INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

AUTOMATIC GENERATION OF TEST CASES AND ANTICIPATED TEST OUTCOME BASED ON A TABULAR DESIGN SPECIFICATION

Thatipamala Ramakrishnaiah

A Thesis

in

the Department of Computer Science

Presented in Partial Fulfilment of the Requirements

for the Degree of Master of Computer Science at

Concordia University

Montréal, Québec, Canada

April 1999

ABSTRACT

AUTOMATIC GENERATION OF TEST CASES AND ANTICIPATED TEST OUTCOME BASED ON A TABULAR DESIGN SPECIFICATION

Thatipamala Ramakrishnaiah

At the present time, even for safety-critical applications, it is very difficult, if not impossible, to produce a software that is "completely error-free". One of the important issues associated with this realistic situation is how to minimise the number of errors in a given software. Effective testing of software using trusted CASE tools is one possible strategy.

This thesis discusses the development of a prototype CASE tool, called Apollo, that automates some of the "tedious, complex and error-prone" manual activities that are associated with the unit testing of software modules. The input to Apollo is a design specification document where the design is specified using a tabular notation. This specification is sufficiently detailed to enable execution by a machine. The tool generates a set of test cases and the anticipated test outcome for each test case by executing the tabular specification. Tabular specification is considered as a "practical" formal method, since it is a method that software developers can easily learn and apply without much mathematical background.

The tabular design specification is parsed and test cases are generated based on the boundary value analysis. The anticipated test outcome for each test case is generated by executing the parsed design specification. The proposed methodology is applied to a hypothetical case study for unit testing of software related to nuclear industry. This application replaces some of the manual generation of test cases and anticipated test outcome thereby reduces the cost of software testing.

Acknowledgements

I would like to express my sincere thanks to Dr. Radhakrishnan, Department of Computer Science, for accepting me as his graduate student and for supervising this work.

I would like to take this opportunity to thank all the members of Technical Committee, Working Party No. 16, CANDU Owners Group (COG) R&D Program, for giving me permission to extend the scope of one of the research projects (that was assigned to me) as a Master's project at Concordia University.

I would like to express my sincere thanks to my management in Information and Control Systems Development Division, AECL. But for the encouragement and financial assistance from the management, this thesis would not have been possible.

To

my parents, who taught me the importance of persistence;

my wife, who proved (once again) that "there is a woman's hand behind every success story"; and

my kids, who had put-up with my long working hours, even on weekends (because of full-time demanding job and part-time Master's program).

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

Abbreviations and Acronyms

BNF	Backus-Naur Form
CAA	Condition and Associated Actions
CANDU	CANada Deuterium Uranium
COG	CANDU Owners Group
EBNF	Extended Backus-Naur Form
L-Bye	Look Before You Execute algorithm
OASES	Ontario Hydro and AECL Software Engineering Standards committee
PVS	Prototype Verification System
RTF	Rich Text Format
SCV	Systematic Code Verification
SDD	Software Design Description
SDT	Structured Decision Table
SDV	Systematic Design Verification
SESM	Software Engineering Standards and Methods
SRS	Software Requirements Specification
UT	Unit Testing

Notation

Italics are used to identify special terminology that is specific to the Design Specification document.

`Courier` font is used to identify the parts that are taken directly from the output files of Apollo.

1. INTRODUCTION

The use of software in safety-critical applications is increasing. Some of the well-known examples of safety-critical applications include: power plant shutdown systems in the nuclear industry, avionics systems in the aerospace industry, and critical components of strategic weapons systems in the defense industry [Leveson, 1995]. Reasons for using software in safety-critical systems are many: software allows many more complex situations to be handled when compared to hardware alone, and it allows changes to be made easily to accommodate new and changing requirements. However, the development of safety-critical software is a relatively new and not a fully mature subject. New techniques and methodologies for safety-critical software are a popular research topics both at universities and in the industry [Bowen and Stavridou, 1992]. The development of safety-critical software should meet the following two important requirements:

- It should achieve a high-degree of reliability, and
- It should demonstrate a high-degree of reliability to the satisfaction of regulatory authorities.

It is accepted by software industry that even for safety-critical applications, **it is very difficult, if not impossible, at the present time to produce a software that is guaranteed to be "completely error-free"**. One of the important issues associated with this realistic situation is how to reduce the number of errors in a particular piece of software.

A problem faced by software development teams in the development of safety-critical software is the amount of manual effort required during the testing of the software. However, the current industrial experience indicates software testing is time consuming and costly, and as much as 50% of the development costs for a project can be attributed to testing [Daich et.al., 1994]. In addition, software testing is error prone as many of the testing activities are manual, tedious and time-consuming [Daich et.al., 1994]. The generation of a set of test cases to satisfy various coverage criteria in order to meet the

stringent criteria imposed by various standards is a difficult process. There are many CASE tools available that tell a tester whether he has satisfied a given coverage, but none is available to provide a tester with a set of test cases that will satisfy a given coverage [Voas et.al., 1993]. In addition, the current industrial experience suggests that often the testing is deferred until the final stages of the development life-cycle. A major challenge to the software engineering community, both in research and industry, remains how to reduce the cost and improve the quality of software testing [Kung et. al., 1998]. Any scheme that can automate at least some of the phases of software testing (such as "unit testing", "sub-system testing", "integration testing", etc.,) could decrease the cost and improve the quality of the software testing process, significantly. The scope of this thesis is limited to unit testing.

The "phase of unit testing" is essentially a manual process consisting of the following activities:

Step 1: Identification of the list of input variables and their ranges (along with the relevant information such as: data type, valid range, list of enumerations, etc.).

Step 2: Identification of the list of output variables that need to be observed.

Step 3: Generation of a set of test values for each input variable based on certain criteria (such as: boundary value analysis etc.).

Step 4: Generation of a set of test cases to satisfy various important coverages in order to meet the stringent criteria imposed by relevant standards and regulatory bodies.

Step 5: Generation of "anticipated test outcome" (i.e. the anticipated value of output variables) for each test case;

Step 6: Preparation of a test driver to test the source code;

Step 7: "Execution of source code" using the test driver;

Step 8: Checking the "actual test results" with the "anticipated test outcome" to decide a "pass or fail" for each test case; and

Step 9: Modification of source code, if any implementation errors are detected during Step (8) [This activity includes raising a software change request (SCR), approval of SCR, Change of source code, etc.].

Step 10: Repetition of steps (3) to (8) after Step (9), if any implementation errors are detected during Step(8).

Many of these activities are time-consuming, tedious, complex and error-prone [Myers, 1979; Pressman, 1992]. The perceived level of difficulty associated with automation of each of these steps of unit testing are presented in the Table 1.

Table 1: "Unit Testing Phase" : Important Activities and Level of Difficulty Associated with Automation

Testing Activity	Level of Difficulty	Explanation
Step(1)	Moderate	Syntax directed analysis could be used.
Step(2)	Moderate	Syntax directed analysis could be used.
Step(3)	Difficult	Criteria for test value generation is complex and the criteria may change.
Step(4)	Easy to Difficult	Degree of difficulty of automation depends on the algorithm required for generation of a set of test cases to satisfy a given criteria
Step(5)	Complex	Generation of the "anticipated test outcome from source code that is being tested" can be considered as the most difficult activity in terms of automation.
Step(6)	Manual (preferred)	Preparation of a test driver need to be considered as a manual activity. Associated difficulty with automation of this step is yet to be analyzed.
Step(7)	Easy	Automation of "execution of source code using test driver" is straight-forward.
Step(8)	Easy	Automation of "comparison of two ASCII text files" is relatively simple.
Step(9)	Manual (automation is not feasible)	Modification of source code, if any implementation errors are detected during step (8) need to be considered as a manual activity, since it involves a number of sub-steps, such as code inspection; raising of a "software change request", etc.. Automation of this activity is not feasible.
Step(10)	Easy	Repetition of already automated steps is relatively simple.

1.1 Motivation

During early 1980s, Atomic Energy of Canada Limited (AECL) and Ontario Hydro (OH) have decided to introduce "a first of its kind software-controlled shutdown systems" in one of their nuclear generating stations [Storey, 1996]. This new approach had presented the Atomic Energy Control Board of Canada (AECB), *the regulatory authority*, with some novel problems during the certification process. AECB was uncertain of the adequacy of such software during its first release, despite considerable amounts of testing [Storey, 1996]. As a result, AECL and Ontario Hydro have decided to rewrite the safety-critical software using a "practical" formal method known as tabular notation that was developed at Naval Research Laboratory, USA [Britton and Parnas, 1981] and improved by Parnas at McMaster University [Parnas, 1992]. Tabular notation is considered as a "practical" formal method, since it is a method that software developers can easily learn and apply without theorem proving skills, knowledge of temporal and higher order logic, or consultation with formal methods experts. Moreover, the tabular notation approach supports formal verification of the design against the requirements as well as the source code against the design.

A series of CASE tools were developed at Atomic Energy of Canada Limited (AECL), in order to support the development of safety-critical software, specified using tabular notation, covering the entire life-cycle. This research activity was a part of an ambitious research program that was jointly initiated by AECL and Ontario Hydro. The following important points need to be noted in this connection:

- These CASE tools are currently being used in Canadian nuclear industry;
- These CASE tools represent an advanced industrial research activity to produce tools that are of "industrial quality";
- These CASE tools support the development of safety-critical software specified using tabular notation approach only;
- These CASE tools are developed using conventional techniques (e.g., requirements are listed using natural language).

The details of these CASE tools are documented by Matias [1998]. The Table 2 summarizes some of the important research work that is being carried out at AECL, in this connection.

Table 2 - CASE Tool Development Work at AECL

CASE Tool	Brief Description
SRS Tool.	Requirements Analysis tool: Analyzes the completeness and correctness of a Requirements Specification using static analysis techniques.
SDD Tool	Design Analysis tool: Analyzes the completeness and correctness of a Design Specification using static analysis techniques.
UT Tool	Test Value generation tool: generates a set of test values for each input parameter of a given *access-program* from design document. (UT: Unit Testing)
SDV Tool	Design Verification tool: Generates a number of files (one PVS specification file for each verification block containing proof obligations that can be analyzed by PVS) to verify the functional mapping between the requirements and the design.
SCV Tool	Code Verification tool (***** being developed *****): Generates a number of files (one PVS specification file for each verification block containing proof obligations that can be analyzed by PVS) to verify the functional mapping between design and the source code.

The author of this thesis [Thatipamala, 1996] is the primary software developer (responsible for requirements; design and implementation) of UT tool; and is serving as a member of the software development team for the other CASE tools in Table 2.

It is noted that the UT tool generates only a set of test values for each input parameter. In other words, it automates only the Steps (1), (2), and (3), that are discussed in Table 1. The other steps were excluded at that time since "automation of Step (4) and Step (5)" were considered too complex to be implemented within the limited time period that was available. However, it was perceived that the usefulness and value of the UT tool could be increased multi-fold if its functionality can be extended to include the generation of a set of test cases along with the anticipated test outcome. As a result, it was decided to develop a prototype tool, called Apollo, to demonstrate the "proof-of-concept" of the automation of Step (4) and Step (5) (i.e., generation of a set of test cases along with the

anticipated test outcome by execution of design specification). This has become the focus of the current thesis.

1.2 Main Objective and Scope of the Thesis

1.2.1 Objective of the Thesis

The main objective of this thesis is to develop a prototype, named Apollo, to demonstrate the "proof-of-concept" of the generation of a set of test cases along with the anticipated test outcome, that could be useful in the Canadian nuclear industry.

The input and output to Apollo are as follows:

The input to Apollo: is a "Design Specification Document" where the design of a safety-critical software is specified using the tabular notation. This specification is sufficiently detailed to enable execution by a machine.

The output from Apollo: is a series of text files consisting of information associated with activities of Step (1) to Step (5) that are part of the unit testing phase. An output file contains the following information:

a) list of input parameters along with associated information;

b) list of output parameters along with associated information;

c) a set of test values for each input parameter;

d) a set of test cases; and

e) anticipated test outcome for each test case.

It should be noted that one separate text file is created for each *access-program* (see section 2.1) which is an independent "unit" for unit testing purposes.

1.2.2 Scope of the Thesis

The overall scope of the thesis work has been divided into the following four stages:

Stage I: Identification of Important Functional Requirements

The identification of important functional requirements is a prerequisite for the successful development of any CASE tool. As a result, it was decided to document the functional requirements as precisely as possible, at the very early stages of the thesis work.

Stage II: Development of Design Details and Implementation

It was planned to develop a set of modules (along with header files), each of which will encapsulate a set of distinct responsibilities and implementation details. The advantages of this approach are: the design is easier to understand; and the code is easier to implement. C-language was chosen as the language for implementation.

Stage III: Preparation of Two Sample Input Design Specification Documents

It was decided to prepare a "sample design specification document" during the third phase of the thesis work. The input document, where the design will be specified using tabular notation, are small and concise to be presented in a thesis. Two input documents were identified: a clean design specification document (i.e., without errors); and a draft design specification document (i.e., with typical errors) in order to reflect a practical situation faced in the Canadian nuclear industry.

Stage IV: Testing of Apollo Using Sample Input Document

It was planned to test Apollo using the clean design specification document as well as draft design specification document, during the final phase of the thesis work, and refine the Apollo tool if required. Both input documents were expected to be updated such that the capabilities of the Apollo tool can be illustrated clearly using the test results.

1.3 Thesis Outline

The Thesis comprises the following chapters:

Chapter 2 gives a brief description of some of the important related research work that is carried out at various universities and international research institutes. In addition, a brief description of the tabular notation and related concepts are also presented in Chapter 2.

Chapter 3 outlines some of the important functional requirements.

Chapter 4 outlines the design details of the CASE tool, Apollo.

Results and Discussion from an industrial case study are presented in Chapter 5 to demonstrate the "proof-of-concept" of the approach that is proposed in this thesis.

Important advantages of the approach that is proposed in this thesis are briefly discussed in Chapter 6.

Conclusions and future work are presented in Chapter 6.

2. RELATED WORK AND RELEVANT CONCEPTS

Some of the important related research work that is carried out at various universities and international research institutes are briefly presented in this chapter. In addition, a brief introduction to relevant concepts is also presented in this chapter.

The section 2.1 describes the important terminology used in this thesis, and the section 2.2 describes the important research work that is related to this thesis.

The section 2.3 describes the relevant concepts and terminology used in this research work.

The section 2.4 describes the tabular notation used in the thesis.

2.1 Terminology

The IEEE standard glossary of software engineering terminology [IEEE 1990], as well as the British standard for software testing vocabulary [BS7925, 1998] define many software-testing related terms. These definitions often clarify the meaning of the technical terms that are otherwise used inconsistently in the field of software engineering. However, it should be noted that, sometimes, these standards are not consistent with one another. As a result, some of the important terminology that are used in this thesis is described in this section.

access-program - The terms subroutine, function, procedure, *access-program* are usually interchangeable. The special term "*access-program*" is used in tabular notation to avoid this confusion, since the other terms (subroutine, function, and procedure) are defined and used differently in different programming languages.

anticipated test outcome - For a given test case, anticipated test outcome represents the anticipated value of each output variable during unit testing. This anticipated test outcome is calculated by Apollo by execution of design specification from the Design Specification document.

deterministic specification - A design specification is considered deterministic if, for every combination of input values, the specification defines **one and only one** value for each output. In other words, for every combination of input values, there should be no ambiguity about the value of each output. The value of the output is completely independent of sequence of execution or implementation strategy.

incomplete specification - A design specification is considered incomplete if, the specification is **under-specified.** In other words, for certain combination of input values, the specification has failed to specify a value for every output. The value of output (or set of outputs) is unknown for certain combination of input values.

non-deterministic specification - A design specification is considered non-deterministic if, for certain combination of input values, the specification **fails to define one and only one** value for each output. In other words, for certain combination of input values, there is ambiguity about the value of each output and the value of output depends on sequence of execution or implementation strategy. More than one value can be assigned to a given output (or set of outputs) for certain combination input values.

unit: Every *access-program* (from input design specification document) is considered as a separate unit, for the purpose of unit testing.

2.2 Related Work

The tabular specification method used by AECL and Ontario Hydro is closely related to the Software Cost Reduction (SCR) Method [Britton and Parnas, 1981] developed at Naval Research Laboratory, United States, and Table Tool System (TTS) developed by Parnas and co-workers [Parnas et al., 1994] at McMaster University, Canada. Another CASE tool, called Tablewise, developed by Hoover and co-workers [Hoover and Chen, 1995] for NASA Langley Research Centre is also closely related to the tabular specification method used by AECL and Ontario Hydro. The details of these works are briefly presented in the following sub-sections.

The CASE tools that are discussed in the following sections deal with errors in requirement specifications or design specifications. Generally, incomplete, non-deterministic or ambiguous specifications are analyzed based on static analysis. In contrast, the approach that is proposed in this thesis and implemented in Apollo software attempts to verify the accuracy of design specification using dynamic testing and execution of design specification.

2.2.1 SCR Toolset: Related Work at Naval Research Laboratory, USA

Formulated in the late 1970s to specify the requirements of the Operational Flight Program (OFP) of the A-7 aircraft, the SCR (Software Cost Reduction) requirements method is based on tables for specifying the requirements of a software system [Britton and Parnas 1981]. Heitmeyer and co-workers have developed the following set of CASE tools, called **SCR Toolset**, in order to provide customized CASE tool support for the SCR method:

- Specification Editor
- Dependency Graph Browser
- Consistency Checker
- Simulator
- Model Checker

Specification Editor [Heitmeyer et al., 1995]: This CASE tool helps the user to create, modify, or display a requirements specification. Each SCR specification is organized into dictionaries and tables. The dictionaries define the static information in the specification, such as the names and values of variables and constants, the user-defined types, etc. The tables specify how the variables change in response to input events.

Dependency Graph Browser (DGB) [Heitmeyer et al., 1997]: This DGB tool captures the dependencies among the variables in a given requirements specification as a directed graph. The user can detect errors such as undefined variables and circular definitions by

examining this directed graph. This tool can help the user in understanding the relationship between different parts of a large specification.

Consistency Checker [Heitmeyer et al., 1996, 1997]: This CASE tool analyzes the tabular specification for consistency. Some of the important checks are: syntax errors, type errors, variable name discrepancies, missing cases, unwanted nondeterminism, and circular definitions.

Simulator [Heitmeyer et al., 1997]: This CASE tool helps the user in validating a given tabular specification. The user enters a sequence of input events associated with a typical test scenario and checks the observed output against the expected output. In other words, the user can run the simulator and analyze the results to ensure that the tabular specification captures the intended behavior.

Model Checker [Heitmeyer et al., 1998a, 1998b]: Currently, the model checker analyzes only the invariant properties of the tabular specification. Heitmeyer and coworkers are planning to extend this tool to check various other properties of the specification.

2.2.2 Table Tool System: Related Work at McMaster University, Canada

Parnas and co-workers [1994] have adapted and refined the SCR method (developed at Naval Research Laboratory, USA) through on-going academic research program at McMaster University, Canada. They have developed the following set of CASE tools called, Table Tool System (TTS), to provide customized CASE tool support for tabular specification method during various phases of software development [Janicki et.al., 1995; Li, 1996; Abraham, 1997; Rastogi, 1998; Shen et.al., 1996]:

- **Table Construction Tool:** This tool helps the user during the preparation of specification document (to construct and edit tabular expressions).
- **Table Formatting / Printing Tool:** This tool helps the user in formatting and printing the tabular specification (by generating a PostScript file).
- **Symbol Editor Tool**: This tool helps the user in creating new symbols and/or modifying their associated information.

- **Inversion / Normalization Tool**: This tool helps the user in transforming normal tables to inverted tables, and vice versa. This feature will be useful in choosing an appropriate table type to present complex design specification.
- **Specialization and Simplification Tool**: This tool helps the user to simplify expressions based on user-defined constraints.
- **Carving and Slicing Tool:** This tool helps the user to extract rows, columns, or slices of tabular expressions.
- **Evaluation Code Generator Tool:** This tool helps the user by generating C++ code from tabular specification. This code can be used for specification checking or for testing software against its specification.
- **Composition Tool:** This tool helps the user to compute the composition of two *function tables*.

The following tools are currently under development.

- **Table Checking Tool**: This tool helps the user to perform completeness and disjointness verification of tabular specification.
- **Transformation Tools**: This tool helps the user in transforming "generalized decision tables" to "structured decision tables", and vice versa. This feature will be useful in choosing an appropriate table type to present complex design specification.
- **LaTeX Output Generation Tool:** This tool generates LaTeX Output from a WordProcessor document.

The TTS supports the production of software documentation through an integrated set of tools which manipulate multi-dimensional tabular expressions. This tabular representation of mathematical expressions improves the readability of complex design documentation. The table cells may contain conventional logic expressions, or even other tables. The TTS project aims to automate checking of software specification and design documents, and to assist in software testing and maintenance.

Test Oracle Generator (TOG) Tool (prototype):

One of the prototype CASE tools, Test Oracle Generator (TOG) that was developed by Parnas and co-workers [Peters and Parnas, 1994; Peters, 1995] is discussed in detail in this section, since it is relevant to the research work reported in this thesis:

Peters [1995] have described an algorithm that can be used to generate a test oracle from design documentation, and have developed a prototype CASE tool that generates a test oracle based on that algorithm. The results from a case study of a commercial network management application demonstrate that these methods can be effective at detecting errors and increase the speed and accuracy of test evaluation when compared with manual evaluation. Peters [1995] has concluded that such oracles can be used for unit testing, and for ensuring consistency between code and documentation. Their work attempts to automate Step (5) from Table 1, only partially. In other words, the following steps associated with unit testing were not automated and has to be done **manually** :

Step(1): identification of the list of input variables (along with the relevant information such as: data type, valid range, list of enumerations, etc.) that will be manipulated;

Step (2): identification of the list of output variables (along with the relevant information such as: data type, valid range, list of enumerations, etc.) that need to be observed;

Step(3): generation of a set of test values for each input parameter based on certain criteria (such as: boundary value analysis, etc.) ;

Step(4): generation of a set of test cases to satisfy various important coverages in order to meet the stringent criteria that is imposed by relevant standards and procedures;

Step(5): generate the test oracle using the approach suggested by the Peters and Parnas [1999] and run the test oracle using the set of test cases that are generated in Step(4) in order to generate the "anticipated test outcome" (i.e. the anticipated value of output variables) for each test case.

The above analysis clearly indicates that Step(5) can be automated only partially, using the approach suggested by Peters [1995].

2.2.3 Tablewise Tool: Work at NASA Langley Research Centre/Odyssey Research Associates, USA

Hoover and co-workers [Hoover et.al., 1996; Hoover and Chen, 1994, 1995] have developed a CASE tool, called **Tablewise**, based on decision tables. Some of the important goals of this research program are:

- to advance the state-of-the-art in formal methods, making it practical for use on life-critical systems developed by the aerospace industry in the United States, and
- to transfer such technology to industry through use of carefully designed demonstration projects.

Hoover and co-workers have concentrated on bringing decision tables up-to-date and closer to "formal methodology". The following are some of the important objectives of their research work:

1. Condensing a decision table.
2. Adding precondition annotations to indicate the cases that are not expected to occur.
3. Checking a decision table by making assertions about what a decision table specifies.
4. Generating a decision table by using assertions about what it should specify.
5. Structural analysis to locate errors in decision tables.

2.3 Relevant Concepts

2.3.1 Boundary Value Analysis

Software testing becomes effective if test values are generated based on the analysis of the conditions at the change-over values called boundaries. Myers [1979] calls this approach to test selection boundary value analysis. The basic idea behind boundary value analysis is that test cases which explore boundary conditions give a higher payoff than those which do not. Specifically, tests just above, at, and just below a boundary are all

important [Myers, 1979]. The following two examples illustrate the test values that are generated based on the boundary value analysis:

Example-1:

input variable: x

data type: integer

valid range: -10 to 100

condition: if(x >= Set_Point)

[NOTE: assume that Set_Point is equal to 70]

Test values generated as per boundary value analysis:

at the condition	:	70
just above the condition	:	71
just below the condition	:	69
maximum range	:	100
minimum range	:	-10

Example-2:

input variable: x

data type: integer

valid range: -10 to 100

condition: if(x >= Set_Point)

[NOTE: assume that Set_Point is a variable set-point with a valid range from 60 to 80]

Test values generated:

Assuming that Set_Point is equal to 60 (the lowest value).

at the condition	:	60
just above the condition	:	61
just below the condition	:	59
maximum range	:	100
minimum range	:	-10

Assuming that Set_Point is equal to 80 (the highest value).

at the condition	:	80
just above the condition	:	81
just below the condition	:	79

maximum range	:	100
minimum range	:	-10

NOTE: From the above examples it is clear that test value generation becomes even more laborious and error-prone when Set_Point is a function of several variables.

2.3.2 Partition Test Values and Partition Testing

Software testing becomes more effective if the test values are partitioned, or subdivided in some way [BS7925-2, 1998]. In general, partition test values are selected by subdividing the range into half. For example, the following partition test values will be generated for the above example discussed in boundary-value analysis:

Test values from boundary value analysis: -10, 69, 70, 71, and 100

Partition test values: 30 (i.e., test value in the middle of -10 and 69)

85 (i.e., test value in the middle of 71 and 100)

Partition testing ensures that the program is tested thoroughly.

2.3.3 Zero Value Inclusion

If the valid range of input variable includes "Zero" value, it should be included as one of the test values. It will test for a range of possible errors including "divide by zero" possibility, which may not be tested otherwise. This testing ensures that the program is tested thoroughly.

2.3.4 Static analysis

Traditionally, static analysis consists of investigation of the source code of software, looking for errors without actually executing the source code. In general, activities such as code review and code walk-through are considered as static analysis activities. However, in the context of development of safety-critical software, activities such as investigation of the requirements specification and design specification, looking for errors

without actually executing the specification can also be considered as a part of static analysis. In general, activities such as requirements review, and design review are also considered as part of static analysis. Even design verification activity (i.e., verification of design against a mathematical requirements specification), code verification activity (i.e., verification of code against a mathematical design specification) can also be considered as a part of static analysis, since execution of either the specification or the source code is not involved in such an activity. In addition more complex static analysis techniques such as control flow analysis, data flow analysis are applied during the development of safety-critical software. Sophisticated tools such as PVS, Malpas [Owre et.al, 1993] can be used to verify the functional mapping between requirements and the design; as well as design and the source code. These sophisticated tools use the concept of symbolic software testing to locate generic problems such as ambiguity, inconsistency, and/or incompleteness of the specification. Various phases of static analysis and their importance has been discussed in detail by [Daich et.al., 1994].

2.3.5 Dynamic Analysis/Testing

Traditionally, dynamic analysis consists of investigation of the source code of software, looking for errors while it is being executed. In general, activities such as white-box testing, black-box testing, gray-box testing, unit testing, sub-system testing, integration testing, etc., are considered as part of the dynamic analysis. However, in the context of development of safety-critical software, activities such as investigation of the requirements-specification and design-specification, looking for errors while the formal specification (either requirements specification or design specification) is being executed also form part of dynamic analysis. In other words, dynamic analysis of a given specification attempts to verify the characteristics of the software by critical analysis of the information that occurs internally during execution as various states of the program are created and executed (starting from a given initial state to a final state, under different execution scenarios).

Advantages of Dynamic Verification

In general, dynamic analysis captures more semantic errors compared to static analysis, since dynamic analysis can capture the critical information during internal state transitions (which represents the true characteristics of the software). Dynamic analysis can detect errors due to misrepresentation of requirements, incorrect specification of requirements, and erroneous translation from specification into implementation language. Dynamic analysis also supports the development and systematic evaluation of test suites, thereby potentially exposing flaws and oversights in a test regime, as well as in the corresponding specification.

However, it should be noted that both static and dynamic analysis activities complement each other. Comparison of static and dynamic analysis activities and their importance have been discussed in detail by [Daich et.al., 1994].

2.3.6 Unit Testing

The unit test is the lowest level of testing performed during software development, where individual units of software are tested in isolation from other parts of a program. In general, Unit Testing can be considered as "testing against Detailed Design Specification", where each unit (basic component) of the software is tested to verify that the detailed design specification for that unit has been correctly implemented. In other words, unit testing is the process of executing software in a controlled manner, in order to answer the question **"Does the unit behave as specified in the Detailed Design Specification of that Unit ?"**.

