SEMI-AUTOMATED KNOWLEDGE ACQUISITION FROM EXISTING TEXTUAL DATABASES

by

Joozar K. Vasi

TR92-068 August 1992

This is an unaltered version of the author's
M.Sc.(CS) Thesis

Faculty of Computer Science
University of New Brunswick
P.O. Box 4400
Fredericton, N.B. E3B 5A3

Phone: (506) 453-4566
Fax: (506) 453-3566

SEMI-AUTOMATED KNOWLEDGE ACQUISITION FROM EXISTING TEXTUAL DATABASES

by

Joozar K. Vasi

BSc(Physics), Bombay University, 1984

BSc(CS), University of New Brunswick, 1990

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science in Computer Science

in the Faculty

of

Computer Science

This thesis is accepted.

.....

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

August, 1992

Table of Contents

Abstract

The introduction of this thesis looks at languages to represent plant operating procedures. Techniques that can be used to translate plant operating procedures from textual form to these languages are also explored.

A context-free grammar representation of a knowledge base for plant operating procedures called GENeric Representation Language (GENRL) was developed. A non-interactive syntax checker for GENRL using Lex and Yacc was built. The design and partial implementation of GEN*RL* Knowledge Acquisition Tool (GENKAT) were completed. GENKAT translated plant operating procedures from WordPerfect text files to a knowledge base whose structure was developed previously. This knowledge base consists of 14 different frames corresponding to fourteen generic tasks typical of plant operating procedures.

GENKAT consists of two main components. The first component accepts input from WordPerfect text files and produces a partially complete knowledge base. The second component allows the user to interactively modify the knowledge base.

GENKAT was developed using Sed, Lex, Yacc and C on the SUN™ Sparcstation 2. It consists of 1975 lines of code. It was used to automatically translate chapter 5 (Normal Operations) and chapter 6 (Abnormal Operations) of the liquid zone control operating manual of the Pt. Lepreau generating station. This resulted in a total of 1125 GENRL frames. Four of the fourteen generic tasks are currently recognized.

Acknowledgments

I would like to thank AECL for their financial support for my work.

Thanks are also due to Professor Brad Nickerson, my supervisor, for his patience and supervision.

Finally, I want to thank my father, Khozema, and my mother, Nafisa, for their support.

List of Figures

List of Tables

1. Introduction

1.1. Background

Plant operating manuals are an established method for making available information on how to operate efficiently any large plant. Power plants, especially nuclear power plants, use operating manuals extensively. For example, there are over 200 operating manuals at Pt. Lepreau comprising of more than 10,000 pages. There is a need to make this voluminous information available to plant operators efficiently (i.e., the right information quickly). To achieve this end many attempts have been made to computerize plant operating procedures. Some of these attempts are described in this chapter.

1.2. The Interactive Operating Advisor

The Interactive Operating Advisor (IOA) built by Maillet [Maillet 90] is an expert system which provides a computerized version of plant operating procedures in the liquid zone control manual [Parker 87] of the Pt. Lepreau nuclear power plant. Like many expert systems the IOA is composed of the following three parts:

1) A set of rules

2) A database of facts

3) An inference engine for firing the rules according to matches with the database of facts

These three parts can be classified into the following categories:

1) Generic component: This component defines the structure and control of the IOA. It consists of all the three parts mentioned above but the database of facts does not contain the declarative knowledge of the manual (the slots of frames are empty).

2) Specific component: This component consists of the declarative knowledge present in the manual. This knowledge is present in the database of facts. The database of facts has all the declarative knowledge of the manual in addition to the general structure that describes them.

The database of facts consisting of both components will henceforth be called frame templates or just frames.

Figure 1 shows the generic and specific components of the IOA.

Two types of frame templates were defined by Maillet. They are:

1) Operating Manual Frame Templates (OMFTs): These templates describe operational procedures and other information of the operating manual when completely filled.

2) Equipment Frame Templates (EFTs): These templates model the physical system of the nuclear power plant as described in the operating manual when completely filled.

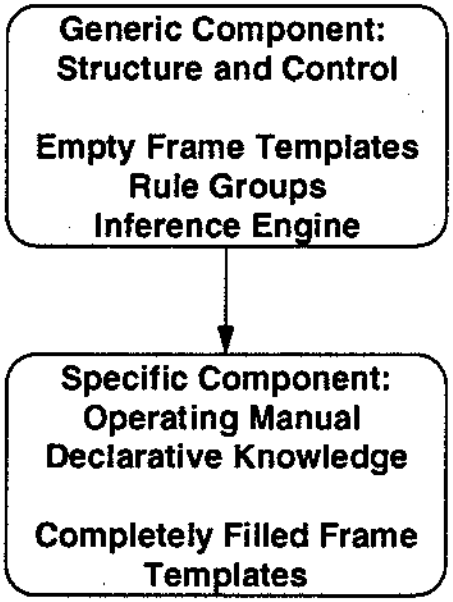


Figure 1. Composition and relation between generic and specific components of the IOA [Maillet 90].

All OMFTs have one slot for the location of the information they contain. This information is divided into two parts. The first part consists of the procedure number and step number if present. The second part consists of a sequence number. The sequence number is used to sequence the OMFTs within a procedure or procedure step.

Within the expert system, a control frame keeps track of what procedure number, step number, and sequence number of the operating manual are currently being processed.

Maillet also defined generic tasks which are small elements of work that require similar processing. For each generic task there is one OMFT and one rule group. OMFTs provide a method of representing the task in a form understandable to the expert system. The rule group is responsible for overseeing the execution of this task. Table 1 shows the primary tasks for which OMFTs and rule groups were developed.

Table 1. List of generic tasks [Maillet 90].

1) Equipment-state-requirement-with-and-logic

2) Equipment-state-requirement-with-or-logic

3) Select-equipment-item

4) Monitor-equipment-item

5) Find-equipment-item

6) Display-text

7) Display-diagram

8) Wait

9) Display-current-context

10) Link-context-with-return

11) Link-context

12) Conditional-link-context

13) Conditional-link-context-with-return

14) Question-link-with-return

As new generic tasks are recognized to be present in operating procedures the generic component (excluding the inference engine) can be extended to include them.

EFTs and OMFTs will vary from manual to manual. They form the basis of the language [Hopcroft 79] of the variable portion of the IOA knowledge base. Adherence of EFTs and OMFTs to the language can be verified using standard language verification techniques [Aho 86].

1.3. Literature Review

1.3.1. Attempts to Computerize Plant Operating Procedures

Five projects related to automating operating procedures at nuclear power plants are described below:

COPMA

A project was undertaken in Halden, Norway, to produce a Computerized Operating Manual (COPMA) for nuclear plant operating procedures [Krogsæter 89] [Sverre 90] [Nelson 90]. On-line COPMA (on the TI explorer) needs a procedure database encoded in the language PROLA. COPMA is connected to the nuclear power plant simulator (on a Norsk Data computer) and can access its process database.

A procedure in PROLA is encoded as follows:

```
Procedure <procedure-identifier>
      Step <step-identifier>
            Instruction
            Instruction
            ...
      Endstep
      Step <step-identifier>
            Instruction
            Instruction
            ...
      Endstep
      ...
Endprocedure
```

Figure 2. A procedure in PROLA [Sverre 90].

Any number of instructions can reside in each procedure step. Instructions are the actions and checks that the procedure designer wants the operator or computer to do at the moment, such as give or read some specific information, ask the operator to manually

perform some checks, let the computer automatically check and evaluate some condition, let the computer monitor a process condition, or branch to another procedure step. The PROLA language syntax checker and compiler were written in Prolog.

PROLA procedures are entered using an off-line Procedure EDitor (PED). PED also supports:

1) An editor for procedure graphs that can be entered within a procedure

2) PROLA editing and compiling

3) Annotations to procedures

4) A variable address table that simplifies access to the simulator database during COPMA execution

The COPMA screen consists of many windows which have different responsibilities of helping the operator. The man machine interface of COPMA simulates a desk of manuals as shown in Figure 3.

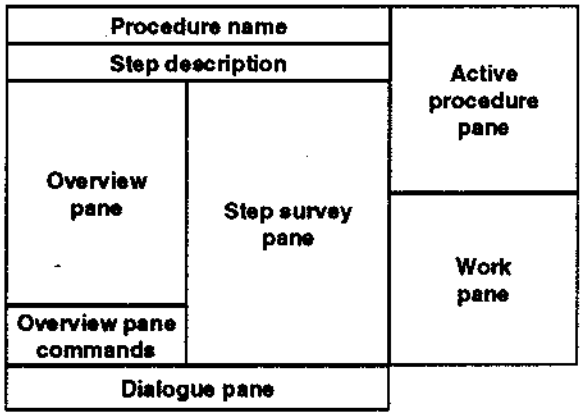


Figure 3. COPMA man machine interface [Krogsæter 89].

Each window is an object with attributes and information to interpret a set of commands. Windows interact with the help of a message coordinator which is itself an object. The message coordinator buffers input messages received from windows and redistributes them to windows that need to be informed with the help of a queue.

Each procedure is a static data structure which can be activated to produce an object of the class 'procedure'. Another module called the kernel is responsible for handling active

procedures. Also a communication module is present between COPMA (on a TI Explorer) and the simulator (on a Norsk Data computer). Communication programs on the two computers communicate by exchanging messages over Ethernet using the TCP/IP protocol (see Figure 4).

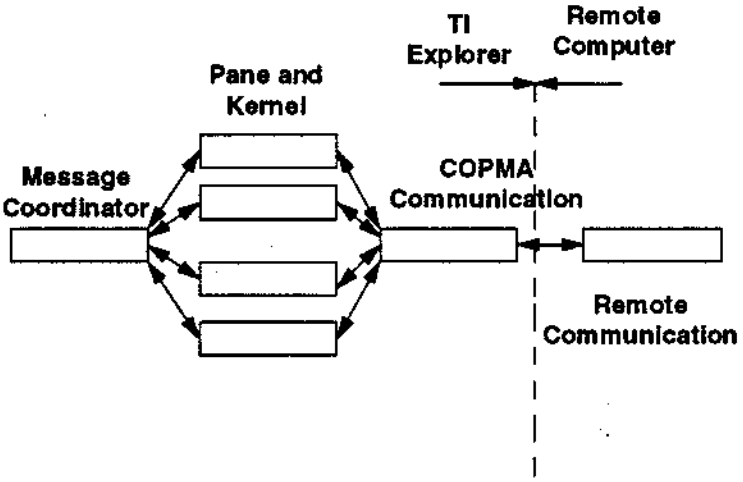


Figure 4. Diagram of overall structure of on-line COPMA [Krogsæter 89]

Integrated OAS

The Integrated Operator Advisor System (OAS) is a knowledge-based system for the plant monitoring, procedure management, and diagnosis. It was developed by the Laboratory for Artificial Intelligence Research at Ohio State University [Bhatnagar 90].

The overall architecture of the OAS is based on four generic tasks [Chandrasekaran 86] from which the four modules of the OAS were developed. Each generic task has the following information:

1) a task specification in the form of generic types of input and output information;

2) specific forms in which the basic pieces of domain knowledge are needed for the task, and specific organizations of this knowledge particular to the task;

3) a family of control regimes appropriate to the task.

Generic tasks can be used in problem solving if:

1) the complex problem can be decomposed into generic tasks;

2) paths and conditions for information transfer from the agents that perform these generic tasks to the others which need the information can be established;

3) knowledge of the domain is available to encode the knowledge structures of the generic task.

Each generic task has its own representation language and control regime.

The OAS consists of four modules as shown in figure 5:

1) An intelligent database (generic task of data abstraction)

2) A Plant Status Monitoring System (PSMS) (generic task of monitoring)

3) A Dynamic Procedure Monitoring System (DPMS) (generic task of plan execution for situation control)

4) A Diagnosis and sensor Validation System (DVS) (generic task of diagnosis)

The detection of abnormal functioning is handled by the PSMS (generic task of monitoring) and the control of abnormal functioning is handled by the DPMS (generic task of plan execution for situation control).

DPMS takes care of:

1) selecting procedures that will best maintain the plant safety in the shortest possible time, when a number of malfunction states are detected and when during the execution of a procedure other more critical safety threats are detected;

2) guiding the operator through the steps of an identified plan;

3) monitoring the success of a procedure;

4) modifying the procedure to maintain plant safety when the executing procedure is not, or cannot be, successful.

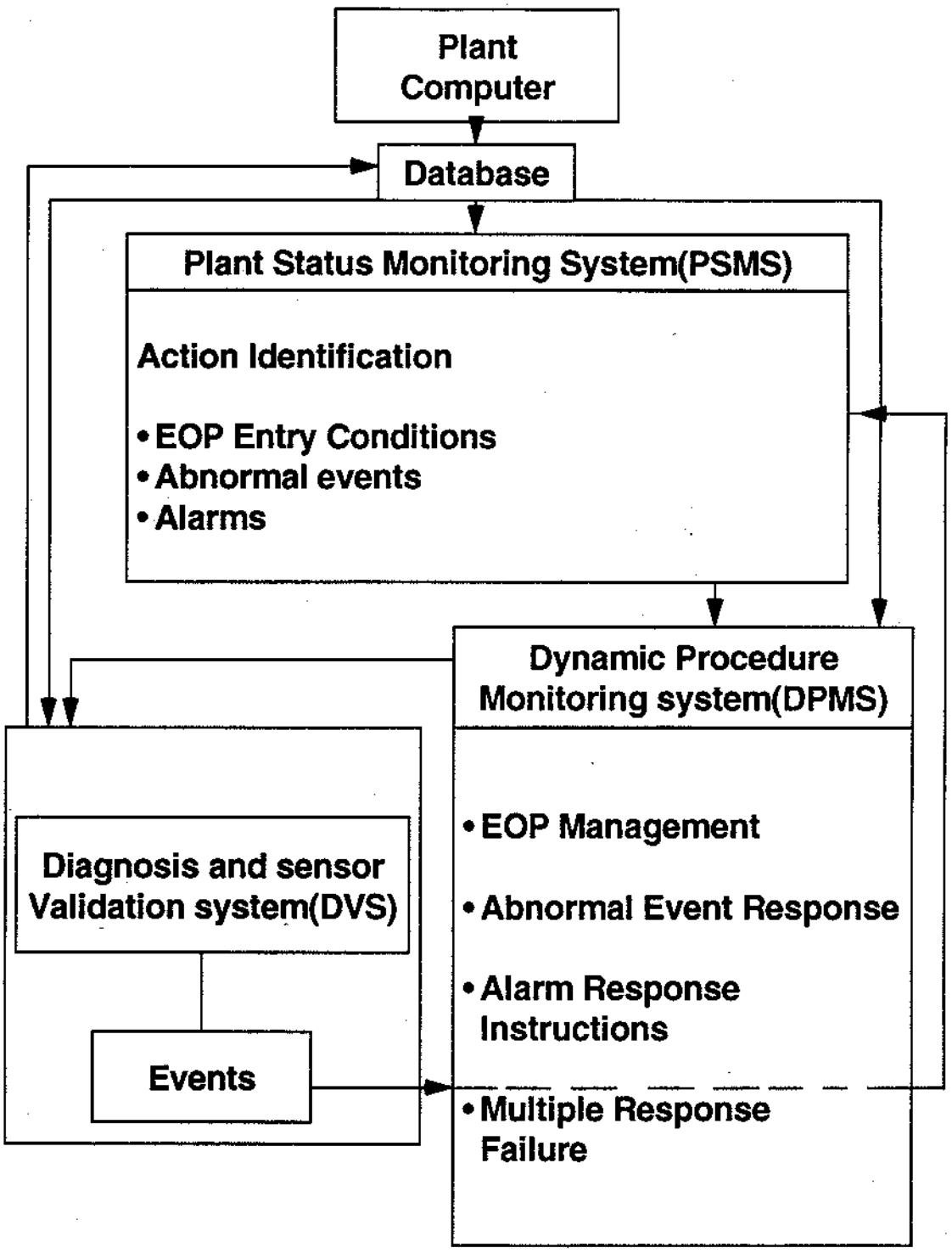


Figure 5. Overall structure of Integrated OAS [Bhatnagar 90].

A high level language called language called Dynamic Procedures Representation Language (DPRL) was developed to represent safety threats and abnormal event procedures, and relationships among these procedures to control abnormal plant functioning.

DPRL provides constructs for the representation of the following:

1) Steps of procedures

2) Objectives of the procedures and how they relate to the overall plant safety

3) Relations among sub-procedures that make up a procedure

4) Success criteria of a procedure

5) Alternatives to maintain plant safety if the procedure is not successful

The procedures and the relationships among them are represented at four levels of abstraction: SPECIALIST, PLAN, PROCEDURE and STEP. The level of abstraction increases from abstract to detailed from SPECIALIST to STEP.

A SPECIALIST is defined for each malfunction state for which a procedure is available. SPECIALISTs initiate required procedures to combat the malfunction state and have the knowledge for safety maintenance if those procedures fail. The knowledge representation of a SPECIALIST is structured as follows [Bhatnagar 90]:

```
(SPECIALIST (NAME )
      (SUPER-SPECIALIST )
      (SAFETY-GOAL )
      (SUB-SPECIALIST )
      (PLANS ))
```

PLAN and PROCEDURE: The PLAN represents the actual control procedure at an abstract level. The PLAN contains the name of the procedure, and the sub procedures and the steps that make up the procedure. A PROCEDURE is a subPLAN. Both PLAN and PROCEDURE contain knowledge for plan execution, monitoring, and modification. A PLAN is represented as follows [Bhatnagar 90]:

```
(PLAN (NAME )
      (SAFETY-GOAL )
      (EXECUTION-TYPE )
      (PREREQUISITE )
      (CRITERIA )
      (USED-BY )
      (BODY ))
```

STEP: This is the most detailed level of knowledge representation. It can contain:

1) A display action

2) Actions that are done on or using some system or component

3) Conditional actions done on or using some system or component

4) Actions to verify the effects of the actions taken

5) Actions to monitor the effects of actions taken for a given amount of time

A display step is structured as follows [Bhatnagar 90]:

```
(DISPLAY (This procedure is a sub procedure of PEI-B13))
```

On identification of each malfunction state the conflict resolution scheme in the OAS updates a plan set that contains the names of the SPECIALISTs for the already detected malfunction states.

A human interface in the form a shell with templates for SPECIALISTs, PLANs, and PROCEDUREs is provided to help in filling in the procedural knowledge available from the plant manuals and from the expertise of the plant personnel. The creation of the knowledge base starts by defining the malfunction states and the procedures to control them. The SPECIALISTs, PLANs, and PROCEDUREs are created with the help of a shell which displays required templates to be filled in. All the information filled in is checked for correct syntax before it is compiled. The shell can help in completing the knowledge base in many ways. One useful check would be to point out which SPECIALISTs, PLANs and PROCEDUREs have been mentioned but have not been compiled.

PCONS

The paper describing PCONS [Krieger 91] begins by introducing different formats for storing procedures. Figure 6 shows an approximate cost versus benefit curve for converting procedures to different formats.

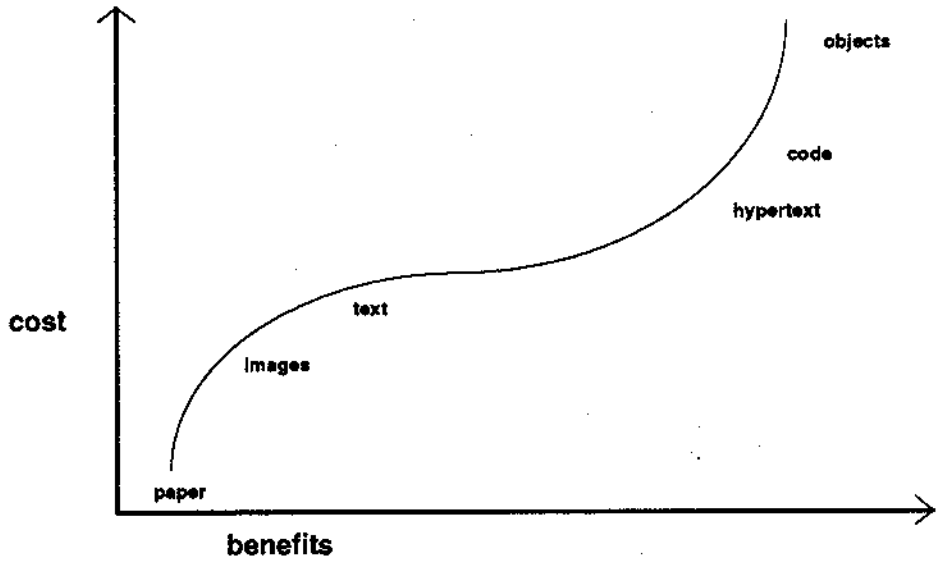


Figure 6. Cost versus benefit curve for converting to different procedure formats [Krieger 91].

The costs associated with representing procedures can be reduced no matter how the procedures are stored provided the format is chosen before hand. To convert procedures from paper to text one does not have to convert the text to images stored on disk and then to text files that can be manipulated by a word processor. Intermediate conversion steps need not be followed.

PCONS was developed in the Symbolics Genera environment. The basic module in PCONS is a clause which is either a complete sentence or a part of a sentence. The writer creates a procedure by using existing modules or creating new ones. Commands to the database allow the writer to locate existing modules based on parameters such as object category (e.g., valve) or operation type (e.g., close).

The successive stages of building a clause are shown in Figure 7.

Objects close, valve V25, and valve 24 are stored in an object base separate from PCONS. They are retrieved from the object base by menus. Other formatting is also added to the clause by the interface to make it readable (e.g., the word "and" in Figure 7).

Close

Close valve V25

Close valves V25 and V24

Figure 7. A clause being built in PCONS [Krieger 91].

Storing plant items in an object base allows the use of multiple formats. Some of the formats developed in the project were:

1) Textual format: Formatting of text is done on each module in a procedure. Annotations can be retrieved from the object base when the procedure refers to the object (i.e., the can be stored with the object).

2) On-screen format: Procedures appear on the screen in a format similar to the textual format. When the operator points to an object in the procedure text (e.g., valve 20), he can perform operations on the object (e.g., close valve 20).

3) Executable code format: A code fragment of a programming language can be created by fetching objects from the object base. For example the following fragment will print a message to the operator on the screen to close valves 25 and 24:

```
printf("Close valves V25 and V24");
```

COMPRO

COmputerized PROcedures (COMPRO) [Lipner, 91] is a computer implementation of emergency operating procedures developed by Westinghouse Electric Corporation. Lipner and Orendi give a description of computerization issues that were addressed during the development process but they provide no implementation details.

The issues considered were:

1) Operator freedom

2) Sequential steps

3) Critical safety function status trees

4) Rediagnosis

5) Foldout page

6) Notes and cautions

7) Continuous monitored parameters and initiated actions

8) Prioritizing the issues

9) Integration with graphic display systems

PASS

Procedure Analysis Software System (PASS) [Robert 89] [Horne 89] is an ongoing EPRI project to investigate computer based methods to improve the development, maintenance, and verification of plant operating procedures. The first main goal of the project was to investigate the applicability of Structured Software Analysis (SSA) to computerizing operating procedures. SSA methods offer benefits only if procedures are transformed into a format that can be easily be used by a computer (e.g., a programming language). A translator, which used natural language techniques, was developed to perform the conversion. Finally, possibilities for automated verification methods for computerized procedures were considered.

Typical Westinghouse Emergency Operating Procedures (EOPs) have a title, a scope, entry conditions, steps, and optional appendices containing supporting material. A step has three components

- *Action*: represents an action or an observation made by the operator
- *ExpectedResponse*: gives an observation to be make by the operator to verify the step *action*, or more information on how to perform an *action*
- *ResponseNotObtained*: gives an action to be performed if the *ExpectedResponse* is not observed

The grammar of a Westinghouse procedure is:

Procedure → Boilerplate Step+

BoilerPlate → TITLE SCOPE CATEGORY DATE REVISION SYMPTOMS NOTES CAUTIONS CONDITIONS

Step → Action ExpectedResponse ResponseNotObtained

Action → TEXT Instruction+

ExpectedResponse → TEXT Instruction+

ResponseNotObtained → Instruction+

Instruction → Rule | IMPERATIVE | NOTE | CAUTION

Rule → IFEXPR THENEXPR ELSEEXPR

In the PASS system all procedures are represented as structured text instead of rules and text corresponding to the rules. The PASS compiler has the information about the input vocabulary of terms and their syntactic classification. It transforms the structured text to assertions and rules for on-line execution. Computerized versions of plant operating procedures were produced using the compile and test approach shown in Figure 8.

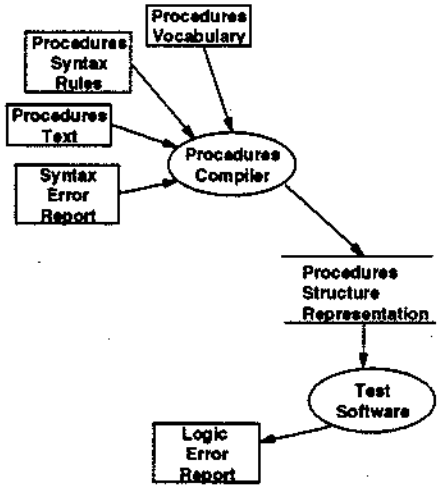


Figure 8. The compile and test approach [Horne 89].

The compiler translates the text into a set of logical assertions and rules containing the form *verb system state*. Verbs in these forms are mapped to one of the following semantic primitives:

- *Query*: Verbs like verify and check correspond to observations of a plant system or component through an instrument.

- *Set*: These verbs (such as set or trip) correspond to operator actions that realign a plant system or component through a controller.

- *Goto*: Procedures may invoke other procedures through this primitive.

