1989

A Natural language interface to MS-DOS

Donna Indovina Blodgett

Rochester Institute of Technology
School of Computer Science

A NATURAL LANGUAGE INTERFACE to MS-DOS

by
Donna Blodgett Indovina

A thesis, submitted to
The faculty of the School of Computer Science and Technology,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science.

Approved by:

Kevin Donaghy

Al Biles

Peter Anderson

July 6, 1989

A Natural Language Interface
to MS_DOS

I, Donna Blodgett Indovina, hereby grant permission to the Wallace Memorial Library of R.I.T. to reproduce my thesis in whole or in part. Any reproductions will not be used for commercial use or profit.

Signed:_____

ACKNOWLEDGMENTS

I would like to express my gratitude to my parents Edward and JoAnn Blodgett and my husband Robert Indovina for their love, encouragement and support throughout this educational endeavor.

ABSTRACT

The goal of this thesis was to design and implement a dialog facility to assist novice users of the MS-DOS operating system. A small natural language interface facility called PE-DOS (Plain English DOS) was built, which serves as a front end for a subset of simple MS-DOS commands. PE-DOS accepts English-like terminology and translates it into the appropriate MS-DOS command using a system of recursive transition networks (RTNs). The translated command then is presented to the user along with an English paraphrase of the user's input sentence in an effort to provide confirmation for the intended command in both English and MS-DOS command language.

PE-DOS was tested by presenting it to two classes of users. One group was made up of sophisticated users of other operating systems who had not used MS-DOS. The second group consisted of unsophisticated users of applications that run under MS-DOS who had not used any operating system commands. The feedback from these user indicates that PE-DOS is probably most useful as an educational device to teach new users a set of basic MS-DOS commands, rather than as a viable command interface.

Table of Contents

1. Introduction.....1

2. Types of User Interfaces.....4
   2.1 Command Languages.....4
   2.2 Menus.....6
   2.3 Icons.....8
   2.4 Natural Languages.....11
   2.5 Implementations of MS-DOS Interfaces.....15

3. An Interface to MS-DOS.....20
   3.1 An Introduction to MS-DOS.....20
   3.3 Goals of a truly Friendly System.....21
   3.3 Target Group.....26

4. Project Development.....28

5. User Feedback.....48

6. Enhancements.....54
   6.1 User Interactions.....54
   6.2 System Design.....62

7. Conclusions.....66

8. Bibliography.....69

9. Appendixes.....73
   appendix A: Recursive Transition Networks.....74
   appendix B: PE_DOS code.....92
   appendix C: Script of PE_DOS 151

CHAPTER 1

INTRODUCTION

The computer operating system can be viewed as a user interface to the system's hardware. Individuals who are interested only in running high level application programs, such as Lotus 123, Wordperfect, or Dbase III, have little need to interact with the operating system, except to load their applications. Other users may require greater interaction with the operating system and will find that their dialogue with it will significantly affect their productivity and enjoyment of system usage [HAYW, 83].

Natural Language Processing is one of the most important tasks underway in Artificial Intelligence. Once the barrier between the human user and the digital machine has been broken, the door to direct dialogs will be opened, bypassing traditional computer command languages. If we can communicate with the computer in our own language, we then can eliminate the time required to learn the monotonous command structures that the computer now requires.

At the present time, the complexities of the English language have hindered the full development of such systems [Lane, 87 & Schi, 87]. The development of Natural Language Processing Systems will serve as an intermediate step to future applications where speech/voice recognition will be the only interface needed with the computer.

The general goal of this thesis was to design an improved dialog facility that would assist in maximizing the benefits of the MS-DOS operating system for its users. The interface should contain English-like terminology and assist in alleviating some of the common anxiety and frustration attributed to learning and working with this operating system. It should accept English sentences or phrases as commands and then translate them into the appropriate command language structure to be executed. Ideally, this interface would allow users the ability to communicate with the computer via their own natural language. Virtually all that must be known about the system is what it is capable of performing (i.e. directory listings, copying files, etc.).

MS-DOS is the most popular operating system for IBM personal computers and compatibles. The majority of home users and businesses use this operating system; however, due to the nontechnical background of its typical users, dialogs with it can be a nightmare. Even highly skilled technical individuals who are exposed to MS-DOS infrequently (or any other system for that matter) experience some form of frustration due to differences when switching between operating environments. Interfaces to this system in the form of menus, icons and natural language have, therefore, been developed to ease some of the confusion (see Types of User Interfaces).

The obvious goal is to make the MS-DOS system easier to use. PE-DOS (Plain English DOS) was developed for the

purpose of this thesis, not to replace the MS-DOS command language, as many of the commercial products claim, but to be utilized as an educational device. The system always paraphrases the user's request and presents the appropriate MS-DOS command. It is up to the user, at the start of a session, to decide whether the system will be used strictly for tutorial purposes or to actually execute the command. For efficiency purposes, it is assumed that eventually the user will be weaned from the use of such a system to interact directly with MS-DOS.

The rest of the thesis is organized as follows. Chapter two investigates four common types of user interfaces available in an operating system environment. Chapter three gives a general overview of an ideal natural language interface to MS-DOS including the use of MS-DOS, aspects of user friendliness and the targeted user group. Chapter four discusses the project development. Chapter five summarizes feedback given during sessions with typical users. Chapter six explores possible system enhancements and Chapter seven concludes the thesis.

CHAPTER 2

TYPES OF USER INTERFACES

There are several types of user interfaces available to system designers. This paper is concerned with those useable in an operating system environment. The characteristics, advantages and disadvantages of the following four such interfaces will be discussed: Command Languages, Menus, Icons, and Natural Languages.

2.1 Command Languages

Characteristics

Command languages are artificial languages. They closely resemble programming languages in that they can be crude, frustrating, painstakingly monotonous and time consuming to memorize, especially to the casual, infrequent computer user. However, command languages can be attractive to the frequent user of a system, due to that user's acquired knowledge of the task domain and their ability to comprehend computer terminology [SHNE, 87].

MS-DOS is a typical example of a command language used as an interface between a human user and the computer hardware. It requires precise syntax of commands using English-like directives (i.e. copy for copying files, dir for directory), yet the language used to communicate is not

natural to humans. It is something that must be learned, and this learning experience often causes headaches for users.

Advantages

Command languages can be attractive to the user who is already very astute in their use. The experienced user may find that typing in a single command can be faster then wading through a series of cumbersome menus. They are more flexible and efficient, and the user can specify parameter differences or numerous functions that can be combined with ease. These languages are also much faster for the machine, especially when display rates are slow.

Though a significant amount of time is spent training and memorizing the syntax, individuals with a commitment to learn will achieve high productivity. The users must fully understand the nature of the commands they are issuing as well as their consequences. It is also common for command languages to offer little structure. This shortcoming, however, can be viewed as beneficial due to the greater amount of flexibility that is attained.

Disadvantages

Command languages can differ from one system to another. The time to develop a working vocabulary of commands for a particular system can be lengthy due to the strenuous nature of their structure. Most command languages share inadequate commonalities such as a lack of

descriptiveness in their wording or abbreviations, an overly restrictive or limiting vocabulary of expression, rigid syntax, and complex and confusing abstractions. Along with the pompous and unnatural dialogs they engender, they lack the necessary ingredients of a truly appropriate human-computer relationship.

2.2 Menus

Characteristics

Menus may be incorporated in a command interface to present all the permissible options to the user. A menu selection system can be thought of as a tree structure or network, with the root node and intermediate nodes being menu of options, and the leaf nodes reporting on a solution or executing a desired command, depending on what kind of system has been employed. Individual menus display a descriptive text of the available branch options. Decisions made will affect the order in which the network of menus is navigated.

Several design alternatives for menu formats are available. A fill-in-the-blank style would be appropriate for a data input environment, but a series of menus containing choices would be typical for an operating system.