In general, unit testing activity is classified into the following four types based on test case selection process [Morell and Deimel, 1992]:

- requirement-specification-oriented testing;
- design-specification-oriented testing;
- implementation-oriented testing; and
- Error-oriented testing.

Various types of unit testing and their importance has been discussed in detail by Morell and Deimel [1992]. It should be noted that none of the unit testing techniques is superior

to others so that its exclusive use can be justified. Various types of unit testing are best seen as complementary rather than competing with each other [Morell and Deimel, 1992].

2.3.6.1 Test Coverage Criteria

The following are a number of unit test coverage criteria (metrics) that are associated with unit testing activity:

- Statement Coverage
- Branch Coverage
- Path Coverage
- Condition/Decision Coverage
- Multiple Condition/Decision Coverage

It should be noted that various international standards have different test coverage criteria, depending on the application. "Multiple Condition/ Decision Coverage" criteria is discussed below, in order to illustrate the complexity (and practical difficulties) associated with meeting a specified test coverage criteria from a relevant standard.

"**Multiple Condition/ Decision Coverage**" is defined as follows in FAA standard DO-178B [RTCA/FAA DO-178B, 1985; Voas et.al., 1993]:

Every point of entry and exit in the program has been invoked at least once, and every condition in a decision in the program has taken on all possible outcomes at least once, every decision in the program has taken on all possible outcomes at least once, and each condition in a decision has been shown to independently affect that decision's outcome.

In other words, a condition should be shown to independently affect a decision's outcome by varying just that condition while holding all other possible conditions fixed.

For example, there are 2^n possible combinations of condition outcomes for every given complex decision consisting of the following form:

if ((c1) or (c2) or .... or (cn)).

As a result, the number of test cases will increase in the same order (in order to satisfy the "Multiple Condition/ Decision Coverage" criteria).

This analysis indicates that the activity of test value generation as well as the activity of the generation of a set of test cases become complex in order to meet a given test coverage criteria.

Different types of coverages and their importance have been discussed in detail by Morell and Deimel [1992].

Unit testing is an opportunity to "catch" software bugs early, before the cost of correction escalates too far. Unit tests are simpler to create, easier to maintain and more convenient to repeat than later stages of testing. When all costs are considered, unit tests are cheap compared to the alternative of complex integration testing, or unreliable software.

2.4 Introduction to Tabular Notation

2.4.1 Hypothetical Example

Table based specification has been discussed in a number of publications [Parnas 1992, Parnas et.al. 1994, McDougall et.al. 1994, Hoover et.al. 1994, Hoover 1995, and Matias 1998]. A brief introduction to tabular notation is given in this section using an hypothetical example associated with "Inlet Coolant Temperature in a Critical Cooling Loop for a Nuclear Reactor" and its Display Status on the Control Panel. The status of the light on the control panel is decided based on the following general conditions:

- if the temperature is with-in the range which is calculated based on the set-point, then green light shall be turned-on;
- if the temperature is below the set-point then red light shall be turned-on;
- if the temperature is above the set-point then red light shall flash on the control panel;
- use a reasonable dead-band for a steady display on the control panel.

2.4.1.1 Declaration of Constants

Based on the analysis of the problem, the following five constants are found to be necessary:

1. valid lowest range of the temperature signal : C_Lower_Range (assumed as 0° C for illustrative purposes);
2. valid highest range of the temperature signal : C_Upper_Range (assumed as 100° C for illustrative purposes);
3. valid lowest range of the set-point : C_Lowest_SP (assumed as 15° C for illustrative purposes);
4. valid highest range of the set-point : C_Highest_SP (assumed as 25° C for illustrative purposes);
5. a reasonable dead band : C_Deadband (assumed as 4° C for illustrative purposes).

In the tabular notation such constants are declared using a *Constants Table* (see Table 3):

Table 3 - Constants Table

	Name	*Value*	*Type*
Constants:	C_Lower_Range	0	INTEGER
	C_Upper_Range	100	INTEGER
	C_Lowest_SP	15	INTEGER
	C_Highest_SP	25	INTEGER
	C_DeadBand	4	INTEGER

2.4.1.2 Declaration of Data Types

The analysis reveals the need for three data types:

1. an abstract data type to deal with the coolant temperature which can be called as T_Coolant_Temp;

2. an abstract data type to deal with set-point variable which can be called as T_Temp_Set_Point;
3. an enumeration data type to deal with variable "display status on the control panel" which can be called as T_Display_Status; This data type contains three elements to deal with green, red and flashing red display which can be called as e_Green, e_Red and e_Flashing_Red;

These data types are specified using the *Types Definition Table* (see Table 4):

Table 4 - Types Definition Table

	Name	*Definition*
Types:	T_Coolant_Temp	C_Lower_Range TO C_Upper_Range
	T_Temp_Set_Point	C_Lowest_SP TO C_Highest_SP
	T_Display_Status	{e_Green, e_Red, e_Flashing_Red }

2.4.1.3 Declaration of Inputs

Input parameters are declared using the Inputs Table (see Table 5):

- an input parameter to deal with the coolant temperature named V_Temp, whose data type is T_Coolant_Temp;
- an input parameter to deal with the set-point named V_Set_Point, whose data type is T_Temp_Set_Point.

Table 5 - Inputs Table

	Name	Type
Inputs:	V_Temp	T_Coolant_Temp
	V_Set_Point	T_Temp_Set_Point

2.4.1.4 Declaration of Outputs

Output parameters are declared using the Outputs Table (see Table 6):

- an output parameter to deal with the display status of the light named V_Display_Status, whose data type is T_Display_Status.

Table 6 - Outputs Table

	Name	Type
Outputs:	V_Display_Status	T_Display_Status

2.4.1.5 Declaration of the Logic

The analysis of the problem yields the three rules given below:

R1: if the temperature is with in the desired range (i.e. set-point plus or minus half-of-the dead-band) then assign e_Green;

R2: if the temperature is less than the desired range then assign e_Red;

R3: if the temperature is more than the desired range then assign e_Flashing_Red;

Such rules are specified using a *Function Table* (see Table 7). In this table each rule has one row and there are two columns.

Table 7 - Function Table

Table display_Status

	Result
Condition	V_Display_Status
V_Temp < (V_Set_Point - (0.5 * C_DeadBand))	e_Red
(V_Temp >= (V_Set_Point - (0.5 * C_DeadBand)) AND (V_Temp <= (V_Set_Point + (0.5 * C_DeadBand))	e_Green
(V_Temp > (V_Set_Point + (0.5 * C_DeadBand))	e_Flashing_Red

The following observations can be made from the above table specification:

- The value of the output variable given in the second column, is decided based on the condition that is satisfied in the first column.
- The conditions given in the first column are expected to be mutually exclusive and the specification is non-deterministic otherwise.
- The conditions given in the first column are expected to cover the complete range of input variables. The specification is wrong if they do not cover the complete range, which is called as incomplete specification.

A brief introduction to various *function tables* that are supported by the Apollo tool are presented in section 2.4.2.

2.4.2 Function Tables

Function tables are the main part of the tabular notation. The Apollo tool developed as a part of this thesis supports several different types of *function tables*. The user can choose the most appropriate type of *function table* depending on the complexity of the specification. Apollo translates these *function tables* into a set of Conditions and Associated Actions, called a set of CAA. This generic internal representation used is logically equivalent to the original *function table* specified by the user. This transformation is truth-preserving and simplifies the dynamic analysis of the design specification. Some of the important *function tables* that are supported by the Apollo tool are given below:

a. *Simple Vertical Condition Tables,*

b. *Complex Vertical Complex Condition Tables,*

c. *Simple Horizontal Condition Tables,*

d. *Complex Horizontal Condition Tables,*

e. *Simple Labelled Horizontal Condition Tables,* and

f. *Complex Labelled Horizontal Condition Tables.*

Predicate calculus notation [Kahane 1990] is used, in combination with standard set theory operators and symbols, to provide formal definitions for the various tables that are supported by the Apollo tool. The Extended BNF (EBNF) grammar, predefined functions, symbols and operators that are supported by Apollo are presented in Appendix E.

2.4.2.1 Simple Vertical Condition Table

A sample *Simple Vertical Condition Table* is shown in Table 8. The conditional relationship between a measured or input variable, V_Reactor_Power and a state or output variable, V_Status is specified in this table.

Table 8 - Simple Vertical Condition Table (Sample)

	V_Reactor_Power > C_Danger_Power	V_Reactor_Power <= C_Danger_Power
V_Status	e_Tripped	e_NotTripped

Interpretation of Table 8 (using pseudo-code):

```
if(V_Reactor_Power > C_Danger_Power )
        then V_Status = e_Tripped
if(V_Reactor_Power <= C_Danger_Power )
        then V_Status = e_NotTripped
```

NOTE: Each column is vertically read to create a rule. The order of sequence of execution of such rules is **not implied** in the tabular specification. The value of output is expected to be independent of implementation order of these rules.

2.4.2.2 Complex Vertical Condition Table

A sample *Complex Vertical Condition Table* is shown in Table 9. The conditional relationship between two measured or input variables: V_Reactor_Power and V_Coolant_Flow, and one state or output variable, V_Status is specified in this table.

Table 9 - Complex Vertical Condition Table (Sample)

	V_Reactor_Power > C_Critical_Power		V_Reactor_Power <= C_Critical_Power
	V_Coolant_Flow <> e_High	V_Coolant_Flow = e_High	
V_Status	e_Tripped	e_NotTripped	e_NotTripped

Interpretation (using pseudo-code):

```
if((V_Reactor_Power > C_Critical_Power) AND (V_Coolant_Flow <> e_High ) )
        then V_Status = e_Tripped
if((V_Reactor_Power > C_Critical_Power) AND (V_Coolant_Flow = e_High ) )
        then V_Status = e_NotTripped
if(V_Reactor_Power <= C_Critical_Power)
        then V_Status = e_NotTripped
```

2.4.2.3 Simple Horizontal Condition Table

A sample *Simple Horizontal Condition Table* with two outputs (V_Status and V_Alarm) is shown in Table 10. The conditional relationship between one measured or input variable: V_Reactor_Power, and two state or output variables: V_Status, and V_Alarm is specified in this table.

Table 10 - Simple Horizontal Condition Table (with two outputs) (sample)

	V_Status	V_Alarm
V_Reactor_Power > C_Danger_Power	e_Tripped	e_On
V_Reactor_Power <= C_Danger_Power	e_NotTripped	e_Off

Interpretation (using pseudo-code):

```
if(V_Reactor_Power > C_Danger_Power )
        then
        {
                V_Status = e_Tripped
                V_Alarm = e_On
        }
if(V_Reactor_Power <= C_Danger_Power )
        then
        {
                V_Status = e_NotTripped
                V_Alarm = e_Off
        }
```

NOTE: "the order of sequence of execution" is **not implied** in the tabular specification, even for the assignment of output variables. In other words, if there is any dependency on the order of execution, that should be specified explicitly (using a previous value notation). The value of output is expected to be independent of implementation of details.

2.4.2.4 Complex Horizontal Condition Table

The format of a sample *Complex Horizontal Condition Table* is shown in Table 11.

The conditional relationship between two measured or input variables: V_Reactor_Power and V_Coolant_Flow, and two state or output variables: V_Status, and V_Alarm is specified in this table.

Table 11 - Complex Horizontal Condition Table (Sample)

		V_Status	V_Alarm
V_Reactor_Power > C_Danger_Power	V_Coolant_Flow <> e_High	e_Tripped	e_On
	V_Coolant_Flow = e_High	e_NotTripped	e_Off
V_Reactor_Power <= C_Danger_Power		e_NotTripped	e_Off

Interpretation (using pseudo-code):

```
if((V_Reactor_Power > C_Critical_Power) AND (V_Coolant_Flow <> e_High ) )
        then
        {
                V_Status = e_Tripped
                V_Alarm = e_On
        }
if((V_Reactor_Power > C_Critical_Power) AND (V_Coolant_Flow = e_High ) )
        then
        {
                V_Status = e_NotTripped
                V_Alarm = e_Off
        }
if(V_Reactor_Power <= C_Danger_Power )
        then
        {
                V_Status = e_NotTripped
                V_Alarm = e_Off
        }
```

2.4.2.5 Simple Labelled Horizontal Condition Table

A sample *Simple Labelled Horizontal Condition Table* with the output V_Status is shown in Table 12. The conditional relationship between one measured or input variable, V_Reactor_Power and one state or output variable, V_Status is specified in this table.

Table 12 - Simple Labelled Horizontal Condition Table (Sample)

	Result
Condition	V_Status
V_Reactor_Power > C_Danger_Power	e_Tripped
V_Reactor_Power <= C_Danger_Power	e_NotTripped

Interpretation (using pseudo-code):

```
if(V_Reactor_Power > C_Danger_Power )
    then V_Status = e_Tripped
if(V_Reactor_Power <= C_Danger_Power )
    then V_Status = e_NotTripped
```

2.4.2.6 Complex Labelled Horizontal Condition Table

The format of a sample *Complex Labelled Horizontal Condition Table* is shown in Table 13. The conditional relationship between two measured or input variables: V_Reactor_Power and V_Coolant_Flow, and two state or output variables: V_Status, and V_Alarm is specified in this table.

Table 13 - Complex Labelled Horizontal Condition Table (Sample)

		Result	
Condition		V_Status	V_Alarm
V_Reactor_Power > C_Critical_Power	V_Coolant_Flow <> e_High	e_Tripped	e_On
	V_Coolant_Flow = e_High	e_NotTripped	e_Off
V_Reactor_Power <= C_Critical_Power		e_NotTripped	e_Off

Interpretation (using pseudo-code):

```
if((V_Reactor_Power > C_Critical_Power) AND (V_Coolant_Flow <> e_High ) )
        then {
                V_Status = e_Tripped
                V_Alarm = e_On
        }
if((V_Reactor_Power > C_Critical_Power) AND (V_Coolant_Flow = e_High ) )
        then {
                V_Status = e_NotTripped
                V_Alarm = e_Off
        }
if(V_Reactor_Power <= C_Danger_Power )
        then {
                V_Status = e_NotTripped
                V_Alarm = e_Off
        }
```

The following is a summary of the tables presented in this section:

1. Constants Table (Table 3) defines constants
2. Types Definition Table (Table 4) defines data types
3. Inputs Table (Table 5) defines inputs
4. Outputs Table (Table 6) defines outputs
5. Function Table (Table 7) defines rules : condition - action
6. Simple Vertical Condition Table (Table 8) where each column is read as one rule
7. Complex Vertical Condition Table (Table 9) where a single column can give rise to multiple rules
8. Simple Horizontal Condition Table (Table 10) where each row is read as one rule
9. Complex Horizontal Condition Table (Table 11) where a single row can give rise to multiple rules
10. Simple Labelled Horizontal Condition Table (Table 10) where condition and action columns are labelled for clarity
11. Complex Labelled Horizontal Condition Table (Table 11) where condition and action columns are labelled for clarity

2.5 Software Standards

Table 14 summarizes some of the important standards that are applicable to safety-critical software development. In general, formal mathematical methods are "highly recommended" by most standards for the development of the safety-critical software. All these standards have been discussed in detail by Place and Kang [1993] and Bowen and Stavridou [1992] and IPL [1996].

Table 14 - Safety-Critical Software Development Standards

Standard	Description
IEC880 Software for Computers in the Safety Systems of Nuclear Powers Stations.	A standard for the nuclear industry [IEC880, 1986].
RTCA/FAA DO-178B Software Considerations in Airborne Systems and Equipment Certification.	A standard for avionics and airborne systems [RTCA/FAA DO-178B, 1985].
Defense Standard MOD 00-55 The Procurement of Safety-critical Software in Defense Equipment.	Detailed software standard for safety-critical defense equipment [MOD 00-55, 1991].
Defense Standard MOD 00-56 Safety Management Considerations for Defense Systems Containing Programmable Electronics.	A standard for the defense industry [MOD 00-56, 1991].

3. PROTOTYPE: FUNCTIONAL REQUIREMENTS

This chapter outlines some of the important functional requirements of the prototype Apollo tool.

3.1 High-Level Requirements

The high-level functional requirements of the Apollo tool are as follows:

(a) **Input Document**: The "input document" to the Apollo tool shall be the "Software Design Specification Document" of the software to be tested. It can be produced using one of the commercial word-processing packages. A sample input document is given in Appendix-A.

(b) **Generation of Test Values**: Based on the boundary-value analysis for each condition, the Apollo tool shall generate a set of test values for each input parameter of every *access-program*. The specific rules to be used for generation of test values are given in Section 3.2.7.1.

(c) **Generation of a Set of Test Cases**: The Apollo tool shall generate a set of test cases based on the possible combinations of the test values that are generated in step (b) above.

(d) **Generation of Anticipated Test Outcome:** For each test case, Apollo shall generate the anticipated test outcome, by executing the program's design specification.

(e) **Output File**: The "output file", a set of test cases along with the anticipated test outcome, that is generated by the Apollo tool shall be presented as a simple ASCII text file. A sample output file generated by the Apollo tool should be as shown in Appendix-B.

The typical input and output files for the Apollo tool are illustrated in Figure 1.

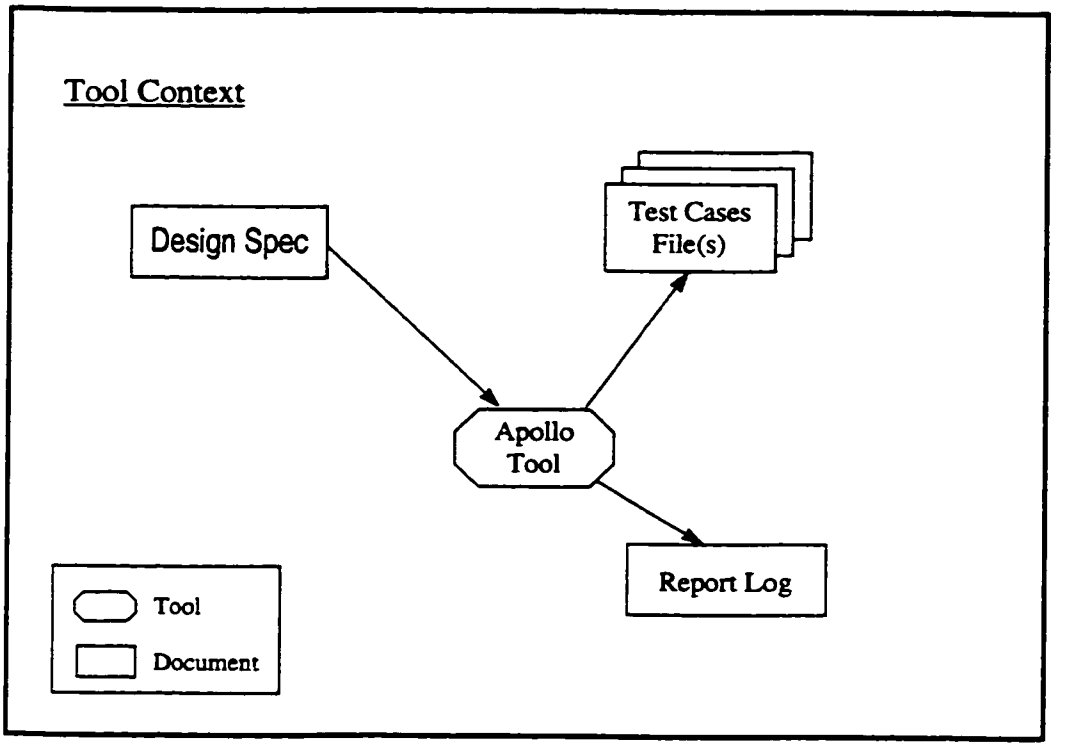

Figure 1 - Apollo Tool Context

3.1.1 Important Assumption

It is assumed that the user will verify the correctness (syntax and other static analysis checks) of the input document using the Design Analysis tool (see Table 2 for details) before using the Apollo tool.

3.2 Detailed Requirements

The detailed functional requirements of some of the high-level requirements (which require further elaboration) are presented in this section.

3.2.1 Identification of the List of Input Parameters of an *Access-Program*

The Apollo tool shall identify all the input parameters for every *access-program*, based on the information provided at the global-scope, module-scope, and the scope of the *access-program.*

3.2.2 Identification of the List of Output Parameters of an *Access-Program*

The Apollo tool shall identify all the output parameters for every *access-program*, based on the information provided at the global-scope, module-scope, and the scope of the *access-program.*

3.2.3 Supported Data Types

The Apollo tool shall support the following four base data types of variables:

(a) INTEGER;
(b) REAL;
(c) ENUMERATED; and
(d) BOOLEAN.

3.2.4 Supported Relational Operators

The Apollo tool shall support the following six relational operators:

(a) greater than: ">";

(b) greater than or equal to: ">=";

(c) less than: "<";

(d) less than or equal to: "<=";

(e) equal to: "=";

(f) not equal to: "<>"

3.2.5 Supported Arithmetic Operators

The Apollo tool shall support the following four arithmetic operators:

(a) addition: "+";

(b) subtraction: "-";

(c) multiplication: "*"; and

(d) division: "/".

3.2.6 Supported Math Functions

The Apollo tool shall support the following five math functions:

(a) ceiling;

(b) floor;

(c) round;

(d) modulus;

(e) absolute.

3.2.7 Test Value Generation Rules for INTEGER and REAL inputs

The Apollo tool shall generate a set of test values for each input parameter (whose data type is either INTEGER or REAL) based on the boundary value analysis using each condition. The specific rules for generation of test values using various forms of simple conditions are given section 3.2.7.1.

3.2.7.1 Boundary Value Analysis of Simple Conditions

(a) The Apollo tool shall isolate the given input parameter to the left-hand-side of a given simple condition before generating the test values using the relevant rules that are specified in this section.

Note:

The terms used in this section are described below:

1. LowerRange and UpperRange : These terms indicate the valid range (lower and upper range) of the input parameter (for which the test values are being generated).
2. MiddleValue: This term represents middle value with in the valid range of the input parameter (for which the test values are being generated). [In general, middle value is equal to (lower range + upper range) divided by 2].
3. ExpMin and ExpMax : These terms indicate the minimum and maximum value of the expression (which is part of the condition that is being analyzed).
4. MIN (a, b) : This term indicates that compare the value of "a" and "b" and choose the lower value.
5. MAX (a, b) : This term indicates that compare the value of "a" and "b" and choose the higher value.
6. DeltaVlaue : This term represents the desired resolution that need to be tested for a given variable. [For REALs, a default delta value of "**0.00001**" will be assigned by the Apollo tool. For INTEGERs, a default delta value of "**1**" will be assigned by the Apollo tool]

3.2.7.1.1 Conditions of the form: (input parameter > expression)

The Apollo tool shall generate the following six test values for each input parameter that is present in a condition of the form (input parameter > expression):

(a) MAX[(ExpMin + DeltaValue), LowerRange]

(b) UpperRange

(c) UpperRange - DeltaValue

(d) ExpMax + DeltaValue

(e) ExpMax

(f) ExpMax - DeltaValue

3.2.7.1.2 Conditions of the form: (input parameter >= expression)

The Apollo tool shall generate the following seven test values for each input parameter that is present in a condition of the form (input parameter >= expression):

(a) MAX[ExpMin, LowerRange]

(b) MAX[ExpMin, LowerRange] + Delta value

(c) UpperRange

(d) UpperRange - DeltaValue

(e) ExpMax + DeltaValue

(f) ExpMax

(g) ExpMax - DeltaValue

3.2.7.1.3 Conditions of the form: (input parameter < expression)

The Apollo tool shall generate the following six test values for each input parameter that is present in a condition of the form (input parameter < expression):

(a) MIN[(ExpMax - DeltaValue); UpperRange]

(b) LowerRange

(c) LowerRange + DeltaValue

(d) ExpMin - DeltaValue

(e) ExpMin

(f) ExpMin + DeltaValue

3.2.7.1.4 Conditions of the form: (input parameter <= expression)

The Apollo tool shall generate the following seven test values for each input parameter that is present in a condition of the form (input parameter <= expression):

(a) MIN[ExpMax; UpperRange]

(b) MIN[ExpMax; UpperRange] - Delta Value

(c) LowerRange

(d) LowerRange + DeltaValue

(e) ExpMin - DeltaValue

(f) ExpMin

(g) ExpMin + DeltaValue

3.2.7.1.5 Conditions of the form: (input parameter = expression)

The Apollo tool shall generate the following four test values for each input parameter that is present in a condition of the form (input parameter = expression):

(a) MIN[ExpMax; UpperRange]

(b) MIN[ExpMax; UpperRange] - Delta Value

(c) MAX[ExpMin, LowerRange]

(d) MAX[ExpMin, LowerRange] + Delta Value

3.2.7.1.6 Conditions of the form: (input parameter <> expression)

The Apollo tool shall generate the following eight test values for each input parameter that is present in a condition of the form (input parameter <> expression):

(a) MIN[ExpMax; UpperRange]

(b) MIN[ExpMax; UpperRange] - Delta Value

(c) MAX[ExpMin, LowerRange]

(d) MAX[ExpMin, LowerRange] + Delta Value

(e) UpperRange

(f) UpperRange - Delta Value

(g) LowerRange

(h) LowerRange + Delta Value

3.2.7.2 Boundary Value Analysis of Compound Conditions

(a) A given Compound Expression shall be broken down into corresponding simple conditions (e.g., input parameter > exp_1; input parameter < exp_2; exp_1 < exp_2), and test values shall be generated (for each simple condition) using the rules that are specified in section 3.2.7.1. This approach ensures that test values are generated using all the conditions that are associated with the given *access-program.*

3.2.7.3 Test Value Generation Rules, for inputs that do not appear in a given condition

The Apollo tool shall generate the following three test values for each input parameter that does not appear in a given condition:

(a) UpperRange;

(b) LowerRange; and

(c) MiddleValue.

3.2.7.4 Partition Range Test Value Generation Rules

(a) The rules that are specified in section 3.2.7.1 provide a set of values for each input parameter. These sets of values partition the range of the inputs into one or more partitions of the range. For each partition the mid-value shall be generated as a test value.

3.2.7.5 Zero inclusion Rule

(a) The Apollo tool shall generate an additional test value of "zero" for all input parameters whose valid range extends from negative to positive.

3.2.8 Test Value Generation Rules for ENUMERATED inputs

(a) The Apollo tool shall generate all enumerations as test values for each enumerated input parameter.

3.2.9 Test Value Generation Rules for BOOLEAN inputs

(a) The Apollo tool shall generate both TRUE and FALSE as test values for each BOOLEAN input parameter.

3.2.10 Design Specification Errors

3.2.10.1 Incomplete design specification

(a) The Apollo tool shall report "Incomplete Spec" as the anticipated outcome for every test case that detects incomplete design specification;

(b) The Apollo tool shall report the following warning message, when appropriate, into the output file:

WARNING: Tool encountered incomplete design specification;
---------- please review design document.

3.2.10.2 Non-deterministic design specification

(a) The Apollo tool shall report "NonDeterministic" as the anticipated outcome for every test case that detects non-deterministic design specification;

(b) The Apollo tool shall report the following warning message, when appropriate, into the output file:

WARNING: Tool encountered non-deterministic design specification;
---------- please review design document.

4. PROTOTYPE: DESIGN DETAILS

This chapter outlines some of the important design details of the prototype Apollo tool. A set of twelve modules have been designed, each of which encapsulates a set of distinct responsibilities and the corresponding implementation details. Each of the modules is further sub-divided into a group of procedures which fulfill specific responsibilities of that module. The advantages of this approach are: the design is easier to document as well as easier to understand; the code is easier to implement; and the software is easier to maintain. In addition, the application gets developed in a well-defined way.

4.1 High-Level Design

The high-level design details of the Apollo tool comprises of the following three sections:

Section 4.1.1 describes the important stages in data and control flow of the Apollo tool.

Section 4.1.2 describes the algorithm followed by the Apollo tool during the generation of test cases along with the anticipated test outcome.