- *ReferTo*: Procedures may refer the operator to additional information in this procedure or some other procedure

The syntax used to compile procedure text is a subset of standard English specialized for procedures. Augmented Transition Networks (ATN) perform syntax analysis. The current PASS syntax rules have 31 independent ATNs.

It was concluded that the following was essential to make procedures suitable for automated computer analysis

- The final state of the plant on executing a procedure should be precisely defined.

- The vocabulary of plant actions, components and states should be constrained.

- Procedures should be written using well specified and limited grammatical structures.

- Whenever possible a plant action should be accompanied by a description to verify that the action succeeded.

- Care should be taken to consistently represent and correlate blocks of AND/OR portions of procedures.

- Procedures should be written on a computer file in a form that allows the writer to add comments about the issues and information sources considered in writing those procedures.

1.3.2. Important features of Attempts to Computerize Plant Operating Procedures

Most of the attempts described previously had the following in common:

1) The language to represent procedures is built around procedure steps as its basic building block. Each step contains operator actions and checks typical of plant operating procedures. Therefore each step has restricted content.

2) Object oriented programming techniques were used to handle complexity.

3) An elaborate man machine interface was developed. The system has to be "operator-friendly."

4) Monitoring of system parameters is done automatically. The computerized manual is connected to the simulator.

5) Procedures have a hierarchy among them. Procedures that handle safety threats have precedence over procedures that handle abnormal events.

6) An intelligent editor for the knowledge base was developed to check for correct syntax and semantics.

7) If a translator to computerize procedures is to be developed then the format of the input procedures should have the characteristics identified in the PASS project.

1.3.3. A General Overview of Formal Languages

Translation between one machine format and another, for example, between a programming language and assembler is a well-documented problem in computing literature (e.g., [Aho 86]). These methods, however, are defined for only certain classes of texts — only those that can be described by context-free languages (or subsets of them).

Context-free languages are part of a hierarchy of languages, called the "Chomsky hierarchy" [Luger 89] [Hopcroft 79]. These classes are defined because they form

potential models for natural languages. The languages, in order of increasing complexity, are:

1) Regular languages: In this class of language all productions are of the form

$A \rightarrow wB$

$A \rightarrow w$

where A and B are non-terminals and w is a string (possibly empty) of terminals. The machine that recognizes a regular language is called a Deterministic Finite Automaton (DFA) and an example is shown in Figure 9. This machine makes a decision on what state to go to on the basis of its current state (say A in Figure 9) and the current input symbol (say 0 in Figure 9). The input symbol is consumed. If the machine is in some acceptable final state (e.g., A in Figure 9) on scanning the input stream symbols, the machine halts and declares that the sequence of input symbols is in the language. A regular language can also be recognized by a Nondeterministic Finite Automaton (NFA) because every language recognized by a NFA can be recognized by a DFA. Regular expressions [Hopcroft 79] can also be used to represent regular languages.

2) Context-Free Languages (CFLs): In this class of language all productions are of the form:

$A \rightarrow \alpha$

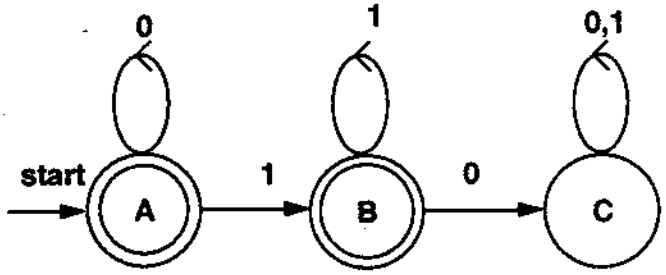


Figure 9. Example of a DFA recognizing a regular language.

where A is a non-terminal and α consists of terminals and non-terminals in any order. The machine that recognizes a context-free language is called a nondeterministic Push Down Automaton (PDA) and an example is shown in Figure 10.

The instantaneous description of the machine is fully determined by the current state (q in Figure 10) and the stack of symbols. The next set of instantaneous descriptions is determined by its current state (q in Figure 10), the current input symbol (0 in Figure 10 which is consumed) and its current symbol on top of the stack (A in Figure 10). The next state of the stack is obtained by popping the current stack symbol of the stack and pushing zero or more stack symbols on the stack. A sequence of input symbols is said to be in the language if the stack becomes empty, or equivalently, the machine reaches a designated final state.

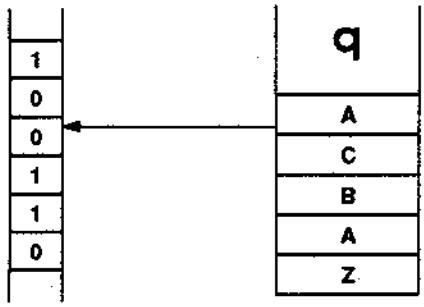


Figure 10. Example of PDA.

The CFLs contain Deterministic Context-Free Languages (DCFLs). The syntax of most programming languages can be described by DCFLs. LR grammars [Hopcroft 79] generate exactly the DCFLs. To prove a language is a DCFL usually involves building a Deterministic Push Down Automaton (DPDA) or LR-grammar to describe the language.

LR parsing is attractive for the following reasons [Aho 86]:

1) LR parsing method is the most general non backtracking shift-reduce parsing method known, yet its implementation is efficient.

2) The class of grammars that can be parsed using LR methods is a proper superset of the class of grammars that can be parsed with a predictive parser.

3) An LR parser can detect errors in a left-to right scan of the input as soon as possible.

It is very tedious to construct a LR parser by hand. Therefore parser generators like Yacc (see section 1.4.2) are used to do the job.

3) Context-sensitive languages: In this class of language all productions are of the form:

$\alpha \rightarrow \beta$

where α and β consist of terminals and non-terminals in any order with $|\alpha| \leq |\beta|$. The machine that recognized context-sensitive languages is the Linear Bounded Automaton (LBA). An LBA is a nondeterministic Turing machine satisfying the following two conditions:

1) Its input alphabet includes two special symbols ¢ and $, the left and right end markers, respectively.

2) The LBA has no moves left from ¢ or right from $, nor may it print another symbol over ¢ or $.

4) Recursively enumerable languages: In this class of language all productions are of the form:

$\alpha \rightarrow \beta$

where α and β consist of terminals and non-terminals in any order. The machine that recognizes a recursively enumerable language is the deterministic Turing Machine (TM) which is shown in Figure 11.

The instantaneous description of this machine determined by the input symbols to the rightmost non blank or the symbol to the left of the head, whichever is rightmost; and the current state of the head within this string (e.g., abcqaaa in Figure 11. The current input symbol being scanned is to the left of the state q). On the basis of the current instantaneous description the head of the machine moves right or left. During this motion the head writes a symbol on the input tape. At the end of the move the finite control of the head changes state. The deterministic TM accepts a sequence of input symbols on tape if its head starts scanning from the left edge of the input tape and the finite control reaches a designated final state.

It is possible that the deterministic TM may never halt on an input sequence of symbols not in the language.

Note that this machine is more powerful than the other two machines because of its rescanning and writing capabilities.

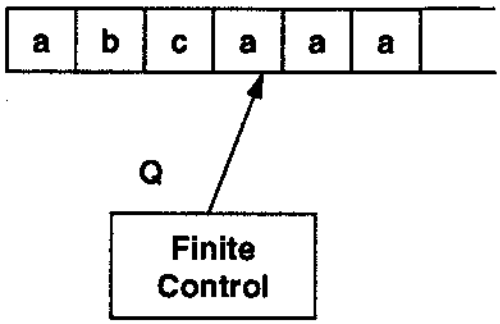


Figure 11. Example of a deterministic Turing Machine (TM).

Figure 12 gives an overall view of the hierarchy of these languages showing complete containment of one language within another.

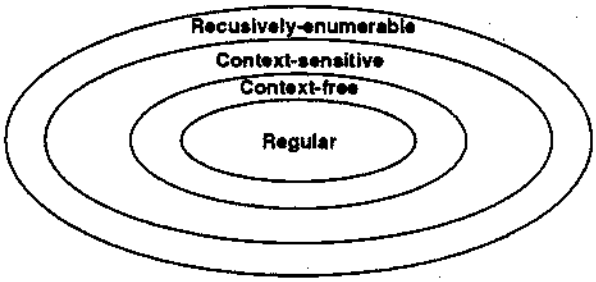


Figure 12. Chomsky hierarchy of languages.

When text is being analyzed, the regular part of text is recognized first since the recognition machine (the DFA) is simple and easy to program. If this machine (the DFA) is unable to do the job (when constructs of the form $a^n b^n$ have to be recognized) then the context-free part of the text is recognized by the PDA. Some constructs cannot be recognized by a PDA (e.g., constructs of the form $a^n b^n c^n$). These constructs include declaration of a variable before use and the problem of checking that the number of formal parameters in the declaration of a procedure agrees with the number of actual parameters in a use of the procedure [Aho 86]. The semantic analysis phase of a compiler takes care of these two problems.

1.3.4. Techniques for Analysis of Textual Documents

There are many levels of analysis of natural language. They include [Luger 89]:

1) *Prosody* deals with the rhythm and intonation of language.

2) *Phonology* studies how sounds form language. This is important in computerized speech recognition and generation.

3) *Morphology* is concerned with what constitutes words.

4) *Syntax* studies rules that determine legal sentence and rules to recognize and generate sentences.

5) *Semantics* considers meaning of sentences and its components and the way in which it is conveyed in natural language.

6) *Pragmatics* determines the way in which language is used and its effects on the listener.

7) *World knowledge* includes of the physical world, the world of human social interaction, and the role of goals and intentions in communication. This background knowledge is essential to understand the full meaning of text or a conversation.

All of these levels interact extensively.

The problem of natural language analysis of textual data can be divided into three phases as shown in Figure 13.

1) Parsing: This process involves use of language syntax (relationship between tokens). The end result is normally a parse tree.

2) Semantic interpretation: This stage produces a representation that reflects the meaning of the text (i.e., how the text relates to the real world). The end result are normally frames or logic based representations. Semantic consistency checks (analysis) are also performed during this phase.

3) Contextual knowledge/world interpretation: Structures from a knowledge base are added to the internal representation of the sentence. This knowledge base contains world knowledge required to complete the meaning of the sentence.

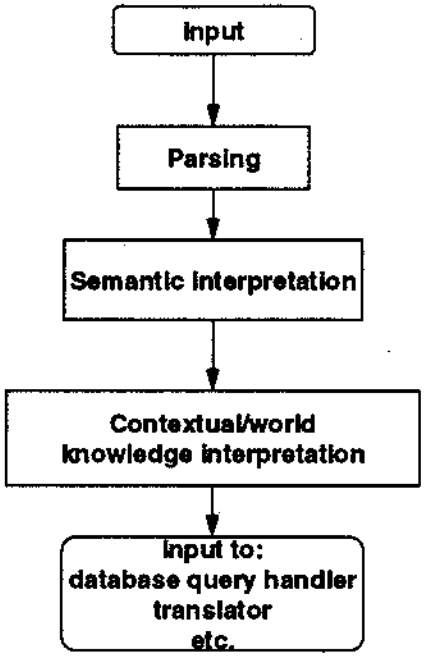


Figure 13. Overview of phases of natural language processing [Luger 89].

Machines that can be used for parsing include [Gazdar 89]:

1) Finite-State Transition Networks

2) Recursive Transition Networks

3) Augmented Transition Networks

4) Chart Parsers

Simple modifications of these machines are used to perform translations of natural language. Determining which machine to use is guided by three factors:

1) Mathematical adequacy: The machine cannot recognize constructs that are required to be recognized.

2) Notational adequacy: The machine is a good model of the Natural Language Processing (NLP) problem. RTNs are preferred over FSTNs since they allow commonly occurring subpatterns to be expressed as a named subnetwork, and large networks to be built up in a modular way. A RTN does not become too large because of repetitive specification.

3) Efficiency: A machine that takes less time is preferred.

Two papers discussing text analysis were studied, along with a review paper of 15 techniques.

Salton and Smith

This paper describes how text analysis can be used for automatic construction of book indices by using the PLNLP syntactic analyzer.

It was recognized that the syntactic analysis phase of natural language processing can encounter the following problems:

1) Ambiguous parse trees

2) Incomplete vocabulary

3) Requirement of extensive storage and computer speed

To overcome these three obstacles additional contextual information is obtained by machine dictionaries and manually prepared knowledge bases that reflect the semantic properties of the particular area of discourse. The knowledge base can form a semantic network. The following information retrieval strategy can be used to gather information relevant to the area of discourse from the knowledge base:

1) The available search request is analyzed into a formal representation similar to that used for the knowledge base.

2) A fuzzy matching operation is performed to compare the formalized search requests with the elements of the knowledge representation.

3) An answer to the search request is constructed if the degree of match between knowledge base and search request is sufficiently great.

The PLNLP syntactic analysis system has been developed at the IBM Research Laboratory in Yorktown Heights. This system analyzes complete sentences, as well as sentence fragments, producing in each case one or more syntactic parses for each sentence, ranked in decreasing order of presumed correctness. When the input cannot be analyzed using the normal grammar rules, a "fitted" parsing system is used to produce a

reasonable analysis for the apparently intractable fragment. Figure 14 shows PLNLP recognizing part of a sentence.

```
CMPD  DECL  NP    NP    NOUN  today
                  AJP   ADJ   large
                  NP    NOUN  disk
                  NOUN  arrays
            VERB  are
            ADJ   ADP   ADV   usually
                  ADJ   available
      CONJ  but
      DECL  NP    AJP   ADJ   using
                  AJP   ADJ   short
                  NP    NP    NOUN  texts
                        CONJ  and
                        NP    NOUN  small
                  NOUN  dictionary
```

Figure 14. PLNLP recognizing "today large disk arrays are usually available but using short texts and small dictionary" [Salton 89].

This paper demonstrates the power of syntactic analysis.

Jacobs and Rau

The System for Conceptual Information Summarization, Organization, and Retrieval (SCISOR) [Jacobs 90] is a prototype system that performs text analysis, creates a knowledge base, and answers queries with the conceptual representation of information in the knowledge base. This system works in constrained domains. The present implementation of the system analyzes text from an on-line financial service (Dow Jones ™) about corporate mergers and acquisitions. It runs on the Sun™ workstation and consists of 50,000 lines of Common Lisp code.

SCISOR processes news at the rate of approximately six stories per minute performing the following tasks:

- Lexical analysis of the input character stream including extracting names, dates, numbers.
- Reorganizing the raw news feed into a definite structure with separate headline, byline, and dateline designations.

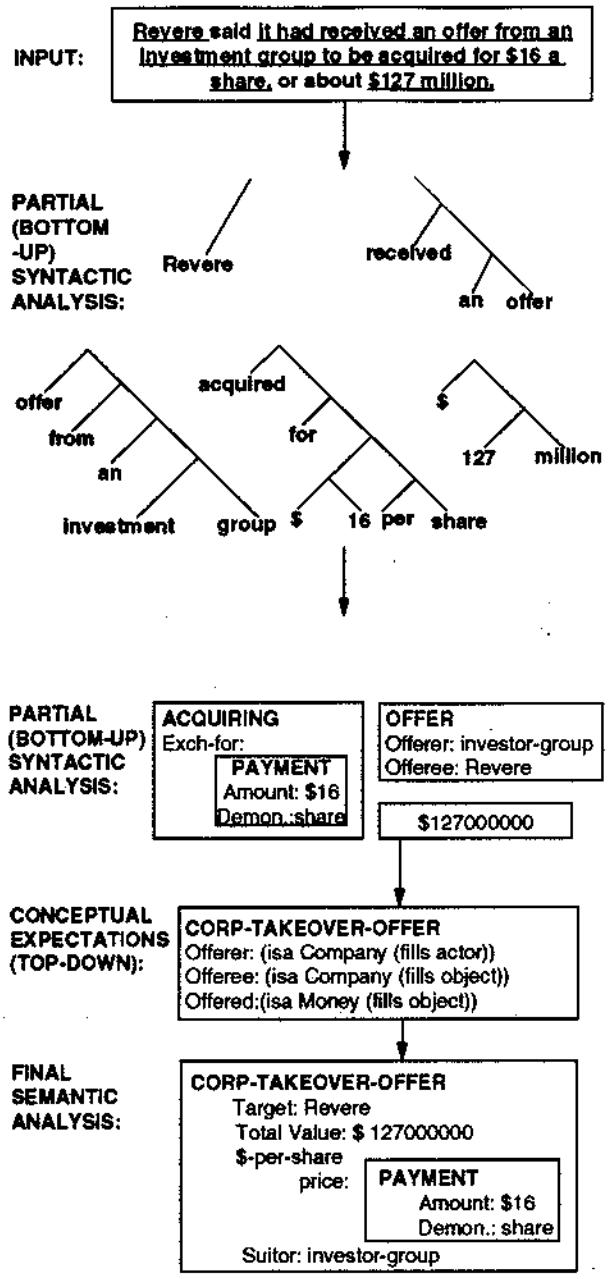


Figure 15. "Bottom-up" linguistic analysis and "top-down" conceptual interpretation in SCISOR [Jacobs 90].

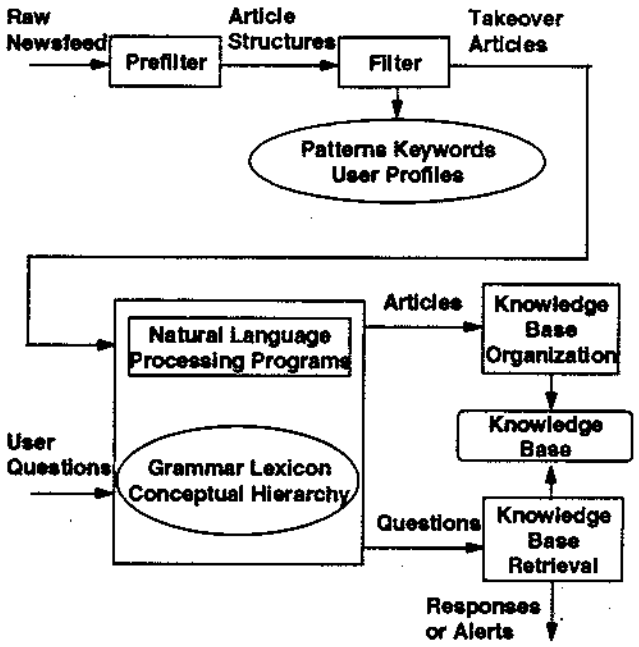


Figure 16. Overall structure of SCISOR [Jacobs 90].

- Classifying a new item into a corporate merger, acquisition, or other topic; a natural language analysis of the news item is performed using "bottom-up" linguistic analysis and "top-down" conceptual interpretation as shown in Figure 15.

- A knowledge base that handles the storage and retrieval of the conceptual representations of the news items.

The overall structure of SCISOR in Figure 16 clearly shows how the tasks mentioned above are performed.

Lehnert and Sundheim

Lehnert and Sundheim [1991] compare natural language processing methods with more conventional approaches to text analysis. This comparison is based on a recent evaluation of text-analysis technologies by the US Defense Advanced Research Projects Agency (DARPA). Lehnert and Sundheim [Lehnert 91] conclude that natural language techniques are better for certain types of applications:

> "...text-analysis techniques incorporating natural language processing are superior to traditional information-retrieval techniques based on statistical classification when applications require structured representation of the information present in texts."

Lehnert and Sundheim include the following summary of the state-of-the-art in text analysis:

> "...text-analysis techniques have progressed far beyond database interface applications and have demonstrated clear viability for information extraction from unconstrained text."

These conclusions were reached by comparing 15 text analysis systems. The input for each system comes from news articles concerning terrorism. The answer key to the translation consisted of output templates. These templates were developed for each article that was considered for testing the text analysis system. The template generated by the system and the template that formed the answer key were compared with regard to:

1) Recall or the completeness of the output templates

2) Precision or the accuracy with which the output template was filled

3) Overgeneration or the amount of irrelevant information generated by the system

4) Fallout or the tendency to fill slots incorrectly as number of potentially incorrectly filled slots increases

The two best performers had a recall of over 40%, with precision over 60%. Many sites where this experiment was performed had trouble with discourse analysis. In discourse analysis information of input sentences have to be reorganized into target template instantiations.

1.4. Tools in UNIX available for textual analysis

Figure 17 shows the UNIX tools available for textual analysis. The higher the location of the tool in the pyramid the more complex it is to use. The functionality of a tool at the bottom of the pyramid is much less than the one present on the top.

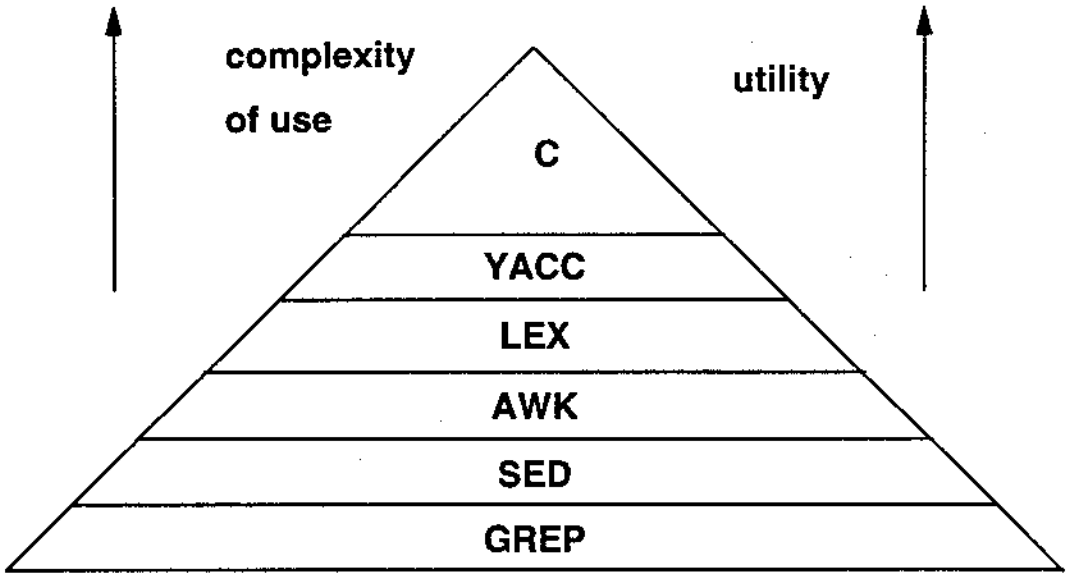


Figure 17. Hierarchy of UNIX tools [Mason 90].

Using tools in general has two main advantages:

1) No duplication of effort which spanned many years

2) The problem can be solved at a more abstract level using more problem oriented constructs

Lex and Yacc were used extensively in this thesis. A brief description of the nature of these tools follow.

1.4.1. Lex

Lex is a translator that converts regular expressions and their associated C routines that follow it into C code [Sun 90]. The regular expressions that match input text follow two rules:

1) The longest regular expression is matched if two regular expressions match the input text.

2) If two expressions of equal size are matched then the expression that comes first is matched to the input text.

Lex provides right and left context checking for matching regular expressions.

1.4.2. Yacc

Yacc is a translator that converts its input specification to C code [Sun 90]. The input specification is in terms of an LALR(1) grammar. This input specification recognizes a valid sequence of tokens and performs some actions on recognition of the sequence. These actions, which are specified in C code, can access values from the value stack (which contains inherited attributes attached to terminals and non-terminals) which runs in parallel with the Yacc state stack. When Yacc is invoked with the -v option, a file called *y.output* is produced. This file contains a human-readable description of the parser.

1.4.3. Interaction of Lex and Yacc

Figure 18 shows how Lex and Yacc can be used to produce an executable program which can be used as a translation tool or a language recognition tool. The Gnu C compiler (gcc) is present in the Figure 18 since it was used in this thesis.

Figure 19 shows how `main()`, the scanning routine produced by Lex (`yylex()`), and the parsing routine produced by Yacc (`yyparse()`), interact when processing input text. `Main()` calls `yyparse()`, and `yyparse()` calls `yylex()`. `yylex()` scans input text and returns the token number recognized to `yyparse()`, or zero if it is the end of input text.

1.5. Statement of Problem

As mentioned in section 1.2 the IOA consists of a generic component and a specific component. The generic component includes the structure of the database of facts that consists of empty EFTs and OMFTs. To get a working IOA this structure has to be filled in the appropriate places by the declarative knowledge present in the liquid zone control manual. This corresponds to the area of discourse analysis [Lehnert 91].

The following was proposed to be implemented considering sections 1.2, 1.3 and 1.4:

1) Develop a context-free grammar for OMFTs. This will give plant procedures a formal representation.

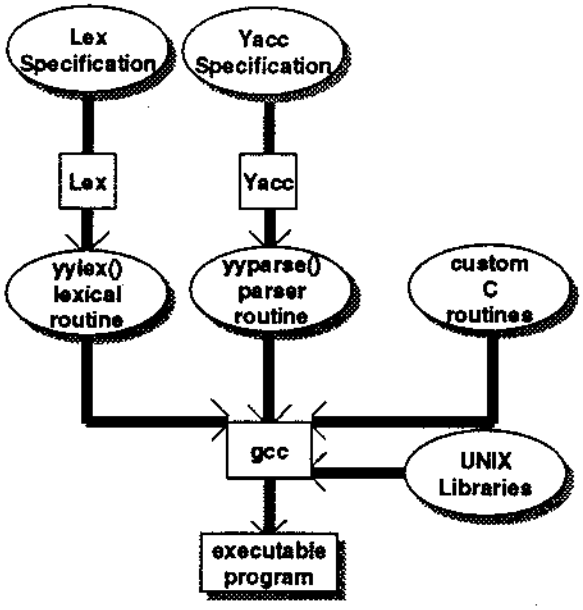


Figure 18. Using Lex and Yacc [Mason 90].