Advantages

A function of a menu is to provide informative descriptions and instructions for implementation by the

inexperienced user. Menus often can be very useful in situations where the range of commands is very limited or restricted. The total number of keystrokes typed by the user is also reduced in comparison to command languages.

Since the user makes decisions based on a set of alternative choices, training time can be cut, and the need for memorization of commands is reduced or even eliminated.

Disadvantages

Experienced users may complain that this type of interface may slow them down because of the monotonous series of menus. The possibility exists for the menu scheme to become so cumbersome, due to several layers of submenus, that users become lost or confused as to where they are in the system.

Reading time is another factor that needs to be considered in this type of system. A user unfamiliar with the menu system will have to spend time reading the available options. The amount time spent on reading may well exceed the time saved from the reduction of keystrokes. Typing at the system's command language prompt may in fact, prove less time consuming, especially when the user knows what command to issue. After a period of time, an individual usually memorizes the structure of the menu and thus types the selections without even waiting for the menu to appear on the screen. It seems more productive to train individuals in the use of the command language, which would,

in the long run, greatly increase efficientcy.

The menu strategy is also extremely inflexible, limiting the user to a set of admissible actions. In time this may cause suspicion on the part of the user who may surmise that a certain function unavailable in the menus could be performed, or perhaps a function that is available may be performed more efficiently (i.e. easier access). On a broader scale, the menu structure does not allow thought processes or experimentation on the part of the user.

2.3 Icons

Characteristics

Icons are standardized pictographic symbols and signs used as universal recognition for highways, airports, electronics, packaging, etc. Icons possess a general descriptive power that transcends language barriers and reading abilities. They are used to represent objects that are found in "real" world situations. Gittins (1986) interestingly notes that the International Standards Organization (ISO) approach is to create a written statement for any icon, which describes:

- The graphic contents of the symbol;
- The underlying function/object it represents;
- The fields to which it may be applied.

The ultimate goal of the ISO is to develop some overall consistency in the designs, although the end icons may differ somewhat.

These pictorial elements also can be used successfully to represent objects in computer science as graphic-based interfaces. The concepts behind using icons are similar to those behind the menu selection approach; a series of options are displayed for the user, and a tree structure is used to organize the commands.

Additionally, pointing devices such as mice can be incorporated into icon based systems. The mouse is a tool that offers a convenient alternative to keyboard input to interface with a system. This device, as well as others, should be an available option, rather than a mandatory user requirement, to allow experienced users the opportunity for more personal use on the DOS level.

Advantages

Icons seem to have had their share of success. They easily lend themselves to the popular pointing devices such as mice, which eliminate the drudgery of manually typing commands or memorizing complex syntax while improving the speed of the system. They have been designed to be self-explanatory and generally are developed using the ISO standards.

Icons are extremely easy to understand and use. They offer the path of least resistance and are easily grasped without extensive explanation and effort. Most people respond to them for the same reason that they

watch the news on television rather than reading the newspaper - it's easier. They simply don't want to be bothered with the monotonous memorization of computer commands. With icons, learning time is relatively short and painless, so that potential and infrequent users need only a brief demonstration of the capabilities of the system.

Design of icon-based interfaces is becoming more prevalent and standardized to meet the growing demand and need of new users who have willingly or unwillingly entered the unavoidable realm of computers. Computers are becoming more and more a part of our lives, and it is only natural that we adapt them to be more like us. Icons are a universal language we can readily understand.

Disadvantages

Despite the excitement over icon interfaces, there are some who are still not impressed by their prospects. Fisher (1987) says that "It's condescending. Someone out there thinks you can't understand simple English." Many long-time programmers and/or computer experts who are familiar with a particular system prefer the use of the keyboard; they believe the mouse (or other pointing device) only gets in the way, and that they would be much more efficient and productive without those "pesty creatures."

2.4 Natural Languages

Characteristics

"The goal of natural language understanding is to allow computers to understand people as well as people understand people" [Mish, 85]. Unlike the traditional forms of communications with a computer, a true natural language interface should allow users the ability to communicate with the computer via their own natural language (i.e. English). This concept of an interface design may enhance the user's ability and desire to utilize the power of the computer system due to its ease and simplicity of use. The user only needs a basic understanding of the computer's functional capabilities. Much work still needs to be done if this goal is ever to be realized fully.

A natural language understanding system must be able to comprehend natural language input from the user, as well as to compose natural language output. In artificial intelligence these processes are termed natural language understanding and generation respectively. The generation of a natural language sentence can be relatively straightforward, given the context of the message and the appropriate rules of grammar; however, natural language comprehension can be a definite hassle. The computer has difficulty in resolving ambiguity, imprecision, incomplete sentences,

and other miscellaneous inaccuracies that people tend to overlook in human to human conversations. In other words, the computer requires precision in its communications, whereas the human is more flexible.

To overcome the difficulties of natural language understanding, humans, due to their acquired "world" knowledge, develop a perception or are able to put the subject into context. They also develop a sense of familiarity for the situation and expectations due to their past experiences. Attempts have been made to simulate these aspects of human intelligence. Stevens (1985) questions that if we could create a machine that impersonates a human, then can that machine be said to bear human-like intelligence? He goes on to argue that human intelligence is presumably greatly different from that of a digital machine. Therefore, the way in which the human mind works may be an inefficient algorithm for the machine.

Given a natural language interface, how can we be sure if the computer really understands what has been intended? If there is correct understanding, then the appropriate action will be performed. For example, if we go into a restaurant and order chicken, and the waitress brings us a steak, we know that there was not correct understanding. If she brings us chicken, then correct understanding is implied.

Advantages

A natural language interface would be highly desirable for occasional users who lack the time needed to acquire proficiency in the system's use or those unfamiliar with a particular system's syntax. Conceptually, a natural language system should allow the same flexibility as the command language, only worded in English. Though the user's memory is not taxed by having to remember the syntax of the traditional query language, they would still need to comprehend the underlying abilities of the system. In essence, a natural language interface would perform a simulation of recognition; the user would type a command using "normal" human syntax and, through some translation process (using key words for example), generate the appropriate command to be invoked at the operating system level.

Future developments in the natural language processing arena could lead to dramatic advancements. Perhaps it is not inconceivable to imagine a computer such as H.A.L. from 2001: A Space Odyssey, which can carry on human-like conversations. It clearly would be a productivity gain if we could verbally tell the computer what to do, freeing up the use of our hands for other tasks. Natural language processing is a step in this direction.

Disadvantages

The main disadvantage of a natural language interface is that the number of keystrokes required are dramatically increased. For the prudent user who is used to quick, abbreviated commands, a natural language system would clearly not enhance "ease of use". These users would be annoyed at having to type "show files" when all that is traditionally needed is "dir".

Spelling is another concern. With the additional typing involved, the possibility of mis-typing is magnified. If the system were capable of understanding common misspellings (like a spell checker for a word processing program), the user would need to be prompted for each mis-typed word's correction. If this were not done, then we are allowing the system to derive what "it" thinks we mean, which may lead to disastrous results.

Natural languages, especially English, are full of complex and ambiguous terminology. There are many ways of relaying the same meaning. A natural language system should be able to cope with all the variations of a command sentence; however, on the practical side, limitations may have to be imposed. Misinterpretation of a command may prove common, thus creating undesirable results if not carefully monitored.

Though the command language may seem at times

monstrous to learn, it is at least precise; the command entered has a single exact meaning. On the other hand, natural language systems could require an exorbitant amount of time to enter the command, check and verify possible misspellings, and confirm potential misinterpretations. Even then there is still no guarantee of the success of the translation.