Section 4.1.3 summarizes responsibilities and encapsulation of individual modules that are specific to the Apollo tool.

4.1.1 Data and Control Flow

Figure 2 illustrates the important stages in data and control flow of the Apollo tool.

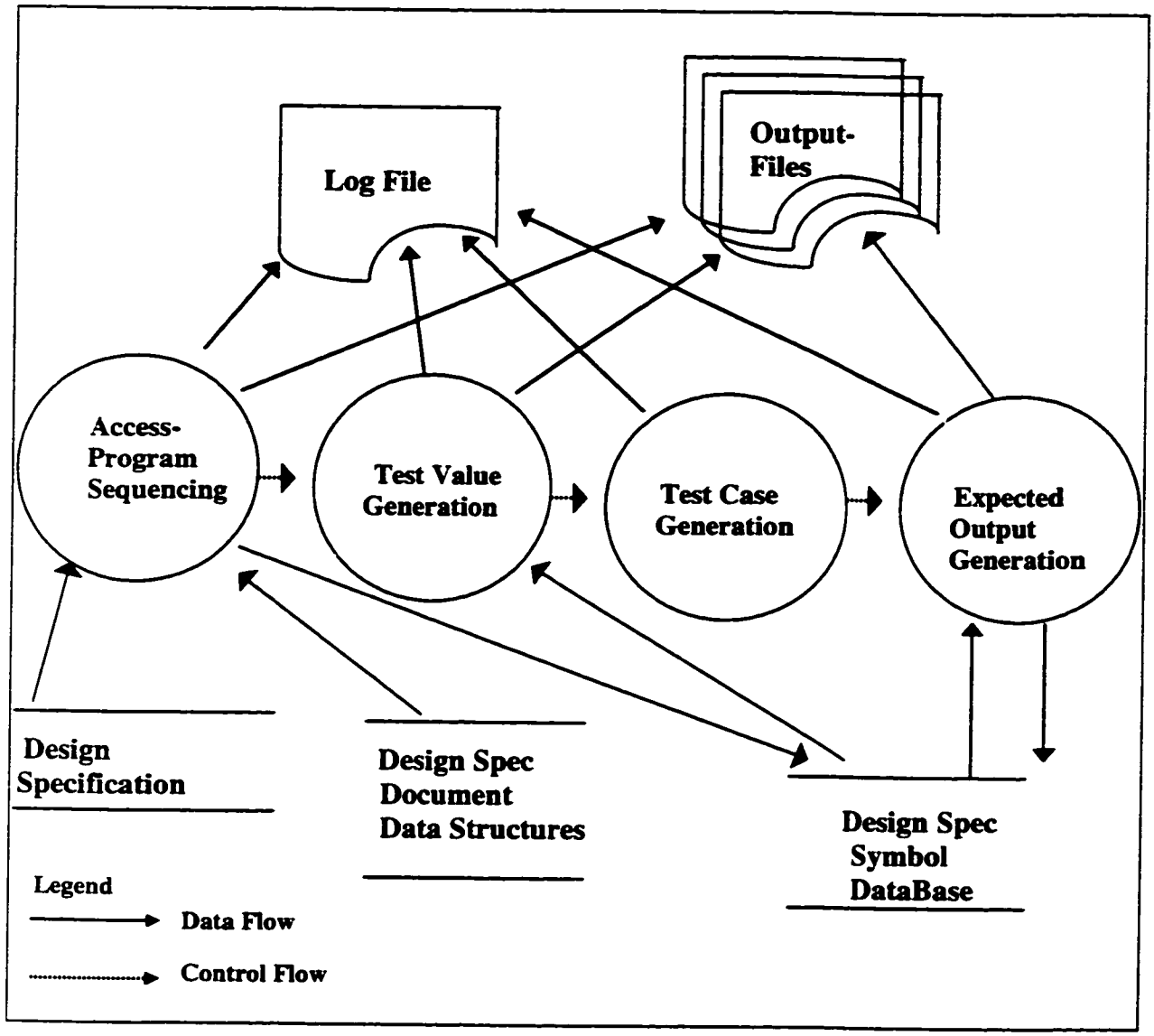

Figure 2 - Apollo Tool: Data and Control Flow Diagram

The four important stages in data and control flow are briefly described below:

***Access-program* Sequencing**

The Apollo tool parses the input design specification document and builds the data structures that represent the input document. The Apollo tool subdivides the task to the *access-program* level by loading all the symbols that are with in the scope of the *access-program* that is being analyzed.

Test Value Generation

The Apollo tool generates the test values for each input parameter by applying boundary-value analysis and the associated rules.

Test Case Generation

The Apollo tool generates a set of test cases, for unit testing of the given *access-program*, using all possible combinations of the test values.

Anticipated Output Generation

For each test case, the Apollo tool executes the design specification and generates the anticipated value(s) of the output parameter(s). After executing all the test cases, the output file is written to the specified directory.

4.1.2 Algorithm

The algorithm followed by the Apollo tool during the test case generation along with the anticipated test outcome, is presented in two parts. Figure 3 illustrates the algorithm followed by the Apollo tool during the generation of a set of test cases; whereas Figure 4 illustrates the Look Before You Execute (L-Bye) algorithm followed by the Apollo tool, during the execution of the design specification, for generating the anticipated test outcome, for each test case.

4.1.2.1 Algorithm for Generation of Test Cases

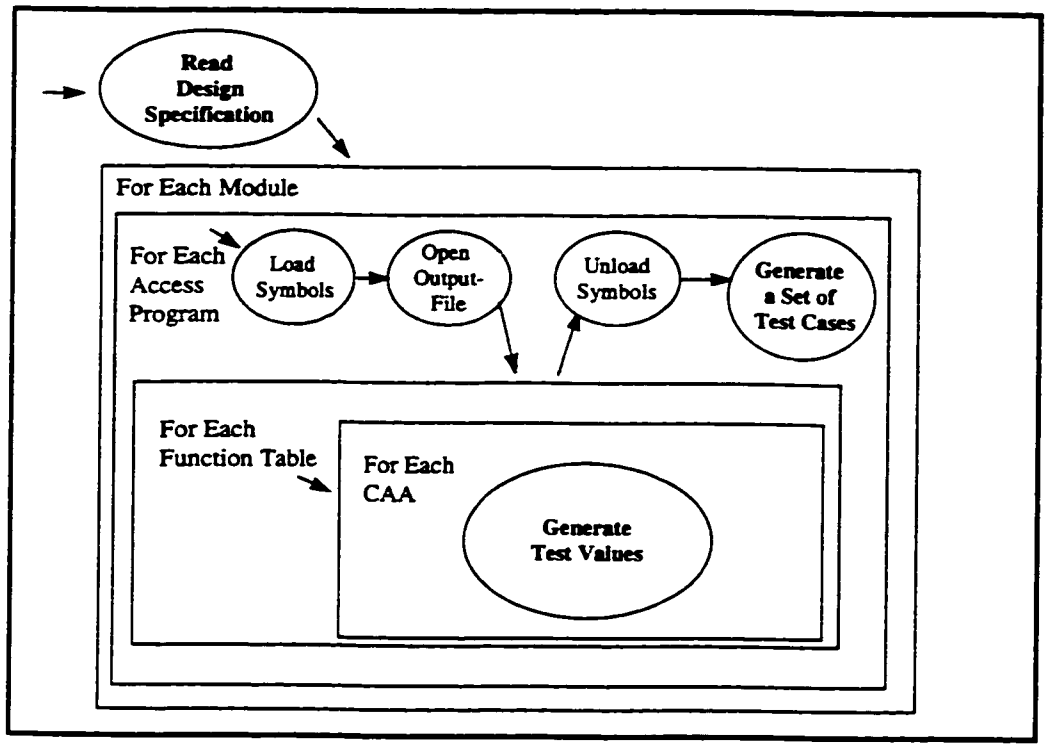

Figure 3 - Algorithm for Generation of a Set of Test Cases

The important steps in test case generation algorithm (Figure 3) are briefly described below:

- After parsing the design specification document, the Apollo tool sub-divides the task to *access-program* level, and loads global-scope, module-scope, and *access-program*-scope symbols.
- After loading the symbols, the Apollo tool creates a new output file, identifies the list of input parameters to the *access-program,* identifies the list of output parameters to the *access-program,* and sub-divides the task to *function table* level.
- The Apollo tool generates test values for each input parameter based on boundary-value analysis and associated rules, using all the conditions that are specified in the given *access-program.*

- The Apollo tool generates more test values for each input parameter using partition and zero inclusion rules.
- After completing the test value generation (for each input parameter), the tool generates a set of test cases using all possible combination of the input parameters.

This algorithm is presented in Table 15 using pseudo-code.

Table 15: Algorithm for Generation of a Set of Test Cases

```
Input:      parsed design specification
Output:     a set of test cases
Algorithm:
begin
       create and open a new output file;
       write version control information to output file;
       identify the list of input parameters;
       identify the list of output parameters;
       write input and output parameter information to output file;
       load global-scope symbols;
       load module-scope symbols;
       load access-program-scope symbols;
       for each function table begin
              for each condition begin
                     for each input parameter begin
                            generate test values from boundary value analysis;
                     end;
              end;
       end;
       for each input parameter begin
              generate partition test values;
              apply zero-inclusion rule;
       end;
       for each input parameter begin
              write test value summary to output file;
       end;
       generate a set of test cases using all possible combinations of test
       values of all input parameters;
end {generation of a set of test cases};
```

4.1.2.2 Look Before You Execute (L-Bye) Algorithm : Execution of Design Specification

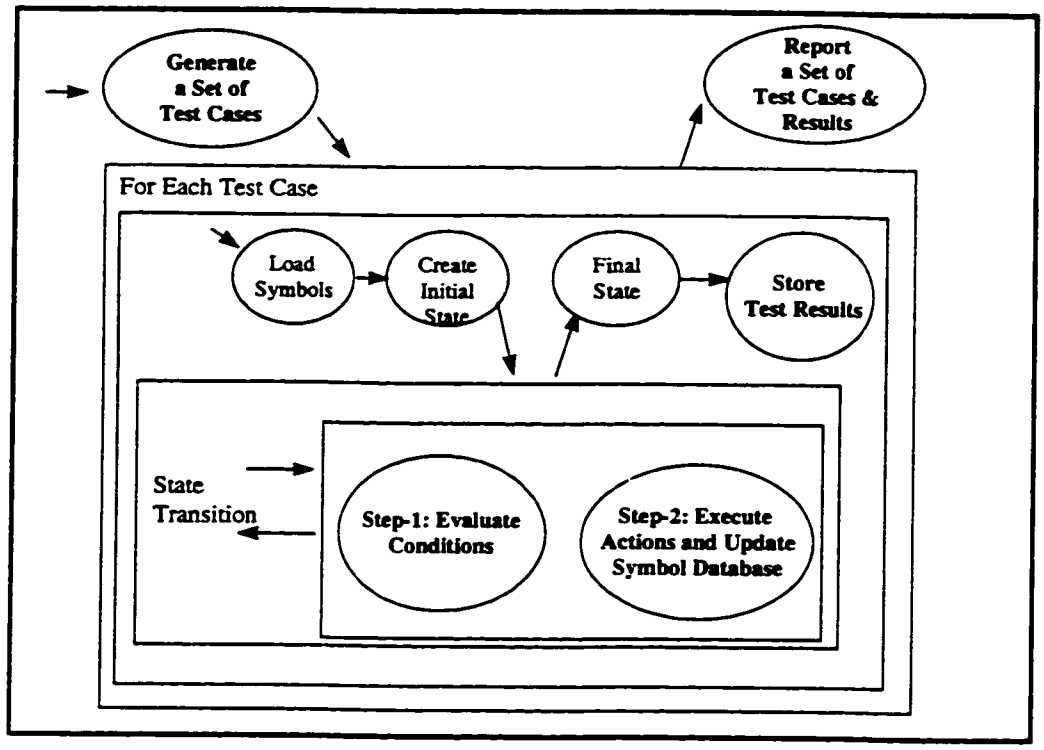

Figure 4 - Execution of Design Specification : L-Bye Algorithm

The primary steps in Look Before You Execute algorithm (Figure 4) are briefly described below:

For each test case:

- **Symbol database:** The Apollo tool loads global-scope, module-scope, and *access-program*-scope symbols; and initializes all the variables with their default values.
- **Initial State:** all the input parameters of the *access-program* are initialized based on the current test case information, "before entering into the *access-program*".

- **State Transition:** The Apollo tool follows a two-step strategy given below at every state transition:

  Step-One: The Apollo tool evaluates all of the conditions (that are relevant at a given state) from design specification (and reports exceptions) in order to determine that one and only one condition is valid for state transition.

  Step-Two: The Apollo tool executes all the actions that are associated with that "one and only one condition", and updates the symbol database appropriately after execution of "each action".

- **Final State:** After reaching the final state, the Apollo tool collects the anticipated values of the output parameters of the *access-program* from symbol database and reports a summary (along with the input parameter information) "prior to exiting the *access-program*".

Exception Handling:

If an exception is encountered during the execution of design specification, for a given test case, Apollo aborts the execution with respect to that specific test case alone and reports the details of the test case along with an appropriate error message. For example, (a) incomplete specification is reported if the execution reaches a dead-end state, i.e., none of the relevant conditions at the given point of the execution are satisfied; (b) non-determinism is reported if the execution reaches an ambiguous state, i.e., if more than one path is valid for state transition.

It should be noted that Apollo continues execution of remaining test cases even after reporting an exception.

This algorithm is presented in Table 16 using pseudo-code.

Table 16: Execution of Design Specification : L-Bye Algorithm

```
Input:      a set of test cases
Output:     anticipated test outcome for each test case
Algorithm:
begin
       for each test case begin
              {load symbol database}
              load global-scope symbols;
              load module-scope symbols;
              load access-program-scope symbols;
              {Initial State}
              for each symbol begin
                     initialize the symbol with default value;
              end;
              for each input parameter begin
                     initialize input parameter based on current test case
                     information;
              end;

              {State Transition}
              for each function table begin
                     {Step-1: Check for Exceptions}
                     ValidConditionCount = zero;
                     for each condition begin
                            evaluate the condition;
                            if condition is valid then begin
                                   increment ValidConditionCount;
                            end;
                     end;
                     if ValidConditionCount > 1 then begin
                            report nondeterminism;
                            and continue with the next test case;
                     else if ValidConditionCount is equal to zero then begin
                            report incomplete specification;
                            and continue with the next test case;
                     end;
                     {Step-2: Execute associated actions}
                     for each condition begin
                            evaluate the condition;
                            if condition is valid then begin
                                   execute associated actions;
                                   update symbol database;
                                   continue with execution of next function
                                   table;
                            end;
                     end;
              end;
              {Final State}
              for each output parameter begin
                     collect the value of output parameter from symbol database;
              end;
              write test case and anticipated test outcome information to output
              file;
       end;
end {generation of anticipated test outcome};
```

4.1.3 Module Responsibilities and Encapsulation

Table 17 summarizes the responsibilities and encapsulation of twelve modules that are specific to the Apollo tool. Section 0 gives a detailed design description of the individual modules.

Table 17 - Module Responsibilities and Encapsulation

Module	Responsibilities	Encapsulation
UMN	• load Delta Values document • load SDD document • user interface • subdivide the task to *access-program* level	• command line options • overall data and control flow
UGA	• manage generation of anticipated test outcome for a given *access-program* • identify the list of input and output parameters for a given *access-program* • generate all valid test values, for each input parameter • generate a set of test cases along with anticipated test outcome • check for non-determinism and incomplete specification	• logic for identification of input and output parameters • structure of an *access-program* in a design specification document • test case generation algorithm • strategy for execution of a given test case • algorithm for checking non-determinism and incomplete specification
UEX	• execute condition • execute a set of actions • check for overflow, underflow, divide-by-zero, etc.	• strategy for execution of different types of conditions and a set of actions. • algorithm for checking overflow, underflow, divide-by-zero, etc.

continued...

Table 17 - Module Responsibilities and Encapsulation (Continued)

Module	Responsibilities	Encapsulation
UGC	• generate valid test values for a list of input parameters using a given condition	• structure of various conditions • logic for classification of conditions • data and control flow that is specific to each condition
UGR	• generate valid test values for a given input parameter using a given range-expression	• logic for generation of test values for a range expression.
UGS	• generate test values for a given input parameter using a given Simple Condition	• test value generation algorithm
UGG	• generate test values using partition and other general rules	• partition and other general rules
UCA	• library of utility functions	• algorithms that are specific to each utility function
UVH	• hold and provide access to values • associate a set of values with a given symbol	• holder data structure
UOR	• module initialization and clean-up	• correct sequence of initialization and termination of all modules.
UOF	• generate output file for a given *access-program*	• format and contents of an output file

In addition to these modules, the Apollo tool makes use of several generic libraries that are developed or obtained by AECL for the development of CASE tools.

4.2 Detailed Design

4.2.1 UMN (Mainline) Module

Responsibilities

The UMN module implements the mainline functionality for the Apollo tool. Its important responsibilities are to

- read the input design specification document;
- analyze the user's request and subdivide the task to the *access-program* level; and
- invoke the MGT module to generate a set of test cases along with the anticipated test outcome for an *access-program* that is under consideration.

Encapsulation

The UMN module encapsulates

- the command line options,
- the overall control flow of the Apollo tool,
- the logic to subdivide the task to the *access-program* level, and
- the correct sequence of loading the symbol database.

4.2.2 UGA (Generate Test Cases for *Access-program*) Module

Responsibilities

The UGA module implements the data and control flow that is associated with the generation of test cases along with anticipated test outcome for a given *access-program*. Its important responsibilities are to

- manage the generation of anticipated test outcome for a given *access-program*;
- identify the list of input parameters for a given *access-program*;

- identify the list of output parameters for a given *access-program*; and
- generate all the valid test values, for each input parameter;
- generate a set of test cases;
- generate the anticipated test result for each test case;
- check for non-determinism; and
- check for incomplete specification.

Encapsulation

The UGA module knows about the formal contents of various sections and tables in an *access-program* in a given design specification document. This module encapsulates

- the structure of an *access-program* in a design specification document;
- the data and control flow that is specific to each part of a given *access-program*;
- the logic for identification of input and output parameters, along with the relevant symbol information;
- test case generation algorithm;
- strategy for execution of a given test case; and
- algorithm for checking non-determinism and incomplete specification.

4.2.3 UEX (Execution) Module

Responsibilities

The UEX module manages the execution of conditions and actions. Its important responsibilities are to

- execute a given condition,
- execute a set of actions, and
- check for overflow, underflow, divide-by-zero, etc.

Encapsulation

The UEX module encapsulates

- the strategy for execution of different types of conditions and actions, and
- the algorithm for checking overflow, underflow, divide-by-zero, etc.

4.2.4 UGC (Generate Test Values for a Condition) Module

Responsibilities

The UGC module implements the data and control flow that is associated with the test value generation for a given condition. Its important responsibilities are to

- identify the type of the condition (Compound Condition (CC); Range Condition (RC), Simple Condition (SC), etc.), and
- generate valid test values for each input parameter using a given condition (by applying the rules that are associated with the type of the given condition).

Encapsulation

The UGC module knows about the format of various conditions that are expected in input design specification document. This module encapsulates

- the structure of various conditions in input design specification document,
- the data and control flow that is specific to each type of a given condition, and
- the logic for classification of conditions.

4.2.5 UGR (Generate Test Values for a Range Condition) Module

Responsibilities

The UGR module implements the data and control flow that is associated with the test value generation for a given Range Condition. Its important responsibilities are to

- identify a valid Range Condition, and
- generate valid test values for a given input parameter by analyzing the given Range Condition.

Encapsulation

The UGR module encapsulates

- the logic for identification of a valid Range Condition,
- the data and control flow that is specific to a given Range Condition, and
- the algorithm(s) for generation of valid test values using a given Range Condition

4.2.6 UGS (Generate Test Values for a Simple Condition) Module

Responsibilities

The UGS module implements the data and control flow that is associated with the test value generation for a given Simple Condition. Its important responsibility is to generate valid test values for a given input parameter by analyzing the given Simple Condition.

Encapsulation

The UGS module encapsulates

- data and control flow that is specific to various Simple Conditions, and
- algorithms and rules that are specified in section 3.2.7.1 for generation of test values using a given Simple Condition.

4.2.7 UGG (Generate Test Values using General Rules) Module

Responsibilities

The UGG module's important responsibility is to generate valid test values for a given input parameter by applying partition and zero-inclusion rules.

Encapsulation

The UGS module encapsulates

- algorithm for partition rule(s), and
- algorithm for zero-inclusion rule.

4.2.8 UCA (Common Access-Programs) Module

Responsibilities

The UCA module consists of general utility functions which are called by a number of modules that are specific to the Apollo tool.

Encapsulation

The UCA module encapsulates

- the logic for getting and checking the lower and upper range of a given symbol, and
- the logic for getting and checking the lower and upper bounds of a given expression.

4.2.9 UVH (Value Holder) Module

Responsibilities

The UVH module holds and provides access to test values associated with an input parameter. Its important responsibilities are to

- hold and provide access to values, and
- associate a set of values with a given symbol.

Encapsulation

The type of data structure of the holder is internal to the module.

4.2.10 UOR (Initialization and Clean-up) Module

Responsibilities

The UOR module implements a part of the mainline functionality for the UT tool. Its important responsibilities are to initialize and terminate all modules that are specific to the Apollo tool.

Encapsulation

The UOR module encapsulates the correct sequence of initialization and termination of all the modules that are specific to the Apollo tool.

4.2.11 UOF (Ouput-File) Module

Responsibilities

The UOF module manages the generation of an output-file. The format of a typical output file(s) is given in Appendix-B. Its important responsibilities are to

- generate a unique name for each output-file for a every *access-program*, and
- present the summary of a set of test cases along with the anticipated test outcome along with all pertinent information, using a specific output format.

Encapsulation

The UOF module encapsulates

- the location of the output-files,
- the logic for generation of unique names for each output-file,
- the format for presentation of the summary of a set of test cases along with the anticipated test outcome.

5. CASE STUDY AND DISCUSSION

The following two sample "Design Specification" documents are prepared, based on an industrial example, in order to test the feasibility of the approach proposed in this thesis.

- A sample **clean design specification** document (i.e., without any errors) was prepared describing the design details of a hypothetical industrial example.
- A sample **draft design specification** document was prepared by introducing certain typical/subtle errors (into clean document), in order to reflect a practical situation.

This chapter gives a brief description of both the sample documents which are used as separate "input document" for Apollo and discusses the results from this case study.

5.1 Sample Design Specification (clean)

The sample design specification (clean) using tabular notation (along with a brief explanation using natural language) is presented in Appendix_A. This design specification describes a small part of the design details of one of the modules (which can be considered as a representative sample) associated with a safety system of a nuclear reactor. It should be noted that a hypothetical POWER module was created to illustrate the capabilities of the tool. The design details of POWER module are described in the following section.

5.1.1 POWER Module

Some of the important responsibilities of the POWER module are:

- check the set-point value associated with the reactor power, and give appropriate feedback (OK or warning message) to the operator;
- determine the status of the reactor by comparing the measured power with the set-point and give appropriate feedback to the operator;

- determine the "indicator-status of the reactor on the control panel" by comparing the measured power with the set-point and turn-on the appropriate indicator-light (green, blue, red, flashing-red etc.); and
- determine the "alarm-status of the reactor on the control panel" by comparing the measured power with the set-point and turn-on the warning alarm as and when required.

The following four *access-programs* fulfill the specific responsibilities of the POWER module:

- PWR$Check_Set_Point
- PWR$Power_Status
- PWR$Display_Status
- PWR$Alarm_Status

Only these four *access-programs* from POWER module have been discussed in this thesis, since the primary purpose is only to illustrate the capabilities of the Apollo tool.

5.1.2 Access-Program: PWR$Check_Set_Point

The responsibilities (i.e., high-level functional requirements), design description and tabular specification of this *access-program* are presented in this section.

Responsibilities

This *access-program* verifies that the user-supplied set-point (of reactor power) is with in the valid range and gives an appropriate feed-back to the operator. If the value of set-point is outside valid-range, a default value is used.

Design Description (Natural-Language):

High-Level Design:

- check the value of the power-set-point using the valid range of the expected-power-output from the reactor and the dead-band associated with the power-set-point;

- if the value of the power-set-point is valid (i.e., with in the valid range of power-set-point), then do not change the value; and
- if the value of the power-set-point is invalid (i.e., outside of the valid range of power-set-point), then give an appropriate warning message to the operator, and assign a valid value to the power-set-point.

Detailed Design:

The value of the output variable, V_Power_Set_point, is decided based on the following logic:

- if the value is less than the lower-valid-range (i.e. valid_lower_range_of_power plus half-of-the dead-band) then assign Ceiling(valid_lower_range_of_power plus half-of-the dead-band) to the output variable, V_Power_Set_point;
- if the value is greater than the upper-valid-range (i.e. valid_upper_range_of_power minus half-of-the dead-band) then assign Floor(valid_lower_range_of_power minus half-of-the dead-band) to the output variable, V_Power_Set_point;
- if the value is with in the valid range then do not change the value of the output variable, V_Power_Set_point.

The state of the output variable, V_Check_Result, is decided based on the following logic:

- if the value of the power-set-point is with in the valid range then give a valid-set-point message.
- if the value of the power-set-point is out-of- range (i.e., invalid) then give a warning message.

Tabular Specification:

The design details of this *access-program* are specified in the following vertical condition table [Please refer to Appendix-A, for more design details].

Table Check_Set_Point

	V_Power_Set_Point < (C_P_Lower_Range + (0.5 * C_Power_Dead_Band))	(V_Power_Set_Point >= (C_P_Lower_Range + (0.5 * C_Power_Dead_Band))) AND (V_Power_Set_Point <= (C_Danger_Power - (0.5 * C_Power_Dead_Band)))	V_Power_Set_Point > (C_Danger_Power - (0.5 * C_Power_Dead_Band))
V_Power_Set_Point	CEILING(C_P_Lower_Range + (0.5 * C_Power_Dead_Band))	V_Power_Set_Point	FLOOR(C_Danger_Power - (0.5 * C_Power_Dead_Band))
V_Check_Result	e_P_SP_Warning	e_Valid_P_SP	e_P_SP_Warning

5.1.3 Access-Program: PWR$Power_Status

The responsibilities (i.e., high-level functional requirements), design description and tabular specification of this *access-program* are presented in this section.

Responsibilities

This *access-program* determines the status of the reactor by comparing the measured power (from the reactor) with the set-point (for the power).

Design Description (Natural-Language):

High-Level Design:

- determine the status of the reactor by comparing the measured power with the set-point

Detailed Design:

The status of the output variable, V_Power_Status, is decided based on the following logic:

- if the value is less than the desired range (i.e. power-set-point minus half-of-the dead-band) and outside the danger zone then assign e_Sub_Normal;

- if the value is with in the desired range and outside the danger zone then assign e_Normal;
- if the value is more than the desired range (i.e. power-set-point plus half-of-the dead-band) and outside danger zone then assign e_Above_Normal;
- if the value inside danger zone but outside critical zone then assign e_Init_P_SetBack (i.e., initialize the process of decreasing the Power);
- if the value inside critical zone then assign e_Emergency_Shut_Down (i.e., initialize the process of emergency shut-down of the reactor).

Tabular Specification:

The design details of this *access-program* are specified in following vertical condition table [Please refer to Appendix-A, for more design details].

Table Power_Status

	Result
Condition	V_Power_Status
V_Reactor_Power < (V_Power_Set_Point - (0.5 * C_Power_Dead_Band)) AND (V_Reactor_Power < C_Danger_Power)	e_Sub_Normal
(V_Reactor_Power >= (V_Power_Set_Point - (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power <= (V_Power_Set_Point + (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power < C_Danger_Power)	e_Normal
(V_Reactor_Power > (V_Power_Set_Point + (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power < C_Danger_Power)	e_Above_Normal
(V_Reactor_Power >= C_Danger_Power) AND (V_Reactor_Power < C_Critical_Power)	e_Init_P_SetBack
(V_Reactor_Power >= C_Critical_Power)	e_Emergency_Shut_Down

5.1.4 Access-Program: PWR$Display_Status

The responsibilities (i.e., high-level functional requirements), design description and tabular specification of this *access-program* are presented in this section.