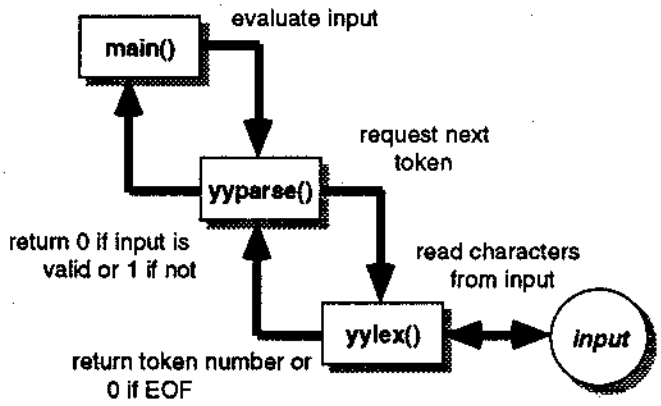


Figure 19. Interaction between C routines `main()`, `yylex()`, and `yyparse()` [Mason 90].

2) Develop a context-free grammar representation of EFTs. This will give equipment items a formal representation.

3) Build a scanner and parser for the EFTs and OMFTs. This constitutes the first step in the construction of an interactive completion and verification tool for EFTs and OMFTs.

4) Perform a translation from the electronic text [Johnson 91] to OMFTs.

5) Design and build a prototype for interactive completion and verification of the generated instances.

Tasks 4 and 5 were partially completed.

2. Context-free Grammar Representation of a Language for Plant Operating Procedures

Context-free productions are used to represent computer languages since they offer significant advantages to the language designer. The advantages are [Aho 86]:

1) A grammar gives a precise, yet easy-to-understand syntactic specification of a language.

2) From certain classes of grammars we can automatically construct an efficient parser that determines whether a source program is syntactically well formed.

3) New constructs can be easily added to the language as it evolves.

A GENeric Representation Language (GENRL) for plant operating procedures was developed. This language in the form of context-free productions gives OMFTs and EFTs (see section 1.2) a precise and concise representation.

2.1. GENRL representation of OMFTs

The grammar for OMFTs can be found in Appendix 1. There are a total of 48 productions in the grammar. This grammar shows in detail the structure of all the generic tasks enumerated by Maillet (see Table 1). Figure 20 shows three productions of the grammar. The instance-of slot determines which generic task this production corresponds to.

The first production corresponds to the generic task of equipment-state-requirement-with-and-logic (due to the slot (instance-of requirement)). Text for non-terminals <generic-task-name>, <context-name>, <context-seq-no>, and <equipment-desc-seq> has to be acquired from a manual containing operating procedures. The non-terminal <documentation> can contain the empty string. The non-terminal <generic-task-name> should be unique (this is required by ART) and should reflect the characteristics of the frame recognized (this provides good documentation). The non-terminals <context-name> and <context-seq-no> indicates where in the manual this frame is located. The last non-terminal <equipment-desc-seq> consists of a list of equipment items together with the state each equipment item is supposed to be in. The next 2 productions are for the generic

tasks equipment-state-requirement-with-or-logic and select-equipment-item. The corresponding frames that appear in Maillet's code are shown in Figure 21.

```
<requirement-frame> ::=
   (defschema <generic-task-name> <documentation>
   (instance-of requirement)
   (context (<context-name> <context-seq-no>))
   (equipment-list <equipment-desc-seq>+))

<requirement-or-frame> ::=
   (defschema <generic-task-name> <documentation>
   (instance-of requirement-or)
   (context (<context-name><context-seq-no>))
   (equipment-list <equipment-desc-seq>+))

<select-frame> ::=
   (defschema <generic-task-name> <documentation>
   (instance-of select)
   (context (<context-name> <context-seq-no>))
   (equipment-list <equipment-desc-seq>+))
```

Figure 20. GENRL description of three OMFTs.

```
(defschema req-5-2-1-step-1
   (instance-of requirement)
   (context (5-2-1-step-1 1))
   (equipment-list (1 v73 state "=" closed) (2
   v78 state "=" closed) (3 v103 state "="
   closed)))

(defschema req-5-2-1-step-3
   (instance-of requirement-or)
   (context (5-2-1-step-3 1)
   (equipment-list (1 iec1 state isolated) (2
   iec2 state isolated)))

(defschema select-5-2-4-step-3
   (instance-of select)
   (context (5-2-4-step-3 1))
   (equipment-list (1 p1 state on) (2 p2 state
   on) (3 p3 state on)))
```

Figure 21. Sample ART code for three OMFTs [Maillet 90].

2.2. GENRL representation of EFTs

The grammar for EFTs can be found in Appendix 2. There are a total of 33 productions in the grammar. This grammar shows in detail the structure of all the equipment items found in the liquid zone control manual [Parker 87]. Figure 22 shows productions that

recognize two equipment items, namely a tank and a pump. The instance-of slot determines which equipment item is being recognized.

Let us consider how a tank EFT is represented by the first production in Figure 22. The non-terminal <equipment-name> represents a unique string and has to reflect the type of equipment that is being recognized. The non-terminal <documentation> contains the empty string or some description of the equipment item. The instance-of slot determines the type of equipment this frame represents. The other slots determine other characteristics of the tank instance such as:

1) where it is located (the belongs-to slot)

2) its description (the equip-desc slot)

3) its pressure (the pressure slot)

4) its state (the state slot)

There are restrictions on the expansion of the non-terminals associated these slots. These restrictions serve to give an accurate description of any physical instance of a tank. Note that other properties can be added to the tank specification by introducing new slots. Figure 23 shows Maillet's code representing these two equipment items.

```
<tank-frame> ::= (defschema <equipment-name>
   <documentation>
(instance-of tank)
(belongs-to <tank-placement>)
(equip-desc <string>)
(pressure <number>)
(state <tank-state>) )

<pump-frame> ::= (defschema <equipment-name>
   <documentation>
(instance-of pump)
(belongs-to demineralized-water-system)
(equip-desc <string>)
(state <pump-state>)
(breakers <breakers-state>) )
```

Figure 22. GENRL description of two EFTs.

2.3. Recognizer of GENRL Using Lex and Yacc

This recognizer can form an essential part of the interactive completion and verification tool (see chapter 5) for GENRL.

Appendix 3 and Appendix 4 give the Lex and Yacc specifications to recognize OMFTs specified by the grammar in Appendix 1. The process of producing an executable program is depicted in Figure 18.

```
(defschema TK1
   "system tank 1"
   (instance-of tank)
   (belongs-to helium-cover-gas-system)
   (equip-desc "Delay Tank")
   (pressure 80)
   (state open))

(defschema P1
   "system pump 1"
   (instance-of pump)
   (belongs-to demineralized-water-system)
   (equip-desc "System Pump 1")
   (state off)
   (breakers closed))
```

Figure 23. Sample ART code for two EFTs [Maillet 90].

```
PARSING STATISTICS:
Lines parsed 200  Schemas parsed 31
SCHEMA BREAKUP:
Requirement 3
Requirementor 1
Select 0
Constraint 0
Findequipment 0
Cleanupfind 0
Defcontext 8
Display 11
Displayfigure 0
Connect 7
Connectreturn 0
Condconnect 0
Condconnector 1
Questionconnect 0
Cleanupselected 0
Holdcontext 0
Clearcontext 0
Wait 0
Error schemas: 0
END OF STATISTICS
```

Figure 24. Parsing statistics for OMFTs.

The output obtained on running this executable program on part of Maillet's ART code is shown in Figure 24. The output obtained clearly shows the breakdown of recognized frames. It also shows there were no errors detected in the GENRL code. Error recovery provided by Yacc is implemented. Whenever an error is encountered the C routine `yyerror()` is invoked. Normal parsing of OMFTs will continue after three valid tokens representing the beginning of an OMFT are recognized. This feature prevents multiple invocations of `yyerror()`. This routine prints the line number in which the error occurred and the frame where the error occurred. The lookahead token present when the error occurred is also printed.

Figure 25 shows the output obtained on running the executable program on ART code produced by GENT (see section 3.3). The parser completely recovered from the error on line 4771 in frame 687.

The recognizer for EFTs was implemented in a similar fashion as OMFTs. Appendix 5 and Appendix 6 give the Lex and Yacc specification to recognize valid EFTs according to the grammar given in Appendix 2. The output obtained from a run of the executable program is shown in Figure 26. This output gives the breakdown of all valid equipment item frames recognized. Once again error recovery is available, but there were no errors in this example.

```
a.out: syntax error near  line 4771
near schema 687
lookahead token 40
lookahead token (
PARSING STATISTICS:
Lines parsed 8060  Schemas parsed 1125
SCHEMA BREAKUP:
Requirement 116
Requirementor 0
Select 0
Constraint 0
Findequipment 0
Cleanupfind 0
Defcontext 247
Display 494
Displayfigure 0
Connect 267
Connectreturn 0
Condconnect 0
Condconnector 0
Questionconnect 0
Cleanupselected 0
Holdcontext 0
Clearcontext 0
Wait 0
Error schemas: 1
END OF STATISTICS
```

Figure 25. Parsing statistics for GENT code.

```
PARSING STATISTICS:
Lines parsed 737  Schemas parsed 103
SCHEMA BREAKUP:
Tanks 2
Pumps 3
Compressors 2
Heaters 1
Valves 87
Ionexchanges 2
Pressure-indicators 5
Handswitchs 1
Error schemas 0
END OF STATISTICS
```

Figure 26. Parsing statistics for EFTs.

3. Translation of Operating Manual Text to GENRL

3.1. Structure of Operating Manuals

NB Power has published a document entitled "How to Write a Power Plant Operating Manual, RD-01364-P2" [Johnson 88] which describes in detail the structure and content of a plant operating manual. Plant Operating Manuals (OMs) contain instructions for:

1) Normal operations for start-up, running and shutdown

2) Abnormal events which are foreseeable and appropriate operator action

3) Additional information to permit system monitoring, trouble shooting, and where appropriate, corrective action

Generally there is one manual for each independent plant system. There can be many manuals for one system if the system is very complex, or one manual for many systems if each system is not complex and possibly interrelated. Figure 27 shows the major parts of a manual. A description of each chapter follows [Johnson 88].

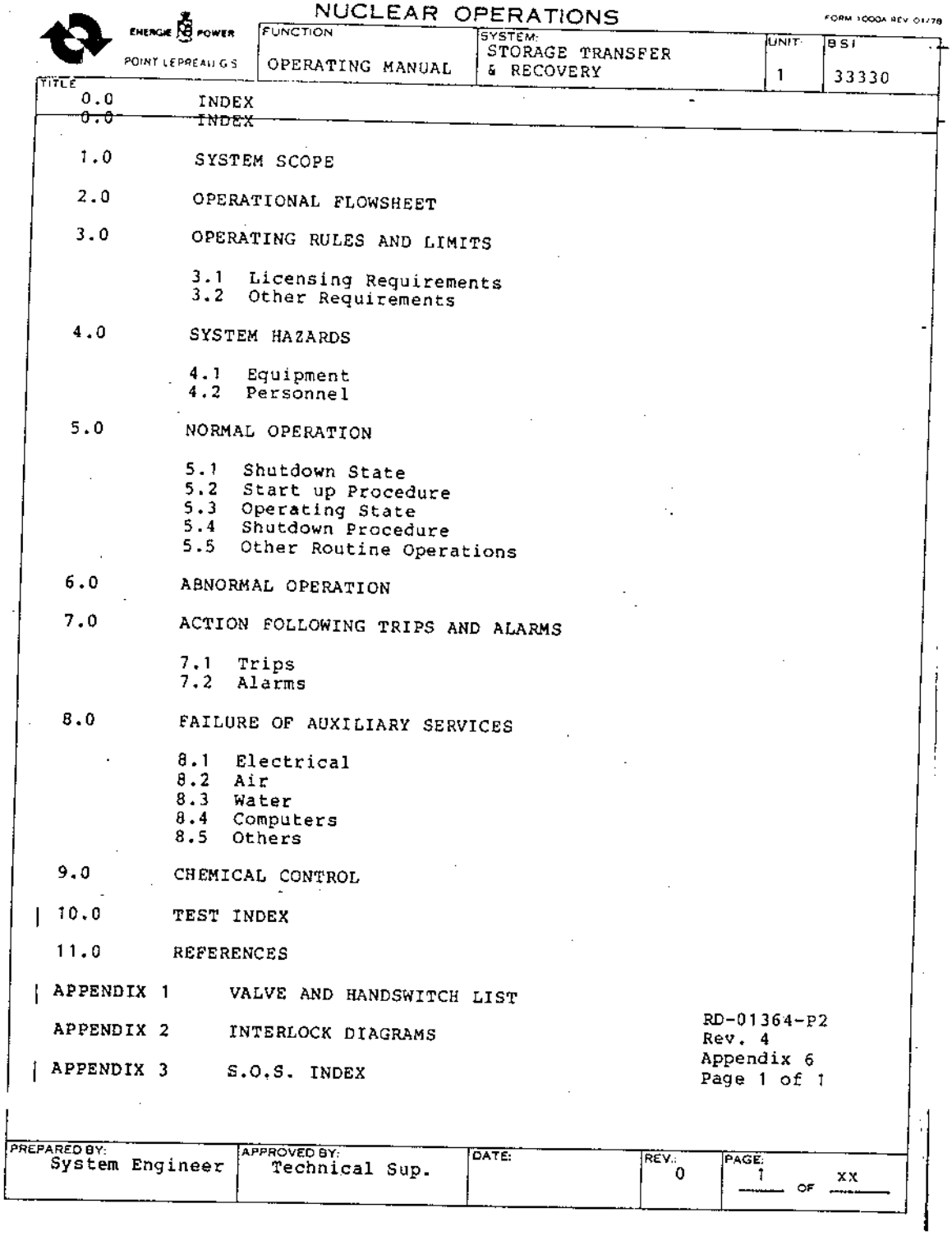
Chapter 1 (System Scope) gives a short description of the system covered by the operating manual. The purpose of this chapter is to ensure there is no confusion as to which system is covered by this manual.

Chapter 2 (Operational Flow sheet) contains references to the applicable system flow sheets, location diagrams, and control schematics.

Chapter 3 (Operating Rules and Limits) is divided into two groups. The first group (Licensing Requirements) contains operating rules and limits which guarantee compliance with operating licenses and are repeated verbatim from the original document. The second group (Other Requirements) consists of rules and limits that may relate to NB Power practice, personnel safety, economic considerations, etc.

Chapter 4 (System Hazards) contains a brief list of the primary personnel and equipment dangers associated with the system.

Chapter 5 (Normal Operation) includes routine procedures (e.g., Start up procedure).



NUCLEAR OPERATIONS

ENERGIE POWER
POINT LEPREAU G S

FORM 1000A REV 01/78

FUNCTION	SYSTEM:	UNIT:	BSI
OPERATING MANUAL	STORAGE TRANSFER & RECOVERY	1	33330

TITLE

0.0 INDEX
~~0.0 INDEX~~

1.0 SYSTEM SCOPE

2.0 OPERATIONAL FLOWSHEET

3.0 OPERATING RULES AND LIMITS

- 3.1 Licensing Requirements
- 3.2 Other Requirements

4.0 SYSTEM HAZARDS

- 4.1 Equipment
- 4.2 Personnel

5.0 NORMAL OPERATION

- 5.1 Shutdown State
- 5.2 Start up Procedure
- 5.3 Operating State
- 5.4 Shutdown Procedure
- 5.5 Other Routine Operations

6.0 ABNORMAL OPERATION

7.0 ACTION FOLLOWING TRIPS AND ALARMS

- 7.1 Trips
- 7.2 Alarms

8.0 FAILURE OF AUXILIARY SERVICES

- 8.1 Electrical
- 8.2 Air
- 8.3 Water
- 8.4 Computers
- 8.5 Others

9.0 CHEMICAL CONTROL

10.0 TEST INDEX

11.0 REFERENCES

APPENDIX 1 VALVE AND HANDSWITCH LIST

APPENDIX 2 INTERLOCK DIAGRAMS

APPENDIX 3 S.O.S. INDEX

RD-01364-P2
Rev. 4
Appendix 6
Page 1 of 1

PREPARED BY:	APPROVED BY:	DATE:	REV.:	PAGE:
System Engineer	Technical Sup.		0	1 OF XX

Figure 27. Major parts of a manual containing operating procedures [Johnson 88].

Chapter 6 (Abnormal Operation) describes procedures to correct abnormal functioning of the system.

Chapter 7 (Action Following Trips and Alarms) contains both automatic action and recommended operator action following a computer annunciated alarm. Alarms are listed by priority

Chapter 8 (Failure of Auxiliary Services) describes equipment response to a failure of one of the following auxiliary services: electrical, instrument air, water, computers, and other. Required operator action is not covered here.

Chapter 9 (Chemical Control) identifies system parts requiring chemical control.

Chapter 10 (Test Index) contains only an index to the relevant system tests. The procedures for all regularly performed tests are in the operational testing manual.

Chapter 11 (References) gives a list of reference material.

The liquid zone control manual was chosen as the manual to be used for computerization because it was proposed by Point Lepreau staff as being representative of power plant operating manuals in general [Maillet 90]. Maillet computerized chapters 5 and 6 of the manual. The same material was chosen to be translated in this thesis because a comparison could be made between the knowledge base encoded by Maillet and the knowledge base obtained by the GEN*RL* Knowledge Acquisition Tool (GENKAT) (see section 3.3). However, an accurate comparison could not be made because the manual used for the translation had changed since Maillet used it to develop EFTs and OMFTs.

3.2. Structure of Text Used for Translation

The input text can be considered to consist of chapters, sections, subsections and steps. The relationship between these components is shown in Figure 28. A typical character sequence indicates the beginning of any of these components. Table 2 shows the tokens (see section 3.4) that indicate the start of these components. The previous component also ends at this point. The contents of textual paragraphs within these components require natural language textual analysis to extract information from them.

3.3. Overall Architecture of GENRL Knowledge Acquisition Tool

The first step of the knowledge acquisition process involves recognizing an instance of a generic task present in the WordPerfect file. On recognition, the declarative knowledge of the generic task has to be acquired from this file. This is done by filling the slots of the frame associated with the generic task. This is what the GEN*RL* Translation tool (GENT) does.

Table 2. Tokens indicating the start of major components of input text.

Token name	Component
CHAPTERTAG	Chapter
SECTIONTAG	Section
SUBSECTIONTAG	Subsection
STEPTAG	Step

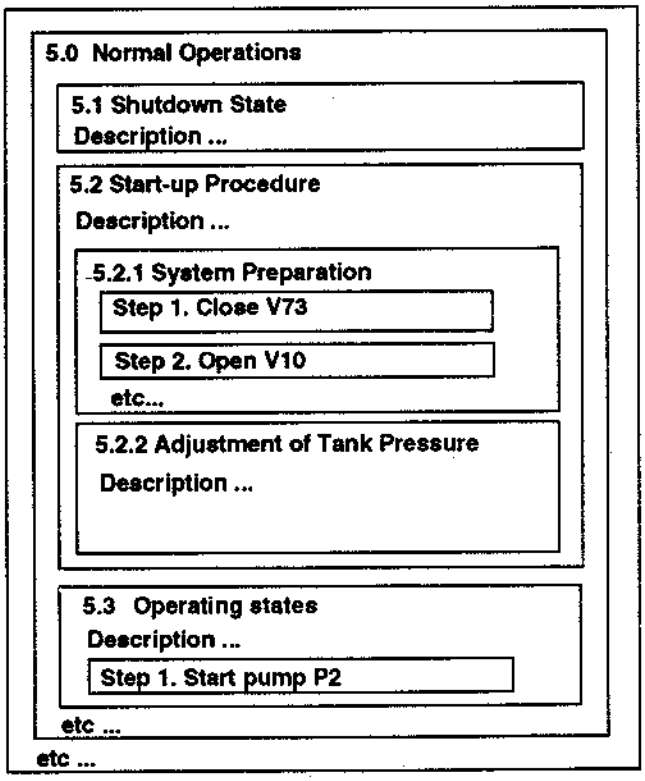


Figure 28. Example components of input text, and their relationships.

Figure 29 shows the overall architecture of GEN*RL* Knowledge Acquisition Tool (GENKAT). GENT and the design of GEN*RL* Interactive COmpletion and VErification tool (GENICOVE) will be explained in the following sections.

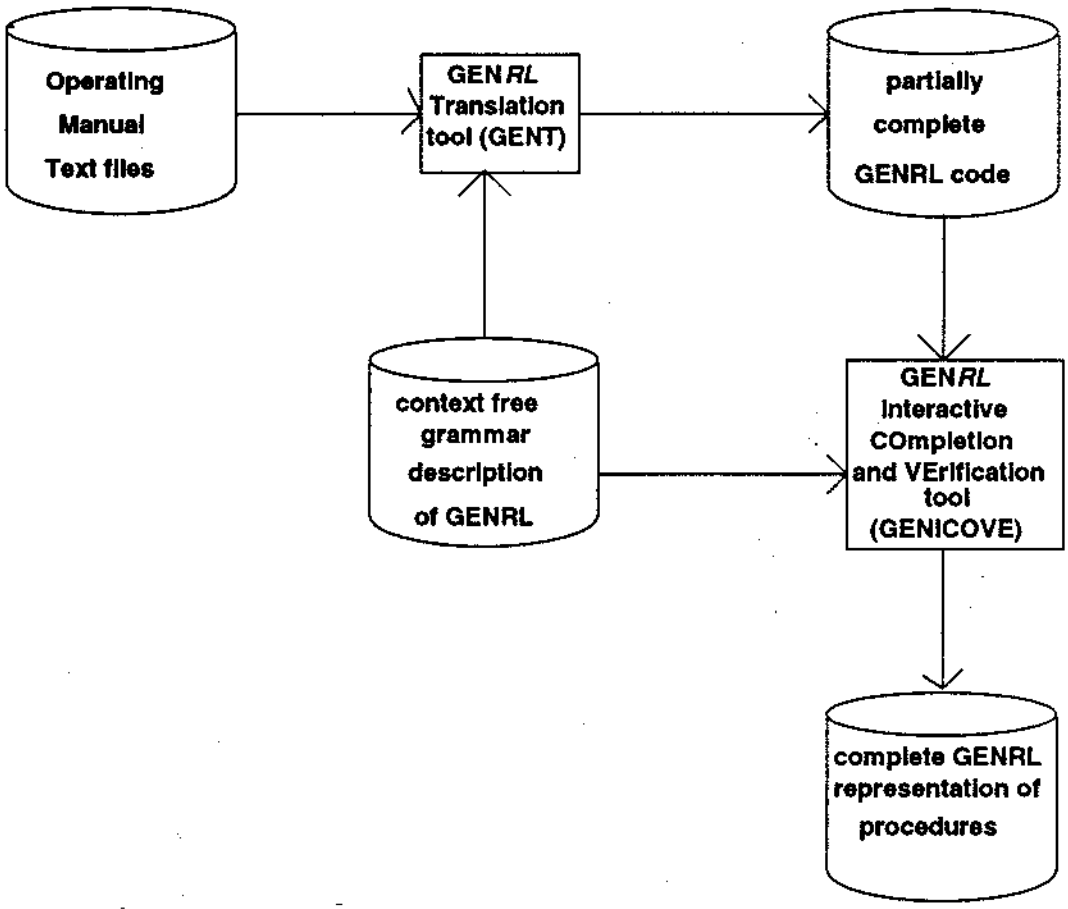


Figure 29. Architecture of GENKAT.

3.4. Important Tokens

The tokens recognized in the translation process can be seen in the Yacc specification for the input text in Appendix 8. A short description of each of the tokens follows. An instance of each token is included with the description. These instances appear in Figure 28. The recognition of these tokens in input text is accomplished by the Lex specification shown in Appendix 7.

1. CHAPTERTAG: This is the chapter number (e.g., 5.0).

2. CHAPTERHEADER: This is the chapter heading that follows a chapter number (e.g., Normal Operations that follows 5.0).

3. SECTIONTAG: This is the section number within the chapter (e.g., 5.1).

4. SUBSECTIONTAG: This is the subsection number within a section (e.g., 5.2.1).

5. HEADER: This is the heading following a section number or subsection number (e.g., Shutdown State after 5.1).

6. STEPTAG: This consists of the word "Step" followed by the step number within a section or subsection (e.g., Step 1 in subsection 5.2.1).

7. PARAGRAPH: This token follows a section heading (token HEADER), a subsection heading (token HEADER), or a step number (token STEPTAG) (e.g., The text between 5.1 Shutdown State and 5.2 Start-up Procedure). The end of a paragraph token is recognized when a new chapter, section, subsection or step begins. This happens when tokens with a "TAG" suffix are recognized.

3.5. The Parsing Process

This process consists of three stages. They are:

1) Preprocessing

2) Breakdown into main components

3) Rescanning

These stages seem to be the most natural way to divide the problem considering the structure of input text. The complexity of processing increases from the former stages to the latter stages. Figure 30 gives an overview of the three stages together with the inputs and outputs files involved in the processing of input text.

3.5.1. Preprocessing

This stage eliminates unwanted text and also makes some changes to the text considering ART syntax. The following is done during this stage:

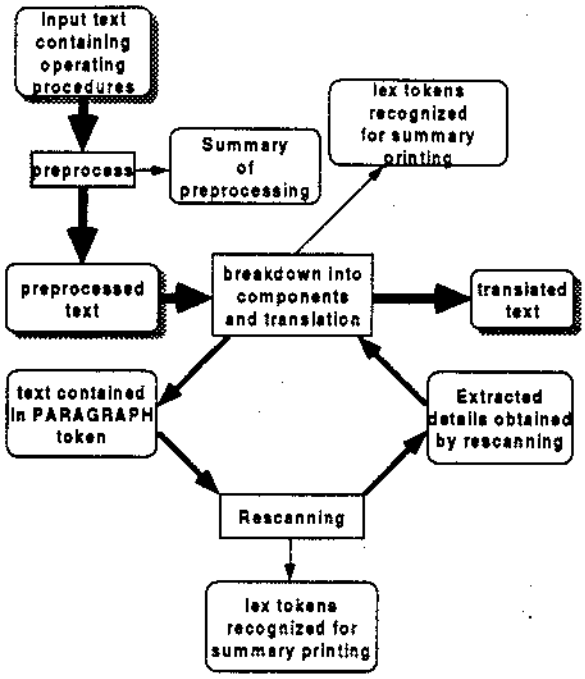


Figure 30. Overall architecture of GENT.

