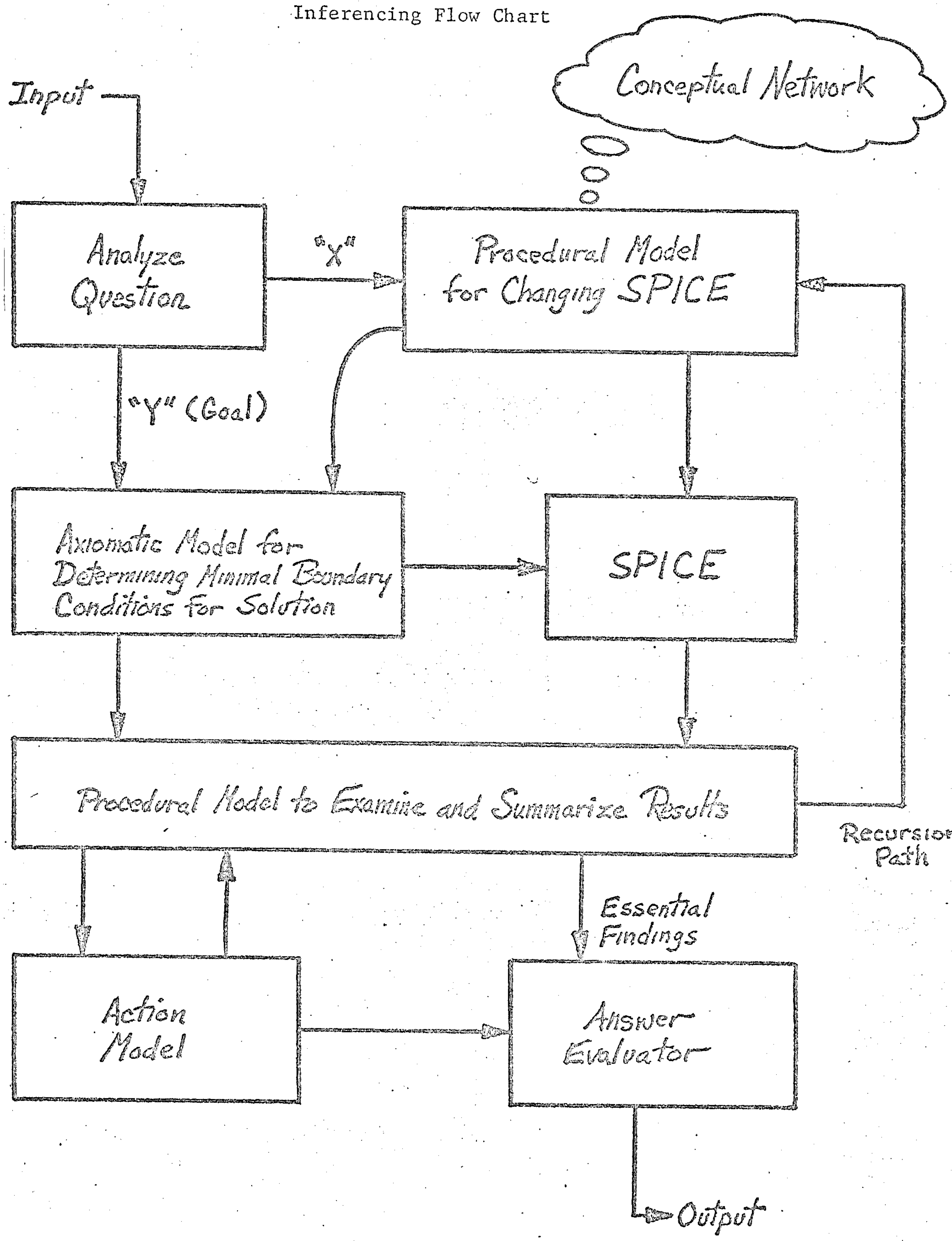
Voltage regulation is achieved via comparison of the output of the Darlington with the output of a reference voltage, the source of which is the half-wave rectifier on the other winding of the power transformer (diode D3). The output of the half wave rectifier is filtered by capacitor C1. Zener diode ZD4 insures that voltage will drop across R3 to a steady 56 volts (the output of the rectifier, and whatever ripple that remains after C1, is up around 80 volts). ZD5 and R4 act similarly to drop this voltage in turn to 36 volts. The use of two zener diodes insures that the output (final) zener will remain in a temperature stable region. The very clean reference voltage thus produced is applied across rheostat R7 (Voltage Control), one terminal of which is connected to the output of the power supply. The difference between the voltage drop in the reference branch of the circuit to the wiper of R7 and the voltage across the output load appears across the base - emitter junction of Q5, thereby controlling the amount of current that transistor will draw away from the base of the Darlington pair. C5 is used to reduce the output impedance of the power supply, and C6 insures that the device will not oscillate due to switching transients.