Responsibilities

This *access-program* determines the visual display status of the reactor (i.e., blue/green/amber etc.) on the control panel by comparing the measured power (from the reactor) with the set-point (for the power).

Design Description (Natural-Language):

High-Level Design:

- determine the visual display status of the reactor (i.e., blue/green/amber etc.) on the control panel by comparing the measured power with the set-point

Detailed Design:

The status of the output variable, V_Display_Status, is decided based on the following logic:

- if the value is less than the desired range (i.e. power-set-point minus half-of-the dead-band) and outside the danger zone then assign e_Blue;
- if the value is with in the desired range and outside the danger zone then assign e_Green;
- if the value is more than the desired range (i.e. power-set-point plus half-of-the dead-band) and outside danger zone then assign e_Amber;
- if the value inside danger zone but outside critical zone then assign e_Red;
- if the value inside critical zone then assign e_Flashing_Red.

Tabular Specification:

The design details of this *access-program* are specified in following vertical condition table [Please refer to Appendix-A, for more design details].

Table display_Status

	Result
Condition	V_Display_Status
V_Reactor_Power < (V_Power_Set_Point - (0.5 * C_Power_Dead_Band)) AND (V_Reactor_Power < (1.05 * C_Danger_Power))	e_Blue
(V_Reactor_Power >= (V_Power_Set_Point - (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power <= (V_Power_Set_Point + (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power < (1.05 * C_Danger_Power))	e_Green
(V_Reactor_Power > (V_Power_Set_Point + (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power < (1.05 * C_Danger_Power))	e_Amber
(V_Reactor_Power >= (1.05 * C_Danger_Power)) AND (V_Reactor_Power < (1.05 * C_Critical_Power))	e_Red
(V_Reactor_Power >= (1.05 * C_Critical_Power))	e_Flashing_Red

5.1.5 Access-Program: PWR$Alarm_Status

The responsibilities (i.e., high-level functional requirements), design description and tabular specification of this *access-program* are presented in this section.

Responsibilities

This *access-program* determines the audible alarm status of the reactor (i.e., off/intermittent/continuous) on the control panel by comparing the measured power (from the reactor) with the set-point (for the power).

[NOTE: A slight time-delay between visual display and audible alarm is preferred].

Design Description (Natural-Language):

High-Level Specification:

- determine the audible alarm status of the reactor (i.e., off/intermittent/continuous) on the control panel by comparing the measured power with the set-point

Detailed Specification:

The status of the output variable, V_Alarm_Status, is decided based on the following logic:

- if the value is outside the danger zone then assign e_Off (i.e. do not turn-on the alarm);
- if the value inside danger zone but outside critical zone then assign e_Intermittent (i.e., turn-on intermittent audible alarm);
- if the value inside critical zone then assign e_Continuous (i.e., turn-on continuous audible alarm);.

[NOTE: A slight time-delay between visual display and audible alarm is introduced by using a factor of 1.1, while deciding the status of the alarm]

Tabular Specification:

The design details of this *access-program* are specified in following vertical condition table [Please refer to Appendix-A, for more design details].

Table Alarm_Status

	Result
Condition	V_Alarm_Status
(V_Reactor_Power < (1.1 * C_Danger_Power))	e_Off
(V_Reactor_Power >= (1.1 * C_Danger_Power)) AND (V_Reactor_Power < (1.1 * C_Critical_Power))	e_Intermittent
(V_Reactor_Power >= (1.1 * C_Critical_Power))	e_Continuous

5.2 Results from Apollo

The results, four output-files (one for each access program), that are generated by the Apollo tool are presented in Appendix-B.

5.2.1 Access-Program: PWR$Check_Set_Point

The set of test values and the set of test cases that are generated by Apollo, for *access-program* PWR$Check_Set_Point are presented in Table 18 (This section is taken directly from the output file: "B.1 pwr_ch.etr (version 1.0)", in Appendix-B. For complete results, see Appendix-B).

Table 18 - Results from Apollo

```
------------------------------------------------------------------------
Section 3: Test Values from Boundary Value Analysis
    V_Power_Set_Point     : 10, 11, 12, 14, 15, 16, 65, 114, 115, 116, 117,
                            119, 120

------------------------------------------------------------------------

------------------------------------------------------------------------
Section 5: Test Cases along with Expected Test Results

Test Case#  Input              Expected Output  Expected Output
            V_Power_Set_Point  V_Check_Result   V_Power_Set_Point
            ------------       ------------     ------------
1           10                 e_P_SP_Warning   15
2           11                 e_P_SP_Warning   15
3           12                 e_P_SP_Warning   15
4           14                 e_P_SP_Warning   15
5           15                 e_Valid_P_SP     15
6           16                 e_Valid_P_SP     16
7           65                 e_Valid_P_SP     65
8           114                e_Valid_P_SP     114
9           115                e_Valid_P_SP     115
10          116                e_P_SP_Warning   115
11          117                e_P_SP_Warning   115
12          119                e_P_SP_Warning   115
13          120                e_P_SP_Warning   115

------------------------------------------------------------------------
```

Verification:

A sample verification activity is presented, only for this *access-program.*

[To avoid the repetition, verification is not presented for the remaining three *access-programs*].

Part-1: Test Values:

The following set of test values are expected from each condition:

Step-A:

Condition: V_Power_Set_Point < (C_P_Lower_Range + 0.5 * C_Power_Dead_Band)

Test values (from rules in section 3.2.7.1.3): 10, 11, and 14

Step-B:

Condition: V_Power_Set_Point >= (C_P_Lower_Range + 0.5 * C_Power_Dead_Band)

Test values (from rules in section 3.2.7.1.2): 15, 16, 119, and 120

Step-C:

Condition: V_Power_Set_Point <= (C_Danger_Power - (0.5 * C_Power_Dead_Band)

Test values (from rules in section 3.2.7.1.4): 10, 11, 114 and 115

Step-D:

Condition: V_Power_Set_Point > (C_Danger_Power - (0.5 * C_Power_Dead_Band)

Test values (from rules in section 3.2.7.1.1): 116, 119, and 120

Step-E:

Set of Test Values: 10, 11, 14, 15, 16, 114, 115, 116, 119, and 120

Step-F: Partition range test values (from rules in section 3.2.7.4): 12, 65, and 117

Step-G:

Final test value set (thirteen test values):

10, 11, 12, 14,

15, 16, 65, 114, 115,

116, 117, 119, and 120

Part-2: Test Cases

A set of thirteen test cases are expected since this *access-program* has only one input .

Part-3: Anticipated Outcome:

1. For first four test cases a warning message and Set-Point of 15 are expected, since the input is below the valid range,
2. For next five test cases a valid message and Set-Point same as the input value are expected, since the input is within the valid range, and
3. For last four test cases a warning message and Set-Point of 115 are expected, since the input is above the valid range.

Actual Results from Apollo:

Part-1: Test Values: The thirteen test values that are presented in section 3, in Table 18 match the expected test values.

Part-2: Test Cases: The thirteen test cases that are presented in section 5, in Table 18 match the set of expected test cases.

Part-3: Anticipated Outcome: The thirteen test cases along with the anticipated test outcome that are presented in section 5, in Table 18 match the set of anticipated outcome.

General Observations:

The following general observations can be made from a through analysis of the results that are presented in the Table 18:

- the Apollo tool identifies and reports the relevant input and output parameters (i.e. one input parameter: V_Power_Set_Point; and two output parameters: V_Power_Set_Point, and V_Check_Result);
- the Apollo tool generates a total of thirteen test values for the input parameter, V_Power_Set_Point, based on the boundary-value analysis and other specific rules;
- the Apollo tool generates a set of thirteen test cases; and
- for each test case, the Apollo tool calculates and reports appropriate anticipated test outcome (the status of both output parameters: V_Power_Set_Point, and V_Check_Result) as expected.

5.2.2 Access-Program: PWR$Power_Status

The set of test values and a small sample of test cases that are generated by Apollo are presented in Table 19 [This section is taken directly from the output file: "B.2 pwr_po.etr (version 1.0)", in Appendix-B. For complete results, see Appendix-B.].

Table 19 - Results from Apollo

```
================================================================================
Section 3: Test Values from Boundary Value Analysis
    V_Reactor_Power      : 0, 1, 12, 13, 24, 25, 26, 52, 65, 77, 104, 105,
                           106, 116, 117, 129, 130, 131, 132, 134, 135,
                           136, 145, 159, 160, 161, 192, 205, 249, 250

    V_Power_Set_Point    : 10, 11, 12, 14, 15, 16, 65, 67, 119, 120

================================================================================
================================================================================
Section 5: Test Cases along with Expected Test Results

Test Case#  Input           Input              Expected Output
            V_Reactor_Power V_Power_Set_Point V_Power_Status
            ------------    ------------      ------------
1           0               10                e_Normal
--          --              --                --
56          25              16                e_Normal
--          --              --                --
67          26              65                e_Sub_Normal
--          --              --                --
78          52              67                e_Normal
--          --              --                --
89          65              119               e_Sub_Normal
--          --              --                --
156         129             16                e_Above_Normal
--          --              --                --
167         130             65                e_Init_P_SetBack
--          --              --                --
189         132             119               e_Init_P_SetBack
--          --              --                --
200         134             120               e_Init_P_SetBack
--          --              --                --
234         159             14                e_Init_P_SetBack
--          --              --                --
245         160             15                e_Emergency_Shut_Down
--          --              --                --
267         192             65                e_Emergency_Shut_Down
--          --              --                --
289         249             119               e_Emergency_Shut_Down
--          --              --                --
300         250             120               e_Emergency_Shut_Down

================================================================================
```

The following general observations can be made from a through analysis of the results that are presented in Table 19 (and the output file: "B.2 pwr_po.etr (version 1.0)", in Appendix-B):

- the Apollo tool identifies and reports the relevant input and out parameters (i.e. two input parameters: V_Reactor_Power, and V_Power_Set_Point; and one output parameter: V_Power_Status);
- the Apollo tool generates a total of ten test values for the input parameter, V_Power_Set_Point; and thirty test values for the input parameter, V_Reactor_Power based on the boundary-value analysis and other specific rules;

- the Apollo tool generates a set of three hundred test cases, using all possible combinations of the test values of the input parameters; and
- for each test case, the Apollo tool determines and reports an appropriate anticipated test outcome (the status of V_Power_Status) as expected.

5.2.3 Access-Program: PWR$Display_Status

The set of test values and a small sample of test cases that are generated by Apollo are presented in Table 20 [This section is taken directly from the output file: "B.3 pwr_di.etr (version 1.0)", in Appendix-B. For complete results see Appendix-B.].

Table 20 - Results from Apollo

```
================================================================================
Section 3: Test Values from Boundary Value Analysis
    V_Reactor_Power      : 0, 1, 12, 13, 24, 25, 26, 52, 65, 68, 80, 104,
                           105, 106, 119, 120, 134, 135, 136, 137, 138,
                           152, 167, 168, 169, 193, 209, 249, 250

    V_Power_Set_Point    : 10, 11, 12, 14, 15, 16, 65, 67, 119, 120

================================================================================
================================================================================
Section 5: Test Cases along with Expected Test Results

Test Case#  Input            Input              Expected Output
            V_Reactor_Power V_Power_Set_Point V_Display_Status
            ------------    ------------      ------------
1           0               10                e_Green
--          --              --                --
20          1               120               e_Blue
--          --              --                --
41          24              10                e_Green
--          --              --                --
61          26              10                e_Amber
--          --              --                --
82          65              11                e_Amber
--          --              --                --
98          68              67                e_Green
--          --              --                --
120         104             120               e_Blue
--          --              --                --
140         106             120               e_Green
--          --              --                --
161         134             10                e_Amber
--          --              --                --
180         135             120               e_Green
--          --              --                --
191         137             10                e_Red
--          --              --                --
210         138             120               e_Red
--          --              --                --
230         167             120               e_Red
231         168             10                e_Flashing_Red
--          --              --                --
250         169             120               e_Flashing_Red
--          --              --                --
270         209             120               e_Flashing_Red
--          --              --                --
290         250             120               e_Flashing_Red

================================================================================
```

The following general observations can be made from a through analysis of the results that are presented in Table 20 (and the output file: "B.3 pwr_di.etr (version 1.0)", in Appendix-B):

- the Apollo tool identifies and reports the relevant input and out parameters (i.e. two input parameters: V_Reactor_Power, and V_Power_Set_Point; and one output parameter: V_Display_Status);
- the Apollo tool generates a total of ten test values for the input parameter: V_Power_Set_Point; and twenty nine test values for the input parameter: V_Reactor_Power based on the boundary-value analysis and other specific rules;
- the Apollo tool generates a set of two hundred and ninety test cases, using all possible combinations of the test values of the input parameters; and
- for each test case, the Apollo tool determines and reports an appropriate anticipated test outcome (the status of the output parameter, V_Display_Status) as expected.

5.2.4 Access-Program: PWR$Alarm_Status

The set of test values and set of test cases that are generated by Apollo are presented in Table 21 [This section is taken directly from the output file: "B.4 pwr_al.etr (version 1.0)", in Appendix-B. For complete results see Appendix-B.].

Table 21 - Results from Apollo

```
--------------------------------------------------------------------------
Section 3: Test Values from Boundary Value Analysis
    V_Reactor_Power      : 0, 1, 71, 142, 143, 144, 159, 175, 176, 177, 213,
                           249, 250

--------------------------------------------------------------------------

--------------------------------------------------------------------------
Section 5: Test Cases along with Expected Test Results

Test Case#  Input           Expected Output
            V_Reactor_Power V_Alarm_Status
            -----------     ------------
1           0               e_Off
2           1               e_Off
3           71              e_Off
4           142             e_Off
5           143             e_Intermittent
6           144             e_Intermittent
7           159             e_Intermittent
8           175             e_Intermittent
9           176             e_Continuous
10          177             e_Continuous
11          213             e_Continuous
12          249             e_Continuous
13          250             e_Continuous

--------------------------------------------------------------------------
```

The following general observations can be made from a through analysis of the results that are presented in Table 21 (and the output file: "B.4 pwr_al.etr (version 1.0)", in Appendix-B):

- the Apollo tool identifies and reports the relevant input and out parameters (i.e. one input parameter: V_Reactor_Power; and one output parameter: V_Alarm_Status);
- the Apollo tool generates a total of thirteen test values for the input parameter: V_Reactor_Power based on the boundary-value analysis and other specific rules;
- the Apollo tool generates a set of thirteen test cases; and
- for each test case, the Apollo tool determines and reports an appropriate anticipated test outcome (the status of the output parameter, V_Alarm_Status) as expected.

5.2.5 General Discussion

From the results presented in these four output files it can be observed that:

- The Apollo tool can evaluate both simple and complex conditions and also complex mathematical expressions;
- The Apollo tool can generate appropriate test values for each input parameter based on boundary-value analysis (using the conditions) and other specific rules;
- The Apollo tool can generate a set of test cases using all the possible combinations of the test values of the input parameters; and
- For each test case, the Apollo tool can determine appropriate expected value(s) of the output parameter(s), by executing the design specification.

5.3 Sample Design Specification (draft)

In general, any draft design specification is expected to contain some implicit errors (typographic, logical, human errors). As a result, in order to reflect a practical situation, a sample draft design specification is prepared by introducing certain typical/subtle errors (into the clean design specification) and presented in Appendix-C. The results that are generated by the Apollo tool, using this draft design specification document, are presented in Appendix-D, and are briefly discussed in the following sections.

5.4 Results from Apollo

5.4.1 Access-Program: PWR$Check_Set_Point (draft)

The preliminary design details of this *access-program* are specified in following vertical condition table [Please refer to Appendix-C, for more design details].

Draft Design Specification:

Table Check_Set_Point

	V_Power_Set_Point < (C_P_Lower_Range + (0.5 * C_Power_Dead_Band))	(V_Power_Set_Point >= (C_P_Lower_Range + (0.5 * C_Power_Dead_Band))) AND (V_Power_Set_Point <= (C_Danger_Power - (0.5 * C_Power_Dead_Band)))	V_Power_Set_Point >= (C_Danger_Power - (0.5 * C_Power_Dead_Band))
V_Power_Set_Point	CEILING(C_P_Lower_Range + (0.5 * C_Power_Dead_Band))	V_Power_Set_Point	FLOOR(C_Danger_Power - (0.5 * C_Power_Dead_Band))
V_Check_Result	e_P_SP_Warning	e_Valid_P_SP	e_P_SP_Warning

Description of Error introduced:

The error that was introduced into the design specification (clean), and its consequences are as follows:

Error:

A subtle error was introduced by modifying the condition in the last column as given below:

"V_Power_Set_Point > (C_Danger_Power - (0.5 * C_Power_Dead_Band))"

is changed to

"V_Power_Set_Point >= (C_Danger_Power - (0.5 * C_Power_Dead_Band))"

Consequences:

This design specification is non-deterministic.

In other words, the following two conditions will be satisfied, when input parameter, V_Power_Set_Point has a value of "(C_Danger_Power - (0.5 * C_Power_Dead_Band))":

Conditon-1:

"V_Power_Set_Point >= (C_Danger_Power - (0.5 * C_Power_Dead_Band))"; and

Condition-2:

(V_Power_Set_Point >= (C_P_Lower_Range + (0.5 * C_Power_Dead_Band))) AND
(V_Power_Set_Point <=
(C_Danger_Power - (0.5 * C_Power_Dead_Band)))

Results:

The set of test values and set of test cases that are generated by Apollo, for *access-program* PWR$Check_Set_Point, are presented in Table 22 [This section is taken directly from the output file: "D.1 pwr_ch.etr (version 0.0)", in Appendix-D. For complete results see Appendix-D.].

Table 22 - Results from Apollo

```
================================================================================
Section 3: Test Values from Boundary Value Analysis
    V_Power_Set_Point     : 10, 11, 12, 14, 15, 16, 65, 114, 115, 116, 117,
                            119, 120

================================================================================
================================================================================
Section 5: Test Cases along with Expected Test Results

WARNING: Tool encountered non-deterministic test case(s);
-------  please review design document.


Test Case#  Input              Expected Output Expected Output
            V_Power_Set_Point V_Check_Result  V_Power_Set_Point
            ------------      ------------    ------------
1           10                e_P_SP_Warning  15
2           11                e_P_SP_Warning  15
3           12                e_P_SP_Warning  15
4           14                e_P_SP_Warning  15
5           15                e_Valid_P_SP    15
6           16                e_Valid_P_SP    16
7           65                e_Valid_P_SP    65
8           114               e_Valid_P_SP    114
9           115                               NonDeterministic
10          116               e_P_SP_Warning  115
11          117               e_P_SP_Warning  115
12          119               e_P_SP_Warning  115
13          120               e_P_SP_Warning  115

================================================================================

WARNING: Tool encountered non-deterministic test case(s);
-------  please review design document.
```

Verification:

Expected Output:

The non-deterministic nature of the draft design specification is expected to be detected by a test case when input parameter, V_Power_Set_Point has a value of "(C_Danger_Power - (0.5 * C_Power_Dead_Band))"

(i.e., when V_Power_Set_Point is equal to 115).

Actual Output from Apollo:

Test case # 9 (i.e., when V_Power_Set_Point is equal to 115) reports non-deterministic nature of the draft design specification, as expected.

General Observations:

The following general observations can be made from the results that are presented in Table 22 :

- the Apollo tool generates a total of thirteen test values for the input parameter; and a set of thirteen test cases; and
- the Apollo tool detects the specification error while executing one of the test cases and reports an appropriate warning message.

5.4.2 Access-Program: PWR$Power_Status (draft)

The preliminary design details of this *access-program* are specified in following vertical condition table [Please refer to Appendix-C, for more design details].

Draft Design Specification:

Table Power_Status

	Result
Condition	V_Power_Status
V_Reactor_Power < (V_Power_Set_Point - (0.5 * C_Power_Dead_Band)) AND (V_Reactor_Power < C_Danger_Power)	e_Sub_Normal
(V_Reactor_Power >= (V_Power_Set_Point - (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power <= (V_Power_Set_Point + (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power < C_Danger_Power)	e_Normal
(V_Reactor_Power > (V_Power_Set_Point + (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power < C_Danger_Power)	e_Above_Normal
(V_Reactor_Power >= C_Danger_Power) AND (V_Reactor_Power < C_Critical_Power)	e_Init_P_SetBack
(V_Reactor_Power > C_Critical_Power)	e_Emergency_Shut_Down

Description of Error introduced:

The error that was introduced into the design specification (clean), and its consequences are as follows:

Error:

A subtle error was introduced by modifying the condition in the last row as given below:

(V_Reactor_Power >= C_Critical_Power)

is changed to

(V_Reactor_Power > C_Critical_Power)

Consequences:

This design specification is an incomplete specification.

In other words, neither of the following two conditions will be satisfied, when one of the input parameters, V_Reactor_Power, has a value of "C_Critical_Power":

Conditon-1:

(V_Reactor_Power >= C_Danger_Power) AND (V_Reactor_Power < C_Critical_Power)

Condition-2:

(V_Reactor_Power > C_Critical_Power)

Results:

The set of test values and a small sample of test cases that are generated by Apollo, for *access-program* PWR$Power_Status, are presented in Table 23 [This section is taken directly from the output file: "D.2 pwr_po.etr (version 0.0)", in Appendix-D. For complete results see Appendix-D.].

Table 23 - Results from Apollo

```
================================================================================
Section 3: Test Values from Boundary Value Analysis
    V_Reactor_Power      : 0, 1, 12, 13, 24, 25, 26, 52, 65, 77, 104, 105,
                           106, 116, 117, 129, 130, 131, 132, 134, 135,
                           136, 145, 159, 160, 161, 192, 205, 249, 250

    V_Power_Set_Point    : 10, 11, 12, 14, 15, 16, 65, 67, 119, 120

================================================================================
================================================================================
Section 5: Test Cases along with Expected Test Results

WARNING: Tool encountered incomplete design specification;
-------  please review design document.


Test Case#  Input           Input              Expected Output
            V_Reactor_Power V_Power_Set_Point  V_Power_Status
            ------------    ------------       ------------
1           0               10                 e_Normal
--          --              --                 --
241         160             10                 Incomplete Spec
242         160             11                 Incomplete Spec
243         160             12                 Incomplete Spec
244         160             14                 Incomplete Spec
245         160             15                 Incomplete Spec
246         160             16                 Incomplete Spec
247         160             65                 Incomplete Spec
248         160             67                 Incomplete Spec
249         160             119                Incomplete Spec
250         160             120                Incomplete Spec
--          --              --                 --
300         250             120                e_Emergency_Shut_Down

================================================================================

WARNING: Tool encountered incomplete design specification;
-------  please review design document.
```

Verification:

Anticipated Output:

The "incomplete" draft design specification is expected to be detected by a test case when one of the input parameters, V_Reactor_Power, has a value of "C_Critical_Power".

(i.e., when V_Reactor_Power is equal to 160).

Actual Output from Apollo:

A set of test cases (from #241 to #250) (i.e., when V_Reactor_Power is equal to 160) report "incomplete design specification" error as expected.

General Observations:

The following general observations can be made from the results that are presented in Table 23:

- the Apollo tool generates a total of ten test values for the input parameter, V_Power_Set_Point; and thirty test values for the input parameter, V_Reactor_Power based on the boundary-value analysis and other specific rules;
- the Apollo tool generates a set of three hundred test cases, using all possible combinations of the test values of the input parameters; and
- the Apollo tool detects the specification error while executing ten test cases and reports an appropriate warning message.

5.4.3 Access-Program: PWR$Display_Status (draft)

The preliminary design details of this *access-program* are specified in following vertical condition table [Please refer to Appendix-C, for more design details].

Draft Design Specification:

Table Display_Status

	Result
Condition	V_Display_Status
V_Reactor_Power < (V_Power_Set_Point + (0.5 * C_Power_Dead_Band)) AND (V_Reactor_Power <=(1.05 * C_Danger_Power))	e_Blue
(V_Reactor_Power >= (V_Power_Set_Point - (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power <= (V_Power_Set_Point + (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power < (1.05 * C_Danger_Power))	e_Green
(V_Reactor_Power > (V_Power_Set_Point + (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power < (1.05 * C_Danger_Power))	e_Amber
(V_Reactor_Power >= (1.05 * C_Danger_Power)) AND (V_Reactor_Power < (1.05 * C_Critical_Power))	e_Red
(V_Reactor_Power >= (1.05 * C_Critical_Power))	e_Flashing_Red

Description of Error introduced:

The typographical error that was introduced into the design specification (clean), and its consequences are as follows:

Error:

A subtle error was introduced by modifying the condition in the first row as given below:

V_Reactor_Power < (V_Power_Set_Point - (0.5 * C_Power_Dead_Band)) AND (V_Reactor_Power <=(1.05 * C_Danger_Power))

is changed to

V_Reactor_Power < (V_Power_Set_Point + (0.5 * C_Power_Dead_Band)) AND (V_Reactor_Power <= (1.05 * C_Danger_Power))

Consequences:

This design specification is non-deterministic specification.

Results:

The set of test values and a small sample of test cases that are generated by Apollo, for *access-program* PWR$Display_Status, are presented in Table 24 [This section is taken directly from the output file: "D.3 pwr_di.etr (version 0.0)", in Appendix-D. For complete results see Appendix-D.].

Table 24 - Results from Apollo

```
================================================================================
Section 3: Test Values from Boundary Value Analysis
    V_Reactor_Power      : 0, 1, 12, 13, 24, 25, 26, 65, 68, 80, 104, 105,
                           106, 120, 134, 135, 136, 137, 138, 152, 167,
                           168, 169, 193, 209, 249, 250

    V_Power_Set_Point    : 10, 11, 12, 14, 15, 16, 64, 65, 67, 119, 120
================================================================================


================================================================================
Section 5: Test Cases along with Expected Test Results

WARNING: Tool encountered non-deterministic test case(s);
-------  please review design document.


Test Case#  Input            Input              Expected Output
            V_Reactor_Power  V_Power_Set_Point  V_Display_Status
            ------------     ------------       ------------
--          --               --                 --
132         105              120                NonDeterministic
--          --               --                 --
143         106              120                NonDeterministic
--          --               --                 --
154         120              120                NonDeterministic
--          --               --                 --
165         134              120                NonDeterministic
--          --               --                 --
297         250              120                e_Flashing_Red

================================================================================

WARNING: Tool encountered non-deterministic test case(s);
-------  please review design document.
```

Verification:

Anticipated Output:

The non-deterministic nature of draft design specification is expected to be detected by a test case when one of the input parameters, V_Reactor_Power, has a value between (V_Power_Set_Point - (0.5 * C_Power_Dead_Band)) and (V_Power_Set_Point + (0.5 * C_Power_Dead_Band)).

(i.e., a set of test cases will detect this error. For example, when V_Reactor_Power is greater than or equal to105 and less than 135 (when V_Power_Set_Point is 120) will detect this error.

Actual Output from Apollo:

The following sub-set of test cases: #132, #143, #154, and #165 have the following input values:

V_Power_Set_Point: 120

V_Reactor_Power: 105, 106, 120, and 134

All the above test cases report "non-deterministic specification" error as expected.

[In all, fifty-two test cases report "non-deterministic specification" error, since a number of input combinations can detect this error].

General Observations:

The following general observations can be made from the results that are presented in Table 24 (and the output file: "D.3 pwr_di.etr (version 0.0)", in Appendix-D):

- the Apollo tool generates a total of eleven test values for the input parameter, V_Power_Set_Point; and twenty-seven test values for the input parameter, V_Reactor_Power based on the boundary-value analysis and other specific rules;

- the Apollo tool generates a set of two hundred and ninety seven test cases, using all possible combinations of the test values of the input parameters; and
- the Apollo tool detects the specification error while executing fifty-two test cases and reports an appropriate warning message.

5.4.4 Access-Program: PWR$Alarm_Status (draft)

The preliminary design details of this *access-program* are specified in following vertical condition table [Please refer to Appendix-B, for more design details].