1. All carriage return (^M in vi, \r in UNIX, 015 as an octal number) characters are eliminated.

2. All formfeeds (^L in vi, \f in UNIX, 014 as an octal number) that represent beginning of pages are replaced by two newline (not displayed in vi, \n in UNIX, 012 in octal) characters. This will make newline characters the only line delimiters in the text.

3. All lines containing blanks are converted to null lines. This makes delimiters between the main components (chapters, sections, subsections, and steps) of the input text two or more newline characters.

4. All double quote (") characters are prefixed by the backslash (\) character. This conversion is required because of the ART use of double quotes as string delimiters. The backslash allows the double quote character to be included in strings.

5. All lines present at the beginning of a page (after the formfeed character) which indicate continuation of a chapter, section, or subsection are deleted. The

Lex specification that deletes these lines is shown in Appendix 10. All deleted lines are printed to a diagnostic message file. A warning message is printed when a chapter, section, or subsection number is out of sequence. An out of sequence major component number (tokens CHAPTERTAG, SECTIONTAG, and SUBSECTIONTAG) causes the breakdown into components stage not to recognize the end of a chapter, section, or subsection. This major component number is considered part of the present component.

The UNIX shell script responsible for this preprocessing is shown in Figure 31. The executable deleteheadings program present in Figure 31 is obtained from the Lex specification of Appendix 10.

```
# preprocessor for operating plant
  procedures
# Author : Joozar Vasi
# Date : 29/6/92
#

sed 's/^M//g' $* |
sed 's/^L/\
\
/g' |
sed 's/^[ ][ ]*$//' |
sed 's/\"/\\\"/g' |
deleteheadings
```

Figure 31. UNIX preprocessor for input text as seen in the vi editor.

3.5.2. Breakdown into components

This stage divides the text of the liquid zone control manual into its main components. These components were described in section 3.2. Lex and Yacc were used to perform the processing following the same scheme depicted in Figure 18. The generic tasks of display-current-context, display-text, and link-context are recognized during this stage.

The structure of the text as seen in Figure 28 is reflected in the top level Yacc rules shown in Figure 32. The complete Yacc specification used during this stage can be found in Appendix 8. These rules form a LALR(1) grammar since Yacc does not produce any shift-reduce or reduce-reduce conflicts. The relationship between the tokens specified in capital letters in the Yacc grammar and input text can be found in section 3.4.

```
chapters : chapter
   | chapters chapter
   ;
chapter : CHAPTERTAG CHAPTERHEADER sections
   ;
sections : section
   | sections section
   ;
section : SECTIONTAG HEADER PARAGRAPH
   | SECTIONTAG HEADER PARAGRAPH subsections
   | SECTIONTAG HEADER subsections
   | SECTIONTAG HEADER PARAGRAPH steps
   | SECTIONTAG HEADER PARAGRAPH steps subsections
   ;
subsections : subsection
   | subsections subsection
   ;
subsection : SUBSECTIONTAG HEADER PARAGRAPH
   | SUBSECTIONTAG HEADER steps
   | SUBSECTIONTAG HEADER PARAGRAPH steps
   ;
steps : STEPTAG PARAGRAPH
   | steps STEPTAG PARAGRAPH
```

Figure 32. Yacc specification for input text (truncated).

When a Yacc rule is recognized all the inherited attributes [Aho 86] associated with tokens can be accessed by the C code associated with the rules (see Appendix 8). These attributes are provided by Yacc for semantic analysis. All the tokens with a "TAG" suffix have an attribute that is a number. All other tokens have an attribute which is a string. The content of these attributes is determined by `yylex()` and is passed to `yyparse()` by the global variable `yylval`.

In the rescanning stage the inherited attribute associated with each PARAGRAPH token is rescanned and any information obtained is stored in a data structure. A detailed explanation of this process is given in section 3.5.3.

After completion of this stage and the rescanning stage all the knowledge of the input text is stored in a data structure (see chapter 4). This data structure contains information of an entire section. On successful recognition of a section, the content of this data structure is printed according to GENRL syntax.

3.5.3. Rescanning

This stage involves the recognition of generic tasks other than ones recognized during stage 2. The string attribute of a PARAGRAPH token has to be rescanned to extract

information about other generic tasks. The string attribute contains English sentences. Accordingly, Natural Language Processing (NLP) methods have to be use to analyze the text and extract information from them.

The following method was used to gather information for a requirement frame (generic task of equipment-state-requirement-with-and-logic):

The string attribute of the PARAGRAPH token is scanned line by line. The end of the line is indicated by a period followed by a blank or a newline character. One line of the PARAGRAPH token is scanned to determine if it contains any equipment items which are known in advance. These equipment items can be obtained by considering the components of the liquid zone control system. The presence of other words such as "off" is also detected while searching for equipment items. These words help in obtaining settings of equipment items. If some information cannot be obtained regarding equipment settings, then default information is inserted. When the end of a line is reached the process of scanning the next line begins.

The Lex specification that uses the above method is given in Appendix 9. Important words are only recognized if they are preceded and succeeded by non-alphanumeric characters. This rescanning is invoked by a call to the program *equipparse*. Places where this invocation takes place can be seen in the Yacc specification of Appendix 8.

Figure 33 shows contents of input and output files after the rescanning stage of GENT is completed. The strings ***state*** and **"="** are present in the output file since no information from input text can be obtained for these slots for valves, V73, V78 and V103. These are the default values associated with all equipment items. An equipment number is added to the beginning since it is necessary for the requirement frame. The word "closing" is recognized by the Lex specification and the word "closed" is printed in the equipment state slot.

There are shortcomings of the above method that essentially uses a FSTN with memory to extract information. They are:

1) The Lex specification has a limited vocabulary. In Figure 34 important words like "if", "then" and "otherwise" are not recognized.

2) Information peculiar to certain equipment items is not differentiated in the specification. Equipment items can be stored as objects. When a reference is

made to them the valid settings or default settings can be retrieved from the object base. The contextual/world knowledge information is not complete in the Lex specification.

```
Input file:

Step 6. Isolate the balance header by closing
   valves V73, V78, and V103.

Output file:

1 V73 state "=" closed
2 V78 state "=" closed
3 V103 state "=" closed
```

Figure 33. Input and Output files of rescanning stage.

```
If V73 is closed, then open V72; otherwise open valve V48
```

Figure 34. Sentence not properly interpreted by GENT.

3) The basic unit for gathering information into a data structure is a sentence. Dependencies across sentences and PARAGRAPH tokens will have to be stored in memory.

4) More information can be extracted if valid sequences of tokens are recognized. For instance, if the "if then otherwise" sequence is recognized then the frame corresponding to the appropriate generic task should be output (see Figure 34). Given the text in Figure 34, the Lex specification will infer that all the valves have to be opened. This is so because the word "open" is present at the end of the sentence. Again, this can be remedied if valid sequences of tokens are recognized.

5) When more than one generic task is recognized by the rescanning stage of the parsing process, a conflict resolution scheme will have to be devised. The Lex specification in Appendix 9 on scanning the text in Figure 34 will print the requirement frame which indicates the presence of the generic task of equipment-state-requirement-with-and-logic. This is not correct.

Some advantages of this method are:

1) It does not require a complete vocabulary of all English words because the Lex specification has the knowledge of the area of discourse. If this knowledge was not utilized, extensive storage requirements will be needed for the complete English vocabulary.

2) No complete grammar has to developed for English sentences. Problems of ambiguity and time-consuming backtracking parsing methods are avoided.

3) Variations of an important word (e.g., auto, Auto and AUTO) result in only one word being printed in the output obtained after rescanning (i.e., auto).

3.6. The Printing Process

The net result after the parsing process is a data structure. The content of this data structure is described in detail in chapter 4. According to the content of this data structure, the printing routine generates ART code. It is invoked by `yyparse()` on successful recognition of a section (see Appendix 8). For the section being recognized and each of its subsections there is a node in the linked list. For each node a def-context and two display frames are printed. A section or subsection node can have a linked list of steps attached to it. The same three frames are printed for each step present in a section or its subsections. If a linked list of equipment requirements is attached to a section, subsection, or step node, then a requirement frame is printed. Connect frames are printed as links in the data structure are traversed. Appendix 11 contains the complete code for the printing process.

3.7. An Example Translation

Figure 35 shows part of the input WordPerfect document and the file obtained on preprocessing this part of the input document. Step 6 seen in Figure 35 is present in chapter 5, section 5.1, and subsection 5.2.1. Figure 36 shows the data structure which is created on completion of stages 2 and 3 of the parsing process. Figure 37 shows the knowledge base created on completion of the printing process.

Step 7 follows Step 6. Therefore a final connect frame to step 7 is printed. This frame corresponds to the generic task of link-context. If the token SUBSECTIONTAG is found after Step 6 (i.e., there was no step 7), then the final frame printed will be a display frame (generic task of display-text) indicating the end of the subsection.

```
Input WordPerfect text:

^M
Step 6.  Isolate the balance header by closing
   valves V73, V78,^M
and V103.^M
^M

Preprocessed output file:

Step 6.  Isolate the balance header by closing
   valves V73, V78,
and V103.
```

Figure 35. Input WordPerfect text and preprocessed output file as seen by the vi editor.

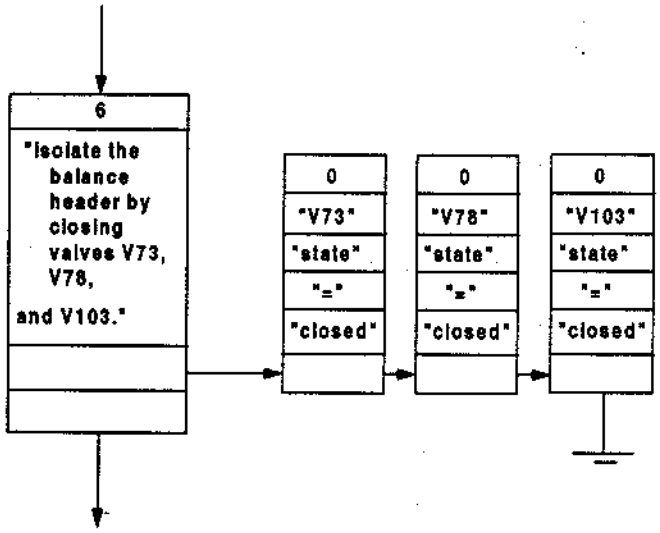


Figure 36. Data structure for sample translation.

```
(defschema context-5-2-1-Step-6
     (instance-of def-context)
     (context 5-2-1-Step-6)
     (header "5.2.1 Realign System Components For Start-Up")
     (desc "Step 6. Isolate the balance header by closing
     valves V73, V78, and V103. "))


(defschema display-5-2-1-Step-6-a
     (instance-of display)
     (context (5-2-1-Step-6 1))
     (header "5.2.1 Realign System Components For Start-Up")
     (desc "Proceed with 5.2.1 Step 6 Verification."))


(defschema req-5-2-1-Step-6
     (instance-of requirement)
     (context (5-2-1-Step-6 2))
     (equipment-list (1 V73 state "=" closed) (2 V78 state
"="  closed) (3 V103 state "=" closed)))


(defschema display-5-2-1-Step-6
     (instance-of display)
     (context (5-2-1-Step-6 3))
     (header "5.2.1 Realign System Components For Start-Up")
     (proc-stream "Completed")
     (desc "5.2.1 Step 6 completed"))


(defschema connect-5-2-1-Step-6-a
     (instance-of connect)
     (context (5-2-1-Step-6 4))
     (go-context (5-2-1-Step-7 1)))
```

Figure 37. The ART knowledge base obtained from input WordPerfect text of Figure 35.

4. The Implementation of GENT

GENT was developed on the SUN™ Sparcstation 2 running SunOS Release 4.1.1. OpenWindows™ Version 2 was also used to enter, test and run GENT. GENT utilizes UNIX utilities Sed, Lex and Yacc [Pike 84], and the C programming language [Ritchie 78]. The GENT parsing process consists of 1100 lines of code; the GENT printing process consists of 370 lines of code; the supporting C routines for the parsing process consist of 439 lines of code. The common header file used by all files in GENT consists of 66 lines of code. Table 3 gives the breakdown of files of GENT among the various file systems mentioned above.

Table 3. GENT file system.

File system	File sub-system	Files	File type
GENT parsing process	Preprocessing	deleteheadings.l preprocessor	Lex specification UNIX shell script
	Breakdown into components	main.c lzcman.l lzcman.y	C code Lex specification Yacc specification
	Rescanning	main1.c eqiupreq.l	C code Lex specification
GENT Printing process		printsect.lib.c print.lib.c	C code C code
Supporting C routine for GENT parsing process		utilities.lib.c equiputilities.lib.c	C code C code
Header file for GENT		header.h	C code

The GENT parsing process and the GENT printing process are explained in sections 3.5 and 3.6 respectively. The data structure used to gather information during the breakdown into components and rescanning stages of the parsing process is shown in Figure 38. Information of a complete section is held in it. The data structure used to store information about step details appears in Figure 39 and the data structure for equipment details appears in Figure 40. The header file (*header.h*) that contains the C description of these data structures is present in Appendix 12.

Appendix 13 contains the code for supporting C routines of GENT. Appendix 14 contains the driver program for the breakdown into components stage of the parsing process (*main.c*) and the driver program for the rescanning stage of the parsing process (*main1.c*).

GENT translated the complete content (2209 lines) of Chapter 5 (Normal Operations) and Chapter 6 (Abnormal Operations) of the liquid zone control manual [Johnson 91]. This consists of 75759 characters. The final outcome of the translation was 8059 lines of GENRL code consisting of 1125 frames.

One error was detected by the non-interactive syntax checker of GENRL on line 4771 in the generated GENRL code for chapters 5 and 6 (see section 2.3). This error occurred because a requirement frame was not properly printed due to wrong information present in the data structure containing equipment details (see Figure 40). This data structure has character pointers (pointers equipment attribute, equipment operator, and equipment value) that point to strings which describe equipment details. These pointers were set to "null" instead of the default values associated with the equipment items (see section 3.5.3). The GENICOVE tool discussed in chapter 5 could be used to fill in the correct information.

The time taken (the real time of the UNIX time command) to run the preprocessor of GENT (see Figure 31) was 2.8 seconds, and the time taken to run the command to perform the rest of the translation was 23.7 seconds.

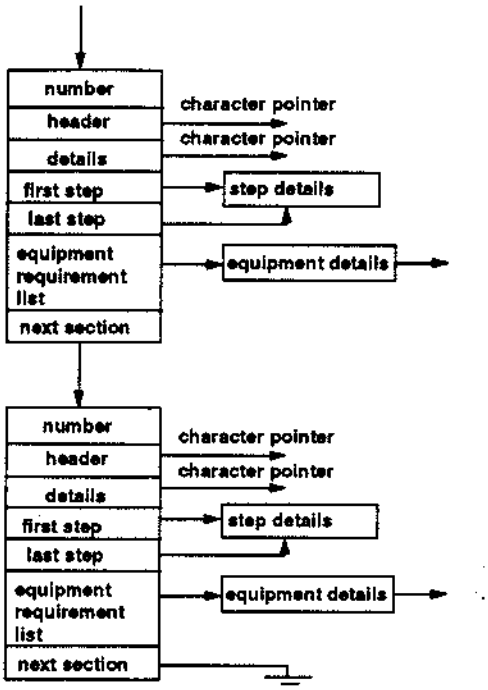


Figure 38. Data structure for storing section details.

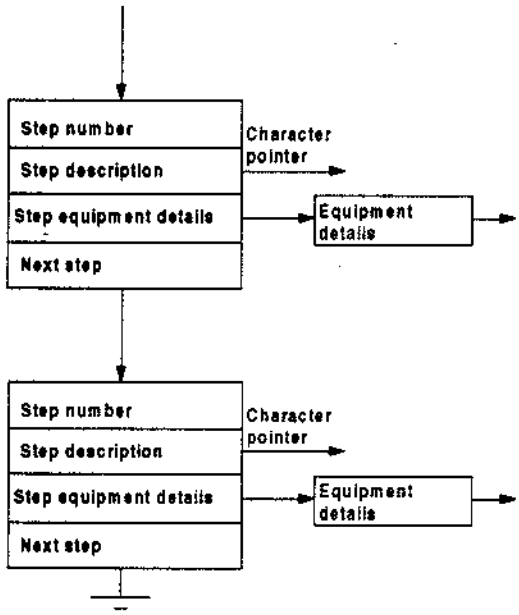


Figure 39. Data structure for step details.

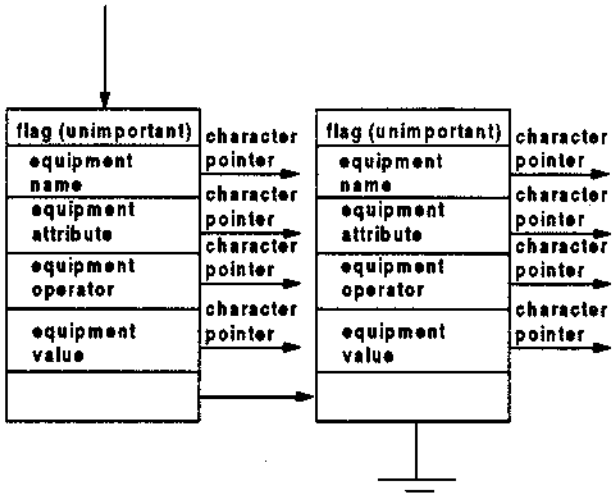


Figure 40. Data structure for equipment details.

5. Interactive Completion and Verification Tool

The screen layout of GENICOVE (GEN*RL* Interactive COmpletion and VErification tool) is shown in Figure 41. This tool provides the capabilities of being able to see translated text directly alongside original text. It provides windows for

1) Preprocessed operating procedures text,

2) Partially complete GENRL knowledge base, and

3) Interactive prompts from GENICOVE.

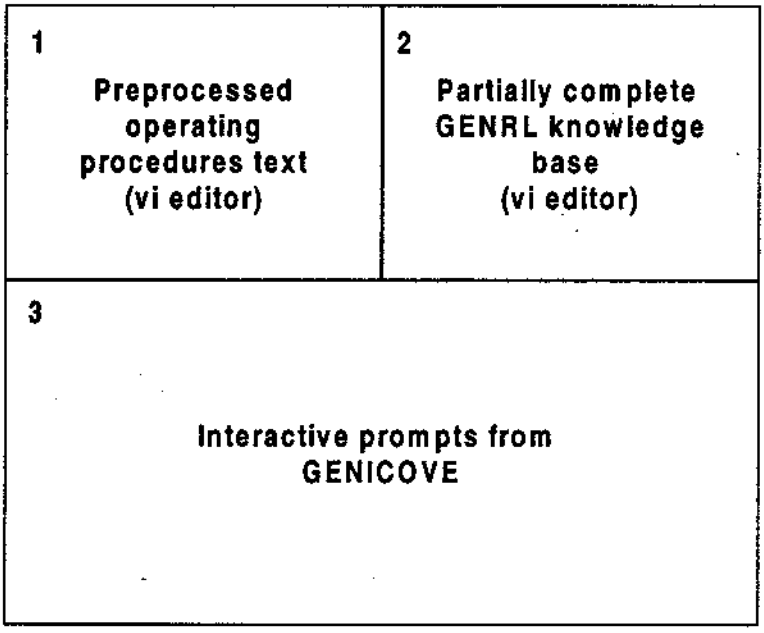


Figure 41. Proposed screen layout for GENICOVE.

There is interaction between windows 1 and 2 and between windows 2 and 3. GENICOVE works as follows:

1) The preprocessed operating procedure text is loaded into window 1. A sample of this text appears in Figure 35.

2) The partially complete GENRL knowledge base (see Figure 37) is loaded into window 2.

3) GENICOVE scans the partially complete knowledge base in window 2 until it finds an error. The scanning is interrupted and the cursor is automatically placed on the line where the error occurred.

4) Window 1 is updated to reflect the location of error in the preprocessed operating procedure text. Links exist from the partially complete GENRL knowledge base to the preprocessed text, i.e., from window 2 to window 1. These links can be established by GENT during the translation.

5) The user makes corrections based on the error message in window 3.

6) Parsing continues from the start of the OMFT or EFT which caused the error.

The GENRL grammar (see sections 2.1 and 2.2) provides the basis for a syntax checker (see section 2.3) that scans EFTs and OMFTs, and flags entries which are incomplete or incorrect. This is only part of the error checking that GENICOVE must be capable of. It should be able to track down semantic errors as well as errors in procedure logic in EFTs and OMFTs.

A common semantic error that can occur is the recognition of a wrong generic task. A frame associated with the generic task of equipment-state-requirement-with-and-logic may be printed when the frame associated with equipment-state-requirement-with-or-logic has to be printed. Checks for duplicate frame names can also be included in GENICOVE. ART deletes previous frames with the same name in a knowledge base. This is normally not wanted.

Some of the logical errors typical of plant operating procedures are [Horne 89]:

1) Compliance with Explicit Specification: This test assumes that the plant's final state at the end of the procedure is well-defined. In this test a search is made to determine if any prior plant states along with its procedure steps required to arrive at the final state violate the procedure's final state specification.

2) Consistency: This checks whether concurrent active procedures do not require plant settings which are incompatible. For example, one procedure requires a valve to be closed, and a concurrent procedure requires it to be open.

3) Compliance with constraints: This test searches states consistent with procedures but inconsistent with constraints specified in technical specifications of procedures.

4) Acceptability of Operator Information Rate: This checks whether the rate of information processing warranted by procedures is well within reasonable limits for human operators.

5) Absence of Cycles: A cycle occurs if the final state of one procedure is the entry state of another and conversely.

6) Existence of instrumentation: Checks are made to see that reference to instrumentation is valid for a particular system.

7) Agreement in State: State names are valid for certain components. They should refer to the right component. For example, a pump being open is invalid; it can only be on or off.

8) Proof of Coverage: Undesirable plant states should be generated and abnormal state operating procedures should be simulated to check whether the plant has reached a normal state.

Detecting some errors in procedure logic (e.g., Proof of Coverage) requires a complete computerized model of the plant system for which the operating procedure applies.

Use of the vi editor is a simple way of providing the user with a tool to modify the translated text as necessary in window 2. Window 1 runs vi in read-only mode. Use of interactive prompts from the scanning tool provides a method of verifying that the translated code is free of syntactic, semantic and procedure logic errors.

Alternatively, a user can run GENICOVE in batch mode on a complete selection. The user can select one or more frames from window 2. These frames will be checked for errors. This use of GENICOVE has the possible disadvantage of producing many errors when only one error has occurred. This is typical of other translation tools such as compilers.

6. Summary

GENRL, a context-free language for plant operating procedures, was developed based on a knowledge base consisting of frames [Maillet 90]. A syntax checker for GENRL using Lex and Yacc was developed. A knowledge acquisition tool (GENKAT) for GENRL was designed and partially implemented. GENKAT accepts WordPerfect operating procedure text and generates GENRL frames. GENKAT consists of two tools, namely GENT and GENICOVE. GEN*RL* Translation tool (GENT) is responsible for producing a partially complete GENRL knowledge base of frames. These frames are used by the GEN*RL* interactive COmpletion and VErification tool (GENICOVE) to produce computerized plant operating procedures. The present implementation of GENT does not result in any loss of input text because the all input text appears in the frame that indicates the beginning of a new section, subsection or step (the def-context frame).

Presently GENT is capable of recognizing four generic tasks [Maillet 90]. It has to be enhanced to recognize all fourteen generic tasks. This may require more sophisticated NLP methods [Gazdar 89] than the one used for extracting details for a requirement frame. This frame corresponds to the generic task of equipment-state-requirement-with-and-logic.

GENT translated 2209 lines of WordPerfect text to produce 8059 lines of GENRL code consisting of 1125 frames. These lines represent approximately half of the entire liquid zone control manual. The bulk of the information representing operating procedures is present in this part of the manual (i.e., chapters 5 and 6) together with chapters 7 and 8 (see Figure 27).

Equipment frames could be generated using equipment lists (e.g., Appendix 1 Valve and Handswitch List in Figure 27). Information from other parts of the manual (other than chapters 5, 6, 7, and 8) should be inserted in the computerized operating procedures. These parts normally represent information that can be used as annotations to procedures. It is also possible to use the GENT approach to capture textual descriptions in these chapters.

The recall and precision percentages [Lehnert 91] of GENT have yet to be obtained. These numbers give a measure of how accurately GENT has translated the text to GENRL. Recall and precision percentages will also determine how much work has to be

done by GENICOVE to obtain a knowledge base free of errors in syntax, semantics, and procedure logic.

GENICOVE has been designed. Its implementation will require the following to be completed (see chapter 5):

1. The user-interface shown in Figure 41 has to be developed.

2. Links from partially complete GENRL code to preprocessed operating procedure text have to be established.

3. Errors in GENRL syntax are presently recognized non-interactively by Lex and Yacc (see section 2.3). This parser has to be enhanced to make it interactive and to include checks for semantic errors and errors in procedure logic.