Though the conception of systems that carry on actual conversation may appear desirable, Nickerson (1981) points out that the development of human-like conversational capabilities for human-computer interaction must await further developments in the techniques for natural language processing on the computer as well as a better understanding of interpersonal communication.

2.5 Implementations of MS-DOS Interfaces

DOS has repeatedly been condemned for being a user "unfriendly" system. Operating systems, in particular PC-DOS and MS-DOS, can be a "beast to learn." [Beec, 87] DOS shells that rescue the user from the need to type in lengthy, memory-taxing commands are available. Many of these shells are menu based windowing packages, and they offer a range of features that insulates the user from the annoying exactness of DOS. Recently, natural language packages also have entered the market

place, allowing users to issue commands in their own language.

GEM (Graphics Environment Manager), is a product of Digital Research Corporation. The GEM package extends some of the friendly icon appeal of Apple's Macintosh and Atari's ST due to its bit-mapped graphics display. GEM behaves as a front end menu to DOS. It is also capable of calling various application programs. Aside from icons, GEM incorporates the use of dropdown menus, adjustable size windows, a pop up calculator, a clock, and a print spooler. It also gives the user the option of using a mouse or keyboard.

DESQview, a product of Quarterdeck Office Systems, uses a limited menu approach to interface with DOS due to the restrictions of using only the standard character set. The menu scheme allows the user to issue DOS commands and to call applications. It is able to support windowing and multitasking, as well as cutting and pasting. DESQview also gives the user the option of using either a mouse or keyboard.

Naturallink, developed by Texas Instruments, uses menu options and windowing, within what seems to be a natural language environment. This package allows the user to create a proper English sentence from a series of menus containing options of English phrases. [MISH, 85] It is, therefore, not a true natural language

system, but a menu system bastardizing the concepts of what natural language processing is. Since performance is based on a series of menu selections, what you can't see, you can't do. This may eliminate some of the ambiguities of natural language, but, it does not allow the user the expressiveness of true natural language.

DOSTALK, as it implies by its name, offers a "plain English" interface to DOS. Developed by SAK Technologies, it enables users to simply type their commands as "normal" English sentences instead of following rigid computer syntax. DOSTALK attaches itself to the operating system, acting for the most part, as a superset of DOS. (DOSTALK allows the user to issue MS-DOS commands as well as additional facilities such as a product defined undo command.)

DOSTALK is an attempt to create a friendlier, human-like dialog between the user and the DOS environment. The product does offer some additional niceties [RICH, 88] such as: prompting the user for optional parameters of the given command in a menu form; however, this method swallows more time than it is probably worth. DOS would treat these missing parameters as purposely missing and, therefore, use the system's defaults. An undo function also exists for the user who inadvertently destroys information; however, recovery is not guaranteed. The entire hard

disk, not only the current directory, will be searched when a file is declared in a command. There is no need to specify a directory either. The product also serves as a tutorial; it displays the executed DOS command to the user. The casual user tends to delve into their applications, with little thought about the power of the computer system. In light of this, DOSTALK will not alleviate the basic necessity for users to achieve some understanding of the operating system on their own.

A set of restrictive rules are placed upon the product's performance, thus requiring some memorization of syntax. This contradicts one of the principle justifications for the use of natural language. Though the DOSTALK prompt looks just like the DOS prompt, and DOS commands can be issued from within the product, when issuing a true DOS command, it must be preceeded by the $ symbol. When specifying file names, the first character must be capitalized. All other letters in the command, including the first letter of the sentence, must be lower case. Furthermore, DOSTALK expects sentences that are not ambiguous; therefore, it will either understand the command fully or not at all. [GRAL, 88]

Overall, there are many DOS enhancers available to the user who desires some insulation from the command

language. Icons seem to be one of the most popular interfaces in use today. Natural languages are awkward at the present; however, their capabilities are evolving and this area remains a hot research topic. It is becoming more possible to envision a future where human-computer interaction may one day be carried on just as any two people converse today.

CHAPTER 3

AN INTERFACE TO MS-DOS

3.1 An Introduction to MS-DOS

What is MS-DOS?

MS-DOS is the standard "disk operating system" developed for IBM compatible personal computers by Microsoft Corporation. MS-DOS manages the resources and controls the operations and facilities of the computer. It manages the connected peripheral devices, controls the execution of programs, and processes commands it is given. It is the intermediary between the human user and the computer hardware.

Major Influences

DOS (Disk Operating System) was introduced to the marketplace in August of 1981 as the operating system of the new line of IBM personal computers. It has become one of the dominant operating systems for personal computers of the 1980's.

At the time of its development, CP/M (Control Program/ Microprocessor) was the leading personal computer operating system. CP/M was developed by Digital Research for 8-bit microcomputers. The designers of DOS felt that is was necessary to create a system that allowed the current programs to easily adapt to the new operating system without having to be rewritten. Thus the first versions of DOS

closely imitated CP/M. [NORT, 86]

Microsoft believed that the original DOS contained too many restrictions and limitations. UNIX, a multiuser, multitasking operating system was developed at Bell Laboratories. Xenix, a variation of UNIX, was an inspiration to Microsoft. As a result, later versions of DOS incorporated styles and features found in Xenix, such as redirection, piping and hierarchical directory structures. [NORT, 86]

Norton (1986) notes two other dominating features in the evolution of DOS, the fact that DOS is as hardware independent as possible, and the fact that features are added to the system on an as-needed basis in a sort of "ad-hoc" fashion.

3.2 Goals of a truly Friendly system

It is well documented that users tend to overestimate the degree of expertise necessary to interact and utilize a computer efficiently. They are intimidated and view computers as mysterious mechanical contraptions only understood by those who are willing to invest hours of their time in extensive training. Such individuals view computers as "dull, dreary, complex, unreliable, depersonalizing, bossy, domineering and cold". [GILR, 86]

Shneiderman (1984) points out that enthusiastic users report positive feelings such as:

- Mastery of the system.
- Competence in performance of their task.

- Ease in learning to use the system originally and in acquiring new features.
- Confidence in their capacity to retain mastery over time.
- Enjoyment in using the system.
- Eagerness to show off to novices.
- Desire to explore more powerful aspects of the system.

What makes an "enthusiastic user"? What are the requirements for a successful interface? What makes an interface adequate and useful?

Research in the area of human-computer interface design has shown that a substantial difference in learning time, performance speed, error rates, and user satisfaction occurs if an adequate amount of time is spent in the developmental stages. Design alternatives have been tested for their impact upon these human performance measures. It is recognized that the simpler the system is to operate, the greater its competitive edge will be in the information retrieval, office automation, and personal computing arenas. [SHNE, 86] The design of the system should be oriented towards the capabilities and needs of the end user and not around the ideas of the programmer. The most important objective of human factors design is "Know thy user, for he is not thyself." [RUBI, 84]

According to Shneiderman (1987) there are eight golden rules of dialog design:

1. Strive for consistency.
2. Enable frequent users to use shortcuts.
3. Offer informative feedback.
4. Design dialogs to yield closure.
5. Offer simple error handling.
6. Permit easy reversal of actions.
7. Support internal locus of control.

8. Reduce short term memory load.

These rules should serve as a guideline for developing adequate and useful environments for the user. The system design should encourage the user to develop positive feelings towards the computer and enable a sense of mastery.

Schneider (1984) describes five developmental levels of computer expertise in users. The first level is the "parrot," who has minimal knowledge of the computer system, who does not think, question, understand, or synthesize commands. The second is the "novice," who has some experience, is beginning to understand several isolated concepts, and has a minimal command of the grammar. The remaining levels are that of "intermediate," "expert," and "master." Ideally, a single interface should accommodate the needs of all of its users; however, it is doubtful that this much flexibility can be incorporated. Care must be taken so as not to overwhelm the beginning user, or frustrate the advanced user.