Draft Design Specification:

Table Alarm_Status

	Result
Condition	V_Alarm_Status
(V_Reactor_Power < (1.1 * C_Danger_Power))	e_Off
(V_Reactor_Power >= (1.1 * C_Danger_Power)) AND (V_Reactor_Power <= (1.1 * C_Critical_Power))	e_Intermittent
(V_Reactor_Power > (1.11 * C_Critical_Power))	e_Continuous

Description of Error introduced:

The typographical error that was introduced into the design specification, and its consequences are as follows:

Error:

A subtle error was introduced by modifying the condition in the last row as given below:

(V_Reactor_Power > (1.1 * C_Critical_Power))

is changed to

(V_Reactor_Power > (1.11 * C_Critical_Power))

Consequences:

This design specification is an incomplete specification.

Results:

The set of test values and set of test cases that are generated by Apollo, for *access-program* PWR$ Alarm_Status, are presented in Table 25 [This section is taken directly from the output file: "D.4 pwr_al.etr (version 0.0)", in Appendix-D. For complete results see Appendix-D.].

Table 25 - Results from Apollo

```
=============================================================================
Section 3: Test Values from Boundary Value Analysis
    V_Reactor_Power       : 0, 1, 71, 142, 143, 144, 159, 175, 176, 177, 178,
                            179, 213, 214, 249, 250

=============================================================================


=============================================================================
Section 5: Test Cases along with Expected Test Results

WARNING: Tool encountered incomplete design specification;
-------  please review design document.


Test Case#  Input           Expected Output
            V_Reactor_Power V_Alarm_Status
            ------------    ------------
1           0               e_Off
2           1               e_Off
3           71              e_Off
4           142             e_Off
5           143             e_Intermittent
6           144             e_Intermittent
7           159             e_Intermittent
8           175             e_Intermittent
9           176             e_Intermittent
10          177             Incomplete Spec
11          178             e_Continuous
12          179             e_Continuous
13          213             e_Continuous
14          214             e_Continuous
15          249             e_Continuous
16          250             e_Continuous

=============================================================================

WARNING: Tool encountered incomplete design specification;
-------  please review design document.
```

Verification:

Anticipated Output:

The "incomplete" draft design specification is expected to be detected by a test case when the input parameter, V_Reactor_Power, has a value of between "1.1 * C_Critical_Power" and "1.11 * C_Critical_Power".

(i.e., when V_Reactor_Power is greater than 176 and less than or equal to 177.6).

Actual Output from Apollo:

Test case # 10 (i.e., when V_Reactor_Power is equal to 177) reports "incomplete design specification" error as expected.

General Observations:

The following general observations can be made from the results that are presented in Table 25 :

- the Apollo tool generates a total of sixteen test values for the input parameter; and a set of sixteen test cases; and
- the Apollo tool detects the specification error while executing one of the test cases and reports an appropriate warning message.

5.4.5 General Discussion

From the results presented in these four output files it can be observed that:

- The Apollo tool detects all the typical/subtle design specification errors that were introduced and reports an appropriate/meaningful message;
- the user can easily locate the error and correct the design specification by examining the "execution path" for a given failed test case in conjunction with the error message generated by Apollo.

6. IMPORATNT ADVANTAGES AND LIMITATIONS

6.1 Important Advantages

The advantages and benefits of the partial automation of the "design-specification-oriented unit testing" are briefly discussed in this section. It should be noted that these advantages are relevant for safety-critical applications, because for other applications, tabular design specification is not being used in software industry.

6.1.1 "Design-vs-Implementation" Consistency

It can be observed that any functional mismatch between the "implementation" and the "design specification" will be detected in the form of failed test cases during unit testing phase by the current approach. Thus, the current approach will be useful in demonstrating that the "implementation" matches the "design specification" at the unit level. However, one should not underestimate the importance and necessity of integration testing and subsystem level testing.

6.1.2 Significant Reduction in Cost of Unit Testing

As of now, generation of test cases along with the anticipated test outcome during unit testing phase is mostly manual, tedious and error-prone. Automation of this manual process is expected to reduce the over-all cost of unit testing. Apollo illustrates that it is feasible to automatically generate a set of test cases along with the anticipated test outcome from design specification. The failed test cases, if any, will help to pin-point the mismatch between the design specification and the implementation.

The manual generation of test cases account for about 50 percent of the total time spent during unit testing. The use of Apollo like tool is expected to result in a saving of 30 to 40% of this time.

6.1.3 L-Bye Algorithm: Design Specification Errors

In general, most tools attempt to identify design specification errors such as incomplete, non-deterministic or ambiguous specification based on static analysis. In contrast, the current approach attempts to verify the accuracy of design specification using a dynamic testing strategy. The following typical inconsistencies, errors, and anomalies can be detected using the L-Bye algorithm used in the current approach:

- incomplete specification;
- non-deterministic specification;
- potential "divide-by-zero" error conditions during execution;
- potential non-executable statements in design specification; or
- potential unreachable and dead regions in design specification.

6.1.4 L-Bye Algorithm: Potential Problems Associated with Target Environment

For safety-critical software, dynamic test results are only valid in the target environment. However, testing on target environment is quite slow and time-consuming. In addition, target environment facilities (for testing purposes) are scarce. As a result, test-case input-files and test-drivers are prepared and debugged in a software development environment during the initial stage, whereas the target environment is used only during the final stage. The current approach can complement such testing efforts since the following typical inconsistencies, errors, and anomalies can be detected using the L-Bye algorithm used the current approach:

- potential over-flow error conditions during execution (on target environment);
- potential under-flow error conditions during execution (on target environment); and
- potential "divide-by-zero" error conditions during execution (on target environment)

6.1.5 Regression Testing and Software Maintenance

Industrial experience indicates that software constantly changes, over a period of time, for a number of reasons (e.g., corrective, perfective, adaptive, etc.[Pressman, 1992]). The impact of any software change that is associated with a change in the design specification can be easily tested and verified at the unit level, using the current approach. In turn, it will improve the regression testing.

6.1.6 Design Documentation

The current industrial experience indicates that most documentation associated with software development becomes out-of-date very easily, due to number of reasons: frequent releases of software; time and cost associated with updating documentation; and it is a lot easier to modify the code when compared to updating of documentation (in order to be consistent with the code). However, it can be observed that a number of test cases will fail whenever a unit testing (based on the current approach) is repeated, if there is a mismatch between design specification and implementation (since the set of test cases along with anticipated test outcome are derived from "design specification" rather than implementation). As a result, the software development group is forced to keep the design documentation up to date.

Peters [1995] had also arrived at a similar conclusion and reported that "generation of a test oracle from a design document" enforces consistency between design and code; and enhances the value and usefulness of design documentation.

6.2 Limitations

Important limitations of the current approach are presented in this section.

6.2.1 Valid Only For Safety-Critical Applications

The approach discussed in the thesis is valid only for safety-critical applications.

Resolution: The approach discussed in this thesis requires a very detailed design specification. Such detailed specifications are common in the development of software for safety-critical applications. It is not practical to expect such a detailed design specification for general software applications.

6.2.2 Too Many Test Cases

Apollo may generate thousands of test cases if there are three or more inputs for a given *access-program* because all possible combinations of test values are used while generating a set of test cases.

Resolution: A suitable test case generation algorithm needs to be developed and implemented to avoid the explosion of the number of test cases.

Automation of Step(7) and Step(8) in Table 1 may reduce the impact of this limitation significantly.

6.2.3 Test Coverage Analysis

Apollo does not report any information regarding the test coverage and the set of test cases.

Resolution: In this thesis, test coverage analysis is **not** addressed. A suitable test coverage analysis algorithm can be developed and implemented to provide information about test coverage analysis. This additional feature can be added to Apollo and its modular design will be of use in this case.

6.2.4 Detection of Semantic Errors

Apollo detects only certain types of semantic errors. It may fail to detect certain other types of semantic errors (e.g., errors associated with basic design decisions).

6.2.5 Complex Functional Requirements

Apollo may fail to meet some of its complex functional requirements in an industrial application. For example, it may generate either more or less number of test values than expected from boundary value analysis. In that case, testing at different stages is expected to detect the errors that are missed at the unit testing stage.

6.2.6 Validation

The current version of Apollo was not validated since it is only a prototype CASE tool.

Resolution: In general, the validation activity of CASE tools is carried out by the user

champions (who are independent of CASE tool software development group). Both pros and cons of a given CASE tool need to be evaluated for the specific industrial application.

7. CONCLUSIONS AND FUTURE WORK

Systematic testing is important for all software systems and it becomes more important for safety-critical systems. Effective testing of software using trust-worthy CASE tools is one of the several strategies, which can minimise the number of errors in given software and keep the testing costs under control.

Testing a large software system involves several stages: unit testing, sub-system testing, integration testing, acceptance testing, and regression testing. The scope of this thesis is restricted only to unit testing. Further, we require as input a very detailed design specification written in a tabular notation. Such detailed specifications are commonly found in the development of software for safety-critical applications. One such application area is in Canadian nuclear industry. Hypothetical examples are drawn from this application throughout this thesis. A case study has been used to justify the "proof of concept" of the proposed software-testing tool named "Apollo".

7.1 Conclusion

The major contribution of this thesis is the conception, design, development and use of the testing tool Apollo. It is based on the well-known tabular method of specification. In this method, specification can be made detailed enough to facilitate their execution. This thesis is an application of an existing technology (tabular method) to a new problem. The analysis of results from the case study has demonstrated the "proof-of-concept" of the automation of the generation of a set of test cases and the anticipated test outcome for each test case.

The design of Apollo tool discussed in this thesis consists of the following three major steps:

Step 1: The input design specification document is parsed and stored in an intermediate form.

Step 2: Test values based on the boundary value analysis are generated.

Step 3: The parsed design specification is executed in order to generate the anticipated test outcome for each test case.

This thesis serves as one humble step towards increased automation of effective testing strategy to minimise the number of errors in safety-critical software.

7.2 Future Work

The current version of Apollo is only a prototype CASE tool. It has been tested with simple programs, since the intention was to demonstrate the "proof-of-concept". It is planned to extend this prototype tool into a full-fledged tool for future safety-critical applications in Canadian nuclear industry.

The Apollo tool could generate a very large set of test cases for complex applications. One problem to solve in future would be to minimize the number of test cases without affecting a specified coverage criterion.

References

Abraham, R.F., "Evaluating Generalized Tabular Expressions in Software Documentation", *CRL Report 346*, McMaster University, CRL, TRIO, NSERC, February 1997.

AECB . *Regulatory Document: Requirements for Shutdown Systems for CANDU Nuclear Power Plants; A Regulatory Policy Statement*, R-8, Effective February 21, 1991. Atomic Energy Control Board. Ottawa, ON, February 1991.

Bowen, J. and Stavridou, V., "Safety-Critical Systems, Formal Methods and Standards", *Software Engineering Journal*, vol. 8, no. 4, July 1993, pp 189-209.

Britton, K. H. and D. L. Parnas., *A-7E Software Module Guide*, NRL Memorandum Report 4702. Computer Science and Systems Branch, Information Technology Division; Naval Research Laboratory, Washington, D. C., 1981.

BS7925-1, *Software Testing: Vocabulary*, British Standard 7925-1: 1998.

BS7925-2, *Software Testing: Software Component Testing*, British Standard 7925-2: 1998.

Daich, G.T., Price, G., Ragland, B. and Dawood, M. *Software Test Technologies Report*, Test Reengineering Tool Evaluation Project, Software Technology Support Center (STSC) , Utah, USA, August 1994.

Heitmeyer, C., Bull, A., Gasarch, C. and Labaw, B. "SCR*: A Toolset for Specifying and Analyzing Requirements," *Proceedings of the Tenth Annual Conference on Computer Assurance (COMPASS '95)*, Gaithersburg, MD, June, 1995.

Heitmeyer, C., Jeffords, R. and Labaw, B. "Automated consistency checking of requirements specifications." *ACM Trans. Software Eng. and Method.* 5(3), 1996.

Heitmeyer, C., Kirby, J. and Labaw, B. "Tools for Formal Specification, Verification, and Validation of Requirements," *Proceedings of 12th Annual Conference on Computer Assurance (COMPASS '97)*, Gaithersburg, MD, June 16-19, 1997.

Heitmeyer, C., Kirby, J., Labaw, B., Archer, M., Bharadwaj, R., "Using Abstraction and Model Checking to Detect Safety Violations in Requirements Specifications," *IEEE Transactions on Software Engineering*, vol. 24, no. 11, November 1998a.

Heitmeyer, C., Kirby, J., Labaw, B. and Bharadwaj, R. "SCR*: A Toolset for Specifying and Analyzing Software Requirements," *Proc. Computer-Aided Verification, 10th Ann. Conf. (CAV'98)*, Vancouver, Canada, 1998b.

Heitmeyer, C., Kirby, J., and Labaw, B. "Applying the SCR Requirements Method to a Weapons Control Panel: An Experience Report." *Formal Methods in Software Practice '98*, Clearwater Beach, FL, March 4-5, 1998c.

Heitmeyer, C. "Using the SCR* Toolset to Specify Software Requirements." *Proceedings, Second IEEE Workshop on Industrial Strength Formal Specification Techniques (WIFT'98)*, Boca Raton, FL, Oct. 19, 1998.

Hoover, D. N., and Chen, Zewei. *TBell: A Mathematical Tool for Analyzing Decision Tables*. NASA Contractor Report 195027, November 1994.

Hoover, D.N. *Three Extensions of Decision Table Syntax and Semantics*. Technical Report, Odyssey Research Associates, Inc., Ithaca NY 14850-1326, March 1995.

Hoover, D. N., and Chen, Zewei. "Tablewise: A decision table tool." *Proceedings of the Tenth Annual Conference on Computer Assurance (COMPASS '95)*, Gaithersburg, MD, June, 1995..

Hoover, D. N., Guaspari, D. and Humenn, P. *Applications of Formal Methods to Specification and Safety of Avionics Software*. NASA Contractor Report 4723, April 1996.

IEC880-1986. *Software for Computers in the Safety Systems of Nuclear Power Stations*, International Electrotechnical Commission Standard, 1986.

IEEE/ANSI, *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Standard 610.12-1990.

Information Processing Limited (IPL): *An Introduction to Safety-critical Systems: Software Testing White Papers*, IPL, Bath, UK, August 1996.

Janicki, R., "On a Formal Semantics of Tabular Expressions" *CRL Report 355*, McMaster University, CRL, NSERC, TRIO, October 1997.

Janicki, R., Parnas, D.L., Zucker J.I., "Tabular Representations in Relational Documents", *CRL Report 313*, McMaster University, CRL, TRIO, November 1995.

Jeffords, R. and Heitmeyer, C. "Automatic Generation of State Invariants from Requirements Specifications," *6th International Symposium on the Foundations of Software Engineering (FSE-6)*, Orlando FL, Nov. 3-5, 1998.

Joannou, P., J. Harauz, M. Viola, R. Cirjanic, D. Chan, R. Wittall and D. Tremaine., *Standard for Software Engineering of Safety-critical Software*, CANDU

Computer System Engineering Centre of Excellence, Ontario Hydro and AECL. CE-1001-STD Rev. 1, 1995.

Kahane, H., *Logic and Philosophy, A Modern Introduction, Sixth Edition.* Wadsworth Publishing Co. Delmont CA. ISDN 0-534-12330-9, 1990.

Knight, J. and Littlewood, B. "Critical Task of Writing Dependable Software", *IEEE Software*, vol. 11, no. 1, January 1994, pp 16-20.

Kung, D.C., Hsia, P. and Gao J. *Testing Object-Oriented Software.* IEEE Computer Society, 1998

Leveson, N.G. *Safeware: System Safety and Computers*, Addison-Wesley, New York, 1995.

Li, ChunMing, "Software Reliability Estimation Tool", *CRL Report 337*, McMaster University, CRL (Communications Research Laboratory), TRIO (Telecommunications Research Institute of Ontario), December 1996.

Li, W., 'Table Construction Tool", *CRL Report 330*, McMaster University, CRL, TRIO, July 1996.

Matias, E. *Software Engineering, Standards and Methods (SESM) Tools: Overview,* COG Report, COG-98-013-I (R0), Atomic Energy of Canada Limited, Ontario, Canada, March 1998.

M^{c}Dougall, J., Moum, G., Viola, M., "Tabular Representation of Mathematical Functions for the Specification and Verification of Safety-critical Software", *SAFECOMP'94 Proceedings of the 13th International Conference on Computer Safety, Reliability and Security*, Anaheim, California, USA, 24-26 October, 1994.

Morell, L.J. and Deimel, L.E. *Unit Analysis and Testing.* Technical Report SEI-CM-9-2.0, Software Engineering Institute, Carnegie Mellon University, Pennsylvania, June 1992.

MOD 00-55. *The Procurement of Safety-critical Software in Defence Equipment.* Defence Standard 00-55, Ministry of Defence, Great Britain, April 1991.

MOD 00-56. *Hazard Analysis and Safety Classification of the Computer and Programmable Electronic System Elements of Defence Equipment. Ministry of Defence.* Defence Standard 00-56, Ministry of Defence, Great Britain, April 1991.

Myers, G.J. *The Art of Software Testing*, John Wiely & Sons, New York, 1979.

NASA, *Software Assurance Standard.* NASA Office of Safety and Mission Assuamce, November 1992.

NASA *Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems: Volume II: A Practitioner's Companion.* NASA Office of Safety and Mission Assuamce, May 1997.

Owre, S., N. Shankar, and J. M. Rushby., *User Guide for the PVS Specification and Verification System (Beta Release).* Computer Science Laboratory, SRI International, Menlo Park, CA 94025, March 1993.

Parnas, D.L., "Tabular Representation of Relations", *Technical Report 260*, McMaster University, CRL, October 1992.

Parnas, D.L., Madey, J., IglewskI, M., "Precise Documentation of Well-Structured Programs", *IEEE Transactions on Software Engineering*, Vol. 20, No.12, December 1994, pp. 948-976.

Peters, D.K., *Generating a Test Oracle from Program Documentation*, M.Eng. Thesis, Department of Electrical and Computer Engineering, McMaster University, Hamilton, ON (April, 1995). 97 pgs. Also printed as *CRL Report 302*, McMaster University, CRL, TRIO, April 1995.

Peters, D.K., Parnas, D.L., "Generating a Test Oracle from Program Documentation--work in progress", *Proceeding of the 1994 International Symposium on Software Testing and Analysis (ISSTA)*, August, 1994, pp. 58-65.

Place, P.R.H. and Kang, K.C. *Safety-Critical Software: Status Report and Annotated Bibliography*, Technical Report CMU/SEI-92-TR-5, Software Engineering Institute, Carnegie Mellon University, Pennsylvania, June 1993.

Pressman, R.S. *Software Engineering A Practitioner's Approach.* 3rd Ed. McGraw-Hill, Inc. New York, 1992.

Rastogi, P., "Specialization: An Approach to Simplifying Table in Software Documentation", *CRL Report 360*, McMaster University, CRL, TRIO, March 1998.

RTCA/FAA DO-178B. *Software Considerations in Airborne Systems and Equipment Certification*, do-178a edition, Radio Technical Commission for Aeronautics/ FAA Standard, Washington, D.C., 1985.

Shen H., "Implementation of Table Inversion Algorithms", *CRL Report 315*, McMaster University, NSERC, CRL, TRIO, December 1995.

Shen H., Zucker J.I., Parnas, D.L., "Table Transformation Tools: Why and How", *Proceedings of the Eleventh Annual Conference on Computer Assurance (COMPASS '96)*, published by IEEE and NIST, Gaithersburg, MD., June 1996, pp. 3-11. Also published as CRL Report 328, McMaster University, CRL, TRIO, May 1996.

Storey, N. *Safety-Critical Computer Systems*, Addison-Wesley, New York, 1996.

Thatipamala, R., "Function Table Processing in SESM CASE Tools", *Table Tools Workshop* December 11-12, 1996, McMaster University, Hamilton, ON, Canada, December 1996.

Voas, J.M., Payne, J.E., and Miller, K.W. *Automating Test Case Generation for Coverages Required by FAA Standard DO-178B*, Reliable Software Technologies (RST) Corporation, Reston, VA., 1993.

Yamaura, T. How to Design Practical Test Cases, IEEE Software, vol. 15, no. 6, Nov/Dec 1998, pp 30-36.

Appendix A - Sample Design Specification Document (Clean)

Design Specification Document
(Clean)

Revision: 1.0

Issue Date: 98/10/20

Disclaimer

This Sample Design Specification is purely fictitious.
It is used ONLY to test Apollo and to illustrate its capabilities.
The logic in the *access-programs* does NOT reflect the current practice in Industry.

INTRODUCTION

Purpose

Organization

DESIGN OVERVIEW

DEVIATIONS FROM SDD PROCEDURE

NOTATION

MODULE GUIDE

Anticipated Changes

ITEM	ANTICIPATED CHANGE	REF.

Module Hierarchy Diagram

Module Responsibilities and Secrets

ALTERNATIVE VIEWS OF THE SOFTWARE DESIGN

Processing Unit Diagrams

Call Hierarchy Diagrams

DETAILED DESIGN

Global Definitions

Include-file: file_1.inc

	Name	Value	Type
Constants:	C_Critical_Power	160	INTEGER
	C_Power_Dead_Band	30	INTEGER
	C_Danger_Power	130	INTEGER
	C_P_Lower_Range	0	INTEGER
	C_P_Higher_Range	250	INTEGER
	C_Lowest_P_SP	10	INTEGER
	C_Highest_P_SP	120	INTEGER

	Name	Definition
Types:	T_Power_Set_Point	C_Lowest_P_SP TO C_Highest_P_SP
	T_Reactor_Power	C_P_Lower_Range TO C_P_Higher_Range
	T_Power_Status	{e_Sub_Normal, e_Normal, e_Above_Normal,e_Init_P_SetBack, e_Emergency_Shut_Down}
	T_P_Display_Status	{e_Blue, e_Green, e_Amber, e_Red, e_Flashing_Red }
	T_P_Alarm_Status	{e_Off, e_Intermittent, e_Continuous }
	T_Check_Power_SP	{e_P_SP_Warning, e_Valid_P_SP}

Leaf Modules

MODULE PWR

	Name	Definition
Types:	(None)	

Natural Language Description:
Some of the important responsibilities of the POWER module are:

- *check the set-point value associated with the reactor power, and give feedback to the operator;*
- *determine the status of the reactor by comparing the measured power with the set-point and announce the message to the operator;*
- *determine the "indicator-status of the reactor on the control panel" by comparing the measured power with the set-point and turn-on the appropriate light (green, blue, red, flashing-red etc.); and*
- *determine the "alarm-status of the reactor on the control panel" by comparing the measured power with the set-point and turn-on the alarm as and when required.*

The following four access-programs fulfill the specific responsibilities of the POWER module:

- *PWR$Check_Set_Point*
- *PWR$Power_Status*
- *PWR$Display_Status*
- *PWR$Alarm_Status*

Access Programs:

PWR$Check_Set_Point
V_Power_Set_Point: T_Power_Set_Point - in/out
V_Check_Result: T_Check_Power_SP - out

PWR$Power_Status
V_Reactor_Power: T_Reactor_Power - in
V_Power_Set_Point: T_Power_Set_Point - in
V_Power_Status: T_Power_Status - out

PWR$Display_Status
V_Reactor_Power: T_Reactor_Power - in
V_Power_Set_Point: T_Power_Set_Point - in
V_Display_Status: T_P_Display_Status - out

PWR$Alarm_Status
V_Reactor_Power: T_Reactor_Power - in
V_Alarm_Status: T_P_Alarm_Status - out

MODULE PWR Internal Declarations

	Name	Type
State Data:	(None)	

ACCESS PROGRAM PWRSCheck_Set_Point
V_Power_Set_Point: T_Power_Set_Point - in/out
V_Check_Result: T_Check_Power_SP - out

Natural Language Description:

High-Level Specification:

- *check the value of the power-set-point using the valid range of the expected-power-output from the reactor and the dead-band associated with the power-set-point;*
- *if the value of the power-set-point is valid (i.e., with in the valid range of power-set-point), then do not change the value; and*
- *if the value of the value of the power-set-point is invalid (i.e., outside of the valid range of power-set-point), then give an appropriate warning message to the operator, and assign a valid value to the power-set-point.*

Detailed Specification:

The value of the output variable, V_Power_Set_point, is decided based on the following logic:

- *if the value is less than the lower-valid-range (i.e. valid_lower_range_of_power plus half-of-the dead-band) then assign Ceiling(valid_lower_range_of_power plus half-of-the dead-band);*
- *if the value is greater than the upper-valid-range (i.e. valid_upper_range_of_power minus half-of-the dead-band) then assign Floor(valid_lower_range_of_power minus half-of-the dead-band);*
- *if the value is with in the valid range then do not change the value.*

The state of the output variable, V_Check_Result, is decided based on the following logic:

- *if the value of the power-set-point is with in the valid range then give a valid-set-point message.*
- *if the value of the power-set-point is out-of- range/invalid then give a warning message.*

	Name	Ext_value	Type	Origin
Updates:	V_Power_Set_Point	-	T_Power_Set_Point	Param

	Name	Ext_value	Type	Origin
Outputs:	V_Check_Result	-	T_Check_Power_SP	Param

Table Check_Set_Point

	V_Power_Set_Point < (C_P_Lower_Range + (0.5 * C_Power_Dead_Band))	(V_Power_Set_Point >= (C_P_Lower_Range + (0.5 * C_Power_Dead_Band))) AND (V_Power_Set_Point <= (C_Danger_Power - (0.5 * C_Power_Dead_Band)))	V_Power_Set_Point > (C_Danger_Power - (0.5 * C_Power_Dead_Band))
V_Power_Set_Point	CEILING(C_P_Lower_Range + (0.5 * C_Power_Dead_Band))	V_Power_Set_Point	FLOOR(C_Danger_Power - (0.5 * C_Power_Dead_Band))
V_Check_Result	e_P_SP_Warning	e_Valid_P_SP	e_P_SP_Warning

ACCESS PROGRAM PWRSPower_Status
V_Reactor_Power: T_Reactor_Power - in
V_Power_Set_Point: T_Power_Set_Point - in
V_Power_Status: T_Power_Status - out

Natural Language Description:

High-Level Specification:

- *determine the status of the reactor by comparing the measured power with the set-point*

Detailed Specification:

The status of the output variable, V_Power_Status, is decided based on the following logic:

- *if the value is less than the desired range (i.e. power-set-point minus half-of-the dead-band) and outside the danger zone then assign e_Sub_Normal;*
- *if the value is with in the desired range and outside the danger zone then assign e_Normal;*
- *if the value is more than the desired range (i.e. power-set-point plus half-of-the dead-band) and outside danger zone then assign e_Above_Normal;*
- *if the value inside danger zone but outside critical zone then assign e_Init_P_SetBack (i.e., initialize the process of decreasing the Power);*
- *if the value inside critical zone then assign e_Emergency_Shut_Down (i.e., initialize the process of emergency shut-down of the reactor).*

	Name	Ext_value	Type	Origin
Inputs:	V_Reactor_Power	-	T_Reactor_Power	Param
	V_Power_Set_Point	-	T_Power_Set_Point	Param

	Name	Ext_value	Type	Origin
Outputs:	V_Power_Status	-	T_Power_Status	Param

Table Power_Status

	Result
Condition	V_Power_Status
V_Reactor_Power < (V_Power_Set_Point - (0.5 * C_Power_Dead_Band)) AND (V_Reactor_Power < C_Danger_Power)	e_Sub_Normal
(V_Reactor_Power >= (V_Power_Set_Point - (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power <= (V_Power_Set_Point + (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power < C_Danger_Power)	e_Normal
(V_Reactor_Power > (V_Power_Set_Point + (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power < C_Danger_Power)	e_Above_Normal
(V_Reactor_Power >= C_Danger_Power) AND (V_Reactor_Power < C_Critical_Power)	e_Init_P_SetBack
(V_Reactor_Power >= C_Critical_Power)	e_Emergency_Shut_Down

ACCESS PROGRAM PWRSDisplay_Status
V_Reactor_Power: T_Reactor_Power - in
V_Power_Set_Point: T_Power_Set_Point - in
V_Display_Status: T_P_Display_Status - out