7. References

[Aho 86] A. Aho, R. Sethi, and J. Ullman . Compilers, Principles, Techniques, and Tools. Addison-Wesley Publishing Company, 1986.

[Bhatnagar 90] R. Bhatnagar, D. Miller, B. Hajek, and J. Stasenko. "An integrated operator advisor system for plant monitoring, procedure management, and diagnosis". Nuclear Technology 89 (3), March 1990, pages 281-317.

[Chandrasekaran 86] B. Chandrasekaran. "Generic tasks in knowledged-based reasoning: high-level building blocks for expert system design". IEEE Expert, Fall 1986, pages 23-30.

[Gazdar 89] G. Gazdar and C. Mellish. Natural Language Processing in PROLOG. Addison-Wesley Publishing Company, 1989.

[Hopcroft 79] J. Hopcroft and J. Ullman. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley Publishing Company, 1979.

[Horne 89] C. P. Horne. "Methods for testing the logical structure of plant procedure documents". Proceedings of 1989 Conference on Advanced Computer Technology for the Power Industry (Scottsdale, AZ), December 4-6, 1989, 17 pages.

[Jacobs 90] P. S. Jacobs and L.F. Rau. "SCISOR: extracting information from on-line news". ACM Comm 33 (11), November 1990, pages 88-97.

[Johnson 88] A. R. Johnson. "How to write a power plant operating manual". Point Lepreau, NB: RD-01364-P2 Rev. 4, 1988.

[Johnson 91] A. R. Johnson. "Liquid zone control". Electronic WordPerfect text of the Point Lepreau Generating Station Operating Manual 34810 , April 1991.

[Krieger 91] J. W. Krieger. "From paper to objects: improving the information value of procedures". Proceedings of Frontiers in Innovative Computing for the Nuclear Industry (Jackson, WY), September 15-18, 1991, pages 566-574.

[Krogsæter 89] M. Krogsæter, J. Larsen, S. Nilsen, and F. Owre. "The computerized procedure system COPMA and its user interface". Proceedings of the IAEA Specialist Meeting on Artificial Intelligence in Nuclear Power Plants (Helsinki, Finland), October 10-12, 1989, pages 1-13.

[Lehnert 91] W. Lehnert and B. Sundheim. "A performance evaluation of text-analysis technologies". AI Magazine, Fall 1991, pages 81-94.

[Lipner 91] M. H. Lipner and R. G. Orendi. "Issues involved with computerizing emergency operating procedures". Proceedings of Frontiers in Innovative Computing for the Nuclear Industry (Jackson, WY), September 15-18, 1991, pages 556-565.

[Luger 89] G. Luger and W. Stubblefield. Artificial Intelligence and the Design of Expert Systems. Benjamin/Cummings Publishing Company, Inc., 1989.

[Maillet 90] G. Maillet. "Knowledge Representation for a Power Plant Interactive Advisor", Technical Report TR90-050, Faculty of Computer Science, UNB, Fredericton, N.B., July 1990, 114 pages.

[Mason 91] T. Mason and D. Brown. lex & yacc. O'Reilly & Associates, Inc., January 1991.

[Nelson 90] W. Nelson, N. Fordestrommen, C. Holmstrom, M. Krogsæter, T. Karstad, and O. Tunold, "Experimental evaluation of the computerized procedure system COPMA: preliminary results". Presented at the Enlarged Halden Programme Group Meeting on Computerised Man-Machine Communication (Bolkesjo, Norway), February 11-16, 1990, pages 1-22.

[Parker 87] D. Parker and S. LeClair. "Liquid zone control". Point Lepreau, NB: Point Lepreau Generating Station Operating Manual 34810 Rev. 4/8, December 17, 1987.

[Pike 84] R. Pike and B. Kernighan. The UNIX Programming Environment. Prentice-Hall, Inc., 1984.

[Ritchie 78] D. Ritchie and B. Kernighan. The C programming language. Prentice-Hall, Inc., 1978.

[Robert 89] C. Robert, C. P. Horne, and J. M. Fahley. "Methods for improving the development and maintenance of plant operating procedures". Proceedings of the EPRI Conference on the Applications of

Expert Systems to the Utility Industry, June 1989, 14 pages.

[Salton 89] G. Salton and M. Smith. "On the application of syntactic methodologies in automatic text analysis". Proceedings of the 12th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (Cambridge, MA), June 25-28, 1989, pages 137-150.

[Sun 90] Sun Microsystems, Inc.. "Programmer's Overview Utilities & Libraries". Revision A of 27 March, 1990, pages 203-264.

[Sverre 89] J. Sverre, M. Krogsæter, J. Sverre, S. Nilsen, W. Nelson, and F. Owre. "Computerized procedures -- the COPMA system -- and its proposed validation program". Proceedings of the Expert Systems Applications for the Electric Power Industry (Orlando, FL), June 5-8, 1989, 13 pages.

Appendix 1. GENRL representation of OMFTs

```
<gt-knowledge-base> ::= ε | <gt-knowledge-base> <gt-frame>

<gt-frame> ::= <requirement-frame> |
                        <requirement-or-frame> |
                        <select-frame> |
                        <constraint-frame> |
                        <find-equip-frame> |
                        <cleanup-find-equip-frame> |
                        <def-context-frame> |
                        <display-frame> |
                        <display-figure-frame> |
                        <connect-frame> |
                        <connect-return-frame> |
                        <cond-connect-frame> |
                        <cond-connect-or-frame> |
                        <question-connect-return-frame> |
                        <cleanup-selected-frame> |
                        <hold-context-frame> |
                        <clear-context-frame> |
                        <wait-frame>

<requirement-frame> ::= (defschema <generic-task-name> <documentation>
                        (instance-of requirement)
                        (context (<context-name> <context-seq-no>))
                        (equipment-list <equipment-desc-seq>+))
<requirement-or-frame> ::= (defschema <generic-task-name>
                        <documentation>
                        (instance-of requirement-or)
                        (context (<context-name> <context-seq-no>))
                        (equipment-list <equipment-desc-seq>+))
<select-frame> ::= (defschema <generic-task-name> <documentation>
                        (instance-of select)
                        (context (<context-name> <context-seq-no>))
                        (equipment-list <equipment-desc-seq>+))
<constraint-frame> ::= (defschema <generic-task-name> <documentation>
                        (instance-of constraint)
                        (context (<context-name> <context-seq-no>))
                        (until-context <context-name> <context-seq-no>)
                        (equipment-id <equipment-desc>))
<find-equip-frame> ::= (defschema <generic-task-name> <documentation>
                        (instance-of find-equip)
                        (context (<context-name> <context-seq-no>))
                        (key <key-name>)
                        (properties (<equipment-prop>))
<cleanup-find-equip-frame> ::= (defschema <generic-task-name>
                        <documentation>
                        (instance-of cleanup-find-equip)
                        (context (<context-name> <context-seq-no>))
                        (key <key-name>))
<def-context-frame> ::= (defschema <generic-task-name> <documentation>
                        (instance-of def-context)
                        (context <context-name>)
                        (header <header-string>)
                        (desc <desc-string>))
```

```
<display-frame> ::= (defschema <generic-task-name> <documentation>
                        (instance-of display)
                        (context (<context-name> <context-seq-no>))
                        (header <header-string>)
                        {(proc-stream <proc-stream-string>)}
                        (desc <desc-string>))
<display-figure-frame> ::= (defschema <generic-task-name>
                        <documentation>
                        (instance-of display-figure)
                        (context (<context-name> <context-seq-no>))
                        (header <header-string>)
                        (diagram <diagram-name>))
<connect-frame> ::= (defschema <generic-task-name> <documentation>
                        (instance-of connect)
                        (context (<context-name> <context-seq-no>))
                        (go-context (<context-name> <context-seq-no>)))
<connect-return-frame> ::= (defschema <generic-task-name>
                        <documentation>
                        (instance-of connect-return)
                        (context (<context-name> <context-seq-no>))
                        (go-context (<context-name> <context-seq-no>))
                        (stay-until-context (<context-name> <context-
                        seq-no>))
                        (return-context (<context-name> <context-seq-
                        no>)))
<cond-connect-frame> ::= (defschema <generic-task-name> <documentation>
                        (instance-of cond-connect)
                        (context (<context-name> <context-seq-no>))
                        (equipment-list <equipment-desc-seq>+)
                        (go-context (<context-name> <context-seq-no>))
                        (stay-until-context (<context-name> <context-
                        seq-no>)))
                        (return-context (<context-name> <context-seq-
                        no>)))
<cond-connect-or-frame> ::= (defschema <generic-task-name>
                        <documentation>
                        (instance-of cond-connect-or)
                        (context (<context-name> <context-seq-no>))
                        (equipment-list <equipment-desc-seq>+)
                        (go-context (<context-name> <context-seq-no>))
                        (stay-until-context (<context-name> <context-
                        seq-no>))
                        (return-context (<context-name> <context-seq-
                        no>)))
<question-connect-return-frame> ::= (defschema <generic-task-name>
                        <documentation>
                        (instance-of question-connect-return)
                        (context (<context-name> <context-seq-no>))
                        (question <question-string>)
                        {(go-no <go-no-symbol>)}
                        (go-context (<context-name> <context-seq-no>))
                        (stay-until-context (<context-name> <context-
                        seq-no>))
                        (return-context (<context-name> <context-seq-
                        no>)))
<cleanup-selected-frame> ::= (defschema <generic-task-name>
                        <documentation>
                        (instance-of cleanup-selected)
                        (context (<context-name> <context-seq-no>)))
<hold-context-frame> ::= (defschema <generic-task-name> <documentation>
                        (instance-of hold-context)
                        (context (<context-name> <context-seq-no>)))
```

```
<clear-context-frame> ::= (defschema <generic-task-name> <documentation>
                          (instance-of clear-context)
                          (context (<context-name> <context-seq-no>)))
<wait-frame> ::= (defschema <generic-task-name> <documentation>
                          (instance-of wait)
                          (context (<context-name> <context-seq-no>))
                          (time <time-no>))

<generic-task-name> ::= <symbol>
<documentation> ::= ε | <string>
<context-name> ::= <symbol>
<context-seq-no> ::= <digits>
<equipment-desc-seq> ::= (<equipment-seq-no> <atomic-value>+)
<equipment-desc> ::= (<atomic-value>+)
<equipment-prop> ::= (<atomic-value> <atomic-value> <atomic-value>)
<key-name> ::= <symbol>
<header-string> ::= <string>
<desc-string> ::= <string>
<proc-stream-string> ::= <string>
<diagram-name> ::= <symbol>
<question-string> ::= <string>
<go-no-symbol> ::= <symbol>
<time-no> ::= <digits>

<symbol> ::= "Any sequence of <alphas> that in not a <number> ,<digits>
                          or <reserved-word>. Also the sequence does not
                          begin with a '?','-', or '_'"
<string> ::= <string-quote><string-character><string-quote>
<digits> ::= <digit>+
<equipment-seq-no> ::= <digits>
<atomic-value> ::= <symbol> | <string> | <number> | <digits>

<alpha> ::= <alphabetic> | <digit> | _ | - | ?
<number> ::= <sign><digit>+ |
                          {<sign>}<digit>*.<digit>+ |
                          {<sign>}<digit>+E{<sign>}<digit>+ |
                          {<sign>}<digit>*.<digit>+E{<sign>}<digit>+
<reserved-word> ::= defschema |
                          requirement |
                          requirement-or |
                          select |
                          constraint |
                          find-equip |
                          cleanup-find-equip |
                          def-context |
                          display |
                          display-figure |
                          connect |
                          connect-return |
                          cond-connect |
                          cond-connect-or |
                          question-connect-return |
                          cleanup-selected |
                          hold-context |
                          clear-context |
                          wait |
                          context |
                          equipment-list |
                          until-context |
                          equipment-id |
                          key |
                          properties |
```

```
header |
                          desc |
                          proc-stream |
                          diagram |
                          go-context |
                          stay-until-context |
                          return-context |
                          question |
                          go-no |
                          time
<string-quote> ::= "The character '"'"
<string-character> ::= "Any <character>. A sequence \<character>
                          includes <character> to be <string-character>"
<digit> ::= 0|1|2|3|4|5|6|7|8|9

<sign> ::= + | -
<character> ::= "Any ASCII character"
```

Appendix 2. GENRL representation of EFTs

```
<eq-knoledge-bse> ::= ε | <eq-knowledge-base> <eq-frame>

<eq-frame> ::= <tank-frame> |
                  <pump-frame> |
                  <compressor-frame> |
                  <heater-frame> |
                  <valve-frame> |
                  <ion-exchange-frame> |
                  <pressure-indicator-frame> |
                  <hand-switch-frame>

<tank-frame> ::= (defschema <equipment-name> <documentation>
                  (instance-of tank)
                  (belongs-to <tank-placement>)
                  (equip-desc <string>)
                  (pressure <number>)
                  (state <tank-state>) )
<pump-frame> ::= (defschema <equipment-name> <documentation>
                  (instance-of pump)
                  (belongs-to demineralized-water-system)
                  (equip-desc <string>)
                  (state <pump-state>)
                  (breakers <breakers-state>) )
<compressor-frame> ::= (defschema <equipment-name> <documentation>
                  (instance-of compressor)
                  (belongs-to heluim-gas-cover-system)
                  (equip-desc <string>)
                  (state <compressor-state>)
                  (breakers <breakers-state>) )
<heater-frame> ::= (defschema <equipment-name> <documentation>
                  (instance-of heater)
                  (belongs-to helium-cover-gas-system)
                  (equip-desc <string>)
                  (state <heater-state>)
                  (breakers <breakers-state>) )
<valve-frame> ::= (defschema <equipment-name> <documentation>
                  (instance-of valve)
                  (belongs-to associated-piping)
                  (equip-desc <string>)
                  {(operation-mode <valve-operation-mode>)}
                  {(pressure <number>)}
                  {(state <valve-state>)} )
<ion-exchange-frame> ::= (defschema <equipment-name> <documentation>
                  (instance-of IEC)
                  (belongs-to demineralized-water-system)
                  (equip-desc <string>)
                  (state <ion-exchange-state>)
                  (resin fresh) )
<pressure-indicator-frame> ::= (defschema <equipment-name>
                  <documentation>
                  (instance-of pressure-indicator)
                  (belongs-to associated-piping)
                  (equip-desc <string>)
                  (pressure <number>) )
<hand-switch-frame> ::= (defschema <equipment-name> <documentation>
                  (instance-of hand-switch)
                  (belongs-to asssociated-piping)
                  (equip-desc <string>)
                  (state <string>) )
```

```
<equipment-name> ::= <symbol>
<documentation> ::= ε | <string>
<tank-placement> ::= helium-cover-gas-system |
                     demineralized-water-system
<string> ::= <string-quote><string-character>*<string-quote>
<number> ::= {<sign>}<unsigned-number>
<tank-state> ::= open | closed
<pump-state> ::= on | off
<breakers-state> ::= closed | open
<compressor-state> ::= auto | standby
<heater-state> ::= on | off
<valve-operation-mode> ::= manual | auto
<valve-state> ::= open | closed | locked-open | auto | manual
<ion-exchange-state> ::= not-isolated | isolated

<symbol> ::= "Any sequence of <alpha>s beginning with a <alphabetic>
                     character that is not a <reserved-word>"
<string-quote> ::= "The character '"'"
<string-character> ::= "Any <character>. A sequence \<character> makes
                     <character> a <string-character>"
<sign> ::= + | -
<unsigned-number> ::= <digit>+ |
                     <digit>*.<digit>+ |
                     <digit>+E{<sign>}<digit>+ |
                     <digit>*.<digit>+E{<sign>}<digit>+

<alpha> ::= <alphabetic> | <digit> | - | _
<alphabetic> ::= "A single character between 'A' and 'Z, or between 'a'
                     and 'z'"
<reserved-word> ::= defschema |
                     instance-of |
                     tank |
                     pump |
                     compressor |
                     heater |
                     valve |
                     IEC |
                     pressure-indicator |
                     hand-switch |
                     belongs-to |
                     demineralized-water-system |
                     helium-cover-gas-system |
                     associated-piping |
                     equip-desc |
                     pressure |
                     breakers |
                     state |
                     open |
                     closed |
                     on |
                     off |
                     auto |
                     standby |
                     operation-mode |
                     locked-open |
                     manual |
                     not-isolated |
                     isolated |
                     resin |
                     fresh
<character> ::= "Any ASCII character"
<digit> ::= 0|1|2|3|4|5|6|7|8|9
```

Appendix 3. Lex specification for OMFTs

```
SIGN [+-]?
ALPHABETIC [A-Za-z]
DIGIT [0-9]
ALPHA [0-9A-Za-z\-\_\?]
WS [ \b\t\r\v\f\n]
%a 5000
%o 5000
%%
{WS}        {/* delete whitespace */}
;[^\(]*           {/* delete comments from input */}
\(          {     /* detect a left parethesis */
                  return('(');
            }
\)          {     /* detect a right parenthesis */
                  return(')');
            }
\"          {     /* detect a string */
                  int c;

                  start: while ((c = input()) != '\\' && c  != '"');
                  if (c == '\\')
                        {
                        input();
                        goto start;
                        }
                  else
                        return STRING;

            }
defschema         {     return DEFSCHEMA; }
instance\-of      {     return INSTANCEOF;      }
requirement       {     return REQUIREMENT;     }
requirement-or    {     return REQUIREMENTOR;   }
select            {     return SELECT;    }
constraint        {     return CONSTRAINT;      }
find\-equip       {     return FINDEQUIP; }
cleanup\-find\-equip    {     return CLEANUPFIND;     }
def\-context      {     return DEFCONTEXT;      }
display           {     return DISPLAY;   }
display\-figure   {     return DISPLAYFIGURE;   }
connect           {     return CONNECT;   }
connect\-return   {     return CONNECTRETURN;   }
cond\-connect     {     return CONDCONNECT;     }
cond\-connect\-or {     return CONDCONNECTOR;   }
question\-connect\-return     {     return QUESTIONCONNECT; }
cleanup\-selected {     return CLEANUPSELECTED; }
hold\-context     {     return HOLDCONTEXT;     }
clear\-context    {     return CLEARCONTEXT;    }
wait        {     return WAIT;      }
context           {     return CONTEXT;   }
equipment\-list   {     return EQUIPLIST; }
until\-context    {     return UNTILCONTEXT;    }
equipment\-id     {     return EQUIPID;   }
key               {     return KEY;}
properties        {     return PROPERTIES;      }
header            {     return HEADER;    }
desc              {     return DESC;      }
proc\-stream      {     return PROCSTREAM;      }
diagram           {     return DIAGRAM;   }
```

```
go\-context         {     return GOCONTEXT; }
stay\-until\-context    {     return STAYUNTILCONTEXT;      }
return\-context     {     return RETURNCONTEXT;    }
question            {     return QUESTION;  }
go\-no              {     return GONO;      }
time                { return DIGITS; }
[+-]{DIGIT}+                |
{DIGIT}*\.{DIGIT}+          |
{DIGIT}+E{SIGN}{DIGIT}+     |
{DIGIT}*\.{DIGIT}+E{SIGN}{DIGIT}+   {     return NUMBER;}
{ALPHA}+    {     if (yytext[0] == '?')
                        return(VARIABLE);
                  else if (yytext[0]=='-' || yytext[0]=='_')
                        return(ERROR);
                  else
                        return(SYMBOL);
            }
.     { return(ERROR);}
```

Appendix 4. Yacc specification for OMFTs.

```
%{
# include <ctype.h>
# include <stdio.h>
%}

%start list

%token VARIABLE
%token DEFSCHEMA INSTANCEOF
%token NORMAL REQUIREMENT REQUIREMENTOR SELECT CONSTRAINT FINDEQUIP
CLEANUPFIND DEFCONTEXT
%token DISPLAY DISPLAYFIGURE CONNECT CONNECTRETURN CONDCONNECT
CONDCONNECTOR QUESTIONCONNECT
%token CLEANUPSELECTED HOLDCONTEXT CLEARCONTEXT WAIT
%token CONTEXT EQUIPLIST UNTILCONTEXT EQUIPID
%token KEY PROPERTIES HEADER DESC PROCSTREAM DIAGRAM GOCONTEXT
%token STAYUNTILCONTEXT RETURNCONTEXT QUESTION GONO TIME
%token STRING NUMBER SYMBOL
%token '(' ')'
%token ERROR
%token DIGITS

%%    /* beginning of rules section */

list  :     /* empty */
      |     list stat
      ;

stat  : requirementframe      {schemano++;}
      | requirementorframe    {schemano++;}
      | selectframe           {schemano++;}
      | constraintframe {schemano++;}
      | findequipframe  {schemano++;}
      | cleanupfindframe      {schemano++;}
      | defcontextframe {schemano++;}
      | displayframe          {schemano++;}
      | displayfigureframe    {schemano++;}
      | connectframe          {schemano++;}
      | connectreturnframe    {schemano++;}
      | condconnectframe      {schemano++;}
      | condconnectorframe    {schemano++;}
      | questionconnectframe  {schemano++;}
      | cleanupselectedframe  {schemano++;}
      | holdcontextframe      {schemano++;}
      | clearcontextframe     {schemano++;}
      | waitframe       {schemano++;}
      | error
      ;

requirementframe  : '(' DEFSCHEMA framename documentation
            '(' INSTANCEOF REQUIREMENT ')'
            '(' CONTEXT '(' contextname contextseqno ')' ')'
            '(' EQUIPLIST equipmentdescseq ')' ')'
            {reqctr++;}
      ;

requirementorframe      : '(' DEFSCHEMA framename documentation
            '(' INSTANCEOF REQUIREMENTOR ')'
            '(' CONTEXT '(' contextname contextseqno ')' ')'
```

```
            '(' EQUIPLIST equipmentdescseq ')' ')'
            {reqorctr++;}
      ;

selectframe : '(' DEFSCHEMA framename documentation
            '(' INSTANCEOF SELECT ')'
            '(' CONTEXT '(' contextname contextseqno ')' ')'
            '(' EQUIPLIST equipmentdescseq ')' ')'
            {selectctr++;}
      ;

constraintframe   : '(' DEFSCHEMA framename documentation
            '(' INSTANCEOF CONSTRAINT ')'
            '(' CONTEXT '(' contextname contextseqno ')' ')'
            '(' UNTILCONTEXT '(' contextname contextseqno ')' ')'
            '(' EQUIPID equipmentdesc ')' ')'
            {constraintctr++;}
      ;

findequipframe    : '(' DEFSCHEMA framename documentation
            '(' INSTANCEOF FINDEQUIP ')'
            '(' CONTEXT '(' contextname contextseqno ')' ')'
            '(' KEY keyname ')'
            '(' PROPERTIES equipprop ')' ')'
            {findequipctr++;}
      ;

cleanupfindframe  : '(' DEFSCHEMA framename documentation
            '(' INSTANCEOF CLEANUPFIND ')'
            '(' CONTEXT '(' contextname contextseqno ')' ')'
            '(' KEY keyname ')' ')'
            {cleanupfindctr++;}
      ;

defcontextframe   : '(' DEFSCHEMA framename documentation
            '(' INSTANCEOF DEFCONTEXT ')'
            '(' CONTEXT  contextname ')'
            '(' HEADER headerstring ')'
            '(' DESC descstring ')' ')'
            { defcontextctr++;}
      ;

displayframe      : '(' DEFSCHEMA framename documentation
            '(' INSTANCEOF DISPLAY ')'
            '(' CONTEXT '(' contextname contextseqno ')' ')'
            '(' HEADER  headerstring ')'
            procstreamspec ')'
            {displayctr++;}
      ;


displayfigureframe      : '(' DEFSCHEMA framename documentation
            '(' INSTANCEOF DISPLAYFIGURE ')'
            '(' CONTEXT '(' contextname contextseqno ')' ')'
            '(' HEADER headerstring ')'
            '(' DIAGRAM diagramname ')' ')'
            {displayfigurectr++;}
      ;

connectframe      : '(' DEFSCHEMA framename documentation
            '(' INSTANCEOF CONNECT ')'
            '(' CONTEXT '(' contextname contextseqno ')' ')'
            '(' GOCONTEXT '(' contextname contextseqno ')' ')' ')'
            {connectctr++;}
```

```
      ;

connectreturnframe      : '(' DEFSCHEMA framename documentation
            '(' INSTANCEOF CONNECTRETURN ')'
            '(' CONTEXT '(' contextname contextseqno ')' ')'
            '(' GOCONTEXT '(' contextname contextseqno ')' ')'
            '(' STAYUNTILCONTEXT '(' contextname contextseqno ')' ')'
            '(' RETURNCONTEXT '(' contextname contextseqno ')' ')' ')'
            {connectreturnctr++;}
      ;

condconnectframe  : '(' DEFSCHEMA framename documentation
            '(' INSTANCEOF CONDCONNECT ')'
            '(' CONTEXT '(' contextname contextseqno ')' ')'
            '(' EQUIPLIST equipmentdescseq ')'
            '(' GOCONTEXT '(' contextname contextseqno ')' ')'
            '(' STAYUNTILCONTEXT '(' contextname contextseqno ')' ')'
            '(' RETURNCONTEXT '(' contextname contextseqno ')' ')' ')'
            {condconnectctr++;}
      ;

condconnectorframe      : '(' DEFSCHEMA framename documentation
            '(' INSTANCEOF CONDCONNECTOR ')'
            '(' CONTEXT '(' contextname contextseqno ')' ')'
            '(' EQUIPLIST equipmentdescseq ')'
            '(' GOCONTEXT '(' contextname contextseqno ')' ')'
            '(' STAYUNTILCONTEXT '(' contextname contextseqno ')' ')'
            '(' RETURNCONTEXT '(' contextname contextseqno ')' ')' ')'
            {condconnectorctr++;}
      ;

questionconnectframe    : '(' DEFSCHEMA framename documentation
            '(' INSTANCEOF QUESTIONCONNECT ')'
            '(' CONTEXT '(' contextname contextseqno ')' ')'
            '(' QUESTION questionstring ')'
            gonospec ')'
            {questionconnectctr++;}
      ;

cleanupselectedframe    : '(' DEFSCHEMA framename documentation
            '(' INSTANCEOF CLEANUPSELECTED ')'
            '(' CONTEXT '(' contextname contextseqno ')' ')' ')'
            {cleanupselectedctr++;}
      ;

holdcontextframe  : '(' DEFSCHEMA framename documentation
            '(' INSTANCEOF HOLDCONTEXT ')'
            '(' CONTEXT '(' contextname contextseqno ')' ')' ')'
            {holdcontextctr++;}
      ;

clearcontextframe : '(' DEFSCHEMA framename documentation
            '(' INSTANCEOF CLEARCONTEXT ')'
            '(' CONTEXT '(' contextname contextseqno ')' ')' ')'
            {clearcontextctr++;}
      ;

waitframe   : '(' DEFSCHEMA framename documentation
            '(' INSTANCEOF WAIT ')'
            '(' CONTEXT '(' contextname contextseqno ')' ')'
            '(' TIME timeno ')' ')'
            {waitctr++;}
      ;
```

```
framename : SYMBOL
      ;

documentation     :     /* empty */
            | STRING
      ;

contextname : SYMBOL
      ;

contextseqno      : DIGITS
      ;

equipmentdescseq :      /* empty */
      |     equipmentdescseq equipmentdescs
      ;

equipmentdesc     : '(' atomicvalues ')'
      ;

equipprop   : '(' atomicvalue atomicvalue atomicvalue ')'
      ;

keyname     : SYMBOL
      ;

headerstring      : STRING
      ;

descstring  : STRING
      ;


procstreamspec    :     '(' DESC descstring ')'
      |     '(' PROCSTREAM procstreamstring ')'
            '(' DESC descstring ')'
      ;

diagramname :     SYMBOL
      ;

questionstring    : STRING

gonospec    :     '(' GOCONTEXT '(' contextname contextseqno ')' ')'
            '(' STAYUNTILCONTEXT '(' contextname contextseqno ')' ')'
            '(' RETURNCONTEXT '(' contextname contextseqno ')' ')'
      |     '(' GONO SYMBOL ')'
            '(' GOCONTEXT '(' contextname contextseqno ')' ')'
            '(' STAYUNTILCONTEXT '(' contextname contextseqno ')' ')'
            '(' RETURNCONTEXT '(' contextname contextseqno ')' ')'
      ;

timeno      : DIGITS
      ;

equipmentdescs    : '(' equipmentseqno atomicvalues ')'
      ;

atomicvalues      :     /*empty*/
      |     atomicvalues atomicvalue
      ;

procstreamstring  : STRING
      ;
```

```
equipmentseqno     : DIGITS
      ;

atomicvalue : SYMBOL
      |     STRING
      |     NUMBER
      |     DIGITS
      ;

%%    /* start of programs */

char *progname;   /* for error messages */
int schemano = 0;
int reqctr = 0,reqorctr = 0,selectctr = 0,constraintctr = 0,
findequipctr = 0, cleanupfindctr = 0, defcontextctr = 0, displayctr = 0,
displayfigurectr = 0, connectctr = 0, connectreturnctr = 0,
condconnectctr = 0, condconnectorctr = 0, questionconnectctr = 0,
cleanupselectedctr = 0, holdcontextctr = 0, clearcontextctr = 0, waitctr
= 0;
int errorctr = 0;

#include "lex.yy.c"

main(argc,argv)   /* main */

      char *argv[];

      {

      progname = argv[0];
      yyparse();
      fprintf(stderr,"PARSING STATISTICS:\n");
      fprintf(stderr,"Lines parsed %d  Schemas parsed
%d\n",yylineno,schemano);
      fprintf(stderr,"SCHEMA BREAKUP:\n");
      fprintf(stderr,"Requirement %d\nRequirementor %d\nSelect
%d\n",reqctr,reqorctr,selectctr);
      fprintf(stderr,"Constraint %d\nFindequipment %d\nCleanupfind
%d\n",constraintctr, findequipctr, cleanupfindctr);
      fprintf(stderr,"Defcontext %d\nDisplay %d\nDisplayfigure
%d\n",defcontextctr, displayctr, displayfigurectr);
      fprintf(stderr,"Connect %d\nConnectreturn %d\nCondconnect %d\n",
connectctr, connectreturnctr, condconnectctr);
      fprintf(stderr,"Condconnector %d\nQuestionconnect
%d\nCleanupselected %d\n", condconnectorctr, questionconnectctr,
cleanupselectedctr);
      fprintf(stderr,"Holdcontext %d\nClearcontext %d\nWait %d\n",
holdcontextctr, clearcontextctr, waitctr);
      fprintf(stderr,"Error schemas: %d\n",errorctr);
      fprintf(stderr,"END OF STATISTICS\n");
}

yyerror(s)

      char *s;

      {

      fprintf(stderr, "%s: %s ", progname, s);
      fprintf(stderr, "near  line %d\n",yylineno);
      fprintf(stderr, "near schema %d\n", schemano+1);
      fprintf(stderr, "lookahead token %d\n", yychar);
      fprintf(stderr, "lookahead token %s\n", yytext);
```

```
schemano++;
errorctr++;
}
```