As previously stated, the goal of this thesis was to design an interface that characterized the desirable features needed to facilitate acceptance among novices and advanced users. The system represents an improved dialog between the user and the MS-DOS operating system. It enables users to communicate with the computer in a dialog "similar" to that of their own natural language; however, a full blown natural language interface was not a necessary requirement. The system's design makes it more natural than

the traditional command languages, without creating the full capability of a natural language understanding system. [Nick, 81] The intent was to develop a glorified command language interface to MS-DOS which inherits the impressions of natural language. By impressions I mean that the system need not comprehend English or even perform a full grammatical analysis.

The implementation of a truly friendly and useful system should incorporate features offering pleasant usage. The ideal components should consist of:

- o An English-like dialog.
- o A spelling corrector.
- o Capability for the user to define aliases.
- o Error messages that are helpful and descriptive.
- o An informative help facility.
- o Safety factors.
- o The most highly used, basic functions.

An English-like dialog should help users by allowing them to communicate with the computer in their own natural language. This eliminates the need to learn and memorize the special commands of the system.

A spelling corrector would enhance the system by not requiring the user to retype an entire line due to incorrect spelling. A confirmation dialog could be used, which points to the incorrectly spelled word and asks the user to verify and correct it. A menu selection of valid choices may also be a suitable solution.

The user should be given the freedom to define aliases for different commands. In remembering certain commands, the user would feel more comfortable with terminology other

than what has been assigned.

In many systems, error messages are vague and unreliable. However, they should be helpful and descriptive. Error messages should clearly display the reasons for the system's hesitation and encourage the user instead of presenting irritating replies.

Help should always be available when needed. On-line help facilities are important. Also, there is a need for well documented, easy to comprehend manuals.

Safety factors should protect the system from deliberate misuse as well as user errors. For instance, if the user is in the C drive and tries to format a diskette that is in the A drive, he must specify this by typing "format a:". If he fails to use this exact command, then the hard disk could be erroneously formatted. A good user interface should be bullet proof, not allowing such mistakes to occur.

An operating system interface should include the most highly used, basic functions that a novice may require. Among the DOS functions which should be embodied in this system are:

- FORMAT - prepare a floppy disk for use.
- COPY - copy the designated file(s).
- DIR - display the contents of the designated directory.
- TYPE - display the contents of a file.
- RENAME - rename a file.
- ERASE - remove a file.
- CD - change to a new subdirectory.
- MD - create a new directory.
- RD - remove a directory.
- DATE - display the system date.
- TIME - display the system time.

- o PRINT - to send information to the printer.
- o Signing on to various application software.

As stated, these features illustrate the "ideal" system. Most of the MS-DOS functions have been built into the thesis program. However, due to the rudimentary level of the program, the prospect of the other qualities are further discussed as enhancements in chapter six.

3.3 Target Group

The commands of operating systems tend to be cryptic, especially to the user inexperienced with the system. As a consequence, communication with DOS through the ordinary command prompt can be difficult. Alternative operating environments can assist in easing some of this dissatisfaction by offering enhancements to DOS commands and facilities to organize files. These environments appear very attractive to users in visualizing the abstractions of DOS; however, "power users" argue that these special interfaces only get in the way of productivity.

PE-DOS is geared towards those individuals requiring a tutorial type system. It allows users to phrase requests in their own language. Then through the use of paraphrasing and echoing the MS-DOS command users should be able to learn the actual command language. Eventually, the need for PE-DOS should diminish.

The system does not teach the use of computers. Users are expected to have a general understanding of the basic fundamental capabilities and concepts of computer

technology. They must know what it is they want the computer to perform. People familiar with other operating systems, or those ambitious to learn may be typical users of a system like PE-DOS. On the other hand, those individuals who merely parrot commands are unlikely candidates.

Schildt (1987) observes that many programmers may be interested in a Natural Language driven operating system, which would virtually eliminate the time needed to learn to use the computer successfully. These users are in pursuit of a more elementary way to interact with the computer system. Systems such as PE-DOS and DOSTALK are certainly a step towards this direction.

CHAPTER 4

Project Development

The task of interpreting an English sentence or phrase as an MS-DOS command can be thought of as a form of machine translation, similar in effect to the research that has been conducted in the area of translation between two natural languages (i.e. English to Russian). The main difference is the fact that MS-DOS is an artificial language, a command language for a computer operating system.

This chapter discusses the design and implementation of a small system named PE-DOS (Plain English DOS). It uses English sentences and phrases as input and transforms them into the proper MS-DOS command to be executed. The formal structural design is that of a recursive transition network (RTN) operating in a pattern matching environment.

The code for PE-DOS can be found in Appendix B, and a script of its execution is contained in Appendix C.

Prolog was chosen for the implementation of this thesis because of its ease in attempting alternative paths, as well as its flexibility with the use of objects (i.e. words). Arity/Prolog, Turbo Prolog and A.D.A. Prolog were all investigated as possibilities. The comparisons of these different versions was interesting. Arity/Prolog was selected due to its standard syntax and behavior, as well as its execution speed.

The following MS-DOS commands have been implemented in PE-DOS:

- o copy
- o dir
- o type
- o rename
- o erase
- o cd
- o mkdir
- o rmdir
- o date
- o time
- o print
- o format

Many limitations and assumptions are placed upon the types of sentences that can be handled. This chapter describes the design of the system. Chapter 6: Enhancements, offers several improvements that could be incorporated into this program.

The Flow Pattern of PE-DOS

The program finds an MS-DOS command by traversing the RTN's and finding a pattern that fits the input sentence. The overall flow of PE-DOS can be viewed in Figure 4-1.

At the start of each execution of the program a menu is displayed to the user (welcome clause). This menu allows the user to choose between a tutorial session or to actually execute the resulting MS-DOS command. A flag is set indicating the user's option. This flag cannot be altered during the course of execution.

For each iteration, the program will prompt the user to input a sentence, and upon receiving a carriage return, it will read in the input (input clause). If an error is detected via the input routine (i.e. an undefined character

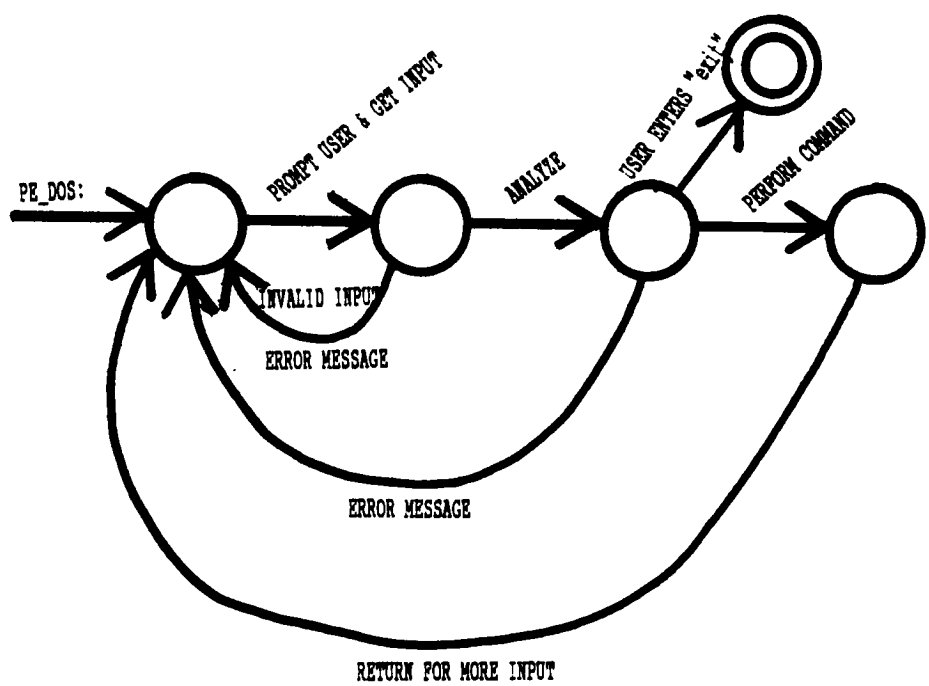


FIGURE 4-1: Network illustration of the flow pattern of PE_DOS.

was entered), an error message is displayed and control returns to the command prompt. Otherwise, the input string is processed. This procedure includes tokenizing the string or breaking it up into its constituent parts (see discussion on the input routine).