Natural Language Description:

High-Level Specification:

- *determine the visual display status of the reactor (i.e., blue/green/amber etc.) on the control panel by comparing the measured power with the set-point*

Detailed Specification:

The status of the output variable, V_Display_Status, is decided based on the following logic:

- *if the value is less than the desired range (i.e. power-set-point minus half-of-the dead-band) and outside the danger zone then assign e_Blue;*
- *if the value is with in the desired range and outside the danger zone then assign e_Green;*
- *if the value is more than the desired range (i.e. power-set-point plus half-of-the dead-band) and outside danger zone then assign e_Amber;*
- *if the value inside danger zone but outside critical zone then assign e_Red;*
- *if the value inside critical zone then assign e_Flashing_Red.*

	Name	Ext_value	Type	Origin
Inputs:	V_Reactor_Power	-	T_Reactor_Power	Param
	V_Power_Set_Point	-	T_Power_Set_Point	Param

	Name	Ext_value	Type	Origin
Outputs:	V_Display_Status	-	T_P_Display_Status	Param

Table display_Status

	Result
Condition	V_Display_Status
V_Reactor_Power < (V_Power_Set_Point - (0.5 * C_Power_Dead_Band)) AND (V_Reactor_Power < (1.05 * C_Danger_Power))	e_Blue
(V_Reactor_Power >= (V_Power_Set_Point - (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power <= (V_Power_Set_Point + (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power < (1.05 * C_Danger_Power))	e_Green
(V_Reactor_Power > (V_Power_Set_Point + (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power < (1.05 * C_Danger_Power))	e_Amber
(V_Reactor_Power >= (1.05 * C_Danger_Power)) AND (V_Reactor_Power < (1.05 * C_Critical_Power))	e_Red
(V_Reactor_Power >= (1.05 * C_Critical_Power))	e_Flashing_Red

ACCESS PROGRAM PWRSAlarm_Status
V_Reactor_Power: T_Reactor_Power - in
V_Alarm_Status: T_P_Alarm_Status - out

Natural Language Description:

High-Level Specification:

- *determine the audible alarm status of the reactor (i.e., off/intermittent/continuous) on the control panel by comparing the measured power with the set-point*

Detailed Specification:

The status of the output variable, V_Alarm_Status, is decided based on the following logic:

- *if the value is outside the danger zone the assign e_Off (i.e. do not turn-on the alarm);*
- *if the value inside danger zone but outside critical zone then assign e_Intermittent (i.e., turn-on intermittent audible alarm);*
- *if the value inside critical zone then assign e_Continuous (i.e., turn-on continuous audible alarm);.*

	Name	Ext_value	Type	Origin
Inputs:	V_Reactor_Power	-	T_Reactor_Power	Param

	Name	Ext_value	Type	Origin
Outputs:	V_Alarm_Status	-	T_P_Alarm_Status	Param

Table Alarm_Status

	Result
Condition	V_Alarm_Status
(V_Reactor_Power < (1.1 * C_Danger_Power))	e_Off
(V_Reactor_Power >= (1.1 * C_Danger_Power)) AND (V_Reactor_Power < (1.1 * C_Critical_Power))	e_Intermittent
(V_Reactor_Power >= (1.1 * C_Critical_Power))	e_Continuous

Appendix B - Results from Apollo

B.1 File pwr_ch.etr(V1.0)

```
================================================
Section 1: Configuration Information
Access Program:      PWRSCheck_Set_Point
Design Document Rev:    1.0
Design Document Date:   98/10/20
AGTR Tool Version:      2.1 Exp
Run Date:            Thu Oct 22 18:14:32 1998

================================================
================================================================================
Section 2.1: Inputs Table

Name              Type              Low  High Delta
----              ----              ---  ---- -----
V_Power_Set_Point T_Power_Set_Point 10   120  1


================================================================================

================================================================================
Section 2.2: Outputs Table

Name              Type              Low  High
----              ----              ---  ----
V_Check_Result    T_Check_Power_SP
V_Power_Set_Point T_Power_Set_Point 10   120


--------------------------------------------------------------------------------
Section 2.3: Enumeration(s)
---------------------------
T_Check_Power_SP = (e_P_SP_Warning,e_Valid_P_SP)

================================================================================

================================================================================
Section 3: Test Values from Boundary Value Analysis
    V_Power_Set_Point    : 10, 11, 12, 14, 15, 16, 65, 114, 115, 116, 117,
                           119, 120

================================================================================

============================================================
 Section 4: Summary of each Condition along with Associated Actions
       ================================================
        Table Check_Set_Point : Condition 1
       ================================================

C : V_Power_Set_Point < (C_P_Lower_Range + (0.5 * C_Power_Dead_Band))
A01: V_Power_Set_Point = CEILING(C_P_Lower_Range + (0.5 * C_Power_Dead_Band))
A02: V_Check_Result = e_P_SP_Warning

       ================================================
        Table Check_Set_Point : Condition 2
       ================================================

C : (V_Power_Set_Point >= (C_P_Lower_Range + (0.5 * C_Power_Dead_Band))) AND (
V_Power_Set_Point <= (C_Danger_Power - (0.5 * C_Power_Dead_Band)))
A01: V_Power_Set_Point = V_Power_Set_Point
A02: V_Check_Result = e_Valid_P_SP

       ================================================
        Table Check_Set_Point : Condition 3
       ================================================

C : V_Power_Set_Point > (C_Danger_Power - (0.5 * C_Power_Dead_Band))
A01: V_Power_Set_Point = FLOOR(C_Danger_Power - (0.5 * C_Power_Dead_Band))
A02: V_Check_Result = e_P_SP_Warning

============================================================
```

Section 5: Test Cases along with Anticipated Test Results

Test Case#	Input V_Power_Set_Point	Anticipated Output V_Check_Result	Anticipated Output V_Power_Set_Point
1	10	e_P_SP_Warning	15
2	11	e_P_SP_Warning	15
3	12	e_P_SP_Warning	15
4	14	e_P_SP_Warning	15
5	15	e_Valid_P_SP	15
6	16	e_Valid_P_SP	16
7	65	e_Valid_P_SP	65
8	114	e_Valid_P_SP	114
9	115	e_Valid_P_SP	115
10	116	e_P_SP_Warning	115
11	117	e_P_SP_Warning	115
12	119	e_P_SP_Warning	115
13	120	e_P_SP_Warning	115

B.2 File pwr_po.etr(V1.0)

```
-----------------------------------------------
Section 1: Configuration Information
Access Program:      PWR$Power_Status
Design Document Rev:    1.0
Design Document Date:   98/10/20
Apollo Version:     2.1 Exp
Run Date:            Thu Oct 22 18:14:34 1998

-----------------------------------------------
-------------------------------------------------------------------------------
Section 2.1: Inputs Table

Name              Type              Low  High Delta
----              ----              ---  ---- -----
V_Reactor_Power   T_Reactor_Power   0    250  1
V_Power_Set_Point T_Power_Set_Point 10   120  1


-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
Section 2.2: Outputs Table

Name           Type           Low  High
----           ----           ---  ----
V_Power_Status T_Power_Status


-------------------------------------------------------------------------------
Section 2.3: Enumeration(s)
--------------------------
T_Power_Status = {e_Sub_Normal,e_Normal,e_Above_Normal,e_Init_P_SetBack,
e_Emergency_Shut_Down}

-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
Section 3: Test Values from Boundary Value Analysis
    V_Reactor_Power      : 0, 1, 12, 13, 24, 25, 26, 52, 65, 77, 104, 105,
                           106, 116, 117, 129, 130, 131, 132, 134, 135,
                           136, 145, 159, 160, 161, 192, 205, 249, 250

    V_Power_Set_Point    : 10, 11, 12, 14, 15, 16, 65, 67, 119, 120

-------------------------------------------------------------------------------

-------------------------------------------------------------
 Section 4: Summary of each Condition along with Associated Actions
      -----------------------------------------------
       Table Power_Status : Condition 1
      -----------------------------------------------

C : V_Reactor_Power < (V_Power_Set_Point - (0.5 * C_Power_Dead_Band ) ) AND (V
_Reactor_Power < C_Danger_Power )
A01: V_Power_Status = e_Sub_Normal

      -----------------------------------------------
       Table Power_Status : Condition 2
      -----------------------------------------------

C : (V_Reactor_Power >= (V_Power_Set_Point - (0.5 * C_Power_Dead_Band ) )) AND
 (V_Reactor_Power <= (V_Power_Set_Point + (0.5 * C_Power_Dead_Band ))) AND (V_
Reactor_Power < C_Danger_Power )
A01: V_Power_Status = e_Normal

      -----------------------------------------------
       Table Power_Status : Condition 3
      -----------------------------------------------

C : (V_Reactor_Power > (V_Power_Set_Point + (0.5 * C_Power_Dead_Band ) )) AND
(V_Reactor_Power < C_Danger_Power )
A01: V_Power_Status = e_Above_Normal
```

```
-------------------------------------------------
 Table Power_Status : Condition 4
-------------------------------------------------

C : (V_Reactor_Power >= C_Danger_Power ) AND (V_Reactor_Power < C_Critical_Pow
er )
A01: V_Power_Status = e_Init_P_SetBack

-------------------------------------------------
 Table Power_Status : Condition 5
-------------------------------------------------

C : (V_Reactor_Power >= C_Critical_Power )
A01: V_Power_Status = e_Emergency_Shut_Down

-------------------------------------------------------------
```

```
-------------------------------------------------------------------------
Section 5: Test Cases along with Anticipated Test Results
```

Test Case#	Input V_Reactor_Power	Input V_Power_Set_Point	Anticipated Output V_Power_Status
1	0	10	e_Normal
2	0	11	e_Normal
3	0	12	e_Normal
4	0	14	e_Normal
5	0	15	e_Normal
6	0	16	e_Sub_Normal
7	0	65	e_Sub_Normal
8	0	67	e_Sub_Normal
9	0	119	e_Sub_Normal
10	0	120	e_Sub_Normal
11	1	10	e_Normal
12	1	11	e_Normal
13	1	12	e_Normal
14	1	14	e_Normal
15	1	15	e_Normal
16	1	16	e_Normal
17	1	65	e_Sub_Normal
18	1	67	e_Sub_Normal
19	1	119	e_Sub_Normal
20	1	120	e_Sub_Normal
21	12	10	e_Normal
22	12	11	e_Normal
23	12	12	e_Normal
24	12	14	e_Normal
25	12	15	e_Normal
26	12	16	e_Normal
27	12	65	e_Sub_Normal
28	12	67	e_Sub_Normal
29	12	119	e_Sub_Normal
30	12	120	e_Sub_Normal
31	13	10	e_Normal
32	13	11	e_Normal
33	13	12	e_Normal
34	13	14	e_Normal
35	13	15	e_Normal
36	13	16	e_Normal
37	13	65	e_Sub_Normal
38	13	67	e_Sub_Normal
39	13	119	e_Sub_Normal
40	13	120	e_Sub_Normal
41	24	10	e_Normal
42	24	11	e_Normal
43	24	12	e_Normal
44	24	14	e_Normal
45	24	15	e_Normal
46	24	16	e_Normal
47	24	65	e_Sub_Normal
48	24	67	e_Sub_Normal
49	24	119	e_Sub_Normal
50	24	120	e_Sub_Normal
51	25	10	e_Normal
52	25	11	e_Normal
53	25	12	e_Normal
54	25	14	e_Normal
55	25	15	e_Normal

56	25	16	e_Normal
57	25	65	e_Sub_Normal
58	25	67	e_Sub_Normal
59	25	119	e_Sub_Normal
60	25	120	e_Sub_Normal
61	26	10	e_Above_Normal
62	26	11	e_Normal
63	26	12	e_Normal
64	26	14	e_Normal
65	26	15	e_Normal
66	26	16	e_Normal
67	26	65	e_Sub_Normal
68	26	67	e_Sub_Normal
69	26	119	e_Sub_Normal
70	26	120	e_Sub_Normal
71	52	10	e_Above_Normal
72	52	11	e_Above_Normal
73	52	12	e_Above_Normal
74	52	14	e_Above_Normal
75	52	15	e_Above_Normal
76	52	16	e_Above_Normal
77	52	65	e_Normal
78	52	67	e_Normal
79	52	119	e_Sub_Normal
80	52	120	e_Sub_Normal
81	65	10	e_Above_Normal
82	65	11	e_Above_Normal
83	65	12	e_Above_Normal
84	65	14	e_Above_Normal
85	65	15	e_Above_Normal
86	65	16	e_Above_Normal
87	65	65	e_Normal
88	65	67	e_Normal
89	65	119	e_Sub_Normal
90	65	120	e_Sub_Normal
91	77	10	e_Above_Normal
92	77	11	e_Above_Normal
93	77	12	e_Above_Normal
94	77	14	e_Above_Normal
95	77	15	e_Above_Normal
96	77	16	e_Above_Normal
97	77	65	e_Normal
98	77	67	e_Normal
99	77	119	e_Sub_Normal
100	77	120	e_Sub_Normal
101	104	10	e_Above_Normal
102	104	11	e_Above_Normal
103	104	12	e_Above_Normal
104	104	14	e_Above_Normal
105	104	15	e_Above_Normal
106	104	16	e_Above_Normal
107	104	65	e_Above_Normal
108	104	67	e_Above_Normal
109	104	119	e_Normal
110	104	120	e_Sub_Normal
111	105	10	e_Above_Normal
112	105	11	e_Above_Normal
113	105	12	e_Above_Normal
114	105	14	e_Above_Normal
115	105	15	e_Above_Normal
116	105	16	e_Above_Normal
117	105	65	e_Above_Normal
118	105	67	e_Above_Normal
119	105	119	e_Normal
120	105	120	e_Normal
121	106	10	e_Above_Normal
122	106	11	e_Above_Normal
123	106	12	e_Above_Normal
124	106	14	e_Above_Normal
125	106	15	e_Above_Normal
126	106	16	e_Above_Normal
127	106	65	e_Above_Normal
128	106	67	e_Above_Normal
129	106	119	e_Normal
130	106	120	e_Normal
131	116	10	e_Above_Normal
132	116	11	e_Above_Normal
133	116	12	e_Above_Normal
134	116	14	e_Above_Normal
135	116	15	e_Above_Normal

136	116	16	e_Above_Normal
137	116	65	e_Above_Normal
138	116	67	e_Above_Normal
139	116	119	e_Normal
140	116	120	e_Normal
141	117	10	e_Above_Normal
142	117	11	e_Above_Normal
143	117	12	e_Above_Normal
144	117	14	e_Above_Normal
145	117	15	e_Above_Normal
146	117	16	e_Above_Normal
147	117	65	e_Above_Normal
148	117	67	e_Above_Normal
149	117	119	e_Normal
150	117	120	e_Normal
151	129	10	e_Above_Normal
152	129	11	e_Above_Normal
153	129	12	e_Above_Normal
154	129	14	e_Above_Normal
155	129	15	e_Above_Normal
156	129	16	e_Above_Normal
157	129	65	e_Above_Normal
158	129	67	e_Above_Normal
159	129	119	e_Normal
160	129	120	e_Normal
161	130	10	e_Init_P_SetBack
162	130	11	e_Init_P_SetBack
163	130	12	e_Init_P_SetBack
164	130	14	e_Init_P_SetBack
165	130	15	e_Init_P_SetBack
166	130	16	e_Init_P_SetBack
167	130	65	e_Init_P_SetBack
168	130	67	e_Init_P_SetBack
169	130	119	e_Init_P_SetBack
170	130	120	e_Init_P_SetBack
171	131	10	e_Init_P_SetBack
172	131	11	e_Init_P_SetBack
173	131	12	e_Init_P_SetBack
174	131	14	e_Init_P_SetBack
175	131	15	e_Init_P_SetBack
176	131	16	e_Init_P_SetBack
177	131	65	e_Init_P_SetBack
178	131	67	e_Init_P_SetBack
179	131	119	e_Init_P_SetBack
180	131	120	e_Init_P_SetBack
181	132	10	e_Init_P_SetBack
182	132	11	e_Init_P_SetBack
183	132	12	e_Init_P_SetBack
184	132	14	e_Init_P_SetBack
185	132	15	e_Init_P_SetBack
186	132	16	e_Init_P_SetBack
187	132	65	e_Init_P_SetBack
188	132	67	e_Init_P_SetBack
189	132	119	e_Init_P_SetBack
190	132	120	e_Init_P_SetBack
191	134	10	e_Init_P_SetBack
192	134	11	e_Init_P_SetBack
193	134	12	e_Init_P_SetBack
194	134	14	e_Init_P_SetBack
195	134	15	e_Init_P_SetBack
196	134	16	e_Init_P_SetBack
197	134	65	e_Init_P_SetBack
198	134	67	e_Init_P_SetBack
199	134	119	e_Init_P_SetBack
200	134	120	e_Init_P_SetBack
201	135	10	e_Init_P_SetBack
202	135	11	e_Init_P_SetBack
203	135	12	e_Init_P_SetBack
204	135	14	e_Init_P_SetBack
205	135	15	e_Init_P_SetBack
206	135	16	e_Init_P_SetBack
207	135	65	e_Init_P_SetBack
208	135	67	e_Init_P_SetBack
209	135	119	e_Init_P_SetBack
210	135	120	e_Init_P_SetBack
211	136	10	e_Init_P_SetBack
212	136	11	e_Init_P_SetBack
213	136	12	e_Init_P_SetBack
214	136	14	e_Init_P_SetBack
215	136	15	e_Init_P_SetBack

216	136	16	e_Init_P_SetBack
217	136	65	e_Init_P_SetBack
218	136	67	e_Init_P_SetBack
219	136	119	e_Init_P_SetBack
220	136	120	e_Init_P_SetBack
221	145	10	e_Init_P_SetBack
222	145	11	e_Init_P_SetBack
223	145	12	e_Init_P_SetBack
224	145	14	e_Init_P_SetBack
225	145	15	e_Init_P_SetBack
226	145	16	e_Init_P_SetBack
227	145	65	e_Init_P_SetBack
228	145	67	e_Init_P_SetBack
229	145	119	e_Init_P_SetBack
230	145	120	e_Init_P_SetBack
231	159	10	e_Init_P_SetBack
232	159	11	e_Init_P_SetBack
233	159	12	e_Init_P_SetBack
234	159	14	e_Init_P_SetBack
235	159	15	e_Init_P_SetBack
236	159	16	e_Init_P_SetBack
237	159	65	e_Init_P_SetBack
238	159	67	e_Init_P_SetBack
239	159	119	e_Init_P_SetBack
240	159	120	e_Init_P_SetBack
241	160	10	e_Emergency_Shut_Down
242	160	11	e_Emergency_Shut_Down
243	160	12	e_Emergency_Shut_Down
244	160	14	e_Emergency_Shut_Down
245	160	15	e_Emergency_Shut_Down
246	160	16	e_Emergency_Shut_Down
247	160	65	e_Emergency_Shut_Down
248	160	67	e_Emergency_Shut_Down
249	160	119	e_Emergency_Shut_Down
250	160	120	e_Emergency_Shut_Down
251	161	10	e_Emergency_Shut_Down
252	161	11	e_Emergency_Shut_Down
253	161	12	e_Emergency_Shut_Down
254	161	14	e_Emergency_Shut_Down
255	161	15	e_Emergency_Shut_Down
256	161	16	e_Emergency_Shut_Down
257	161	65	e_Emergency_Shut_Down
258	161	67	e_Emergency_Shut_Down
259	161	119	e_Emergency_Shut_Down
260	161	120	e_Emergency_Shut_Down
261	192	10	e_Emergency_Shut_Down
262	192	11	e_Emergency_Shut_Down
263	192	12	e_Emergency_Shut_Down
264	192	14	e_Emergency_Shut_Down
265	192	15	e_Emergency_Shut_Down
266	192	16	e_Emergency_Shut_Down
267	192	65	e_Emergency_Shut_Down
268	192	67	e_Emergency_Shut_Down
269	192	119	e_Emergency_Shut_Down
270	192	120	e_Emergency_Shut_Down
271	205	10	e_Emergency_Shut_Down
272	205	11	e_Emergency_Shut_Down
273	205	12	e_Emergency_Shut_Down
274	205	14	e_Emergency_Shut_Down
275	205	15	e_Emergency_Shut_Down
276	205	16	e_Emergency_Shut_Down
277	205	65	e_Emergency_Shut_Down
278	205	67	e_Emergency_Shut_Down
279	205	119	e_Emergency_Shut_Down
280	205	120	e_Emergency_Shut_Down
281	249	10	e_Emergency_Shut_Down
282	249	11	e_Emergency_Shut_Down
283	249	12	e_Emergency_Shut_Down
284	249	14	e_Emergency_Shut_Down
285	249	15	e_Emergency_Shut_Down
286	249	16	e_Emergency_Shut_Down
287	249	65	e_Emergency_Shut_Down
288	249	67	e_Emergency_Shut_Down
289	249	119	e_Emergency_Shut_Down
290	249	120	e_Emergency_Shut_Down
291	250	10	e_Emergency_Shut_Down
292	250	11	e_Emergency_Shut_Down
293	250	12	e_Emergency_Shut_Down
294	250	14	e_Emergency_Shut_Down
295	250	15	e_Emergency_Shut_Down

296	250	16	e_Emergency_Shut_Down
297	250	65	e_Emergency_Shut_Down
298	250	67	e_Emergency_Shut_Down
299	250	119	e_Emergency_Shut_Down
300	250	120	e_Emergency_Shut_Down

B.3 File pwr_di.etr(V1.0)

```
================================================
Section 1: Configuration Information
Access Program:      PWRSDisplay_Status
Design Document Rev:    1.0
Design Document Date:   98/10/20
Apollo Version:     2.1 Exp
Run Date:            Thu Oct 22 18:15:44 1998

================================================


================================================================================
Section 2.1: Inputs Table

Name              Type              Low  High Delta
----              ----              ---  ---- -----
V_Reactor_Power   T_Reactor_Power   0    250  1
V_Power_Set_Point T_Power_Set_Point 10   120  1


================================================================================

================================================================================
Section 2.2: Outputs Table

Name             Type               Low  High
----             ----               ---  ----
V_Display_Status T_P_Display_Status


--------------------------------------------------------------------------------
Section 2.3: Enumeration(s)
--------------------------
T_P_Display_Status = {e_Blue,e_Green,e_Amber,e_Red,e_Flashing_Red}

================================================================================

================================================================================
Section 3: Test Values from Boundary Value Analysis
    V_Reactor_Power      : 0, 1, 12, 13, 24, 25, 26, 52, 65, 68, 80, 104,
                           105, 106, 119, 120, 134, 135, 136, 137, 138,
                           152, 167, 168, 169, 193, 209, 249, 250

    V_Power_Set_Point    : 10, 11, 12, 14, 15, 16, 65, 67, 119, 120

================================================================================

============================================================
 Section 4: Summary of each Condition along with Associated Actions
      ======================================================
       Table Power_Status : Condition 1
      ======================================================

C : V_Reactor_Power < (V_Power_Set_Point - (0.5 * C_Power_Dead_Band ) ) AND (V
_Reactor_Power < (1.05 * C_Danger_Power) )
A01: V_Display_Status = e_Blue

      ======================================================
       Table Power_Status : Condition 2
      ======================================================

C : (V_Reactor_Power >= (V_Power_Set_Point - (0.5 * C_Power_Dead_Band ) )) AND
 (V_Reactor_Power <= (V_Power_Set_Point + (0.5 * C_Power_Dead_Band ))) AND (V_
Reactor_Power < (1.05 * C_Danger_Power) )
A01: V_Display_Status = e_Green

      ======================================================
       Table Power_Status : Condition 3
      ======================================================

C : (V_Reactor_Power > (V_Power_Set_Point + (0.5 * C_Power_Dead_Band ) )) AND
(V_Reactor_Power < (1.05 * C_Danger_Power) )
A01: V_Display_Status = e_Amber
```

```
-------------------------------------------------
 Table Power_Status : Condition 4
-------------------------------------------------

C : (V_Reactor_Power >= (1.05 * C_Danger_Power )) AND (V_Reactor_Power < (1.05
 * C_Critical_Power) )
A01: V_Display_Status = e_Red

-------------------------------------------------
 Table Power_Status : Condition 5
-------------------------------------------------

C : (V_Reactor_Power >= (1.05 * C_Critical_Power) )
A01: V_Display_Status = e_Flashing_Red

-------------------------------------------------------------
```

```
-------------------------------------------------------------------------
Section 5: Test Cases along with Anticipated Test Results
```

Test Case#	Input V_Reactor_Power	Input V_Power_Set_Point	Anticipated Output V_Display_Status
1	0	10	e_Green
2	0	11	e_Green
3	0	12	e_Green
4	0	14	e_Green
5	0	15	e_Green
6	0	16	e_Blue
7	0	65	e_Blue
8	0	67	e_Blue
9	0	119	e_Blue
10	0	120	e_Blue
11	1	10	e_Green
12	1	11	e_Green
13	1	12	e_Green
14	1	14	e_Green
15	1	15	e_Green
16	1	16	e_Green
17	1	65	e_Blue
18	1	67	e_Blue
19	1	119	e_Blue
20	1	120	e_Blue
21	12	10	e_Green
22	12	11	e_Green
23	12	12	e_Green
24	12	14	e_Green
25	12	15	e_Green
26	12	16	e_Green
27	12	65	e_Blue
28	12	67	e_Blue
29	12	119	e_Blue
30	12	120	e_Blue
31	13	10	e_Green
32	13	11	e_Green
33	13	12	e_Green
34	13	14	e_Green
35	13	15	e_Green
36	13	16	e_Green
37	13	65	e_Blue
38	13	67	e_Blue
39	13	119	e_Blue
40	13	120	e_Blue
41	24	10	e_Green
42	24	11	e_Green
43	24	12	e_Green
44	24	14	e_Green
45	24	15	e_Green
46	24	16	e_Green
47	24	65	e_Blue
48	24	67	e_Blue
49	24	119	e_Blue
50	24	120	e_Blue
51	25	10	e_Green
52	25	11	e_Green
53	25	12	e_Green
54	25	14	e_Green
55	25	15	e_Green

56	25	16	e_Green
57	25	65	e_Blue
58	25	67	e_Blue
59	25	119	e_Blue
60	25	120	e_Blue
61	26	10	e_Amber
62	26	11	e_Green
63	26	12	e_Green
64	26	14	e_Green
65	26	15	e_Green
66	26	16	e_Green
67	26	65	e_Blue
68	26	67	e_Blue
69	26	119	e_Blue
70	26	120	e_Blue
71	52	10	e_Amber
72	52	11	e_Amber
73	52	12	e_Amber
74	52	14	e_Amber
75	52	15	e_Amber
76	52	16	e_Amber
77	52	65	e_Green
78	52	67	e_Green
79	52	119	e_Blue
80	52	120	e_Blue
81	65	10	e_Amber
82	65	11	e_Amber
83	65	12	e_Amber
84	65	14	e_Amber
85	65	15	e_Amber
86	65	16	e_Amber
87	65	65	e_Green
88	65	67	e_Green
89	65	119	e_Blue
90	65	120	e_Blue
91	68	10	e_Amber
92	68	11	e_Amber
93	68	12	e_Amber
94	68	14	e_Amber
95	68	15	e_Amber
96	68	16	e_Amber
97	68	65	e_Green
98	68	67	e_Green
99	68	119	e_Blue
100	68	120	e_Blue
101	80	10	e_Amber
102	80	11	e_Amber
103	80	12	e_Amber
104	80	14	e_Amber
105	80	15	e_Amber
106	80	16	e_Amber
107	80	65	e_Green
108	80	67	e_Green
109	80	119	e_Blue
110	80	120	e_Blue
111	104	10	e_Amber
112	104	11	e_Amber
113	104	12	e_Amber
114	104	14	e_Amber
115	104	15	e_Amber
116	104	16	e_Amber
117	104	65	e_Amber
118	104	67	e_Amber
119	104	119	e_Green
120	104	120	e_Blue
121	105	10	e_Amber
122	105	11	e_Amber
123	105	12	e_Amber
124	105	14	e_Amber
125	105	15	e_Amber
126	105	16	e_Amber
127	105	65	e_Amber
128	105	67	e_Amber
129	105	119	e_Green
130	105	120	e_Green
131	106	10	e_Amber
132	106	11	e_Amber
133	106	12	e_Amber
134	106	14	e_Amber
135	106	15	e_Amber