Appendix 5. Lex specification for EFTs

```
SIGN [+-]?
ALPHABETIC [A-Za-z]
DIGIT [0-9]
ALPHA [0-9A-Za-z\-\_]
WS [ \b\t\r\v\f\n]
%%
{WS}            {       /* delete whitespace */ }
;[^\(]*         {       /* delete comments from input */}
\(              {       /* detect a left parethesis */
                        return('(');
                }
\)              {       /* detect a right parenthesis */
                        return(')');
                }
\"              {       /* detect a string */
                        int c;

                        start: while ((c = input()) != '\\' && c  != '"');
                        if (c == '\\')
                                {
                                input();
                                goto start;
                                }
                        else
                                return STRING;

                }
defschema               {       return DEFSCHEMA; }
instance\-of            {       return INSTANCEOF;      }
tank                    {       return TANK;    }
pump                    {       return PUMP;    }
compressor              {       return COMPRESSOR;      }
heater                  {       return HEATER;  }
valve                   {       return VALVE;   }
IEC                     {       return IONEXCHANGE;     }
pressure\-indicator     {       return PRESSUREI; }
hand\-switch            {       return HANDSWITCH;      }
belongs\-to             {       return BELONGSTO; }
demineralized\-water\-system    {       return DEMINERALIZED;   }
helium\-cover\-gas\-system      {       return HELIUMGAS; }
associated\-piping      {       return ASSOCIATEDPIPING;}
equip\-desc             {       return EQUIPDESC; }
pressure                {       return PRESSURE;  }
breakers                {       return BREAKERS;  }
state                   {       return STATE;   }
open                    {       return OPEN;    }
closed                  {       return CLOSED;  }
on                      {       return ON;  }
off                     {       return OFF; }
auto                    {       return AUTO;    }
standby                 {       return STANDBY;   }
operation\-mode         {       return OPERATIONMODE;   }
locked\-open            {       return LOCKEDOPEN;      }
manual                  {       return MANUAL;  }
not\-isolated           {       return NOTISOLATED;     }
isolated                {       return ISOLATED;  }
resin                   {       return RESIN;   }
fresh                   {       return FRESH;   }
{SIGN}{DIGIT}+          |
```

```
{SIGN}{DIGIT}*"."{DIGIT}+      |
{SIGN}{DIGIT}+E{SIGN}{DIGIT}+  |
{SIGN}{DIGIT}+E{SIGN}{DIGIT}+  {
                    /* detect a number */
                    return NUMBER;
      }
{ALPHABETIC}{ALPHA}*     {      return SYMBOL;     }
.                        {      return ERROR;      }
```

Appendix 6. Yacc specification for EFTs.

```
%{
# include <stdio.h>
# include <ctype.h>
%}
%start list

%token NORMAL TANK PUMP COMPRESSOR HEATER VALVE IONEXCHANGE PRESSUREI
HANDSWITCH
%token DEFSCHEMA INSTANCEOF SYMBOL STRING
%token BELONGSTO DEMINERALIZED HELIUMGAS ASSOCIATEDPIPING EQUIPDESC
%token PRESSURE NUMBER
%token BREAKERS
%token STATE OPEN CLOSED ON OFF AUTO STANDBY
%token OPERATIONMODE LOCKEDOPEN
%token MANUAL NOTISOLATED ISOLATED
%token RESIN FRESH
%token '(' ')'
%token ERROR

%%     /* beginning of rules section */
list   :      /* empty */
       |      list stat
       ;

stat   :      tankframe    {schemano++;}
       | pumpframe         {schemano++;}
       | compressorframe {schemano++;}
       | heaterframe              {schemano++;}
       | valveframe               {schemano++;}
       | ionexchangeframe         {schemano++;}
       | pressureiframe  {schemano++;}
       | handswitchframe {schemano++;}
       | error
       ;

tankframe    : '(' DEFSCHEMA equipmentname documentation
             '(' INSTANCEOF TANK ')'
             '(' BELONGSTO tankplacement ')'
             '(' EQUIPDESC STRING ')'
             '(' PRESSURE NUMBER ')'
             '(' STATE tankstate ')' ')'
             {tankctr++;}
      ;

pumpframe    : '(' DEFSCHEMA equipmentname documentation
             '(' INSTANCEOF PUMP ')'
             '(' BELONGSTO DEMINERALIZED ')'
             '(' EQUIPDESC STRING ')'
             '(' STATE pumpstate ')'
             '(' BREAKERS breakerstate ')' ')'
             {pumpctr++;}
      ;

compressorframe    : '(' DEFSCHEMA equipmentname documentation
             '(' INSTANCEOF COMPRESSOR ')'
             '(' BELONGSTO HELIUMGAS ')'
             '(' EQUIPDESC STRING ')'
             '(' STATE compressorstate ')'
             '(' BREAKERS breakerstate ')' ')'
```

```
            {compressorctr++;}
      ;

heaterframe : '(' DEFSCHEMA equipmentname documentation
            '(' INSTANCEOF HEATER ')'
            '(' BELONGSTO HELIUMGAS ')'
            '(' EQUIPDESC STRING ')'
            '(' STATE heaterstate ')'
            '(' BREAKERS breakerstate ')' ')'
            {heaterctr++;}
      ;

valveframe  : '(' DEFSCHEMA equipmentname documentation
            '(' INSTANCEOF VALVE ')'
            '(' BELONGSTO ASSOCIATEDPIPING ')'
            '(' EQUIPDESC STRING ')'
            restvalvespec ')'
            {valvectr++;}
            ;

restvalvespec     : restpressurespec
      |     '(' OPERATIONMODE valveopmode ')' restpressurespec
      ;

restpressurespec  : reststatespec
      |     '(' PRESSURE NUMBER ')' reststatespec
      ;

reststatespec     : /* empty */
      |     '(' STATE valvestate ')'
      ;

ionexchangeframe  : '(' DEFSCHEMA equipmentname documentation
            '(' INSTANCEOF IONEXCHANGE ')'
            '(' BELONGSTO DEMINERALIZED ')'
            '(' EQUIPDESC STRING ')'
            '(' STATE ionexchangestate ')'
            '(' RESIN FRESH ')' ')'
            {ionexchangectr++;}
      ;

pressureiframe    : '(' DEFSCHEMA equipmentname documentation
            '(' INSTANCEOF PRESSUREI ')'
            '(' BELONGSTO ASSOCIATEDPIPING ')'
            '(' EQUIPDESC STRING ')'
            '(' PRESSURE NUMBER ')' ')'
            {pressureictr++;}
      ;

handswitchframe   : '(' DEFSCHEMA equipmentname documentation
            '(' INSTANCEOF HANDSWITCH ')'
            '(' BELONGSTO ASSOCIATEDPIPING ')'
            '(' EQUIPDESC STRING ')'
            '(' STATE STRING ')' ')'
            {handswitchctr++;}
      ;

equipmentname     : SYMBOL
      ;

documentation     :     /* empty */
      | STRING
      ;
```

```
tankplacement      : HELIUMGAS
       |     DEMINERALIZED
       ;

tankstate    : OPEN
       |     CLOSED
       ;

pumpstate    : ON
       |     OFF
       ;

breakerstate       : CLOSED
       |     OPEN
       ;

compressorstate    : AUTO
       |     STANDBY
       ;

heaterstate : ON
       |     OFF
       ;

valvestate   : OPEN
       |     CLOSED
       |     LOCKEDOPEN
       |     AUTO
       |     MANUAL
       ;

valveopmode : MANUAL
       |     AUTO
       ;

ionexchangestate   : NOTISOLATED
       |     ISOLATED
       ;

%%     /* start of programs */

char *progname;    /* for error messages */
int schemano = 0;
int tankctr = 0, pumpctr = 0, compressorctr = 0, heaterctr = 0, valvectr
= 0, ionexchangectr = 0, pressureictr = 0, handswitchctr = 0;
int errorctr = 0;

#include "lex.yy.c"

main(argc,argv)    /* main */

       char *argv[];

       {

       progname = argv[0];
       yyparse();
       fprintf(stderr,"PARSING STATISTICS:\n");
       fprintf(stderr,"Lines parsed %d  Schemas parsed
%d\n",yylineno,schemano);
       fprintf(stderr,"SCHEMA BREAKUP:\n");
       fprintf(stderr,"Tanks %d\nPumps %d\nCompressors %d\nHeaters
%d\n",tankctr,pumpctr,compressorctr,heaterctr);
```

```
      fprintf(stderr,"Valves %d\nIonexchanges %d\nPressure-indicators
%d\nHandswitchs
%d\n",valvectr,ionexchangectr,pressureictr,handswitchctr);
      fprintf(stderr,"Error schemas %d\n",errorctr);
      fprintf(stderr,"END OF STATISTICS\n");
      }

yyerror(s)
      char *s;

      {

      fprintf(stderr, "%s: %s ", progname, s);
      fprintf(stderr, "near  line %d\n", yylineno);
      fprintf(stderr, "near schema %d\n", schemano +1);
      fprintf(stderr, "lookahead token %d\n", yychar);
      fprintf(stderr, "lookahead token %s\n", yytext);
      errorctr++;
      schemano++;
      }
```

Appendix 7. Lex specification for GENT.

```
%{
/* ------------------------------------------------------------------
 *
 * File: lzcman.l
 * System: Liquid zone control manual parsing system
 *
 * Purpose: The lexical analyzer for the preprocessed input of liquid
zone control manual
 *
 * Programmer: Joozar Vasi
 * Date: 24 Oct. 1992
 * Detail:
 *
 * ------------------------------------------------------------------*/

/*---------------------------- Include Files -------------------------
*/

#include <strings.h>
#include "y.tab.h"
#include <stdio.h>
#include "header.h"

/* ---------------------------- Module Definitions --------------------*/

      /* none */

/* ---------------------------- Imported Variables --------------------*/

extern FILE *fp;                /* from main.c */
extern int  chptno;             /* from lzcman.y */
extern YYSTYPE    yylval;       /* from lzcman.y */
extern int  chptctr;            /* from lzcman.y */

/* ---------------------------- Imported Functions --------------------*/

extern char *strsave();         /* from utilities.lib.c */
extern int  findnlchars();      /* from utilities.lib.c  */

/* ---------------------------- Exported Varibles --------------------*/

      /* none */

/* ---------------------------- Local Typedefs ----------------------*/

      /* none */

/* ---------------------------- Local Global Variables ---------------*/

static int chpt;          /* current chapter number for error checking */
static int sectno         /* current section number */
static int sect;          /* current section number for error checking */
static int subsectno;     /* current subsection number */
static int subsect;       /* current subsection number for error checking
*/
static int stepno         /* current step number */
static int step;          /* current step number for error checking*/
static char yytextbuf[BTOKENLEN];   /* a temporary array to store tokens
of lex */
```

```
/* ---------------------------- Local Functions ----------------------*/

      /* none */

/* ---------------------------- Global Functions ---------------------*/

      /* none */

/* -------------------------------------------------------------------*/
%}
%START CHAPTER CHPTHEADING SECTION SCTHEADING PARADESC STEPNUM
ALPHABETIC [A-Za-z\-0-9]
DIGITS [1-9][0-9]*
CHAPTERNO {DIGITS}\.0
SECTIONNO {DIGITS}\.{DIGITS}
SUBSECTIONNO {DIGITS}\.{DIGITS}\.{DIGITS}
STEPNO Step[ ]+{DIGITS}
%%
{CHAPTERNO}[ ]+   {
      /* Found chapter number. Extract current chapter number */
      getsect(yytext,&chpt,&sect,&subsect);
      yylval.number = chpt;
      sectno = subsectno = stepno = 0;
      /* set up to accept chapter heading */
      fprintf(fp,"In token chapterno. Found chapter:!%d! section:!%d!
subsection:!%d!\n",chpt,sect,subsect);
      fflush(fp);
      BEGIN CHPTHEADING;
      return CHAPTERTAG;
     }
<CHPTHEADING>.+[\n]+    {
      /* Found chapter heading. Set up to accept section number */
      BEGIN SECTION;
      fprintf(fp,"Found chapter header with newlines !%s!\n",yytext);
      fflush(fp);
      return CHAPTERHEADER;
      }
<SECTION>{SECTIONNO}[ ]+      {
      /* extract current chapter number and section number. Print error
message if chatper number is incorrect and section numbers are not in
order */
      getsect(yytext,&chpt,&sect,&subsect);
      if (chpt != chptno)
            {
            fprintf(fp,"Expecting chapter number !%d!\n",chptno);
            fprintf(fp,"Received chapter number !%d!\n",chpt);
            fflush(fp);
            }
      if ( sect != ++sectno)
            {
            fprintf(fp,"Expection section number !%d!\n",sectno+1);
            fprintf(fp,"Received section number !%d!\n",sect);
            fflush(fp);
            }
      sectno = sect;
      yylval.number = sect;
      subsectno = stepno = 0;
      /* print found details */
      fprintf(fp,"Found sectionno token. Found chapter:!%d! section:!%d!
subsection:!%d!\n",chpt,sect,subsect);
      fflush(fp);
      /* set up to accept section heading */
      BEGIN SCTHEADING;
```

```
      return SECTIONTAG;
      }
<SECTION>{SUBSECTIONNO}[ ]+   {
      /* Found subsection number. Extract current chapter, section and
subsection number. Print error message if chapter or section number is
incorrect or if the subsection number is not in sequence */
      getsect(yytext,&chpt,&sect,&subsect);
      if (chptno != chpt || sectno != sect)
            {
            fprintf(fp,"Expecting chapter !%d!, Found chapter
!%d!\n",chptno,chpt);
            fprintf(fp,"Expecting section !%d!, Found section
!%d!\n",sectno,sect);
            fflush(fp);
            }
      if ( subsect != ++subsectno)
            {
            fprintf(fp,"Expecting subsection number !%d!\n",subsectno);
            fprintf(fp,"Received subsection number !%d!\n",subsect);
            fflush(fp);
            }
      subsectno = subsect;
      stepno = 0;
      yylval.number = subsectno;
      /* announce to the world the subsection found */
      fprintf(fp,"Found subsection token. Found Chapter:!%d!
Section:!%d! Subsection:!%d!\n",chpt,sect,subsect);
      fflush(fp);
      /* set up to accept section heading */
      BEGIN SCTHEADING;
      return SUBSECTIONTAG;
      }
<SCTHEADING>.+[\n]+/{SUBSECTIONNO}   {
      /* Found section/subsection heading. Delete newline charaters in
token */
      strcpy(yytextbuf,yytext);
      *index(yytextbuf,'\n') = '\0';
      /* Return heading token. Also announce to the world that
section/subsection header is found */
      yylval.tokenstr = strsave(yytextbuf);
      fprintf(fp,"Installed section header !%s!. Subsections
follow\n",yylval.tokenstr);
      fflush(fp);
      /* Set up to accept subsection details */
      BEGIN SECTION;
      return HEADER;
      }
<SCTHEADING>.+[\n]+/{STEPNO}   {
      /* Found section/subsection heading. Delete newline characters in
token*/
      strcpy(yytextbuf,yytext);
      *index(yytextbuf,'\n') = '\0';
      /* Return heading token. Announce to the world that
section/subsection header is found */
      yylval.tokenstr = strsave(yytextbuf);
      fprintf(fp,"Installed section header !%s!. Steps
follow\n",yylval.tokenstr);
      fflush(fp);
      /* Set up to accept step number */
      BEGIN STEPNUM;
      return HEADER;
      }
<SCTHEADING>.+[\n]+     {
```

```
      /* Found section/subsection heading. Delete newline character from
token*/
      strcpy(yytextbuf,yytext);
      *index(yytextbuf,'\n') = '\0';
      /* Return heading token. Annouce to the world that a
section/subsection heading is found */
      yylval.tokenstr = strsave(yytextbuf);
      fprintf(fp,"Installed section header !%s!. Paragraph
follows\n",yylval.tokenstr);
      fflush(fp);
      /* Set up to accept coming paragraph */
      *yytextbuf = '\0';
      BEGIN PARADESC;
      return HEADER;
      }
<STEPNUM>{STEPNO}\.[ ]+ {
      /* Found step number. Extract step number. Print error message if
steps are out of sequence */
      strcpy(yytextbuf,yytext);
      sscanf(yytextbuf,"%*s %d",&step);
      if (step != ++stepno)
            {
            fprintf(fp,"Expecting step !%d!",stepno);
            fprintf(fp,"Found step !%d!",step);
            fflush(fp);
            }
      stepno=step;
      /* Return step number. Announce to the world the step number found
*/
      yylval.number = step;
      fprintf(fp,"found step number !%d!. Step paragragh
follows\n",stepno);
      fflush(fp);
      /* set up ot accept step description */
      *yytextbuf = '\0';
      BEGIN PARADESC;
      return STEPTAG;
      }
<PARADESC>[\n][\n]+{CHAPTERNO}      {
      /* Found beginning of chapter. Store section, subsection or step
details*/
      yyless(findnlchars(yytext));
      yylval.tokenstr = strsave(yytextbuf);
      fprintf(fp,"Installed section/subsection/step description !%s!.
Chapter follows\n",yylval.tokenstr);
      fflush(fp);
      /* set up to accept another chapter */
      BEGIN 0;
      return PARAGRAPH;
      }
<PARADESC>[\n][\n]+{SECTIONNO}      {
      /* Determine whether a new section begins */
      getsect(yytext,&chpt,&sect,&subsect);
      if ((chptno == chpt) && (sectno +1 == sect))
            {
            /* Found new section. Push sectionno back on input stream.
Return token containing details */
            yyless(findnlchars(yytext));
            yylval.tokenstr = strsave(yytextbuf);
            /* Announce to the world about found details. Set up to
accept section */
            fprintf(fp,"Installed section/subsection description !%s!.
Section number follows\n",yylval.tokenstr);
            fflush(fp);
```

```
            BEGIN SECTION;
            return PARAGRAPH;
            }
      else
            {
            /* No new section. Accumalate details */
            fprintf(fp,"Found invalid section number in input
            !%s!\n",yytext);
            fflush(fp);
            yyless(findnlchars(yytext));
            strcat(yytextbuf,yytext);
            }
      }
<PARADESC>[\n][\n]+{SUBSECTIONNO}   {
      /* Determine whether a new subsection begins */
      getsect(yytext,&chpt,&sect,&subsect);
      if ((chptno == chpt) && (sectno == sect) && (subsectno +1 ==
subsect))
            {
            /* Found new subsection. Push subsection number back on the
input stream. Return token containing details */
            yyless(findnlchars(yytext));
            yylval.tokenstr = strsave(yytextbuf);
            /* Announce to the world about found details. Set up to
accept subsection */
            fprintf(fp,"Installed section/subsection/step description
!%s!. Subsection number follows\n",yylval.tokenstr);
            fflush(fp);
            BEGIN SECTION;
            return PARAGRAPH;
            }
      else
            {
            /* No new subsection. Accumalate paragraph */
            fprintf(fp,"Found invalid subsection number in input
!%s!\n",yytext);
            yyless(findnlchars(yytext));
            strcat(yytextbuf,yytext);
            }
      }
<PARADESC>[\n][\n]+{STEPNO}   {
      /* A new step is coming. Push back stepnumber on input stream.
Collect and announce to the world about found token */
      yyless(findnlchars(yytext));
      yylval.tokenstr = strsave(yytextbuf);
      fprintf(fp,"Installed section/subsection/step description !%s!.
Step number follows\n",yylval.tokenstr);
      fflush(fp);
      /* set up to accept another step */
      BEGIN STEPNUM;
      return PARAGRAPH;
      }
<PARADESC>[\n]    {
      /* Paragraph has not terminated. Accumalate paragraph token */
      strcat(yytextbuf,yytext);
      }
<PARADESC>.+      {
      /* Paragraph has not terminated. Accumalate paragraph token */
      strcat(yytextbuf,yytext);
      }
.     {     fprintf(fp,"Error detected chapter no !%d! section no !%d!
subsectionno !%d! token !%s! \n",chptno,sectno,subsectno,yytext);
            fflush(fp);
      }
```

```
%%
yywrap()    /* fprint statistics */
{
fprintf(stderr,"# of lines processed %d\n",yylineno);
fprintf(stderr,"# of chapters processed %d\n",chptctr);
}
```