Control then proceeds to the network ANALYZE. Analyze engages in a full analysis of the tokenized input string. It locates the first command verb and takes the arc that matches that keyword. Control may be sent to a particular network to analyze the object in the remaining token list and then return the MS-DOS command to be performed. Otherwise, it may find a specific MS-DOS command. If all network possibilities are exhausted before a final state is found, an appropriate error message is displayed.

The execution of the clause, perform_command is controlled by the flag set by the user in the welcome clause. The English paraphrase and MS-DOS equivalent command are always displayed. If the flag was set to "tutorial" then perform_command is complete. If the flag was set to "perform" then the user is asked if they want the command to actually be executed. A yes/no response is expected. At the completion of perform_command, control returns to the main clause (pe) for another iteration. The program terminates upon a command of exit.

Noise Words

Schildt (1987) points out two opposing approaches to the analysis of natural language. The first approach is to

consider all words of the sentence as information to be used in the analysis process. The second approach is to eliminate all words that are irrelevant to the sentence's meaning. I incorporated the second approach. The first method would cause considerable chaos in the network structures. By eliminating irrelevant words, or "noise" words, the networks will be a simple framework in which to work.

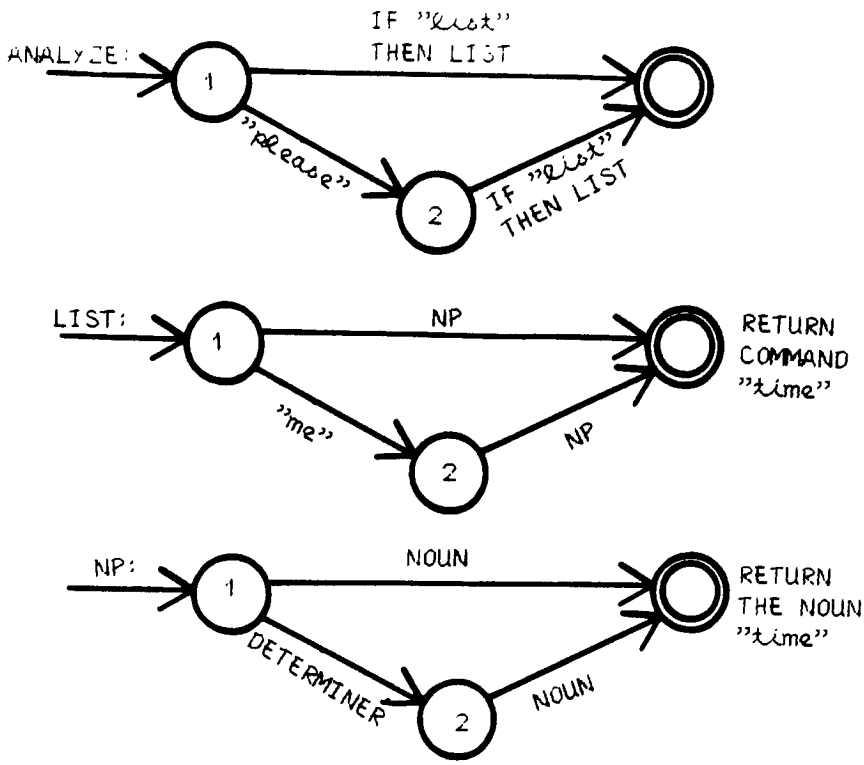
Figure 4-2 illustrates an elementary example of the simplification involved, using sentences dealing with an imperative request for time. The valid sentences are:

"list me the time."
"list the time."
"list me time."
"please list the time."
"list time."

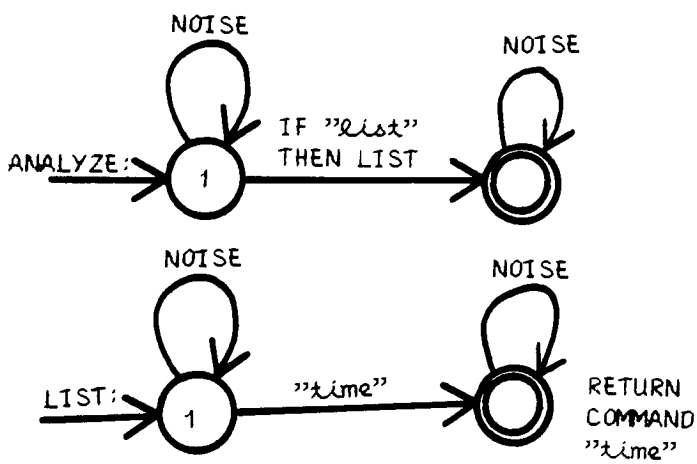
If each word is used in the analysis, Figure 4-2a is the network structure employed. However, by eliminating the unnecessary words (me, the, and please), the only remaining analysis to be preformed is on the sentence "list time." A much less complex network is necessary, as shown in Figure 4-2b.

Notice that after sifting noise words, the remaining structure is that of a verb followed by a noun. The user has been allowed the ability to express themselves with the additional words. Yet, during translation these words are avoided, creating less havoc in the analysis process.

The use of noise words is observed in most of the network structures as an arc that leaves and returns to the



4-2 A: Including noise words in the network.



4-2 B: With the elimination of noise words.

FIGURE 4-2: Illustrates the simplification of the networks through the elimination of noise words for the request for the time (i.e. "display the time", "please show me the time").

same state. Noise words are defined in the lexicon as a blank, English punctuation (comma or semicolon), or English words that are unnecessary (i.e. derive no semantic meaning) to the translation process.

In several cases a noise word is the first arc attempted at any state. Noise word elimination occurs first because all keyword and variable names are expected to be unique. The exception are the keywords "to" and "from". "to" and "from" are found in the lexicon as both a word and a noise word. These words are usually assumed to be noise words except in a few special circumstances. (See section on sentence structures for further discussion.) In a case where a directory, or file name happens to be the same as a noise word, ambiguities could occur. For example, if a string of two or more noise words exist and one of those words is intended as a directory or file name, the processing of the command will eliminate all but the last noise word. The last noise word will be assumed to be the variable name.

Pattern Matching and the RTN

The full set of networks used in the implementation of this program can be found in Appendix A. The design of the networks relies on the semantic properties of the words rather than on their lexical categories. This program is not concerned with a full grammatical analysis of the sentence structure, but on the meaning that can be derived from it.

The networks reveal a set of alternative sentence patterns and specify a particular sequence of words and the positioning of those words. At the point when a final state has emerged, an MS-DOS command should be recognized and returned via a command register. Otherwise, if an error is detected in the analysis process, a message indicating this is displayed, and the user is again returned to the command prompt.

The kind of pattern matching strategy used is "variable pattern matching." A variable pattern is a sequence that matches specific words and variables. [WINO, 83] In this program the variables represent file, directory and drive names.

The network structure consists of a set of states connected by arcs. A state simply characterizes the position within analysis process. The arcs are labeled with a specific keyword and/or a possible subordinate network. Each arc is implemented as a clause in the program emanating from a given state. The clause checks for a condition according to its label. It then proceeds to the state at the end of the arc if a successful traversal of that arc can be accomplished.