Circuit Schematic of IP-28 Power Supply

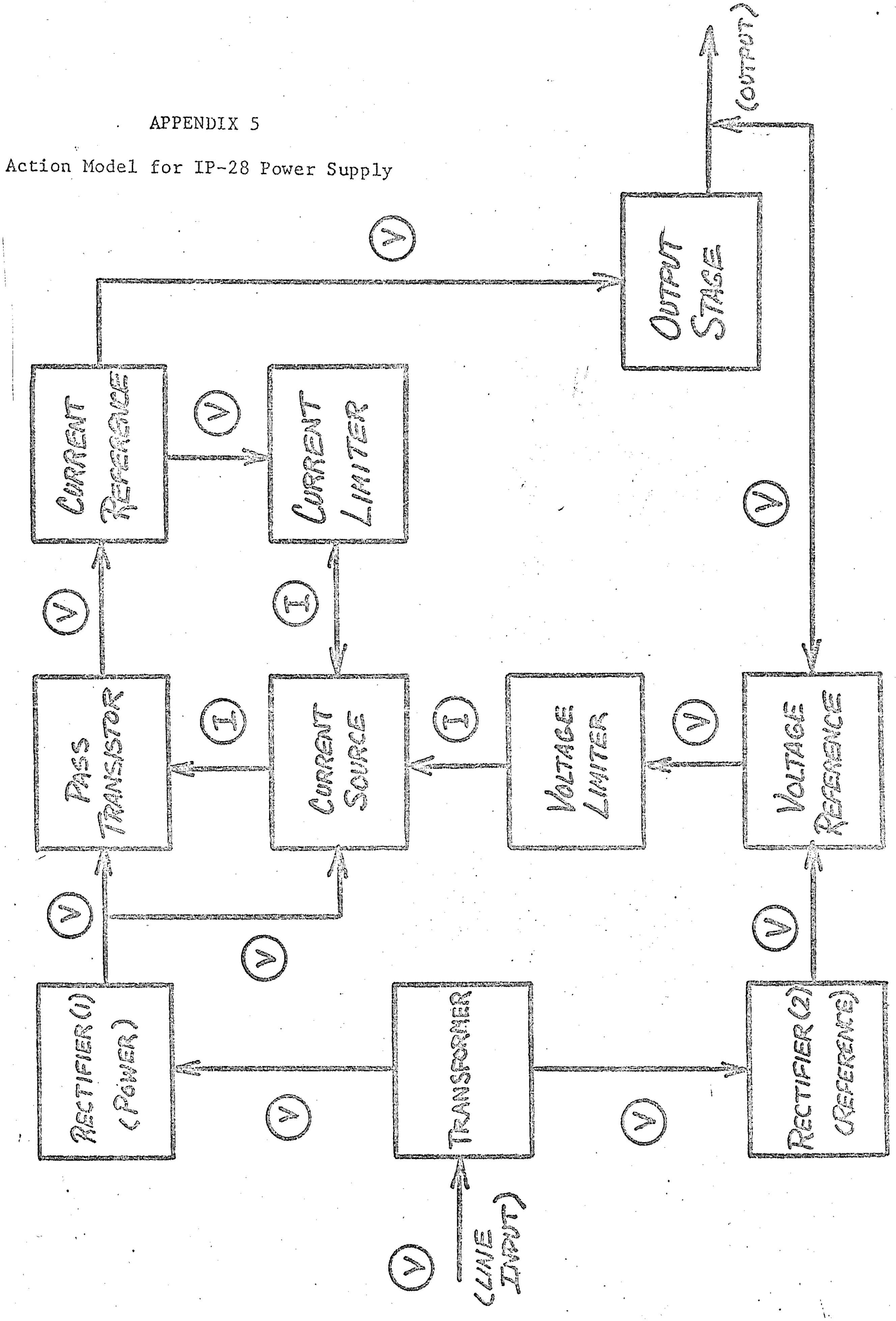
APPENDIX 4

Inferencing Flow Chart


Conceptual Network
Input
Analyze Question
"X"
Procedural Model for Changing SPICE
"Y" (Goal)
Axiomatic Model for Determining Minimal Boundary Conditions for Solution
SPICE
Procedural Model to Examine and Summarize Results
Recursion Path
Essential Findings
Action Model
Answer Evaluator
Output

APPENDIX 5

Action Model for IP-28 Power Supply



APPENDIX 6

State Model Tables for the IP-28 Power Supply

I. Input Space L

Let L be a vector of the following input conditions. Lvr, the setting of the voltage range switch; Lcr, the setting of the current range switch; Lxld, the resistance of the load across the outputs of the power supply; Lstb, the position of the standby switch; Lcc, the setting of the current control rheostat; Lvc, the setting of the voltage control rheostat.

L = (Lcc,Lvc,Lvr,Lcr,Lstb,Lxld)

Lvr = "10 volts","30 volts"

Lcr = ".1 amp","1 amp"

Lxld = "0","infinite","30 ohms"

Lstb = "DC On","Standby"

Lcc = "Full Off","Low","Med","High","Max"

Lvc = "Full Off","Low","Med","High","Max"

II. Symptom Space S

Let (Sj) denote the Symptom Space over a particular Input condition. The Combined Symptom Space S is composed of n-tuples of the form:

(S1,S2,S3, ... ,Sn)

Let (S1) denote the Symptom Space over the following input condition:

L = (Lcc,Lvc,"30 volts",Lcr,"DC On","infinite")

where Lvc is a defining variable and Lcc and Lcr may assume any value. Then:

(S1 = 1) == Normal (no symptom)

(S1 = 2) == No output voltage and no dependency on voltage control.

(S1 = 3) == ($0 \leq$ output voltage $\leq$ .7 volts) and no dependency on voltage control

(S1 = 4) == (30 volts $\leq$ output voltage) and no dependency on voltage control

(S1 = 5) == Output voltage cannot be made to extend down to .7 volts

(S1 = 6) == Output voltage cannot be made to extend up to 30 volts

Let (S2) denote the Symptom Space over the following input condition:

L = (Lcc,Lvc,"10 volts",Lcr,"DC On","infinite")

where Lvc is a defining variable and Lcc and Lcr may assume any value. Then:

(S2 = 1) == Normal (no symptom)

(S2 = 2) == No output voltage and no dependency on voltage control

(S2 = 3) == ($0 \leq$ output voltage $\leq$ .7 volts) and no dependency on voltage control

(S2 = 4) == (10 volts $\leq$ output voltage) and no dependency on voltage control

(S2 = 5) == Output voltage cannot be made to extend down to .7 volts

(S2 = 6) == Output voltage cannot be made to extend up to 10 volts.

Let (S3) denote the Symptom Space over the following input condition:

L = (Lcc,Lvc,Lvr,"1 amp","DC On","0")

where Lcc is a defining variable and Lvc and Lvr may assume any value. Then:

(S3 = 1) == Normal (no symptom)

(S3 = 2) == No output current and no dependency on current control

(S3 = 3) == (1 amp ≤ output current) and no dependency on current control

(S3 = 4) == Output current cannot be made to extend down to .01 amp

(S3 = 5) == Output current cannot be made to extend up to 1 amp.