Appendix 8. Yacc specification for GENT.

```
%{
 /* ------------------------------------------------------------------
 *
 * File: lzcman.y
 * System: Liquid zone control manual parsing system
 *
 * Purpose: Yacc specification for input text
 *
 * Programmer: Joozar Vasi
 * Date: 24 March 1992
 * Detail:
 *
 * ------------------------------------------------------------------*/

/* ---------------------------- Include Files -----------------------*/

#include <stdio.h>
#include "header.h"

/* ---------------------------- Module Definitions ------------------*/

      /* none */

/* ---------------------------- Imported Variables ------------------*/

      /* none */

/* ---------------------------- Imported Functions ------------------*/

extern      void  addtosectlist();        /* from utilities.lib.c */
extern      void  addtosteplist();        /* from utilities.lib.c */
extern      void  sectfree();             /* from utilities.lib.c */
extern      FILE  *efopen();              /* from utilities.lib.c */
extern      void  printsectiondetails();  /* from printsect.lib.c */
extern      char  *progname;              /* from main.c */
extern      int   yylineno;               /* from lex.yy.c */
extern      char  yytext[];               /* from lex.yy.c */
extern  equipptr getequiplist();          /* from equiputilities.lib.c
*/

/* ---------------------------- Exported Varibles -------------------*/

int   chptno;                  /* to lzcman.l and printsect.lib.c */
int   chptctr = 0;             /* to lzcman.l */

/* ---------------------------- Local Typedefs ----------------------*/

      /* none */

/* ---------------------------- Local Global Variables --------------*/

static      sectptr     fsect;
static      sectptr lsect;
static      stepptr     firststep;
static      stepptr     laststep;
static      FILE  *ftl;

/* ---------------------------- Local Functions ---------------------*/
```

```
      /* none */

/* ---------------------------- Local Functions ----------------------*/

      /* none */

%}
%start manual
%union      {
      char  *tokenstr;
      int   number;
      }

%token       FORMFEED CHAPTERTAG CHAPTERHEADER SECTIONTAG HEADER ERROR
%token   SUBSECTIONTAG STEPTAG PARAGRAPH
%% /* beginning of rule section */

      /* The LZC manual consists of one or more chapters */
manual      : chapters
            {
            fprintf(stderr,"# of lines processed %d\n",yylineno);
            fprintf(stderr,"# of chapters processed %d\n",chptctr);
            }
      ;

chapters    : chapter
      | chapters chapter
      ;

      /* A chapter in the LZC manual consists of a chapternumber, a
chapterheader, and the rest of the chapter */
chapter           : CHAPTERTAG
            {
            fsect = NULL;
            lsect = NULL;
            chptno = $1;
            }
            CHAPTERHEADER sections
            {
            chptctr++;
            }
      ;

      /* The rest of the chapter consists of one or more sections */
sections    : section
      | sections section
      ;

      /* A section consists of a sectionnumber, a sectionheader and the
rest of the section. Print section details once a section is recognized
*/
section : SECTIONTAG HEADER PARAGRAPH
            {
            /* Add section details to linked list */
            addtosectlist(&fsect,&lsect,$1,$2,$3,FRONTQ);
            /* Find equipment requirement details if present and add to
linked list */
            ft1 = efopen("temp1","w");
            fprintf(ft1,"%s\n",$3);
            fclose(ft1);
            system("equipparse <temp1 >temp2");
            fsect->sectequiplist = getequiplist();
            /* Print section details and free memory holding these
details */
```

```
            printsectiondetails(fsect);
            sectfree(&fsect,&lsect);
            }
      | SECTIONTAG HEADER PARAGRAPH subsections
            {
            /* Add section details to linked list */
            addtosectlist(&fsect,&lsect,$1,$2,$3,FRONTQ);
            /* Find equipment requirement details if present and add to
linked list */
            ft1 = efopen("temp1","w");
            fprintf(ft1,"%s\n",$3);
            fclose(ft1);
            system("equipparse <temp1 >temp2");
            fsect->sectequiplist = getequiplist();
            /* Print section details and free memory holding these
details  */
            printsectiondetails(fsect);
            sectfree(&fsect,&lsect);
            }
      | SECTIONTAG HEADER subsections
            {
            /* Add section details to linked list */
            addtosectlist(&fsect,&lsect,$1,$2,NULL,FRONTQ);
            /* Print section details and free memory holding these
details  */
            printsectiondetails(fsect);
            sectfree(&fsect,&lsect);
            }
      | SECTIONTAG HEADER PARAGRAPH steps
            {
            /* Add section details to linked list */
            addtosectlist(&fsect,&lsect,$1,$2,$3,FRONTQ);
            /* Find equipment requirement details if present add add to
linked list. Also update list with step details */
            ft1 = efopen("temp1","w");
            fprintf(ft1,"%s\n",$3);
            fclose(ft1);
            system("equipparse <temp1 >temp2");
            fsect->sectequiplist = getequiplist();
            fsect->fstep = firststep;
            fsect->lstep = laststep;
            /* Print section details and free memory holding these
details  */
            printsectiondetails(fsect);
            sectfree(&fsect,&lsect);
            }
      | SECTIONTAG HEADER PARAGRAPH steps
            {
            /* Add section details to linked list */
            addtosectlist(&fsect,&lsect,$1,$2,$3,FRONTQ);
            /* Find equipment requirement details if present add add to
linked list. Also update list with step details */
            ft1 = efopen("temp1","w");
            fprintf(ft1,"%s\n",$3);
            fclose(ft1);
            system("equipparse <temp1 >temp2");
            fsect->sectequiplist = getequiplist();
            fsect->fstep = firststep;
            fsect->lstep = laststep;
            }
            subsections
            {
            /* Print section details and free memory holding these
details  */
```

```
            printsectiondetails(fsect);
            sectfree(&fsect,&lsect);
            }
      ;

      /* A susection consists of one or more subsections */
subsections : subsection
      | subsections subsection
      ;

      /* A subsection consist of a subsection number, a subsection
header and the rest of the subsection */
subsection : SUBSECTIONTAG HEADER PARAGRAPH
            {
            /* Add subsection details to linked list */
            addtosectlist(&fsect,&lsect,$1,$2,$3,BACKQ);
            /* Find equipment requirement details if present and add to
linked list */
            ft1 = efopen("temp1","w");
            fprintf(ft1,"%s\n",$3);
            fclose(ft1);
            system("equipparse <temp1 >temp2");
            lsect->sectequiplist = getequiplist();
            }
      | SUBSECTIONTAG HEADER steps
            {
            /* Add subsection details to linked list. Add step details
to list */
            addtosectlist(&fsect,&lsect,$1,$2,NULL,BACKQ);
            lsect->fstep = firststep;
            lsect->lstep = laststep;
            }
      | SUBSECTIONTAG HEADER PARAGRAPH steps
            {
            /* Add subsection details to linked list */
            addtosectlist(&fsect,&lsect,$1,$2,$3,BACKQ);
            /* Find equipment requirement details if present and add to
linked list */
            ft1 = efopen("temp1","w");
            fprintf(ft1,"%s\n",$3);
            fclose(ft1);
            system("equipparse <temp1 >temp2");
            lsect->sectequiplist = getequiplist();
            /* Add step details to linked list */
            lsect->fstep = firststep;
            lsect->lstep = laststep;
            }
      ;

      /* A section or subsection can contain one or more steps */
steps :     STEPTAG PARAGRAPH
            {
            /* Collect details of first step including any equipment
requirement details if present */
            firststep =  NULL;
            laststep = NULL;
            addtosteplist(&firststep,&laststep,$1,$2);
            ft1 = efopen("temp1","w");
            fprintf(ft1,"%s\n",$2);
            fclose(ft1);
            system("equipparse <temp1 >temp2");
            laststep->stepequiplist = getequiplist();
            }
      | steps STEPTAG PARAGRAPH
```

```
            {
            /* Collect details of all other steps including equipment
requirement details if present */
            addtosteplist(&firststep,&laststep,$2,$3);
            ft1 = efopen("temp1","w");
            fprintf(ft1,"%s\n",$3);
            fclose(ft1);
            system("equipparse <temp1 >temp2");
            laststep->stepequiplist = getequiplist();
            }
      ;
%% /* end of rules */
yyerror(s)        /* for parse error */
      char *s;
{
      fprintf(stderr, "%s: %s", progname,s);
      fprintf(stderr, "near line %d\n", yylineno);
      fprintf(stderr, "in chapter %d\n", chptno);
      fprintf(stderr, "lookahead token %d\n", yychar);
      fprintf(stderr, "lookahead token %s\n", yytext);
}
```

Appendix 9. Lex specification for rescanning

```
%{
/* ------------------------------------------------------------------------
 *
 * File: equipreq.l
 * System: The liquid zone control manual parsing system
 *
 * Purpose: Gather equipment details from PARAGRAPH token attribute
 *
 * Programmer: Joozar Vasi
 * Date: 3 Apr.  1992
 * Detail:
 *
 * ------------------------------------------------------------------------*/

/* ---------------------------- Include Files ----------------------*/

#include    <strings.h>
#include    <malloc.h>
#include    "header.h"

/* ---------------------------- Module Definitions -------------------*/

      /* none */

/* ---------------------------- Imported Variables -------------------*/

extern      FILE  *fp;                      /* from main1.c */

/* ---------------------------- Imported Functions -------------------*/

extern      void  addtoequiplist();         /* from equiputilities.lib.c
*/
extern      void  addinfotoequiplist();     /* from equiputilities.lib.c
*/
extern  void      printreqtofile();         /* from equiputilities.lib.c
*/
extern      void  equipfree();              /* from utilities.lib.c */

/* ---------------------------- Exported Varibles --------------------*/

      /* none */

/* ---------------------------- Local Typedefs -----------------------*/

      /* none */

/* ---------------------------- Local Global Variables ---------------*/

static      char curattribute[EQUIPLEN];
static      char curoperator[EQUIPLEN];
static      char curvalue[EQUIPLEN];
static  equipptr  efrontptr=NULL;           /* for linked list of
equipment details */
static      equipptr    ebackptr=NULL;

/* ---------------------------- Local Functions ----------------------*/

      /* none */
```

```
/* ---------------------------- Global Functions ----------------------*/

      /* none */

%}
%start      KEYWORD
%o 5000
%a 5000
DIGITS [0-9]+
ALPHANUMERIC [a-zA-Z0-9]
OTHER [^a-zA-Z0-9]
%%
      BEGIN KEYWORD;
<KEYWORD>[aA]uto/{OTHER} |
<KEYWORD>AUTO/{OTHER}   {
      /* make auto the desired state of equipment */
      strcpy(curvalue,"auto");
      fprintf(fp,"Found equipment value !%s!\n",yytext);
      BEGIN 0;
      }
<KEYWORD>[sS]tandby/{OTHER} |
<KEYWORD>STANDBY/{OTHER}      {
      /* make standby the desired state of equipment */
      strcpy(curvalue,"standby");
      fprintf(fp,"Found equipment value !%s!\n",yytext);
      BEGIN 0;
      }
<KEYWORD>[pP]ressure/{OTHER}  {
      /* make pressure the desired attribute of equipment */
      strcpy(curattribute,"pressure");
      fprintf(fp,"Found equipment attribute !%s!\n",yytext);
      BEGIN 0;
      }
<KEYWORD>CLOSED?/{OTHER} |
<KEYWORD>[Cc]losed?/{OTHER}     |
<KEYWORD>[Cc]losing/{OTHER}   {
      /* make closed the desired state of equipment */
      strcpy(curvalue,"closed");
      fprintf(fp,"Found equipment value !%s!\n",yytext);
      BEGIN 0;
      }
<KEYWORD>OPEN/{OTHER}    |
<KEYWORD>[Oo]pen/{OTHER}              |
<KEYWORD>[Oo]pening/{OTHER}   {
      /* make open the desired state of equipment */
      strcpy(curvalue,"open");
      fprintf(fp,"Found equipment value !%s!\n",yytext);
      BEGIN 0;
      }
<KEYWORD>OFF/{OTHER}     |
<KEYWORD>[Oo]ff/{OTHER} {
      /* make off the desired state of equipment */
      strcpy(curvalue,"off");
      fprintf(fp,"Found equipment value !%s!\n",yytext);
      BEGIN 0;
      }
<KEYWORD>ON/{OTHER}      |
<KEYWORD>[Oo]n/{OTHER}   {
      /* make on the desired state of equipment */
      strcpy(curvalue,"on");
      fprintf(fp,"Found equipment value !%s!\n",yytext);
      BEGIN 0;
      }
<KEYWORD>controllers?/{OTHER} {
```

```
      /* make controller the desired attribute of equipment */
      strcpy(curattribute,"controller");
      fprintf(fp,"Found equipment value !%s!\n",yytext);
      BEGIN 0;
      }
<KEYWORD>TK{DIGITS}/{OTHER}    |
<KEYWORD>P{DIGITS}/{OTHER}     |
<KEYWORD>CP{DIGITS}/{OTHER}    |
<KEYWORD>HTR{DIGITS}/{OTHER}   |
<KEYWORD>V{DIGITS}/{OTHER}     |
<KEYWORD>PV{DIGITS}/{OTHER}    |
<KEYWORD>PCV{DIGITS}/{OTHER}   |
<KEYWORD>PRV{DIGITS}/{OTHER}   |
<KEYWORD>LCV{DIGITS}/{OTHER}   |
<KEYWORD>FC{DIGITS}/{OTHER}    |
<KEYWORD>FI{DIGITS}/{OTHER}    |
<KEYWORD>IEC{DIGITS}/{OTHER}   {
      /*create and insert node in equipment details linked list */
      addtoequiplist(&efrontptr,&ebackptr,yytext);
      fprintf(fp,"Found new equipment item !%s!\n",yytext);
      BEGIN 0;
      }
<KEYWORD>{DIGITS}/{OTHER}      {
      /* make number the desired state of equipment */
      strcpy(curvalue,yytext);
      fprintf(fp,"Found equipment value !%s!\n",yytext);
      BEGIN 0;
      }
[\.]/[  \n] {
      /* found end of sentence */
      /* set default attribute, operator and state of equipment */
      if (*curattribute == '\0')
          strcpy(curattribute,"state");
      if (*curoperator == '\0')
          strcpy(curoperator,"=");
      if (*curvalue == '\0')
          strcpy(curvalue,"unknown");
      /* put details of new equipment items in linked list */
      addinfotoequiplist(efrontptr,curattribute,curoperator,curvalue);
      /* intialize current attribute,operator, and value */
      *curattribute = '\0';
      *curoperator = '\0';
      *curvalue = '\0';
      BEGIN KEYWORD;
      }
{ALPHANUMERIC} {
      /* delete interesting characters */
      BEGIN 0;
      }
{OTHER}      {
      /* delete everthing of no interest  */
      BEGIN KEYWORD;
      }
%%
yywrap()     /* dump equipment requirement details to file and reclaim
memory */
      {
      printreqtofile(efrontptr);
      equipfree(efrontptr);
      fprintf(fp,"# of lines processed in paragraph %d\n",yylineno);
      }
```

Appendix 10. Lex specification for part of the preprocessor

```
%{
/* ------------------------------------------------------------------
 *
 * File: preprocessor.l
 * System: Liquid zone control manual parsing system
 *
 * Purpose: The lexical analyzer for deleting header information present
on top                                  of page
 * Programmer: Joozar Vasi
 * Date: 26 June 1992
 * Detail:
 *
 * ------------------------------------------------------------------*/

/*---------------------------- Include Files -------------------------*/

#include <strings.h>

/* ---------------------------- Module Definitions --------------------*/

# define GO 2                   /* success in returning from lex */
# define LINELEN 1000           /* length of longest line in input */

/* ---------------------------- Imported Variables --------------------*/

      /* none */

/* ---------------------------- Imported Functions --------------------*/

      /* none */

/* ---------------------------- Exported Varibles ---------------------*/

      /* none */

/* ---------------------------- Local Typedefs ------------------------*/

      /* none */

/* ---------------------------- Local Global Variables ----------------*/

static FILE *fp1;       /* for diagnostic messages */

/* ---------------------------- Local Functions -----------------------*/

      /* none */

/* ---------------------------- Global Functions ----------------------*/

      /* none */

/* ------------------------------------------------------------------*/
%}
%start SCAN
DIGITS [1-9][0-9]*
CHAPTERNO {DIGITS}\.0
SECTIONNO {DIGITS}\.{DIGITS}
SUBSECTIONNO {DIGITS}\.{DIGITS}\.{DIGITS}
%%
```

```
      extern void getsect();                  /* function to get current
chapter, section, subsection */

      static int chptno;                      /* current chapter number for
error checking */
      static int chpt;                  /* current chapter number obtained
from text */

      static int sectno;                      /* current section number for
error checking */
      static int sect;                  /* current section number obtained
from text */

      static int subsectno;                   /* current subsection number
for error checking */
      static int subsect;                     /* current subsection number
obtained from text */

      static char yytextbuf[LINELEN];              /* for deleting unwanted
text */
      static int yytextbufindex;              /* index for yytextbuf */
<SCAN>{CHAPTERNO} {
      /* Found chapter number. Extract current chapter number and write
matched text to output */
      getsect(yytext,&chpt,&sect,&subsect);
      chptno = chpt;
      sectno = subsectno = 0;
      ECHO;
      BEGIN 0;
      return GO;
      }
[\n][\n]+{CHAPTERNO}    {
      /* Found chapter number. Extract chapter number */
      getsect(yytext,&chpt,&sect,&subsect);
      /* if chapter number is already found delete rest of line */
      if (chpt == chptno)
            {
            strcpy(yytextbuf,yytext);
            yytextbufindex = strlen(yytextbuf);
            while ((yytextbuf[yytextbufindex++] = input()) != '\n');
            yytextbuf[yytextbufindex] = '\0';
            fprintf(fp1,"Deleted line !%s! in line
!%d!\n",yytextbuf,yylineno);
            printf("\n\n");
            unput('\n');
            }
      /* if chapter is in order then write matched text to output */
      else if (chpt == ++chptno)
            {
            fprintf(fp1,"found chapter new chapter !%d! in line
!%d!\n",chptno,yylineno);
            sectno = subsectno = 0;
            ECHO;
            }
      else
      /* if chapter is not in order print error message and write
matched text to output */
            {
            fprintf(fp1,"Error: expecting chapter !%d!, found chapter
!%d!\n",chptno--,chpt);
            ECHO;
            }
      return GO;
      }
```

```
[\n][\n]+{SECTIONNO}    {
      /* extract current chapter number and section number */
      getsect(yytext,&chpt,&sect,&subsect);
      /* if invalid chapter is found print error message and write
matched text to output */
      if (chpt != chptno)
            {
            fprintf(fp1,"Error: in section number !%d! in
line\n",sectno,yylineno);
            fprintf(fp1,"Expecting chapter number !%d! but received
chapter !%d!\n",chptno,chpt);
            ECHO;
            }
      /* if section number is already found delete rest of line */
      else if (sect == sectno)
            {
            strcpy(yytextbuf,yytext);
            yytextbufindex = strlen(yytextbuf);
            while ((yytextbuf[yytextbufindex++] = input()) != '\n');
            yytextbuf[yytextbufindex] = '\0';
            fprintf(fp1,"Deleted line !%s! in line
!%d!\n",yytextbuf,yylineno);
            printf("\n\n");
            unput('\n');
            }
      /* if valid next section is found write matched text to output */
      else if (sect == ++sectno)
            {
            fprintf(fp1,"Found new section !%d! in line
!%d!\n",sectno,yylineno);
            subsectno =0;
            ECHO;
            }
      /* if section number is out of order print error message and write
matched text to output */
      else
            {
            fprintf(fp1,"Error: expecting section !%d!, found section
!%d! in line !%d!\n",sectno--,sect,yylineno);
            ECHO;
            }
      return GO;
      }
[\n][\n]+{SUBSECTIONNO} {
      /* extract current chapter, section and subsection number */
      getsect(yytext,&chpt,&sect,&subsect);
      /* if invalid chapter or section number is found print error
message and write matched text to output */
      if (chpt != chptno || sect != sectno)
            {
            fprintf(fp1,"Error: in subsection number !%d! in line !%d!
\n",subsectno, yylineno);
            fprintf(fp1,"Expecting chapter number and section number
!%d! !%d!\n",chptno,sectno);
            fprintf(fp1,"Received chapter number and sectno !%d! !%d!\n"
,chpt,sect);
            ECHO;
            }
      /* if subsection number already found delete rest of line */
      else if (subsect == subsectno)
            {
            strcpy(yytextbuf,yytext);
            yytextbufindex = strlen(yytextbuf);
            while ((yytextbuf[yytextbufindex++] = input()) != '\n');
```

```
            yytextbuf[yytextbufindex] = '\0';
            fprintf(fp1,"Deleted line !%s! in line
!%d!\n",yytextbuf,yylineno);
            printf("\n\n");
            unput('\n');
            }
      /* if valid subsection is found write matched text to output */
      else if (subsect == ++subsectno)
            {
            fprintf(fp1,"Found new subsection no !%d! in line
!%d!\n",subsectno,yylineno);
            ECHO;
            }
      /* if subsection is out of order then print error message and
write matched text to output */
      else
            {
            fprintf(fp1,"Error: expecting subsection !%d!, found
subsection !%d! in line !%d!\n",subsectno--,subsect,yylineno);
            ECHO;
            }
      return GO;
      }
[\n]  { ECHO; return GO;}
.     { ECHO; return GO;}
%%
main()

      {
      fp1 = fopen("preprocess.messages","w");
      BEGIN SCAN;
      while (yylex() == GO);
      fclose(fp1);
      }

yywrap()

      {
      printf("\n\nEOF\n");
      fprintf(fp1,"# of lines processed !%d!",yylineno);
      }

/* fetch chapter, section and subsection number if applicable */
void  getsect(line,chpt,sect,subsect)

      char  *line;
      int   *chpt;
      int   *sect;
      int   *subsect;

      {
      int chapter, section, subsection;
      int i;
      int pointctr = 0;
      char temp[LINELEN];

      strcpy(temp,line);
      for(i=0;i<strlen(temp);i++)
            if( temp[i] == '.' )
                  {
                  temp[i] = ' ';
                  pointctr++;
                  }
      if ( pointctr == 1)
```

```
        strcat(temp," 0");
sscanf(temp,"%d %d %d",&chapter,&section,&subsection);
*chpt = chapter;
*sect = section;
*subsect = subsection;
}       /* getsect(line,chpt,sect,subsect) */
```