As we proceed through the analysis of a sentence, keywords are identified by finding the synonym of a word or phrase at the current point of the sentence. This keyword represents a valid label of an arc in the network. When a keyword is encountered, the arc that matches it then is

traversed if all other possible conditions are met (i.e. possible subordinate networks succeed).

The network also provides for the removal of noise words at strategic locations, where they may possibly exist. Arcs are specifically labeled to strip the random noise words from the sentence being analyzed. Noise word elimination arcs can be observed on most of the network structures. They are permissible between specific keywords, object and file specifications.

The recognition of synonyms and noise words must be a function of the analysis step. If we allowed a preprocessing step to perform the elimination of noise words and replacement of words for their synonyms, then variable names, such as those given to files or directories could be destroyed. Of course this is only if by chance a file or directory name were the same as a noise word or part of a phrase. These cases may seem extreme, but nevertheless, are possible.

The network expresses deterministic properties. However, in the actual programming of the clauses, a non-deterministic property emerges. Chronological backtracking is used; the most recent instantiation is changed to an alternative choice upon finding a dead end in the network. This of course is one of the significant attributes of the Prolog language in general.

A syntactical analysis is not a function of this program. The networks are developed around patterns of

specific words, as stated earlier, and not the lexical categories of those words. Therefore, ungrammatical commands are possible due to the use of noise words; however, since the patterns are constructed as proper English subsets, it is unlikely that immensely ungrammatical commands will exist.

Sentence Structures

As would be expected in any program, restrictions have been made on the structure of permissible commands. Since the purpose of this program is to convert a set of English words to an MS-DOS command, an imperative structure would be the most logical type of sentence. Pattern matching plays the major role in the system analysis process. The command issued by the user must contain the basic underlying pattern of:

command-verb [arguments] [specifications]

where arguments and specifications are not always required.

An argument may be part of the pattern that assists in the translation process. For example, using the LIST network of Appendix A, Figure 2, the verb "list" could render the MS-DOS commands "dir", "type", "time", or "date". The PE-DOS command "list time" sends control to the list network via the command verb "list". The argument "time" returns the MS-DOS command "time".

An argument also could be a true command language argument such as the file name in "list anyfile.txt".

There could be several pieces to a PE-DOS argument as

in the command "list the file anyfile.txt on drive a". Here the key word "file" and "drive" are used to denote a single argument retrieved via a call to the FILE_SPEC network.

Using the prescribed structure, sentences like the following can be declared:

"display directory."
"show files."
"list time."
"remove file anyfile.txt."
"copy one.txt temp.aci."

Certainly, these examples are very limited in their expression. Consequently, to offer a more human-like dialog, the user is allowed to type commands such as:

"display the directory contents."
"show me all the files."
"list the time."
"please remove the file anyfile.txt."
"copy the file temp.aci from one.txt."

These sentences contain the underlying pattern (underlined words) with the use of additional words called noise words, making the command resemble English the way humans use it.

The use of prepositions have been incorporated into the command structure in few instances. Many prepositions have been declared in the lexicon as noise words. However, the use of "to" and "from" in the CHANGE and COPY networks were necessary due to the expectation of two file or object arguments. (See Appendix A, Figure 3 and 5) Typically MS-DOS expects the command in the form:

command <source> <target>

In PE-DOS, it is possible for the command to take an alternative form, accepting:

command ["from"] <source> ["to"] <target>

or

command ["to"] <target> ["from"] <source>

where the use of the keywords "to" and "from" are optional. The use of these additional keywords assists in verifying the legitimacy of the file or object specification. For instance, in the English command:

"copy anyfile.txt drive a anydirectory"

it is not clear where the break in file and object specifications occur. Two different valid MS-DOS interpretations could be surmised:

"copy anyfile.txt a:\anydirectory"

or

"copy a:anyfile.txt \anydirectory"

Though the use of the keywords "to" and "from are not a stringent requirement of PE-DOS, their use to specify the break in argument declarations, as in the above situation, are necessary to clearly indicate the users expectations. In the above example, if no keywords are utilized to separate the two arguments, by default PE-DOS will return:

"copy anyfile.txt a:\anydirectory"

If the users intention was

"copy a:anyfile.txt \anydirectory"

then the PE-DOS command should include the keyword "to" as in:

"copy anyfile.txt drive a to anydirectory"

It is necessary to further review the use of

prepositions in the OBJECT_SPEC, FILE_SPEC and specifications networks. This remains as part of an alternative design consideration (See section 6.2).

The system does recognize valid MS-DOS commands with the exception of spacing of the file and object specifications. The use of symbols (i.e. ":" and "\") can give a near MS-DOS look to a PE-DOS command. PE-DOS spacing constraints between drive, directory, and file declarations of a file or object still apply; a single space is required between each constituent part. This allows the user flexibility in the issuance of commands, giving the option to use a sort of mock command language when the user is familiar enough with the operating system.

Input Routine

The purpose of the input routine is to invite the user to enter a command, retrieve the string of ASCII characters from the terminal, and process the string returning a list of tokens to be analyzed. The code for the input routine can be viewed in Appendix B, file input.ari.

The clause read_command is called from the program's main clause pe (of file pe.ari). This clause prompts the user for a command and passes control to the clause read_character_list to perform the remainder of the input function.

First, read_character_list calls upon Word_chars, which reads the string of ASCII characters. Word_chars then examines all ASCII characters for validity in the PE-DOS

program. Only characters acceptable to PE-DOS are excepted as input. Otherwise, an error message informing the user of the invalid ASCII character (i.e. unrecognized characters such as control characters) is displayed interrupting the input process. Control then resumes at the command prompt for the beginning of a new command. Word_chars also converts all upper case characters to lower case. Upon successful conclusion, word_chars returns a valid list of ASCII characters.

Second, read_character_list calls elim_backspace to take care of any backspace characters found in the ASCII list.

Third, word_break is called to convert the ASCII list into tokens. These tokens are used during the analysis process and include blanks, punctuation ("!", "?", ";", and ","), MS-DOS special characters (".", "/", "¦" and ":") and character strings. This step completes the input routine.

Drive, Directory and File Names

Several network structures have been designed to detect the presence of drive, directory and file names. The main networks, which are called from within the analysis routine are OBJECT_SPEC and FILE_SPEC (Appendix A, Figure 9 and 10). These networks call upon several subordinate networks during their execution.

The object specification (OBJECT_SPEC) network locates a valid object. An object is defined as a drive/directory combination, a drive, or a directory.

The file specification network (FILE_SPEC) finds a valid file name. A file name can consist of an optional drive, optional directory, and a file name in any possible combination.

The product, DOSTALK, requires syntactic declarations in its command to indicate a file name. Their approach restricts the user during the input to the use of lower case letters except when declaring file and directory names. The first letter only of a variable, such as file and directory names, must be capitalized. This forces the user into having to remember syntax when the goal behind natural language processing is a human-like, uninterrupted dialog.

As an attempt to eliminate such a crutch, PE-DOS makes no syntactic requirements upon naming. Therefore, the analysis process is not assisted in identifying the presence of drive, directory or file names. The networks rely upon proper spacing, specific key words and MS-DOS symbolism (i.e. colon, slash and period) to determine the validity of the object or file. Checks are performed on the characters that make up the drive, directory, file and file extension strings. It is assumed that the keywords "drive", "directory", and "file" are not proper MS-DOS directory or file names.

The system will not return an object or file specification that constitutes an invalid MS-DOS argument, yet it is possible that the user's intention will not be recognized (i.e. something unexpected is returned). To

illustrate, lets reconsider the example

"copy anyfile.txt drive a anydirectory"

from above. The user may be intending the MS-DOS command

"copy a:anyfile.txt \anydirectory"

to be derived. When in fact PE-DOS returns

"copy anyfile.txt a:\anydirectory"

unless the keywords "to" and "from" are utilized to indicate differently.