Let (S4) denote the Symptom Space over the following input condition:

L = (Lcc,Lvc,Lvr,".1 amp","DC On","0")

where Lcc is a defining variable and Lvr and Lvc may assume any value. Then:

(S4 = 1) == Normal (no symptom)

(S4 = 2) == No output current and no dependency on current control

(S4 = 3) == (.1 amp ≤ output current) and no dependency on current control

(S4 = 4) == Output current cannot be made to extend down to .001 amp

(S4 = 5) == Output current cannot be made to extend up to .1 amp

Let (S5) denote the Symptom Space over the following input condition:

L = ("Max","Max","30 volts","1 amp","DC On","30 ohms")

Then:

(S5 = 1) == Normal (no symptom)

(S5 = 2) == No presence in this segment of Symptom Space

(S5 = 3) == Circuit does not respond fully to 100≤ increase or decrease in current load within 25 microseconds

(S5 = 4) == AC output voltage exceeds 5 milliwatts.

III. Failure Space F

F1 = Open

F2 = Shorted

F3 = Resistance is too high

F4 = Resistance is too low

F5 = Beta is too low

(Emitr-Base/Emitr-Collr/Base-Collr)

F6 = (OK/OP/OP)

F7 = (OP/OP/OK)

F8 = (OP/OK/OP)

F9 = (OP/OP/OP)

F10 = (OP/SH/OP)

F11 = (OP/OP/SH)

F12 = (SH/OP/OP)

F13 = (SH/SH/SH)

IV. Part Space P

C1,C2,C3,C4,C5,C6
D1,D2,D3,D6,D7,D8
FUSE1
Q1,Q2,Q3,Q4,Q5,Q6
R3,R4,R5,R6,R7A,R7B,R8,R9
R11,R12,R13,R14,R15,R16,R22
S3,S4,S5A,S5B
T1(P),T1(S1A),T1(S1B),T1(S2)
ZD4,ZD5

V. Component Space C

Capacitor
Diode
Fuse
Inductor
Resistor/Rheostat
Switch
Transformer winding
Transistor
Zener diode

VI. Analysis Space A (tentative arrangement)

"AC","DC","Transient","Axiomatic"

VII. Mapping Fp: F x P => S

P = C1	F = F1	S = (6,6,1,1,2-
	F = F2	S = (3,3,1,1,2)
	F = F4	S = (6,6,1,1,2)
P = C2	F = F1	S = (1,1,1,1,4)
	F = F2	S = (3,3,1,1,2)
	F = F4	S = (6,6,1,1,2)
P = C3	F = F1	S = (1,1,1,1,4)
	F = F2	S = (2,2,2,2,2)
	F = F4	S = (6,6,1,1,2)

P = C4	F = F1	S = (1,1,1,1,4)
	F = F2	S = (2,2,2,2,2)
	F = F4	S = (6,6,1,1,2)
P = C5	F = F1	S = (1,1,1,1,4)
	F = F2	S = (2,2,2,2,2)
	F = F4	S = (6,6,1,1,2)
P = C6	F = F1	S = ("High Load Oscillation")
	F = F2	S = (2,2,2,2,2)
	F = F4	S = (6,6,5,5,2)
P = D1	F = F1	S = (6,6,1,1,2)
	F = F2	S = (2,2,2,2,2)
P = D2	F = F1	S = (6,6,1,1,2)
	F = F2	S = (2,2,2,2,2)
P = D3	F = F1	S = (3,3,1,1,2)
	F = F2	S = (6,6,1,1,2)
P = D6	F = F1	S = (4,4,1,1,4)
	F = F2	S = (1,1,1,1,1)
P = FUSE1	F = F1	S = (2,2,2,2,2)
P = Q1	F = F5	S = (5,5,4,4,1)
	F = F6	S = (5,5,4,4,1)
	F = F7	S = (1,1,1,4,1)
	F = F8	S = (5,5,4,4,4)
	F = F9	S = (5,5,4,4,1)
	F = F10	S = (2,2,2,2,2)
	F = F11	S = (2,2,2,2,2)
	F = F12	S = (5,5,4,4,1)
	F = F13	S = (2,2,2,2,2)
P = Q2	F = F5	S = (1,1,1,1,1)
	F = F6	S = (2,2,2,2,2)
	F = F7	S = (2,2,2,2,2)
	F = F8	S = (2,2,2,2,2)
	F = F9	S = (2,2,2,2,2)
	F = F10	S = (5,5,3,3,2)
	F = F11	S = (2,2,2,2,2)
	F = F12	S = (2,2,2,2,2)
	F = F13	S = (5,5,3,3,2)