136	106	16	e_Amber
137	106	65	e_Amber
138	106	67	e_Amber
139	106	119	e_Green
140	106	120	e_Green
141	119	10	e_Amber
142	119	11	e_Amber
143	119	12	e_Amber
144	119	14	e_Amber
145	119	15	e_Amber
146	119	16	e_Amber
147	119	65	e_Amber
148	119	67	e_Amber
149	119	119	e_Green
150	119	120	e_Green
151	120	10	e_Amber
152	120	11	e_Amber
153	120	12	e_Amber
154	120	14	e_Amber
155	120	15	e_Amber
156	120	16	e_Amber
157	120	65	e_Amber
158	120	67	e_Amber
159	120	119	e_Green
160	120	120	e_Green
161	134	10	e_Amber
162	134	11	e_Amber
163	134	12	e_Amber
164	134	14	e_Amber
165	134	15	e_Amber
166	134	16	e_Amber
167	134	65	e_Amber
168	134	67	e_Amber
169	134	119	e_Green
170	134	120	e_Green
171	135	10	e_Amber
172	135	11	e_Amber
173	135	12	e_Amber
174	135	14	e_Amber
175	135	15	e_Amber
176	135	16	e_Amber
177	135	65	e_Amber
178	135	67	e_Amber
179	135	119	e_Amber
180	135	120	e_Green
181	136	10	e_Amber
182	136	11	e_Amber
183	136	12	e_Amber
184	136	14	e_Amber
185	136	15	e_Amber
186	136	16	e_Amber
187	136	65	e_Amber
188	136	67	e_Amber
189	136	119	e_Amber
190	136	120	e_Amber
191	137	10	e_Red
192	137	11	e_Red
193	137	12	e_Red
194	137	14	e_Red
195	137	15	e_Red
196	137	16	e_Red
197	137	65	e_Red
198	137	67	e_Red
199	137	119	e_Red
200	137	120	e_Red
201	138	10	e_Red
202	138	11	e_Red
203	138	12	e_Red
204	138	14	e_Red
205	138	15	e_Red
206	138	16	e_Red
207	138	65	e_Red
208	138	67	e_Red
209	138	119	e_Red
210	138	120	e_Red
211	152	10	e_Red
212	152	11	e_Red
213	152	12	e_Red
214	152	14	e_Red
215	152	15	e_Red

216	152	16	e_Red
217	152	65	e_Red
218	152	67	e_Red
219	152	119	e_Red
220	152	120	e_Red
221	167	10	e_Red
222	167	11	e_Red
223	167	12	e_Red
224	167	14	e_Red
225	167	15	e_Red
226	167	16	e_Red
227	167	65	e_Red
228	167	67	e_Red
229	167	119	e_Red
230	167	120	e_Red
231	168	10	e_Flashing_Red
232	168	11	e_Flashing_Red
233	168	12	e_Flashing_Red
234	168	14	e_Flashing_Red
235	168	15	e_Flashing_Red
236	168	16	e_Flashing_Red
237	168	65	e_Flashing_Red
238	168	67	e_Flashing_Red
239	168	119	e_Flashing_Red
240	168	120	e_Flashing_Red
241	169	10	e_Flashing_Red
242	169	11	e_Flashing_Red
243	169	12	e_Flashing_Red
244	169	14	e_Flashing_Red
245	169	15	e_Flashing_Red
246	169	16	e_Flashing_Red
247	169	65	e_Flashing_Red
248	169	67	e_Flashing_Red
249	169	119	e_Flashing_Red
250	169	120	e_Flashing_Red
251	193	10	e_Flashing_Red
252	193	11	e_Flashing_Red
253	193	12	e_Flashing_Red
254	193	14	e_Flashing_Red
255	193	15	e_Flashing_Red
256	193	16	e_Flashing_Red
257	193	65	e_Flashing_Red
258	193	67	e_Flashing_Red
259	193	119	e_Flashing_Red
260	193	120	e_Flashing_Red
261	209	10	e_Flashing_Red
262	209	11	e_Flashing_Red
263	209	12	e_Flashing_Red
264	209	14	e_Flashing_Red
265	209	15	e_Flashing_Red
266	209	16	e_Flashing_Red
267	209	65	e_Flashing_Red
268	209	67	e_Flashing_Red
269	209	119	e_Flashing_Red
270	209	120	e_Flashing_Red
271	249	10	e_Flashing_Red
272	249	11	e_Flashing_Red
273	249	12	e_Flashing_Red
274	249	14	e_Flashing_Red
275	249	15	e_Flashing_Red
276	249	16	e_Flashing_Red
277	249	65	e_Flashing_Red
278	249	67	e_Flashing_Red
279	249	119	e_Flashing_Red
280	249	120	e_Flashing_Red
281	250	10	e_Flashing_Red
282	250	11	e_Flashing_Red
283	250	12	e_Flashing_Red
284	250	14	e_Flashing_Red
285	250	15	e_Flashing_Red
286	250	16	e_Flashing_Red
287	250	65	e_Flashing_Red
288	250	67	e_Flashing_Red
289	250	119	e_Flashing_Red
290	250	120	e_Flashing_Red

B.4 File pwr_al.etr(V1.0)

```
================================================
Section 1: Configuration Information
Access Program:      PWR$Alarm_Status
Design Document Rev:    1.0
Design Document Date:   98/10/20
Apollo Version:     2.1 Exp
Run Date:            Thu Oct 22 18:16:49 1998

================================================


================================================================================
Section 2.1: Inputs Table

Name             Type             Low  High Delta
----             ----             ---  ---- -----
V_Reactor_Power T_Reactor_Power 0    250  1

================================================================================

================================================================================
Section 2.2: Outputs Table

Name            Type             Low  High
----            ----             ---  ----
V_Alarm_Status T_P_Alarm_Status


--------------------------------------------------------------------------------
Section 2.3: Enumeration(s)
---------------------------
T_P_Alarm_Status = {e_Off,e_Intermittent,e_Continuous}

================================================================================

================================================================================
Section 3: Test Values from Boundary Value Analysis
    V_Reactor_Power       : 0, 1, 71, 142, 143, 144, 159, 175, 176, 177, 213,
                            249, 250

================================================================================

============================================================
 Section 4: Summary of each Condition along with Associated Actions
      ================================================
       Table Power_Status : Condition 1
      ================================================

C : (V_Reactor_Power < (1.1 * C_Danger_Power) )
A01: V_Alarm_Status = e_Off

      ================================================
       Table Power_Status : Condition 2
      ================================================

C : (V_Reactor_Power >= (1.1 * C_Danger_Power )) AND (V_Reactor_Power < (1.1 *
 C_Critical_Power) )
A01: V_Alarm_Status = e_Intermittent

      ================================================
       Table Power_Status : Condition 3
      ================================================

C : (V_Reactor_Power >= (1.1 * C_Critical_Power) )
A01: V_Alarm_Status = e_Continuous

============================================================
```

Section 5: Test Cases along with Anticipated Test Results

Test Case#	Input V_Reactor_Power	Anticipated Output V_Alarm_Status
1	0	e_Off
2	1	e_Off
3	71	e_Off
4	142	e_Off
5	143	e_Intermittent
6	144	e_Intermittent
7	159	e_Intermittent
8	175	e_Intermittent
9	176	e_Continuous
10	177	e_Continuous
11	213	e_Continuous
12	249	e_Continuous
13	250	e_Continuous

Appendix C - Sample Design Specification Document (draft)

Design Specification Document
(Draft)

Revision: 0.0

Issue Date: 98/10/20

Disclaimer

This Sample Design Specification is purely fictitious.
It is used ONLY to test Apollo and to illustrate its capabilities.
The logic in the *access-programs* does NOT reflect the current practice in Industry.

INTRODUCTION

Purpose

Organization

DESIGN OVERVIEW

DEVIATIONS FROM SDD PROCEDURE

NOTATION

MODULE GUIDE

Anticipated Changes

ITEM	ANTICIPATED CHANGE	REF.

Module Hierarchy Diagram

Module Responsibilities and Secrets

ALTERNATIVE VIEWS OF THE SOFTWARE DESIGN

Processing Unit Diagrams

Call Hierarchy Diagrams

DETAILED DESIGN

Global Definitions

Include-file: file_1.inc

	Name	Value	Type
Constants:	C_Critical_Power	160	INTEGER
	C_Power_Dead_Band	30	INTEGER
	C_Danger_Power	130	INTEGER
	C_P_Lower_Range	0	INTEGER
	C_P_Higher_Range	250	INTEGER
	C_Lowest_P_SP	10	INTEGER
	C_Highest_P_SP	120	INTEGER

	Name	Definition
Types:	T_Power_Set_Point	C_Lowest_P_SP TO C_Highest_P_SP
	T_Reactor_Power	C_P_Lower_Range TO C_P_Higher_Range
	T_Power_Status	{e_Sub_Normal, e_Normal, e_Above_Normal,e_Init_P_SetBack, e_Emergency_Shut_Down}
	T_P_Display_Status	{e_Blue, e_Green, e_Amber, e_Red, e_Flashing_Red }
	T_P_Alarm_Status	{e_Off, e_Intermittent, e_Continuous }
	T_Check_Power_SP	{e_P_SP_Warning, e_Valid_P_SP}

Leaf Modules

MODULE PWR

Manage the software / hardware interface for analog inputs and outputs.

	Name	Definition
Types:	(None)	

Access Programs:

PWR$Check_Set_Point
V_Power_Set_Point: T_Power_Set_Point - in/out
V_Check_Result: T_Check_Power_SP - out

PWR$Power_Status
V_Reactor_Power: T_Reactor_Power - in
V_Power_Set_Point: T_Power_Set_Point - in
V_Power_Status: T_Power_Status - out

PWR$Display_Status
V_Reactor_Power: T_Reactor_Power - in
V_Power_Set_Point: T_Power_Set_Point - in
V_Display_Status: T_P_Display_Status - out

PWR$Alarm_Status
V_Reactor_Power: T_Reactor_Power - in
V_Alarm_Status: T_P_Alarm_Status - out

MODULE PWR Internal Declarations

	Name	Type
State Data:	(None)	

ACCESS PROGRAM PWRSCheck_Set_Point
V_Power_Set_Point: T_Power_Set_Point - in/out
V_Check_Result: T_Check_Power_SP - out

	Name	Ext_value	Type	Origin
Updates:	V_Power_Set_Point	-	T_Power_Set_Point	Param

	Name	Ext_value	Type	Origin
Outputs:	V_Check_Result	-	T_Check_Power_SP	Param

Table Check_Set_Point

	V_Power_Set_Point < (C_P_Lower_Range + (0.5 * C_Power_Dead_Band))	(V_Power_Set_Point >= (C_P_Lower_Range + (0.5 * C_Power_Dead_Band))) AND (V_Power_Set_Point <= (C_Danger_Power - (0.5 * C_Power_Dead_Band)))	V_Power_Set_Point >= (C_Danger_Power - (0.5 * C_Power_Dead_Band))
V_Power_Set_Point	CEILING(C_P_Lower_Range + (0.5 * C_Power_Dead_Band))	V_Power_Set_Point	FLOOR(C_Danger_Power - (0.5 * C_Power_Dead_Band))
V_Check_Result	e_P_SP_Warning	e_Valid_P_SP	e_P_SP_Warning

Description of Error introduced:
The error that was introduced into the design specification, and its consequences are as follows:

Error:

The condition:

*"V_Power_Set_Point > (C_Danger_Power - (0.5 * C_Power_Dead_Band))"*

is changed to

*"V_Power_Set_Point >= (C_Danger_Power - (0.5 * C_Power_Dead_Band))"*

Consequences:

The specification becomes non-deterministic.

*In other words, the following two conditions will be satisfied, when input parameter, V_Power_Set_Point has a value of "(C_Danger_Power - (0.5 * C_Power_Dead_Band))":*

Conditon-1:

*"V_Power_Set_Point >= (C_Danger_Power - (0.5 * C_Power_Dead_Band))"; and*

Condition-2:

*(V_Power_Set_Point >= (C_P_Lower_Range + (0.5 * C_Power_Dead_Band))) AND (V_Power_Set_Point <= (C_Danger_Power - (0.5 * C_Power_Dead_Band)))*

ACCESS PROGRAM PWRSPower_Status
V_Reactor_Power: T_Reactor_Power - in
V_Power_Set_Point: T_Power_Set_Point - in
V_Power_Status: T_Power_Status - out

	Name	Ext_value	Type	Origin
Inputs:	V_Reactor_Power	-	T_Reactor_Power	Param
	V_Power_Set_Point	-	T_Power_Set_Point	Param

	Name	Ext_value	Type	Origin
Outputs:	V_Power_Status	-	T_Power_Status	Param

Table Power_Status

	Result
Condition	V_Power_Status
V_Reactor_Power < (V_Power_Set_Point - (0.5 * C_Power_Dead_Band)) AND (V_Reactor_Power < C_Danger_Power)	e_Sub_Normal
(V_Reactor_Power >= (V_Power_Set_Point - (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power <= (V_Power_Set_Point + (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power < C_Danger_Power)	e_Normal
(V_Reactor_Power > (V_Power_Set_Point + (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power < C_Danger_Power)	e_Above_Normal
(V_Reactor_Power >= C_Danger_Power) AND (V_Reactor_Power < C_Critical_Power)	e_Init_P_SetBack
(V_Reactor_Power > C_Critical_Power)	e_Emergency_Shut_Down

Description of Error introduced:
The error that was introduced into the design specification, and its consequences are as follows:

Error:

The condition: *(V_Reactor_Power >= C_Critical_Power)*

is changed to: *(V_Reactor_Power > C_Critical_Power)*

Consequences:

The specification becomes incomplete specification.

In other words, neither of the following two conditions will be satisfied, when input parameter, V_Reactor_Power, has a value of "C_Critical_Power"

Conditon-1:

(V_Reactor_Power >= C_Danger_Power) AND (V_Reactor_Power < C_Critical_Power)

and

Condition-2:

(V_Reactor_Power > C_Critical_Power)

ACCESS PROGRAM PWRSDisplay_Status
V_Reactor_Power: T_Reactor_Power - in
V_Power_Set_Point: T_Power_Set_Point - in
V_Display_Status: T_P_Display_Status - out

	Name	Ext_value	Type	Origin
Inputs:	V_Reactor_Power	-	T_Reactor_Power	Param
	V_Power_Set_Point	-	T_Power_Set_Point	Param

	Name	Ext_value	Type	Origin
Outputs:	V_Display_Status	-	T_P_Display_Status	Param

Table Power_Status

	Result
Condition	V_Display_Status
V_Reactor_Power < (V_Power_Set_Point + (0.5 * C_Power_Dead_Band)) AND (V_Reactor_Power <=(1.05 * C_Danger_Power))	e_Blue
(V_Reactor_Power >= (V_Power_Set_Point - (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power <= (V_Power_Set_Point + (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power < (1.05 * C_Danger_Power))	e_Green
(V_Reactor_Power > (V_Power_Set_Point + (0.5 * C_Power_Dead_Band))) AND (V_Reactor_Power < (1.05 * C_Danger_Power))	e_Amber
(V_Reactor_Power >= (1.05 * C_Danger_Power)) AND (V_Reactor_Power < (1.05 * C_Critical_Power))	e_Red
(V_Reactor_Power >= (1.05 * C_Critical_Power))	e_Flashing_Red

Description of Error introduced:
The error(typographical) that was introduced into the design specification, and its consequences are as follows:

Error:

The condition:

*V_Reactor_Power < (V_Power_Set_Point - (0.5 * C_Power_Dead_Band)) AND (V_Reactor_Power <=(1.05 * C_Danger_Power))*

is changed to

*V_Reactor_Power < (V_Power_Set_Point + (0.5 * C_Power_Dead_Band)) AND (V_Reactor_Power <= (1.05 * C_Danger_Power))*

Consequences:

The specification becomes non-deterministic specification.

ACCESS PROGRAM PWR$Alarm_Status
V_Reactor_Power: T_Reactor_Power - in
V_Alarm_Status: T_P_Alarm_Status - out

	Name	Ext_value	Type	Origin
Inputs:	V_Reactor_Power	-	T_Reactor_Power	Param

	Name	Ext_value	Type	Origin
Outputs:	V_Alarm_Status	-	T_P_Alarm_Status	Param

Table Power_Status

	Result
Condition	V_Alarm_Status
(V_Reactor_Power < (1.1 * C_Danger_Power))	e_Off
(V_Reactor_Power >= (1.1 * C_Danger_Power)) AND (V_Reactor_Power <= (1.1 * C_Critical_Power))	e_Intermittent
(V_Reactor_Power > (1.11 * C_Critical_Power))	e_Continuous

Description of Error introduced:

The error(typographical) that was introduced into the design specification, and its consequences are as follows:

Error:

The condition:

*(V_Reactor_Power > (1.1 * C_Critical_Power))*

is changed to

*(V_Reactor_Power > (1.11 * C_Critical_Power))*

Consequences:

The specification becomes incomplete specification.

Appendix D - Results from Apollo

D.1 File pwr_ch.etr(V0.0)

```
================================================
Section 1: Configuration Information
Access Program:      PWRSCheck_Set_Point
Design Document Rev:    0.0
Design Document Date:   98/10/20
Apollo Version:     2.1 Exp
Run Date:            Thu Oct 22 18:00:50 1998

================================================

================================================================================
Section 2.1: Inputs Table

Name               Type               Low  High Delta
----               ----               ---  ---- -----
V_Power_Set_Point T_Power_Set_Point 10   120  1


================================================================================

================================================================================
Section 2.2: Outputs Table

Name               Type               Low  High
----               ----               ---  ----
V_Check_Result     T_Check_Power_SP
V_Power_Set_Point T_Power_Set_Point 10   120


--------------------------------------------------------------------------------
Section 2.3: Enumeration(s)
---------------------------
T_Check_Power_SP = {e_P_SP_Warning,e_Valid_P_SP}

================================================================================

================================================================================
Section 3: Test Values from Boundary Value Analysis
    V_Power_Set_Point     : 10, 11, 12, 14, 15, 16, 65, 114, 115, 116, 117,
                            119, 120

================================================================================

============================================================
 Section 4: Summary of each Condition along with Associated Actions
      ======================================================
       Table Check_Set_Point : Condition 1
      ======================================================

C : V_Power_Set_Point < (C_P_Lower_Range + (0.5 * C_Power_Dead_Band))
A01: V_Power_Set_Point = CEILING(C_P_Lower_Range + (0.5 * C_Power_Dead_Band))
A02: V_Check_Result = e_P_SP_Warning

      ======================================================
       Table Check_Set_Point : Condition 2
      ======================================================

C : (V_Power_Set_Point >= (C_P_Lower_Range + (0.5 * C_Power_Dead_Band))) AND (
V_Power_Set_Point <= (C_Danger_Power - (0.5 * C_Power_Dead_Band)))
A01: V_Power_Set_Point = V_Power_Set_Point
A02: V_Check_Result = e_Valid_P_SP

      ======================================================
       Table Check_Set_Point : Condition 3
      ======================================================

C : V_Power_Set_Point >= (C_Danger_Power - (0.5 * C_Power_Dead_Band))
A01: V_Power_Set_Point = FLOOR(C_Danger_Power - (0.5 * C_Power_Dead_Band))
A02: V_Check_Result = e_P_SP_Warning

============================================================
```

Section 5: Test Cases along with Anticipated Test Results

WARNING: Tool encountered non-deterministic test case(s);
------- please review design document.

Test Case#	Input V_Power_Set_Point	Anticipated Output V_Check_Result	Anticipated Output V_Power_Set_Point
1	10	e_P_SP_Warning	15
2	11	e_P_SP_Warning	15
3	12	e_P_SP_Warning	15
4	14	e_P_SP_Warning	15
5	15	e_Valid_P_SP	15
6	16	e_Valid_P_SP	16
7	65	e_Valid_P_SP	65
8	114	e_Valid_P_SP	114
9	115		NonDeterministic
10	116	e_P_SP_Warning	115
11	117	e_P_SP_Warning	115
12	119	e_P_SP_Warning	115
13	120	e_P_SP_Warning	115

WARNING: Tool encountered non-deterministic test case(s);
------- please review design document.

D.2 File pwr_po.etr(V0.0)

```
================================================
Section 1: Configuration Information
Access Program:      PWR$Power_Status
Design Document Rev:    0.0
Design Document Date:   98/10/20
Apollo Version:     2.1 Exp
Run Date:            Thu Oct 22 18:00:51 1998

================================================


================================================================================
Section 2.1: Inputs Table

Name              Type              Low  High Delta
----              ----              ---  ---- -----
V_Reactor_Power   T_Reactor_Power   0    250  1
V_Power_Set_Point T_Power_Set_Point 10   120  1

================================================================================

================================================================================
Section 2.2: Outputs Table

Name           Type           Low  High
----           ----           ---  ----
V_Power_Status T_Power_Status

--------------------------------------------------------------------------------
Section 2.3: Enumeration(s)
---------------------------
T_Power_Status = {e_Sub_Normal,e_Normal,e_Above_Normal,e_Init_P_SetBack,
e_Emergency_Shut_Down}

================================================================================

================================================================================
Section 3: Test Values from Boundary Value Analysis
    V_Reactor_Power      : 0, 1, 12, 13, 24, 25, 26, 52, 65, 77, 104, 105,
                           106, 116, 117, 129, 130, 131, 132, 134, 135,
                           136, 145, 159, 160, 161, 192, 205, 249, 250

    V_Power_Set_Point    : 10, 11, 12, 14, 15, 16, 65, 67, 119, 120

================================================================================

============================================================
 Section 4: Summary of each Condition along with Associated Actions
      ======================================================
       Table Power_Status : Condition 1
      ======================================================

C : V_Reactor_Power < (V_Power_Set_Point - (0.5 * C_Power_Dead_Band ) ) AND (V
_Reactor_Power < C_Danger_Power )
A01: V_Power_Status = e_Sub_Normal

      ======================================================
       Table Power_Status : Condition 2
      ======================================================

C : (V_Reactor_Power >= (V_Power_Set_Point - (0.5 * C_Power_Dead_Band ) )) AND
 (V_Reactor_Power <= (V_Power_Set_Point + (0.5 * C_Power_Dead_Band ))) AND (V_
Reactor_Power < C_Danger_Power )
A01: V_Power_Status = e_Normal

      ======================================================
       Table Power_Status : Condition 3
      ======================================================
```

```
C : (V_Reactor_Power > (V_Power_Set_Point + (0.5 * C_Power_Dead_Band ) )) AND
(V_Reactor_Power < C_Danger_Power )
A01: V_Power_Status = e_Above_Normal

      ================================================
       Table Power_Status : Condition 4
      ================================================

C : (V_Reactor_Power >= C_Danger_Power ) AND (V_Reactor_Power < C_Critical_Pow
er )
A01: V_Power_Status = e_Init_P_SetBack

      ================================================
       Table Power_Status : Condition 5
      ================================================

C : (V_Reactor_Power > C_Critical_Power )
A01: V_Power_Status = e_Emergency_Shut_Down

================================================================
```

```
================================================================================
Section 5: Test Cases along with Anticipated Test Results

WARNING: Tool encountered incomplete design specification;
-------  please review design document.
```

Test Case#	Input V_Reactor_Power	Input V_Power_Set_Point	Anticipated Output V_Power_Status
1	0	10	e_Normal
2	0	11	e_Normal
3	0	12	e_Normal
4	0	14	e_Normal
5	0	15	e_Normal
6	0	16	e_Sub_Normal
7	0	65	e_Sub_Normal
8	0	67	e_Sub_Normal
9	0	119	e_Sub_Normal
10	0	120	e_Sub_Normal
11	1	10	e_Normal
12	1	11	e_Normal
13	1	12	e_Normal
14	1	14	e_Normal
15	1	15	e_Normal
16	1	16	e_Normal
17	1	65	e_Sub_Normal
18	1	67	e_Sub_Normal
19	1	119	e_Sub_Normal
20	1	120	e_Sub_Normal
21	12	10	e_Normal
22	12	11	e_Normal
23	12	12	e_Normal
24	12	14	e_Normal
25	12	15	e_Normal
26	12	16	e_Normal
27	12	65	e_Sub_Normal
28	12	67	e_Sub_Normal
29	12	119	e_Sub_Normal
30	12	120	e_Sub_Normal
31	13	10	e_Normal
32	13	11	e_Normal
33	13	12	e_Normal
34	13	14	e_Normal
35	13	15	e_Normal
36	13	16	e_Normal
37	13	65	e_Sub_Normal
38	13	67	e_Sub_Normal
39	13	119	e_Sub_Normal
40	13	120	e_Sub_Normal
41	24	10	e_Normal
42	24	11	e_Normal
43	24	12	e_Normal
44	24	14	e_Normal
45	24	15	e_Normal
46	24	16	e_Normal
47	24	65	e_Sub_Normal
48	24	67	e_Sub_Normal

49	24	119	e_Sub_Normal
50	24	120	e_Sub_Normal
51	25	10	e_Normal
52	25	11	e_Normal
53	25	12	e_Normal
54	25	14	e_Normal
55	25	15	e_Normal
56	25	16	e_Normal
57	25	65	e_Sub_Normal
58	25	67	e_Sub_Normal
59	25	119	e_Sub_Normal
60	25	120	e_Sub_Normal
61	26	10	e_Above_Normal
62	26	11	e_Normal
63	26	12	e_Normal
64	26	14	e_Normal
65	26	15	e_Normal
66	26	16	e_Normal
67	26	65	e_Sub_Normal
68	26	67	e_Sub_Normal
69	26	119	e_Sub_Normal
70	26	120	e_Sub_Normal
71	52	10	e_Above_Normal
72	52	11	e_Above_Normal
73	52	12	e_Above_Normal
74	52	14	e_Above_Normal
75	52	15	e_Above_Normal
76	52	16	e_Above_Normal
77	52	65	e_Normal
78	52	67	e_Normal
79	52	119	e_Sub_Normal
80	52	120	e_Sub_Normal
81	65	10	e_Above_Normal
82	65	11	e_Above_Normal
83	65	12	e_Above_Normal
84	65	14	e_Above_Normal
85	65	15	e_Above_Normal
86	65	16	e_Above_Normal
87	65	65	e_Normal
88	65	67	e_Normal
89	65	119	e_Sub_Normal
90	65	120	e_Sub_Normal
91	77	10	e_Above_Normal
92	77	11	e_Above_Normal
93	77	12	e_Above_Normal
94	77	14	e_Above_Normal
95	77	15	e_Above_Normal
96	77	16	e_Above_Normal
97	77	65	e_Normal
98	77	67	e_Normal
99	77	119	e_Sub_Normal
100	77	120	e_Sub_Normal
101	104	10	e_Above_Normal
102	104	11	e_Above_Normal
103	104	12	e_Above_Normal
104	104	14	e_Above_Normal
105	104	15	e_Above_Normal
106	104	16	e_Above_Normal
107	104	65	e_Above_Normal
108	104	67	e_Above_Normal
109	104	119	e_Normal
110	104	120	e_Sub_Normal
111	105	10	e_Above_Normal
112	105	11	e_Above_Normal
113	105	12	e_Above_Normal
114	105	14	e_Above_Normal
115	105	15	e_Above_Normal
116	105	16	e_Above_Normal
117	105	65	e_Above_Normal
118	105	67	e_Above_Normal
119	105	119	e_Normal
120	105	120	e_Normal
121	106	10	e_Above_Normal
122	106	11	e_Above_Normal
123	106	12	e_Above_Normal
124	106	14	e_Above_Normal
125	106	15	e_Above_Normal
126	106	16	e_Above_Normal
127	106	65	e_Above_Normal
128	106	67	e_Above_Normal