Another type of command which causes unexpected results is

"zap a anyfile.txt"

PE-DOS will convert this to the MS-DOS command

"remove anyfile.txt

"a" has been removed as a noise word which may not have been what the user intended.

The major causes of problems such as those above are the flexibility that PE-DOS offers and the use of noise words.

The only constraint imposed upon the user is the imperative style structure of a command; the action verb must be detected before any arguments. The identification process of drive, directory and file names allows for possible ambiguities. The user does not necessarily indicate the use of these names syntactically. For instance, in the DRIVE network (Appendix A, Figure 14), the keyword "drive" immediately before or after the actual drive name, or a colon immediately after it, specifically

indicates that the name is a drive. However, the single letter of a drive, "a", "b" or "c", will also return successfully. It is possible that "a" could be taken as a noise word. A clarification dialog could be used, but too many queries to the user would become monotonous.

Special flags are used in the OBJECT_SPEC and FILE_SPEC networks indicating the validity of the identification process. These flags are set based on flags returned from subordinate networks (DRIVE, DIRECTORY and FILE_NAME) and certain keywords. The use of these flags are the reason certain clarification dialogs exist. For example, the command

"zap anything"

traverses both the OBJECT_SPEC and FILE_SPEC networks where the flags returned indicate no specific reference to a file or directory. The user is then asked

"Is anything a file or directory?"

and must reply accordingly.

The convention of noise words and too much flexibility really becomes questionable. Depending on when noise word elimination occurs, the PE-DOS command

"erase a a a.ext"

could result in

"erase a.ext"

if noise words are eliminated upfront, or

"erase a:\a\a.ext"

if they are retained. It is necessary to either enforce

syntactic requirements upon the user or have the system check all possibilities. A system check could search the directory structure during the analysis process to assure the existence of a particular object or file specification. However, at times a user may ask for something that is nonexistent and the system would need to handle this.

Lexicon

The lexicon uses the concept of differences lists. Each word clause contains three arguments. The structure of the lexicon appears as:

```
word([word|Rest],synonym,Rest).
```

where the argument word is the stripped token or phrase of the received list of tokens, synonym is the synonym of the word, and Rest is the remaining list of tokens.

Included in the lexicon are words found in the network patterns, noise words and MS_DOS commands. The noise word clauses have only two arguments; it is unnecessary to return the synonym for a noise word. Synonyms are employed to eliminate congestion in the networks.

Noise words are included in the lexicon because of the variable names for drives, directories and files. These variable names should be unique, identifiable names and not just some ordinary name found in the lexicon. Of course, this may not always be true; therefore, the structure of the command will need to identify this (i.e. use of keywords, structure of a file).

Garbage Sentences

Since the analysis system does not cater specifically to grammar, it is certainly possible to have sentences that appear to be garbage, yet an MS-DOS command may still be found. Sentences like:

"the display please files"
"list is is the time"
"duplicate me the anyfile.txt otherfile.aci"

appear nonsensical but will evaluate to the following MS-DOS commands respectively:

"dir"
"time"
"copy afile.txt anyfile.txt"

The system merely looks for a simple patterns like:

"list files"
"list time"
"copy X Y"

(where X and Y are file specifications) and eliminates any noise words found anywhere along the way.

Clarification Dialogs

English is an imprecise language, and MS-DOS is very precise. To convert from English to MS-DOS takes much care. When ambiguities occur, clarification dialogs can help relieve the problem. Validity checks in the form of clarification dialogs have been made at several points during the analysis process.

Clarification dialogs occur in the LIST and REMOVE networks to identify possible misinterpretations.

The most prevalent misunderstanding is whether a specific name is a file or directory when keywords or

structure can not determine the difference. For instance, if the user issues the command

"zap something"

(zap being a synonym for remove) the program is not sure if the user means to get rid of the file something or the directory something. In order to make this determination, the program will ask the user:

"Is something a directory or file?"

The user should respond accordingly.

Chapter 5

User Feedback

Since the objective was to create an interface or tutorial system for the MS-DOS operating system, several feedback sessions were conducted. Individuals were told that the system was capable of routine commands and they were then given an overview of the conceptual goal of the system without a true list of its capabilities (i.e. listing of available commands). Therefore, no ideas of how to word a particular command were suggested to the subjects; they virtually were allowed to follow their instincts.

The resulting system did not meet the original expectations due to the inherent complexities of English. The feedback sessions were conducted to obtain "real" comments from possible users on the idea of such a system. Many of the limitations and/or problems found with the system during these sessions were anticipated and are discussed in Chapter 6: Enhancements.

Two groups of potential users were involved in the feedback sessions. The first group were well versed with operating systems such as the VAX/VMS and UNIX, but were unfamiliar with the MS-DOS system. The original objective of PE-DOS was to support these particular individuals. The second group was relatively inexperienced with the direct use of the operating system. These users were familiar with

PC environments, but only for use with application software (i.e. Lotus, WordStar). The different perspectives of the two groups was enlightening.

The experienced group were bold in the issuance of commands. They expected a wider range of acceptable commands, more flexibility in the structure of a command, and better ways of handling spelling and input errors. To these users, operating system commands are almost second nature. Since they knew what they could expect, they felt that if they needed to adapt to a new system such as MS-DOS, the use of a manual or other help facility to find the answers to their questions would be easier and more informative than PE-DOS was capable of being. The excess verbiage to clarify their intention, as well as the unnecessary time spent to enter the command, made no sense to this group. A good example was the frequent necessity of "to" and "from" as keywords in the copy and rename commands (see discussion in Chapter 4) instead of the positioning of source and target to indicate its arguments.

Individuals come from different backgrounds. Therefore, it is difficult to predict all the possible variations they could use to structure an MS-DOS command using English.

The user of PE-DOS may enter the command

"list X"

where X represents some file or object declaration. If the format of X explicitly designates it as a file or object, as

in the commands

"list directory anydirectory"
"list files on drive a"
"list file anyfile"

then an appropriate MS-DOS command is returned. This seems simplistic enough at first glance, but it could in fact cause the user to be subjected to a series of questions clarifying the intent if X is a simple string that does not identify itself clearly to PE-DOS without additional information (i.e. use of clarification dialogs).

The experienced group was not enthusiastic about the possibilities of English as an interface/tutorial system to execute operating system commands. The feeling in general was that natural language systems were only experimental and/or a waste of one's time. Some felt that the concepts of natural language interfaces could have possibilities in other environments (i.e. futuristic applications involving speech recognition), but within the realm of operating systems would only prove to reduce productivity. MS-DOS, and most other operating systems for that matter, are straightforward. Their symbolism and unique naming conventions for command verbs and arguments make them relatively easy to use so "why reinvent the wheel". In MS-DOS, to view the contents of a particular file, simply issue the command

"type X"

where X is a file specification. In order to get a directory listing to see if a particular file exists or view

several files at a certain location type

"dir X"

where X is an object or file specification.

In the LIST network of PE-DOS even the simple command

"list file anyfile.txt"

is unclear. The keyword "list" is ambiguous. Does the user want to see the file contents displayed (i.e. MS-DOS "type" command), or do they merely expect a directory listing to indicate the existence of the file (i.e. MS-DOS "dir" command). The user, therefore will be queried for clarification upon entering such a command.

MS-DOS doesn't have these complex problems. The command language is direct and concise. It restricts the user to its set of keywords and eliminates the numerous possible synonyms and the myriad of interpretations and ambiguities associated with English. In essence, MS-DOS is a subset of English and by that very fact, is more concise and focused on specific words and meanings. What the user types on the keyboard is exactly what they can expect to have executed, no questions asked. The individual who uses MS-DOS can state specifically what their intentions are. PE-DOS on the other hand, is opening up the possible number of words and meanings to an unmanageable level of interpretation. Command input words must be filtered down to the DOS subset level and even then must be semantically interpreted for the correct translation of user intent.