P = Q3	F = F5	S = (1,1,5,5,2)
	F = F6	S = (1,1,5,5,2)
	F = F7	S = (2,2,2,2,2)
	F = F8	S = (1,1,5,5,2)
	F = F9	S = (1,1,5,5,2)
	F = F10	S = (4,4,3,3,2)
	F = F11	S = (1,1,5,5,2)
	F = F12	S = (1,1,5,5,2)
	F = F13	S = (4,4,3,3,2)
P = Q4	F = F5	S = (6,6,5,5,2)
	F = F6	S = (6,6,2,2,2)
	F = F7	S = (2,2,2,2,2)
	F = F8	S = (2,2,2,2,2)
	F = F9	S = (2,2,2,2,2)
	F = F10	S = (4,4,3,3,2)
	F = F11	S = (2,2,2,2,2)
	F = F12	S = (2,2,2,2,2)
	F = F13	S = (4,4,3,3,2)
P = Q5	F = F5	S = (1,1,1,1,4)
	F = F6	S = (4,4,1,1,2)
	F = F7	S = (4,4,1,1,2)
	F = F8	S = (2,2,2,2,2)
	F = F9	S = (4,4,1,1,2)
	F = F10	S = (2,2,2,2,2)
	F = F11	S = ("ANOMALOUS")
	F = F12	S = (4,4,1,1,2)
	F = F13	S = (2,2,2,2,2)
P = Q6	F = F5	S = (1,1,1,1,1)
	F = F6	S = (1,1,3,3,2)
	F = F7	S = (1,1,1,1,1)
	F = F8	S = (1,1,3,3,2)
	F = F9	S = (1,1,3,3,2)
	F = F10	S = (6,6,2,2,2)
	F = F11	S = ("ANOMALOUS")
	F = F12	S = (1,1,3,3,2)
	F = F13	S = (6,6,2,2,2)
P = R3	F = F1	S = (3,3,1,1,2)
	F = F2	S = (1,1,1,1,4)
	F = F3	S = (6,6,1,1,2)
	F = F4	S = (1,1,1,1,4)
P = R4	F = F1	S = (3,3,1,1,2)
	F = F2	S = (1,1,1,1,4)
	F = F3	S = (6,6,1,1,2)
	F = F4	S = (1,1,1,1,4)

P = R5	F = F1	S = (3,3,1,1,2)
	F = F2	S = (1,1,1,1,1)
	F = F3	S = (6,1,1,1,2)
	F = F4	S = (1,1,1,1,1)
P = R6	F = F1	S = (1,3,1,1,1)
	F = F2	S = (1,1,1,1,1)
	F = F3	S = (1,6,1,1,1)
	F = F4	S = (1,1,1,1,1)
P = R7A	F = F1	S = (3,3,1,1,2)
	F = F2	S = (3,3,1,1,2)
P = R7B	F = F1	S = (3,3,1,1,2)
	F = F2	S = (4,4,1,1,2)
P = R8	F = F1	S = (2,2,2,2,2)
	F = F2	S = (1,1,1,1,1)
	F = F3	S = (1,1,5,5,2)
	F = F4	S = (1,1,1,1,1)
P = R9	F = F1	S = (2,2,2,2,2)
	F = F2	S = (5,5,4,4,1)
	F = F3	S = (6,6,5,5,2)
	F = F4	S = (5,5,4,4,1)
P = R11	F = F1	S = (1,1,1,1,1)
	F = F2	S = (5,5,4,4,1)
	F = F3	S = (6,6,5,5,2-
	F = F4	S = (5,5,4,4,1)
P = R12	F = F1	S = (1,1,1,1,1)
	F = F2	S = (2,2,2,2,2)
	F = F3	S = (1,1,1,1,1)
	F = F4	S = (1,1,5,5,2)
P = R13	F = F1	S = (6,6,2,2,2)
	F = F2	S = (1,1,3,3,2)
P = R14	F = F1	S = (6,6,2,2,2-
	F = F2	S = (1,1,1,4,1)
P = R15	F = F1	S = (6,6,1,2,2)
	F = F2	S = (1,1,1,4,1)
	F = F3	S = (1,1,1,5,1)
	F = F4	S = (1,1,1,4,1)