129	106	119	e_Normal
130	106	120	e_Normal
131	116	10	e_Above_Normal
132	116	11	e_Above_Normal
133	116	12	e_Above_Normal
134	116	14	e_Above_Normal
135	116	15	e_Above_Normal
136	116	16	e_Above_Normal
137	116	65	e_Above_Normal
138	116	67	e_Above_Normal
139	116	119	e_Normal
140	116	120	e_Normal
141	117	10	e_Above_Normal
142	117	11	e_Above_Normal
143	117	12	e_Above_Normal
144	117	14	e_Above_Normal
145	117	15	e_Above_Normal
146	117	16	e_Above_Normal
147	117	65	e_Above_Normal
148	117	67	e_Above_Normal
149	117	119	e_Normal
150	117	120	e_Normal
151	129	10	e_Above_Normal
152	129	11	e_Above_Normal
153	129	12	e_Above_Normal
154	129	14	e_Above_Normal
155	129	15	e_Above_Normal
156	129	16	e_Above_Normal
157	129	65	e_Above_Normal
158	129	67	e_Above_Normal
159	129	119	e_Normal
160	129	120	e_Normal
161	130	10	e_Init_P_SetBack
162	130	11	e_Init_P_SetBack
163	130	12	e_Init_P_SetBack
164	130	14	e_Init_P_SetBack
165	130	15	e_Init_P_SetBack
166	130	16	e_Init_P_SetBack
167	130	65	e_Init_P_SetBack
168	130	67	e_Init_P_SetBack
169	130	119	e_Init_P_SetBack
170	130	120	e_Init_P_SetBack
171	131	10	e_Init_P_SetBack
172	131	11	e_Init_P_SetBack
173	131	12	e_Init_P_SetBack
174	131	14	e_Init_P_SetBack
175	131	15	e_Init_P_SetBack
176	131	16	e_Init_P_SetBack
177	131	65	e_Init_P_SetBack
178	131	67	e_Init_P_SetBack
179	131	119	e_Init_P_SetBack
180	131	120	e_Init_P_SetBack
181	132	10	e_Init_P_SetBack
182	132	11	e_Init_P_SetBack
183	132	12	e_Init_P_SetBack
184	132	14	e_Init_P_SetBack
185	132	15	e_Init_P_SetBack
186	132	16	e_Init_P_SetBack
187	132	65	e_Init_P_SetBack
188	132	67	e_Init_P_SetBack
189	132	119	e_Init_P_SetBack
190	132	120	e_Init_P_SetBack
191	134	10	e_Init_P_SetBack
192	134	11	e_Init_P_SetBack
193	134	12	e_Init_P_SetBack
194	134	14	e_Init_P_SetBack
195	134	15	e_Init_P_SetBack
196	134	16	e_Init_P_SetBack
197	134	65	e_Init_P_SetBack
198	134	67	e_Init_P_SetBack
199	134	119	e_Init_P_SetBack
200	134	120	e_Init_P_SetBack
201	135	10	e_Init_P_SetBack
202	135	11	e_Init_P_SetBack
203	135	12	e_Init_P_SetBack
204	135	14	e_Init_P_SetBack
205	135	15	e_Init_P_SetBack
206	135	16	e_Init_P_SetBack
207	135	65	e_Init_P_SetBack
208	135	67	e_Init_P_SetBack

209	135	119	e_Init_P_SetBack
210	135	120	e_Init_P_SetBack
211	136	10	e_Init_P_SetBack
212	136	11	e_Init_P_SetBack
213	136	12	e_Init_P_SetBack
214	136	14	e_Init_P_SetBack
215	136	15	e_Init_P_SetBack
216	136	16	e_Init_P_SetBack
217	136	65	e_Init_P_SetBack
218	136	67	e_Init_P_SetBack
219	136	119	e_Init_P_SetBack
220	136	120	e_Init_P_SetBack
221	145	10	e_Init_P_SetBack
222	145	11	e_Init_P_SetBack
223	145	12	e_Init_P_SetBack
224	145	14	e_Init_P_SetBack
225	145	15	e_Init_P_SetBack
226	145	16	e_Init_P_SetBack
227	145	65	e_Init_P_SetBack
228	145	67	e_Init_P_SetBack
229	145	119	e_Init_P_SetBack
230	145	120	e_Init_P_SetBack
231	159	10	e_Init_P_SetBack
232	159	11	e_Init_P_SetBack
233	159	12	e_Init_P_SetBack
234	159	14	e_Init_P_SetBack
235	159	15	e_Init_P_SetBack
236	159	16	e_Init_P_SetBack
237	159	65	e_Init_P_SetBack
238	159	67	e_Init_P_SetBack
239	159	119	e_Init_P_SetBack
240	159	120	e_Init_P_SetBack
241	160	10	Incomplete Spec
242	160	11	Incomplete Spec
243	160	12	Incomplete Spec
244	160	14	Incomplete Spec
245	160	15	Incomplete Spec
246	160	16	Incomplete Spec
247	160	65	Incomplete Spec
248	160	67	Incomplete Spec
249	160	119	Incomplete Spec
250	160	120	Incomplete Spec
251	161	10	e_Emergency_Shut_Down
252	161	11	e_Emergency_Shut_Down
253	161	12	e_Emergency_Shut_Down
254	161	14	e_Emergency_Shut_Down
255	161	15	e_Emergency_Shut_Down
256	161	16	e_Emergency_Shut_Down
257	161	65	e_Emergency_Shut_Down
258	161	67	e_Emergency_Shut_Down
259	161	119	e_Emergency_Shut_Down
260	161	120	e_Emergency_Shut_Down
261	192	10	e_Emergency_Shut_Down
262	192	11	e_Emergency_Shut_Down
263	192	12	e_Emergency_Shut_Down
264	192	14	e_Emergency_Shut_Down
265	192	15	e_Emergency_Shut_Down
266	192	16	e_Emergency_Shut_Down
267	192	65	e_Emergency_Shut_Down
268	192	67	e_Emergency_Shut_Down
269	192	119	e_Emergency_Shut_Down
270	192	120	e_Emergency_Shut_Down
271	205	10	e_Emergency_Shut_Down
272	205	11	e_Emergency_Shut_Down
273	205	12	e_Emergency_Shut_Down
274	205	14	e_Emergency_Shut_Down
275	205	15	e_Emergency_Shut_Down
276	205	16	e_Emergency_Shut_Down
277	205	65	e_Emergency_Shut_Down
278	205	67	e_Emergency_Shut_Down
279	205	119	e_Emergency_Shut_Down
280	205	120	e_Emergency_Shut_Down
281	249	10	e_Emergency_Shut_Down
282	249	11	e_Emergency_Shut_Down
283	249	12	e_Emergency_Shut_Down
284	249	14	e_Emergency_Shut_Down
285	249	15	e_Emergency_Shut_Down
286	249	16	e_Emergency_Shut_Down
287	249	65	e_Emergency_Shut_Down
288	249	67	e_Emergency_Shut_Down

```
289          249          119          e_Emergency_Shut_Down
290          249          120          e_Emergency_Shut_Down
291          250          10           e_Emergency_Shut_Down
292          250          11           e_Emergency_Shut_Down
293          250          12           e_Emergency_Shut_Down
294          250          14           e_Emergency_Shut_Down
295          250          15           e_Emergency_Shut_Down
296          250          16           e_Emergency_Shut_Down
297          250          65           e_Emergency_Shut_Down
298          250          67           e_Emergency_Shut_Down
299          250          119          e_Emergency_Shut_Down
300          250          120          e_Emergency_Shut_Down
```

```
WARNING: Tool encountered incomplete design specification;
-------  please review design document.
```

D.3 File pwr_di.etr(V0.0)

```
================================================
Section 1: Configuration Information
Access Program:      PWR$Display_Status
Design Document Rev:    0.0
Design Document Date:   98/10/20
Apollo Version:      2.1 Exp
Run Date:            Thu Oct 22 18:02:01 1998

================================================


================================================================================
Section 2.1: Inputs Table

Name              Type              Low  High Delta
----              ----              ---  ---- -----
V_Reactor_Power   T_Reactor_Power   0    250  1
V_Power_Set_Point T_Power_Set_Point 10   120  1

================================================================================

================================================================================
Section 2.2: Outputs Table

Name             Type               Low  High
----             ----               ---  ----
V_Display_Status T_P_Display_Status

--------------------------------------------------------------------------------
Section 2.3: Enumeration(s)
--------------------------
T_P_Display_Status = (e_Blue,e_Green,e_Amber,e_Red,e_Flashing_Red)

================================================================================

================================================================================
Section 3: Test Values from Boundary Value Analysis
    V_Reactor_Power      : 0, 1, 12, 13, 24, 25, 26, 65, 68, 80, 104, 105,
                           106, 120, 134, 135, 136, 137, 138, 152, 167,
                           168, 169, 193, 209, 249, 250

    V_Power_Set_Point    : 10, 11, 12, 14, 15, 16, 64, 65, 67, 119, 120

================================================================================

============================================================
 Section 4: Summary of each Condition along with Associated Actions
      ======================================================
       Table Power_Status : Condition 1
      ======================================================

C : V_Reactor_Power < (V_Power_Set_Point + (0.5 * C_Power_Dead_Band ) ) AND (V
_Reactor_Power <=(1.05 * C_Danger_Power) )
A01: V_Display_Status = e_Blue

      ======================================================
       Table Power_Status : Condition 2
      ======================================================

C : (V_Reactor_Power >= (V_Power_Set_Point - (0.5 * C_Power_Dead_Band ) )) AND
 (V_Reactor_Power <= (V_Power_Set_Point + (0.5 * C_Power_Dead_Band ))) AND (V_
Reactor_Power < (1.05 * C_Danger_Power) )
A01: V_Display_Status = e_Green

      ======================================================
       Table Power_Status : Condition 3
      ======================================================

C : (V_Reactor_Power > (V_Power_Set_Point + (0.5 * C_Power_Dead_Band ) )) AND
```

```
(V_Reactor_Power < (1.05 * C_Danger_Power) )
A01: V_Display_Status = e_Amber

      ------------------------------------------------
       Table Power_Status : Condition 4
      ------------------------------------------------

C : (V_Reactor_Power >= (1.05 * C_Danger_Power )) AND (V_Reactor_Power < (1.05
 * C_Critical_Power) )
A01: V_Display_Status = e_Red

      ------------------------------------------------
       Table Power_Status : Condition 5
      ------------------------------------------------

C : (V_Reactor_Power >= (1.05 * C_Critical_Power) )
A01: V_Display_Status = e_Flashing_Red

------------------------------------------------------------

------------------------------------------------------------------------
Section 5: Test Cases along with Anticipated Test Results

WARNING: Tool encountered non-deterministic test case(s);
-------  please review design document.
```

Test Case#	Input V_Reactor_Power	Input V_Power_Set_Point	Anticipated Output V_Display_Status
1	0	10	NonDeterministic
2	0	11	NonDeterministic
3	0	12	NonDeterministic
4	0	14	NonDeterministic
5	0	15	NonDeterministic
6	0	16	e_Blue
7	0	64	e_Blue
8	0	65	e_Blue
9	0	67	e_Blue
10	0	119	e_Blue
11	0	120	e_Blue
12	1	10	NonDeterministic
13	1	11	NonDeterministic
14	1	12	NonDeterministic
15	1	14	NonDeterministic
16	1	15	NonDeterministic
17	1	16	NonDeterministic
18	1	64	e_Blue
19	1	65	e_Blue
20	1	67	e_Blue
21	1	119	e_Blue
22	1	120	e_Blue
23	12	10	NonDeterministic
24	12	11	NonDeterministic
25	12	12	NonDeterministic
26	12	14	NonDeterministic
27	12	15	NonDeterministic
28	12	16	NonDeterministic
29	12	64	e_Blue
30	12	65	e_Blue
31	12	67	e_Blue
32	12	119	e_Blue
33	12	120	e_Blue
34	13	10	NonDeterministic
35	13	11	NonDeterministic
36	13	12	NonDeterministic
37	13	14	NonDeterministic
38	13	15	NonDeterministic
39	13	16	NonDeterministic
40	13	64	e_Blue
41	13	65	e_Blue
42	13	67	e_Blue
43	13	119	e_Blue
44	13	120	e_Blue
45	24	10	NonDeterministic
46	24	11	NonDeterministic
47	24	12	NonDeterministic
48	24	14	NonDeterministic
49	24	15	NonDeterministic

50	24	16	NonDeterministic
51	24	64	e_Blue
52	24	65	e_Blue
53	24	67	e_Blue
54	24	119	e_Blue
55	24	120	e_Blue
56	25	10	e_Green
57	25	11	NonDeterministic
58	25	12	NonDeterministic
59	25	14	NonDeterministic
60	25	15	NonDeterministic
61	25	16	NonDeterministic
62	25	64	e_Blue
63	25	65	e_Blue
64	25	67	e_Blue
65	25	119	e_Blue
66	25	120	e_Blue
67	26	10	e_Amber
68	26	11	e_Green
69	26	12	NonDeterministic
70	26	14	NonDeterministic
71	26	15	NonDeterministic
72	26	16	NonDeterministic
73	26	64	e_Blue
74	26	65	e_Blue
75	26	67	e_Blue
76	26	119	e_Blue
77	26	120	e_Blue
78	65	10	e_Amber
79	65	11	e_Amber
80	65	12	e_Amber
81	65	14	e_Amber
82	65	15	e_Amber
83	65	16	e_Amber
84	65	64	NonDeterministic
85	65	65	NonDeterministic
86	65	67	NonDeterministic
87	65	119	e_Blue
88	65	120	e_Blue
89	68	10	e_Amber
90	68	11	e_Amber
91	68	12	e_Amber
92	68	14	e_Amber
93	68	15	e_Amber
94	68	16	e_Amber
95	68	64	NonDeterministic
96	68	65	NonDeterministic
97	68	67	NonDeterministic
98	68	119	e_Blue
99	68	120	e_Blue
100	80	10	e_Amber
101	80	11	e_Amber
102	80	12	e_Amber
103	80	14	e_Amber
104	80	15	e_Amber
105	80	16	e_Amber
106	80	64	e_Amber
107	80	65	e_Green
108	80	67	NonDeterministic
109	80	119	e_Blue
110	80	120	e_Blue
111	104	10	e_Amber
112	104	11	e_Amber
113	104	12	e_Amber
114	104	14	e_Amber
115	104	15	e_Amber
116	104	16	e_Amber
117	104	64	e_Amber
118	104	65	e_Amber
119	104	67	e_Amber
120	104	119	NonDeterministic
121	104	120	e_Blue
122	105	10	e_Amber
123	105	11	e_Amber
124	105	12	e_Amber
125	105	14	e_Amber
126	105	15	e_Amber
127	105	16	e_Amber
128	105	64	e_Amber
129	105	65	e_Amber

130	105	67	e_Amber
131	105	119	NonDeterministic
132	105	120	NonDeterministic
133	106	10	e_Amber
134	106	11	e_Amber
135	106	12	e_Amber
136	106	14	e_Amber
137	106	15	e_Amber
138	106	16	e_Amber
139	106	64	e_Amber
140	106	65	e_Amber
141	106	67	e_Amber
142	106	119	NonDeterministic
143	106	120	NonDeterministic
144	120	10	e_Amber
145	120	11	e_Amber
146	120	12	e_Amber
147	120	14	e_Amber
148	120	15	e_Amber
149	120	16	e_Amber
150	120	64	e_Amber
151	120	65	e_Amber
152	120	67	e_Amber
153	120	119	NonDeterministic
154	120	120	NonDeterministic
155	134	10	e_Amber
156	134	11	e_Amber
157	134	12	e_Amber
158	134	14	e_Amber
159	134	15	e_Amber
160	134	16	e_Amber
161	134	64	e_Amber
162	134	65	e_Amber
163	134	67	e_Amber
164	134	119	e_Green
165	134	120	NonDeterministic
166	135	10	e_Amber
167	135	11	e_Amber
168	135	12	e_Amber
169	135	14	e_Amber
170	135	15	e_Amber
171	135	16	e_Amber
172	135	64	e_Amber
173	135	65	e_Amber
174	135	67	e_Amber
175	135	119	e_Amber
176	135	120	e_Green
177	136	10	e_Amber
178	136	11	e_Amber
179	136	12	e_Amber
180	136	14	e_Amber
181	136	15	e_Amber
182	136	16	e_Amber
183	136	64	e_Amber
184	136	65	e_Amber
185	136	67	e_Amber
186	136	119	e_Amber
187	136	120	e_Amber
188	137	10	e_Red
189	137	11	e_Red
190	137	12	e_Red
191	137	14	e_Red
192	137	15	e_Red
193	137	16	e_Red
194	137	64	e_Red
195	137	65	e_Red
196	137	67	e_Red
197	137	119	e_Red
198	137	120	e_Red
199	138	10	e_Red
200	138	11	e_Red
201	138	12	e_Red
202	138	14	e_Red
203	138	15	e_Red
204	138	16	e_Red
205	138	64	e_Red
206	138	65	e_Red
207	138	67	e_Red
208	138	119	e_Red
209	138	120	e_Red

210	152	10	e_Red
211	152	11	e_Red
212	152	12	e_Red
213	152	14	e_Red
214	152	15	e_Red
215	152	16	e_Red
216	152	64	e_Red
217	152	65	e_Red
218	152	67	e_Red
219	152	119	e_Red
220	152	120	e_Red
221	167	10	e_Red
222	167	11	e_Red
223	167	12	e_Red
224	167	14	e_Red
225	167	15	e_Red
226	167	16	e_Red
227	167	64	e_Red
228	167	65	e_Red
229	167	67	e_Red
230	167	119	e_Red
231	167	120	e_Red
232	168	10	e_Flashing_Red
233	168	11	e_Flashing_Red
234	168	12	e_Flashing_Red
235	168	14	e_Flashing_Red
236	168	15	e_Flashing_Red
237	168	16	e_Flashing_Red
238	168	64	e_Flashing_Red
239	168	65	e_Flashing_Red
240	168	67	e_Flashing_Red
241	168	119	e_Flashing_Red
242	168	120	e_Flashing_Red
243	169	10	e_Flashing_Red
244	169	11	e_Flashing_Red
245	169	12	e_Flashing_Red
246	169	14	e_Flashing_Red
247	169	15	e_Flashing_Red
248	169	16	e_Flashing_Red
249	169	64	e_Flashing_Red
250	169	65	e_Flashing_Red
251	169	67	e_Flashing_Red
252	169	119	e_Flashing_Red
253	169	120	e_Flashing_Red
254	193	10	e_Flashing_Red
255	193	11	e_Flashing_Red
256	193	12	e_Flashing_Red
257	193	14	e_Flashing_Red
258	193	15	e_Flashing_Red
259	193	16	e_Flashing_Red
260	193	64	e_Flashing_Red
261	193	65	e_Flashing_Red
262	193	67	e_Flashing_Red
263	193	119	e_Flashing_Red
264	193	120	e_Flashing_Red
265	209	10	e_Flashing_Red
266	209	11	e_Flashing_Red
267	209	12	e_Flashing_Red
268	209	14	e_Flashing_Red
269	209	15	e_Flashing_Red
270	209	16	e_Flashing_Red
271	209	64	e_Flashing_Red
272	209	65	e_Flashing_Red
273	209	67	e_Flashing_Red
274	209	119	e_Flashing_Red
275	209	120	e_Flashing_Red
276	249	10	e_Flashing_Red
277	249	11	e_Flashing_Red
278	249	12	e_Flashing_Red
279	249	14	e_Flashing_Red
280	249	15	e_Flashing_Red
281	249	16	e_Flashing_Red
282	249	64	e_Flashing_Red
283	249	65	e_Flashing_Red
284	249	67	e_Flashing_Red
285	249	119	e_Flashing_Red
286	249	120	e_Flashing_Red
287	250	10	e_Flashing_Red
288	250	11	e_Flashing_Red
289	250	12	e_Flashing_Red

```
290          250          14           e_Flashing_Red
291          250          15           e_Flashing_Red
292          250          16           e_Flashing_Red
293          250          64           e_Flashing_Red
294          250          65           e_Flashing_Red
295          250          67           e_Flashing_Red
296          250          119          e_Flashing_Red
297          250          120          e_Flashing_Red

------------------------------------------------------------------------------

WARNING: Tool encountered non-deterministic test case(s);
-------  please review design document.
```

D.4 File pwr_al.etr(V0.0)

```
================================================
Section 1: Configuration Information
Access Program:      PWR$Alarm_Status
Design Document Rev:    0.0
Design Document Date:   98/10/20
Apollo Version:      2.1 Exp
Run Date:            Thu Oct 22 18:03:09 1998

================================================


================================================================================
Section 2.1: Inputs Table

Name             Type             Low  High Delta
----             ----             ---  ---- -----
V_Reactor_Power T_Reactor_Power 0    250  1

================================================================================

================================================================================
Section 2.2: Outputs Table

Name            Type             Low  High
----            ----             ---  ----
V_Alarm_Status T_P_Alarm_Status


--------------------------------------------------------------------------------
Section 2.3: Enumeration(s)
--------------------------
T_P_Alarm_Status = (e_Off,e_Intermittent,e_Continuous)

================================================================================

================================================================================
Section 3: Test Values from Boundary Value Analysis
    V_Reactor_Power       : 0, 1, 71, 142, 143, 144, 159, 175, 176, 177, 178,
                            179, 213, 214, 249, 250

================================================================================

============================================================
 Section 4: Summary of each Condition along with Associated Actions
       ======================================================
        Table Power_Status : Condition 1
       ======================================================

C : (V_Reactor_Power < (1.1 * C_Danger_Power) )
A01: V_Alarm_Status = e_Off

       ======================================================
        Table Power_Status : Condition 2
       ======================================================

C : (V_Reactor_Power >= (1.1 * C_Danger_Power )) AND (V_Reactor_Power <= (1.1
* C_Critical_Power) )
A01: V_Alarm_Status = e_Intermittent

       ======================================================
        Table Power_Status : Condition 3
       ======================================================

C : (V_Reactor_Power > (1.11 * C_Critical_Power) )
A01: V_Alarm_Status = e_Continuous

============================================================
```

Section 5: Test Cases along with Anticipated Test Results

WARNING: Tool encountered incomplete design specification;
------- please review design document.

Test Case#	Input V_Reactor_Power	Anticipated Output V_Alarm_Status
	---	---
1	0	e_Off
2	1	e_Off
3	71	e_Off
4	142	e_Off
5	143	e_Intermittent
6	144	e_Intermittent
7	159	e_Intermittent
8	175	e_Intermittent
9	176	e_Intermittent
10	177	Incomplete Spec
11	178	e_Continuous
12	179	e_Continuous
13	213	e_Continuous
14	214	e_Continuous
15	249	e_Continuous
16	250	e_Continuous

WARNING: Tool encountered incomplete design specification;
------- please review design document.

Appendix E - EBNF Grammar

This appendix is a partial adaptation (with a number of modifications) of EBNF grammar presented in an overview document associated with CASE tools research work carried out at AECL [Matias 1998]. The following three topics are presented in this appendix:

- EBNF grammar
- mathematical functions, and
- predicate calculus symbols

EBNF grammar

The syntax of the Apollo tool is described using Extended Backus-Naur Form (EBNF) Notation. The extensions to standard BNF used in this document are described below.

[ ]	Bracketing
|	Or
{ }$_{1+}$	One or More
{ }$_{0+}$	Zero or More
< >	Token defined within the BNF Grammar
' '	The item in quotes should be literally found within the sequence

Condition

Condition statements appear in Function Tables. Each of the specified operators, and math functions has their standard mathematical interpretation.

```
<condition> ::=     '(' condition ')' |
                    'NOT' '(' condition ')' |
                    <condition> [ 'AND' | '&' ] <condition> |
                    <condition> 'OR' <condition> |
                    <expression> <logic_comp> <expression> |
                    <range_expr> |
                    <logical_equ_compare> |
                    <logical_value>

<logical_equ_compare> ::= <expression> '=' <logical_value> |
                    <expression> [ '<>' | '!=' ] <logical_value>

<logical_value> ::= 'TRUE' | 'FALSE'

<range_expr> ::=  <expression> <l_comp> <expression> <l_comp> <expression> |
                    <expression> <g_comp> <expression> <g_comp> <expression>

<expression> ::=  <symbol> |
                    <number> |
                    <expression> '+' <expression> |
                    <expression> '-' <expression> |
                    <expression> '*' <expression> |
                    <expression> '/' <expression> |
                    <math_function> '(' <expression> ')' |
                    '(' <expression> ')'
```

```
<math_function> ::= 'ABS' | 'FLOOR' | 'CEILING' | 'ROUND'

<logic_comp>  ::=       <l_comp> | '=' | '<>' | '!=' | <g_comp>

<g_comp> ::=       '>' | '>='

<l_comp> ::=       '<' | '<='
```

Action

The following production rules define the grammar for action statements occurring within the document. Action statements occur in the condition tables generated by joining the output name with the expression defining the result using an equal operator '='.

```
<action> ::=       <assign_statement>

<assign_statement> ::=  <symbol> '=' <expression> |
                        <symbol> '=' <logical_value>
```

Basic Definitions

The following EBNF productions define basic constructs in the tabular specification language.

```
<number> ::=       <real> | <integer>
<real> ::=         <digits> '.' <digits>

<integer> ::=      <digits> | '-' <digits>

<chars> ::=        {<char>}1.

<char> ::=         <digit> | <letter> | '_' | '.'

<letter> ::=       <lowercase> | <uppercase> | '$'

<lowercase> ::=    a | b | c | d | ... | y | z

<uppercase> ::=    A | B | C | D | ... | Y | Z

<digits> ::=       {<digit>}1.

<digit> ::=        1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0
```

Data Type Definition

The Apollo tool has support for the `INTEGER, REAL` predefined data types. Support is also available for building more complex data types, such as enumerations, and sub-intervals. Table 26 provides a summary of the data types supported by the Apollo tool.

Table 26 - SDD Predefined Data Types and Modifiers

Predefined Data Type/Modifier	Explanation
INTEGER	An integer for which no specific internal representation is defined.
REAL	An external real value, such as an analogue input.
x TO y	Subranges of INTEGERs are defined with "TO". Using this, an 8-bit byte can be defined as "0 TO 255", and a 16-bit word or unsigned integer can be defined as "0 TO 65535".
{ a, b, c }	Curly brackets such as "{" are used to define a list of enumerated values.

When defining a data type in the Design Specification document, the syntax defined by the `<datatype>` production should be used.

```
<data_type> ::=    <base_type> | <enum_type>

<base_type> ::=    'REAL'  | 'INTEGER' |<subrange>

<subrange> ::=     <subrange_index> 'TO' <subrange_index>

<subrange_index> ::= <number> | <constant>

<constant>  ::=    <symbol>

<enum_type> ::=    '{' <enum_def>  { ',' <enum_def> }0- '}'

<enum_def> ::=     <enum> | <enum> = <number>

<enum>  ::=        <symbol>

<symbol>  ::=      <symbol_name>

<symbol_name> ::= <letter> {<char> }0-
```

Mathematical Functions

ABS(α) - *Absolute value of α*, $|\alpha|$

These mathematical functions take on their standard interpretation. The result from the evaluation of one of the predefined functions that yields an error, or when the output is not a natural number, is undefined.

Type Casting

CEILLING (α) - *Smallest integer not less than α*, $\lceil \alpha \rceil$
FLOOR(α) - *Smallest integer not less than α*, $\lfloor \alpha \rfloor$

ROUND (α)

The **ROUND** function returns the closest whole number to α, as defined by Equation (1).

$$ROUND(\alpha) = \begin{cases} \lfloor \alpha \rfloor & ,\alpha < \lfloor \alpha \rfloor + \frac{1}{2} \\ \lceil \alpha \rceil & .\alpha \geq \lfloor \alpha \rfloor + \frac{1}{2} \end{cases} \quad (1)$$

Predicate Calculus Symbols

Table 27 lists the predicate calculus symbols along with their equivalent in tabular notation that is supported by Apollo.

Table 27 - Symbols

Predicate Calculus Symbol	Interpretation	Equivalent in Apollo
∧	Conjunction	AND or &
∨	Disjunction	OR
~	Logical Negation	NOT
=	Equality	=[1]
≠	Not Equal	<> or !=
←	Assignment	=

[1] The equal sign "=" when used in a condition is interpreted as equality; otherwise, it is interpreted as an assignment.