Appendix 11. Printing routines of GENT

```
/* ------------------------------------------------------------------------
 *
 * File: printsect.lib.c
 * System: The liquid zone control parsing system
 *
 * Purpose: Print section details using routines from print.lib.c
 *
 * Programmer: Joozar Vasi
 * Date: 24 March 1992
 * Detail:
 *
 * ------------------------------------------------------------------------*/

/* ---------------------------- Include Files ----------------------*/

#include    <stdio.h>
#include    "header.h"

/* ---------------------------- Module Definitions -------------------*/

#define          TAGLEN 1000       /* length of character strings
holding context information */

/* ---------------------------- Imported Variables -------------------*/

extern      int   chptno;                  /* from lzcman.y */

/* ---------------------------- Imported Functions -------------------*/

extern      void  printContext1();         /* from print.lib.c */
extern      void  printContext2();         /* from print.lib.c */
extern      void  printEndContext();       /* from print.lib.c */
extern      void  printConnect();          /* from print.lib.c */
extern      void  printReq();              /* from print.lib.c */

/* ---------------------------- Exported Varibles --------------------*/

      /* none */

/* ---------------------------- Local Typedefs -----------------------*/

      /* none */

/* ---------------------------- Local Global Variables ---------------*/

static char nullstring[] = "";      /* for printing */

/* ---------------------------- Local Functions ----------------------*/

/* Print one step after another given a linked list of steps */
static
void  printsteps(tag1,tag2,header,sptr,ctr)

      char  *tag1;
      char  *tag2;
      char  *header;
      stepptr     sptr;
      int   ctr;
```

```
        {
        char  steptag1[TAGLEN];
        char  stepdesc[BTOKENLEN];
        char  stepverstring[TAGLEN];
        int   stepctr;
        char  nextsteptag1[TAGLEN];


        /* loop through all the steps */
        while (sptr)
              {

              /* Print context schemas for a step */
              sprintf(steptag1,"%s-Step-%d",tag1,sptr->stepnum);
              sprintf(stepdesc,"Step %d. %s",sptr->stepnum,sptr-
>stepdetails);
              sprintf(stepverstring,"%s Step %d",tag2,sptr->stepnum);
              printContext1(steptag1,tag2,header,stepdesc);
              stepctr = 1;
              printContext2(steptag1,tag2,header,stepctr,stepverstring);

              /* Print an equipment requirement schema if necessary*/
              if (sptr->stepequiplist)
                    printReq(steptag1,sptr->stepequiplist,++stepctr);

              /* Print end context schema */

        printEndContext(steptag1,tag2,header,++stepctr,stepverstring);

              /* Print a connect schema to end of calling
section/subsection or to next step */
              if (sptr->nextstep)
                    {
                    sprintf(nextsteptag1,"%s-Step-%d",tag1,sptr->nextstep-
>stepnum);
                    printConnect(steptag1,nextsteptag1,++stepctr,1);
                    }
              else
                    {
                    printConnect(steptag1,tag1,++stepctr,++ctr);
                    }
              sptr = sptr->nextstep;
              }
        }

/* print one subsection after another if present in a section */
static
void  printsubsections(tag1,tag2,sptr,ctr)

        char  *tag1;
        char  *tag2;
        sectptr     sptr;
        int   ctr;

        {
        char  subsecttag1[TAGLEN];
        char  subsecttag2[TAGLEN];
        char  subsectsteptag1[TAGLEN];
        char  nextsubsecttag1[TAGLEN];
        char  subsectverstring[TAGLEN];
        char  *subsectdesc;
        int   subsectctr;
```

```
      while (sptr)
            {

            /* Print context schemas for beginning of subsection */
            sprintf(subsecttag1,"%s-%d",tag1,sptr->sectnum);
            sprintf(subsecttag2,"%s.%d",tag2,sptr->sectnum);
            subsectdesc = ((sptr->sectdetails == NULL)?nullstring:sptr-
>sectdetails);
            printContext1(subsecttag1,subsecttag2,sptr-
>header,subsectdesc);
            subsectctr = 1;
            printContext2(subsecttag1,subsecttag2,sptr-
>header,subsectctr,sptr->header);

            /* Print equipment requirement schema if necessary */
            if (sptr->sectequiplist)
                  printReq(subsecttag1,sptr-
>sectequiplist,++subsectctr);

            /* Print steps in subsection if present */
            if (sptr->fstep)
                  {
                  /* Connect to first step */
                  sprintf(subsectsteptag1,"%s-Step-%d",subsecttag1,sptr-
>fstep->stepnum);

      printConnect(subsecttag1,subsectsteptag1,++subsectctr,1);
                  /* Print one step after another and a connect schema
at the end */
                  printsteps(subsecttag1,subsecttag2,sptr->header,sptr-
>fstep,subsectctr);
                  }

            /* Print end context schema */
            sprintf(subsectverstring,"%s %s",subsecttag2,sptr->header);
            printEndContext(subsecttag1,subsecttag2,sptr-
>header,++subsectctr,subsectverstring);

            /* Print a connect schema to new subsection if present else
print a connect schema to the calling section */
            if (sptr->nextsect)
                  {
                  sprintf(nextsubsecttag1,"%s-%d",tag1,sptr->nextsect-
>sectnum);

      printConnect(subsecttag1,nextsubsecttag1,++subsectctr,1);
                  }
            else
                  {
                  printConnect(subsecttag1,tag1,++subsectctr,++ctr);
                  }
            sptr=sptr->nextsect;
            }
      }

/* ------------------------ Exported Functions ----------------------*/

/* Print complete section details */
void  printsectiondetails(fsect)

      sectptr     fsect;

      {
      int   sectctr;
```

```
      char  secttag1[TAGLEN];
      char  secttag2[TAGLEN];
      char  *sectdesc;
      char  sectsubsecttag1[TAGLEN];
      char  sectsteptag1[TAGLEN];
      extern      void  printsteps();
      extern      void  printsubsections();

      /* Print context schemas for beginning of section */
      sprintf(secttag1,"%d-%d",chptno,fsect->sectnum);
      sprintf(secttag2,"%d.%d",chptno,fsect->sectnum);
      sectdesc = ((fsect->sectdetails == NULL)?nullstring:fsect-
>sectdetails);
      printContext1(secttag1,secttag2,fsect->header,sectdesc);
      sectctr = 1;
      printContext2(secttag1,secttag2,fsect->header,sectctr,fsect-
>header);

      /* Print equipment requirement schema if necessary */
      if (fsect->sectequiplist)
            printReq(secttag1,fsect->sectequiplist,++sectctr);

      /* Print steps in section if present */
      if (fsect->fstep)
            {
            /* Connect to first step */
            sprintf(sectsteptag1,"%s-Step-%d",secttag1,fsect->fstep-
>stepnum);
            printConnect(secttag1,sectsteptag1,++sectctr,1);
            /* Print one step after another and connect to section */
            printsteps(secttag1,secttag2,fsect->header,fsect-
>fstep,sectctr);
            }

      /* Print subsections if present */
      if (fsect->nextsect)
            {
            /* Connect to first subsection */
            sprintf(sectsubsecttag1,"%s-%d",secttag1,fsect->nextsect-
>sectnum);
            printConnect(secttag1,sectsubsecttag1,++sectctr,1);
            /* Print one subsection after another and connect to last
schema of section */
            printsubsections(secttag1,secttag2,fsect->nextsect,sectctr);
            }

      /* Print end context schema */
      printEndContext(secttag1,secttag2,fsect->header,++sectctr,fsect-
>header);
      }
```

```
/* ------------------------------------------------------------------------
 *
 * Module: print.lib.c
 * System: The liquid zone control manual parsing system
 *
 * Purpose: Functions used by printsectiondetails() in lzcman.y to print
 *        Schemas
 *
 * Programmer: Joozar Vasi
 * Date: 24 March 1992
 * Detail:
 *
 * ------------------------------------------------------------------------*/

/* ------------------------- Include Files ---------------------------*/

#include <stdio.h>
#include "header.h"

/* ---------------------------- Module Definitions --------------------*/

      /* none */

/* ---------------------------  Module Macros -----------------------*/

      /* none */

/* ---------------------------- Imported Variables --------------------*/

extern FILE *fp1;                    /* from main.c */

/* ---------------------------- Imported Functions --------------------*/

      /* none */

/* ---------------------------- Exported Varibles --------------------*/

      /* none */

/* ---------------------------- Local Typedefs -----------------------*/

      /* none */

/* ---------------------------- Local Global Variables ---------------*/

      /* none */

/* ---------------------------- Local Functions ----------------------*/

      /* none */

/* ---------------------------- Global Functions ---------------------*/

/* Print def-context schema for section, subsection, or step, indicating
the context of the displayed instructions */
void  printContext1(tag1,tag2,header,desc)

      char *tag1;
      char *tag2;
      char *header;
      char *desc;

      {
```

```
      fprintf(fp1,"\n\n\(defschema context-%s\n\t",tag1);
      fprintf(fp1,"\(instance-of def-context\)\n\t");
      fprintf(fp1,"\(context %s\)\n\t",tag1);
      fprintf(fp1,"\(header \"%s %s\"\)\n\t",tag2,header);
      fprintf(fp1,"\(desc \"%s\"\)\)",desc);
      fflush(fp1);
      }     /* printContext1(tag1,tag2,header,desc) */

/* Print display schema accompanying def-context schema for section,
subsection, or step */
void  printContext2(tag1,tag2,header,ctr,verstring)

      char *tag1;
      char *tag2;
      char *header;
      int ctr;
      char *verstring;

      {
      fprintf(fp1,"\n\n\(defschema display-%s-a\n\t",tag1);
      fprintf(fp1,"\(instance-of display\)\n\t");
      fprintf(fp1,"\(context \(%s %d\)\)\n\t",tag1,ctr);
      fprintf(fp1,"\(header \"%s %s\"\)\n\t",tag2,header);
      fprintf(fp1,"\(desc \"Proceed with %s Verification.\"\)\)"
,verstring);
      fflush(fp1);
      }     /* printContext2(tag1,tag2,header,ctr,verstring) */

/* print an equipment requirement schema from linked list of equipment
details */
void  printReq(tag1,ptr,ctr)

      char *tag1;
      equipptr ptr;
      int ctr;
      {
      int   equipctr;

      if (ptr != NULL)
            {
            fprintf(fp1,"\n\n\(defschema req-%s\n\t",tag1);
            fprintf(fp1,"\(instance-of requirement\)\n\t");
            fprintf(fp1,"\(context \(%s %d\)\)\n\t",tag1,ctr);
            fprintf(fp1,"\(equipment-list");
            equipctr = 0;
            while(ptr != NULL)
                  {
                  equipctr++;
                  fprintf(fp1," \(");
                  fprintf(fp1,"%d",equipctr);
                  fprintf(fp1," %s",ptr->equipname);
                  fprintf(fp1," %s",ptr->equipattribute);
                  fprintf(fp1," \"%s\"",ptr->equipoperator);
                  fprintf(fp1," %s",ptr->equipvalue);
                  fprintf(fp1,"\)");
                  ptr=ptr->next;
                  }
            fprintf(fp1,"\)\)");
            fflush(fp1);
            }
      }     /* printReq(tag1,ptr,ctr) */

/* Print ending context schema for section, subsection, or step */
void  printEndContext(tag1,tag2,header,ctr,compstring)
```

```
        char  *tag1;
        char  *tag2;
        char  *header;
        int   ctr;
        char  *compstring;

        {
        fprintf(fp1,"\n\n\(defschema display-%s\n\t",tag1);
        fprintf(fp1,"\(instance-of display\)\n\t");
        fprintf(fp1,"\(context \(%s %d\)\)\n\t",tag1,ctr);
        fprintf(fp1,"\(header \"%s %s\"\)\n\t",tag2,header);
        fprintf(fp1,"\(proc-stream \"Completed\"\)\n\t");
        fprintf(fp1,"\(desc \"%s completed\"\)\)",compstring);
        fflush(fp1);
        }     /* printEndContext(tag1,tag2,header,ctr,compstring) */

/* Print connect schema to connect from one context to another */
void  printConnect(fromTag1,toTag1,fromCtr,toCtr)

        char  *fromTag1;
        char  *toTag1;
        int   fromCtr;
        int   toCtr;

        {
        fprintf(fp1,"\n\n(defschema connect-%s-a\n\t",fromTag1);
        fprintf(fp1,"\(instance-of connect\)\n\t");
        fprintf(fp1,"\(context \(%s %d\)\)\n\t",fromTag1,fromCtr);
        fprintf(fp1,"\(go-context \(%s %d\)\)\)",toTag1,toCtr);
        fflush(fp1);
        }     /* printConnect(fromTag1,toTag2,fromCtr,toCtr) */
```

Appendix 12. Header file for GENT

```
/* ------------------------------------------------------------------------
 *
 * File: header.h
 * System: Liquid zone control manual parsing system
 *
 * Purpose: Header file for all files in this system
 *
 * Programmer: Joozar Vasi
 * Date: 24 Oct. 1992
 * Detail:
 *
 *----------------------------------------------------------------------*/

 /* --------------------------- Global Definitions --------------------*/

#define BTOKENLEN 20000 /* maximum length of paragraph token */
#define STOKENLEN 500   /* maximum length of all other tokens */
#define FRONTQ 100      /* append element of linked list in front */
#define BACKQ 200       /* append element of linked list at the back */
#define EQUIPLEN 50     /* maximum length of equipment item details */

/* ---------------------------- Global Macros ------------------------*/

      /* none */

/* ---------------------------- Global Typedefs ----------------------*/

typedef int flag;

typedef struct enode    {           /* a basic equipment node */
      flag newnode;                 /* flag indicating fresh node */
      char *equipname;              /* the name of equipment */
      char *equipattribute;         /* the attribute of equipment item
to be checked */
      char *equipoperator;          /* indication of relation between
equipment attribute and equipment value */
      char *equipvalue;             /* the value of the equipment
attribute */
      struct enode *next;           /* next entry in chain */
      } equipnode, *equipptr;

typedef struct stnode
      {                             /* basic step node */
      int stepnum;                  /* step number */
      char *stepdetails;            /* step details */
      equipptr stepequiplist;       /* equipment requirement details */
      struct stnode *nextstep;      /* next step if present */
      } stepnode, *stepptr;

typedef struct senode
      {                             /* basic section/subsection node */
      int sectnum;                  /* section/subsection number */
      char *header;                 /* section/subsection header */
      char *sectdetails;            /* section/subsection details if
present */
      stepptr fstep;                /* first step in section/subsection
if present */
      stepptr lstep;                /* last step in section/subsection
if present */
```

```
      equipptr sectequiplist;       /* equipment requirement details */
      struct senode *nextsect;      /* next section/subsection if
present*/
      } sectnode, *sectptr;

/* ---------------------------- External Variables --------------------*/

      /* none */

/* ---------------------------- External Functions --------------------*/

      /* none */

/* ------------------------------------------------------------------------*/
```

Appendix 13. Utilities used by GENT

```
/* ------------------------------------------------------------------------
 *
 * File: utilities.lib.c
 * System: The liquid zone control manual parsing system
 *
 * Purpose: Certain utility functions
 *
 * Programmer: Joozar Vasi
 * Date: 24 March 1992
 * Detail:
 *
 * ------------------------------------------------------------------------*/
/* ---------------------------- Include Files ----------------------------*/

#include <stdio.h>
#include <strings.h>
#include <malloc.h>
#include "header.h"

/* --------------------------- Module Definitions -------------------------*/

      /* none */

/* ---------------------------- Module Macros -----------------------------*/

      /* none */

/* ---------------------------- Imported Variables ------------------------*/

      /* none */

/* ---------------------------- Imported Functions ------------------------*/

      /* none */

/* ---------------------------- Exported Varibles -------------------------*/

      /* none */

/* ---------------------------- Local Typedefs ----------------------------*/

      /* none */

/* ---------------------------- Local Global Variables --------------------*/

      /* none */

/* ---------------------------- Local Functions ---------------------------*/

/* allocate an intialized  node that holds section details */
static
sectptr     sectalloc()

      {
      sectptr p;

      p=(sectptr) malloc(sizeof(sectnode));
      p->sectnum = 0;
      p->header = NULL;
```

```
        p->sectdetails = NULL;
        p->fstep = NULL;
        p->lstep = NULL;
        p->sectequiplist = NULL;
        p->nextsect = NULL;
        return(p);
        }       /* sectalloc() */

/* allocate an initialized node that holds step details */
static
stepptr     stepalloc()

        {
        stepptr p;

        p=(stepptr) malloc(sizeof(stepnode));
        p->stepnum = 0;
        p->stepdetails = NULL;
        p->stepequiplist = NULL;
        p->nextstep = NULL;
        return(p);
        }       /* stepalloc() */

/* free memory of equipment requirement details */
void  equipfree(ptr)

        equipptr    ptr;

        {
        equipptr ptr1,ptr2;

        ptr1=ptr;
        while (ptr1)
              {
              free(ptr1->equipname);
              free(ptr1->equipattribute);
              free(ptr1->equipoperator);
              free(ptr1->equipvalue);
              ptr2=ptr1->next;
              free(ptr1);
              ptr1=ptr2;
              }
        }

/* free memory of step details */
static
void  stepfree(ptr)

        stepptr     ptr;

        {
        extern      void  equipfree();
        stepptr ptr1,ptr2;
        equipptr ptr3;

        ptr1 = ptr;
        while(ptr1)
              {
              ptr2=ptr1->nextstep;
              free(ptr1->stepdetails);
              ptr3 = ptr1->stepequiplist;
              equipfree(ptr3);
              free(ptr1);
              ptr1=ptr2;
```

```
            }
        }

/* ------------------------- Exported Functions ---------------------*/

/* free memory of section details */
void  sectfree(fsect,lsect)

      sectptr *fsect;
      sectptr *lsect;

      {
      extern      void  equipfree();
      sectptr ptr0,ptr1;
      stepptr ptr2;
      equipptr ptr3;

      ptr0 = *fsect;
      while (ptr0)
            {
            ptr1 = ptr0->nextsect;
            free(ptr0->header);
            free(ptr0->sectdetails);
            ptr2 = ptr0->fstep;
            stepfree(ptr2);
            ptr3 = ptr0->sectequiplist;
            equipfree(ptr3);
            free(ptr0);
            ptr0=ptr1;
            }
      *fsect = NULL;
      *lsect = NULL;
      }           /* sectfree(fsect,lsect) */

/* fetch chapter, section and subsection number if applicable */
void  getsect(line,chpt,sect,subsect)

      char  *line;
      int   *chpt;
      int   *sect;
      int   *subsect;

      {
      extern      char *strsave();        /* in this file */
      int chapter, section, subsection;
      int i;
      int pointctr = 0;
      char *temp;

      temp = strsave(line);
      for(i=0;i<strlen(temp);i++)
            if( temp[i] == '.' )
                  {
                  temp[i] = ' ';
                  pointctr++;
                  }
      if ( pointctr == 1)
            strcat(temp," 0");
      sscanf(temp,"%d %d %d",&chapter,&section,&subsection);
      *chpt = chapter;
      *sect = section;
      *subsect = subsection;
      free(temp);
      }     /* getsect(line,chpt,sect,subsect) */
```

```
/* put a node that holds subsectiondetails in linked list */
void  addtosectlist(fsect,lsect,sectno,head,details,where)

      sectptr     *fsect;
      sectptr     *lsect;
      int         sectno;
      char        head[];
      char        details[];
      flag        where;

      {
      extern sectptr sectalloc();         /* in this file */
      sectptr ptr;

      ptr = sectalloc();
      ptr->sectnum = sectno;
      ptr->header = head;
      ptr->sectdetails = details;
      if (*lsect == NULL)
            *fsect = *lsect = ptr;
      else if (where == FRONTQ)
            {
            ptr->nextsect = *fsect;
            *fsect = ptr;
            }
      else
            {
            (*lsect)->nextsect = ptr;
            *lsect = ptr;
            }
      }     /* addtosectlist(fsect,lsect,subsect,head,details,where) */

/* put a node that holds step details in linked list */
void  addtosteplist(firststep,laststep,stepno,details)

      stepptr     *firststep, *laststep;
      int         stepno;
      char        *details;

      {
      extern stepptr stepalloc();
      stepptr ptr;

      ptr = stepalloc();
      ptr->stepnum = stepno;
      ptr->stepdetails = details;
      if (*laststep == NULL)
            *firststep = *laststep = ptr;
      else
            {
            (*laststep)->nextstep = ptr;
            *laststep = ptr;
            }
      }     /* addtosteplist(firststep,laststep,stepno,details) */


/* save string s somewhere */
char  *strsave(s)

      char *s;
```

```
        {
        char *p;

        if ((p = malloc(strlen(s)+1))   != NULL)
                {
                strcpy(p,s);
                return(p);
                }
        }       /* *strsave(s) */

/* find number of contiguous newline characters in string */
int     findnlchars(str)

        char *str;

        {
        if (index(str,'\n') != NULL)
                return ( (rindex(str,'\n') - index(str,'\n')) + 1 );
        else
                return 0;
        }       /* findnlchar(str) */

/* open a file or print error message */
FILE  *efopen(file,mode)
        char  *file, *mode;

        {
        FILE  *fp, *fopen();
        extern      char *progname;

        if ((fp = fopen(file,mode)) != NULL)
                return fp;
        fprintf(stderr, "%s: can't open file %s mode
%s\n",progname,file,mode);
        exit(1);
        }       /* *efopen(file,mode) */
```

```
/* ------------------------------------------------------------------------
 *
 * Module: equiputilities.lib.c
 * System:
 *
 * Purpose: Functions for collecting equipment item details.
 *
 * Programmer: Joozar Vasi
 * Date: 24 March 1992
 * Detail:
 *
 * ------------------------------------------------------------------------*/

/* ---------------------------- Include Files ----------------------*/

#include <stdio.h>
#include <malloc.h>
#include "header.h"

/* ---------------------------- Module Definitions -------------------*/

      /* none */

/* ---------------------------- Imported Variables -------------------*/

      /* none */

/* ---------------------------- Imported Functions -------------------*/

extern      char  *strsave();       /* from utilities.lib.c */
extern      FILE  *efopen();        /* from utilities.lib.c */

/* ---------------------------- Local Typedefs -----------------------*/

      /* none */

/* ---------------------------- Local Global Variables ---------------*/

      /* none */

/* ---------------------------- Local Functions ----------------------*/


/* allocate a equipment node somewhere */
static
equipptr elistalloc()

      {

      equipptr    ptr;

      ptr=(equipptr)malloc(sizeof(equipnode));
      ptr->newnode=0;
      ptr->equipname=NULL;
      ptr->equipattribute=NULL;
      ptr->equipoperator=NULL;
      ptr->equipvalue=NULL;
      ptr->next=NULL;
      return(ptr);
      }     /* elistalloc() */


/* ---------------------------- Exported Functions -------------------*/
```

```
/* read equipment details from temp2 and store in linked list */
equipptr    getequiplist()

      {
      FILE  *ft;
      int   equipctr;
      char
      ename[EQUIPLEN],eattribute[EQUIPLEN],eoperator[EQUIPLEN],evalue[EQ
UIPLEN];
      equipptr    frontptr,backptr;
      extern      void  addtoequiplist();

      frontptr = backptr = NULL;
      ft = efopen("temp2","r");
      while(fscanf(ft,"%d %s %s %s
%s",&equipctr,ename,eattribute,eoperator,evalue) != EOF)
            {
            addtoequiplist(&frontptr,&backptr,ename);
            backptr->equipname = strsave(ename);
            backptr->equipattribute = strsave(eattribute);
            backptr->equipoperator = strsave(eoperator);
            backptr->equipvalue = strsave(evalue);
            }
      fclose(ft);
      return frontptr;
      }     /* getequiplist() */


/* add to linked list of equipment details a new node */
void addtoequiplist(frontptr,backptr,equip)

      equipptr *frontptr,*backptr;
      char  *equip;

      {
      equipptr    ptr;

      ptr = elistalloc();
      ptr->equipname = strsave(equip);
      if (*frontptr == NULL)
            *frontptr = *backptr = ptr;
      else
            {
            (*backptr)->next = ptr;
            *backptr = ptr;
            }
      }     /* addtoequiplist(frontptr,backptr,equip) */

      /* update all new equipment detail nodes by current attribute,
operator, and state */
void  addinfotoequiplist(ptr,attribute,operator,value)

      equipptr    ptr;
      char        *attribute;
      char        *operator;
      char        *value;

      {

      while (ptr)
            {
            if (ptr->newnode == 0)
                  {
                  ptr->newnode = 1;
```

```
                ptr->equipattribute = strsave(attribute);
                ptr->equipoperator = strsave(operator);
                ptr->equipvalue = strsave(value);
                }
            ptr=ptr->next;
            }
        }     /* addinfotoequiplist(ptr,attribute,operator,value) */

        /* print  equipment requirement details from linked list to file
*/
void  printreqtofile(ptr)

        equipptr ptr;
        {
        int   equipctr;

        equipctr=0;
        while(ptr)
            {
            equipctr++;
            printf("%d",equipctr);
            printf(" %s",ptr->equipname);
            printf(" %s",ptr->equipattribute);
            printf(" %s",ptr->equipoperator);
            printf(" %s\n",ptr->equipvalue);
            ptr=ptr->next;
            }
        }     /* printreqtofile(ptr) */
```

Appendix 14. Driver programs for GENT

```
/* ------------------------------------------------------------------------
 *
 * File: main.c
 * System: The liquid zone control manual parsing system
 *
 * Purpose: Call yyparse() and output files for translated text and
diagnostic
 *          messages from yylex()
 * Programmer: Joozar Vasi
 * Date: 13 April 1992
 * Detail:
 *
 * ------------------------------------------------------------------------*/
/* ---------------------------- Include Files ---------------------------*/

# include <stdio.h>

/* ---------------------------- Module Definitions ------------------*/

      /* none */

/* ---------------------------- Imported Variables ------------------*/

      /* none */

/* ---------------------------- Imported Functions ------------------*/

extern      int   yyparse();  /*from y.tab.c */
extern      FILE  *efopen();  /*from lexutilities.lib.c*/

/* ---------------------------- Exported Varibles -------------------*/

FILE  *fp;              /* a file for writing Lex messages */
FILE  *fp1;             /* a file for writing translation */
char  *progname;        /* program name  */

/* ---------------------------- Local Typedefs ----------------------*/

      /* none */

/* ---------------------------- Local Global Variables --------------*/

      /* none */

/* ---------------------------- Local Functions ---------------------*/

      /* none */

/* ---------------------------- Local Functions ---------------------*/

      /* none */


main(argc,argv)

      int argc;
      char *argv[];

      {
```

```
        extern FILE *fp;
        extern FILE *fp1;
        extern char *progname;

        progname = argv[0];
        fp = efopen("lexoutput","w");
        fp1 = efopen("artoutput","w");
        yyparse();
        fclose(fp);
        fclose(fp1);
        }


/* ------------------------------------------------------------------------
 *
 * Module: main1.c
 * System: The liquid zone control manual parsing system
 *
 * Purpose: Call yylex() and open output files for equipparse
 *
 * Programmer: Joozar Vasi
 * Date: 13 April 1992
 * Detail:
 *
 * ------------------------------------------------------------------------*/
/* ---------------------------- Include Files ----------------------*/

# include <stdio.h>

/* ---------------------------- Module Definitions -------------------*/

        /* none */

/* ---------------------------- Imported Variables -------------------*/

        /* none */

/* ---------------------------- Imported Functions -------------------*/

extern      int   yylex();    /*from lex.yy.c */
extern      FILE  *efopen();  /*from utilities.lib.c */

/* ---------------------------- Exported Varibles --------------------*/

char  *progname;          /* program name  */
FILE  *fp;                /* for diagnostic messages from yylex() */

/* ---------------------------- Local Typedefs -----------------------*/

        /* none */

/* ---------------------------- Local Global Variables ---------------*/

        /* none */

/* ---------------------------- Local Functions ----------------------*/

        /* none */

/* ---------------------------- Local Functions ----------------------*/

        /* none */
```

```
main(argc,argv)

        int argc;
        char *argv[];

        {
        progname = argv[0];
        fp = efopen("lexoutput1","w");
        yylex();
        fclose(fp);
        }
```

VITA

Candidate's full name:	Joozar Khozem Vasi
Place and date of birth:	Surat, Gujarat, India October 26, 1962
Permanent address:	42, Himgiri Peddar Road Bombay 26, India
Schools Attended:	Campion School, Bombay, Maharashtra, India January 1969 - December 1977
Universities attended:	K. C. College, Bombay, Maharashtra, India May 1978 - May 1984
	Northwest Technical College, Archbold, Ohio, U.S.A. September 1984 - Febuary 1985
	Stevens Institute of Techonology, Hoboken, N.J., U.S.A August 1986 - May 1988
	University of New Brunswick, Fredericton, N.B., Canada September 1988 - August 1992
Publications:	B. G. Nickerson,K. Ward and J. K. Vasi, "Semi-automated Knowledge Acquisition From Plant Operating Procedures", Final Report for AECL Research, Chalk River, Ontario, December 3, 1991, 59 pages.
	J. K. Vasi and K. Ward, "Literature search for computer-assisted operating manuals and representation languages", Contract Report for AECL Research, Chalk River, Ontario, December 2, 1991, 20 pages