P = R16	F = F1	S = (2,2,2,2,2)
	F = F2	S = (1,1,1,1,1)
	F = F3	S = (6,6,5,5,2)
	F = F4	S = (1,1,1,1,1)
P = R22	F = F1	S = (1,1,1,1,3)
	F = F2	S = (2,2,2,2,2)
	F = F3	S = (1,1,1,1,3)
	F = F4	S = (1,1,5,5,2)
P = S3	F = F1	S = (1,1,5,1,2)
	F = F2	S = (1,1,1,1,1)
P = S4	F = F1	S = (6,1,1,1,2)
	F = F2	S = (1,1,1,1,1)
P = S5A	F = F1	S = (1,1,1,1,1)
	F = F2	S = (1,1,1,1,1)
P = S5B	F = F1	S = (1,1,1,1,1)
	F = F2	S = (1,1,1,1,1)
P = T1(P)	F = F1	S = (2,2,2,2,2)
	F = F2	S = (2,2,2,2,2)
P = T1(S1A)	F = F1	S = (6,6,1,1,2)
	F = F2	S = (6,6,1,1,2)
P = T1(S1B)	F = F1	S = (6,6,1,1,2)
	F = F2	S = (6,6,1,1,2)
P = T1(S2)	F = F1	S = (3,3,1,1,2)
	F = F2	S = (3,3,1,1,2)
P = ZD4	F = F1	S = (1,1,1,1,1)
	F = F2	S = (2,2,2,2,2)
P = ZD5	F = F1	S = (5,5,1,1,4)
	F = F2	S = (3,3,1,1,2)

VIII. Mapping Fe: Γ x P => Γ x P

P(1)	Γ(1)		P(2)	Γ(2)
C1	2		FUSE1	1
C3	2		FUSE1	1
C4	2		FUSE1	1
D1	2		FUSE1	1
D2	2		FUSE1	1
D6	2		Q5	8
		OR	Q5	9
		OR	Q5	11
Q2	10		Q5	6
		OR	Q5	7
		OR	Q5	9
		AND	Q6	6
		OR	Q6	7
		OR	Q6	9
Q2	13		Q5	6
		OR	Q5	7
		OR	Q5	9
		AND	Q6	6
		OR	Q6	7
		OR	Q6	9
Q6	6		FUSE1	1
Q6	7		R15	1
		OR	R15	4
		AND	R16	1
		OR	R16	4
Q6	8		FUSE1	1
Q6	9		FUSE1	1
Q6	12		FUSE1	1
R3	2		ZD4	1
		OR	ZD4	2
R3	4		ZD4	1
		OR	ZD4	2
R4	2		ZD5	1
		OR	ZD5	2
R4	4		ZD5	1
		OR	ZD5	2
R9	2		Q2	8
		OR	Q2	9
		OR	Q2	11
		AND	Q5	6
		OR	Q5	7
		OR	Q5	9
		AND	Q6	6

		OR	Q6	7
		OR	Q6	9
R12	2		FUSE1	1
T1(P)	2		FUSE1	1
T1(S1A)	2		FUSE1	1
T1(S1B)	2		FUSE1	1
T1(S2)	2		FUSE1	1
ZD4	2		R3	4
ZD5	2		R3	4
		AND	R4	4