The inexperienced, novice users were timid. This is

attributable to their lack of knowledge about the capabilities of an operating system. Many were confused as to what was expected of them as users. Most of them did not understand even the concepts of files and directories (i.e. files have a name and extension). They would issue a command like

"show the file anyfile"

where anyfile was the name of a text file developed in Display Write. Display Write assigns the extension "txt" to all files developed. The proper MS-DOS conversion of the above PE-DOS command is either

"type anyfile"

or

"dir anyfile"

with no file extension. However, without the proper extension at the operating system, this command would fail on finding the file. These users were frequently unaware of this type of conflict.

It was obvious that a natural language interface/ tutorial was unsuitable for this group. A menu or icon based interface, which displays only the available choices would be more appropriate for this level of user.

PE-DOS as a tutorial/interface was poor at best. The reviews on DOSTALK do not speak optimistically either. They simply introduce the product for what it is and leave the ultimate opinion up to the potential user. Perhaps a system like DOS-Alike [MARS, 89], which caters specifically to

MS-DOS users who are converting to UNIX, is the practical solution for the experienced user requesting some sort of assistance. DOS-Alike performs a translation from MS-DOS into UNIX; the translation process occurs between two precise languages which causes less ambiguities during the conversion process then that with the imprecision of English.

CHAPTER 6

Enhancements

This chapter discusses several enhancements that could be made to the implementation of PE-DOS. Many of the issues had been concerns addressed during the user feedback sessions. The issues noted relate to improvements in user interaction with the system and in system performance. User interaction topics are directed toward a wider acceptance of sentence types and structure, enabling the user to maintain a personal lexicon, how the system may handle spelling errors, a more user friendly input routine, improvements in error handling, specifications for dates and time and requirements of a user manual. System performance items emphasize the restructuring of several assumptions that were made in the program design suggesting a possible alternative architecture using definite clause grammars (DCGs).

6.1 User Interactions

Sentence Types and Structure

The program, as developed, works with a very limited subset of English sentence structures. Commands are limited to an imperative style, where a command verb is optionally followed by arguments and possible specifications. Since humans may phrase a command in any number of ways, a natural language processing system must be much more receptive to a

the possible types of sentences than is PE-DOS. Another form of an imperative command may be

"take the file anyfile.ext and copy it to the a drive"

and should translate to the MS-DOS command

"copy anyfile.ext a:"

The structure of this command is

<argument> <command-verb> <argument>

The use of interrogative structures are also relevant, as in the request to execute the MS-DOS command, "time", "date", and "dir". Such statements may look like:

"what is the time?"
"what's the date?"
"could you show me all files?"

Of course, realistically speaking, some kind of limit on the number of sentence types that are considered appropriate for a given domain must be imposed.

Aside from the structure of the sentence, semantic issues also need more consideration. The command

"list all files"

returns "dir". However, the command

"erase all files"

is not recognized as

"erase *.*"

as might be expected. PE-DOS would eliminated "all" as a noise word and then ask the user if "files" were a directory or file name. The command

"list all files starting with the letter a"

cannot be analyzed and will result in an error message. No

parsing technique has been employed in the specification of files to indicate a particular range.

Availability of different categories of permissible sentence structures would depend on the ultimate processing design, which in the case of PE-DOS conceivably could be altered. (See section 5.2, System Design)

Adding Words to the Lexicon

The available lexicon is restricted to what is initially programmed into PE-DOS. It needs to be expanded to include idioms and other phrases common to an array of environments and users.

If a word is found in the command sentence that does not exist in the lexicon, either as a valid match to the pattern structure or as a noise word, analysis of the sentence will be discontinued and an error message indicating this will be displayed to the user. Instead of halting execution, one possible solution is to query the user at the point of discontinuance for the usage of the particular undefined word and then add it to the lexicon. This may need to take a menu format because a synonym and word category (i.e. regular word, noise word, MS-DOS command) need to be defined if the word is to be appended to the existing lexicon.

A drawback to this method is that the user will need to become more aware of the system. An alternative to systems such as PE-DOS may be many hours of consultation with the consumer to customize the product.

Spelling Errors

As in the case of undefined words, if a spelling mistake occurs, the analysis is discontinued and an error message displayed to the user indicating this. Misspelled words may be taken care of in a similar fashion to how word processors handle them. Through the use of a menu scheme the user could be given a selection of possibilities for the word in question. They could be allowed to choose the appropriate spelling or type in the correction, then the analysis process could resume.

Note that the case of adding words to the lexicon and spelling errors overlap. The system will need to first determine which type of error has occurred through a query of the user.

Input Routine

The input routine of PE-DOS allows the user to type upper and lower case letters (upper case is converted to lower case), numbers and other characters specifically valid to PE-DOS. Other ASCII characters result in an error message at the input routine and kicks the user back to the command prompt.

The input routine needs to be geared to friendlier interactions with the user. The purpose of preventing invalid PE-DOS characters from being input was to curve the time that would be unnecessarily spent attempting to analyze the command. However, the way the program handles it currently could be somewhat annoying to the user.

One alternative may be to continue the processing as it stands, but in a temporary buffer, store the part of the string up to the error. The user would still be given an error message and returned to the command prompt. However, instead of retyping the entire line, a function key could recall that portion in the buffer.

Another possibility is to remain at the current processing status and signal the users attention via a beep at the console. The character does not get input into the system, the user does not get kicked back to the command prompt and the processing of input continues.

Queries to the user in the form of clarification dialogs should allow the user to input some sort of back-out command. It is possible that the user could have entered a command and upon a query decide they didn't want that command after all. The current system only allows a valid response. For example, the query

"Is anything a file or directory?"

only accepts replies indicating "file" or "directory". The system needs to allow the user to break back to the command prompt if need be.

Error Handling

There are certainly many programming environments today which have inappropriate error handling techniques. Errors occur due to a lack of knowledge, incorrect understanding, or inadvertent slips. Users are inclined to become confused, feel inadequate, and apprehensive. Well designed

diagnostic facilities and error messages certainly can help make a system very appealing. It is not that a well groomed and effective message handling system will turn a bad system into a good system; however, it can play a significant role in improving the user's performance and attitude. [SHNE, 1980 & 1986]

Few error messages occur in PE-DOS. An extensive analysis of possible error types needs to be conducted at the analysis stage. Messages need to give a better indication as to the reason why the error has occurred (i.e. what word may not have been in the lexicon, inappropriate syntax, etc.). Also, assistance such as on-line help to explain some of the oddities of the program would improve the usefulness of the system.

Specificity and constructive guidance are needed in the message so that it provides adequate information to correct the error which have occurred. [SHNE, 86] Six general error messages currently exist in PE-DOS. The clause analyze will detect if arguments to a valid command-verb are invalid, if a word is an undefined word in the lexicon, or will display a general message indicating the request cannot be analyzed if no other path or message applies. An error message is also displayed in clause drive and valid_extension if valid strings are not detected. In clause drive, if the special MS-DOS character '\' is found then a valid drive name is expected to follow and clause valid_extension expects to find a proper extension if called upon. Surely, the

messages have to be enhanced to pinpoint the reasons for such error. In the present state, a new user would become annoyed very quickly.

The error messages of PE-DOS need to be more informative (i.e. specify what is missing). If arguments to a command are invalid, specify why they might be invalid (i.e. two arguments are expected and there is only one). Perhaps the use of more queries to guide the program in its analysis phase would be helpful. In checking the validity of a translated command, the product DOSTALK is extreme. It always requires the user to justify the MS-DOS translation before it actually executes it. This may at first glance seem like good ideas for a translation which is full of so many possible ambiguities. However, too much additional dialog with the user, either through the use of more queries or a final query before execution could become quite annoying to the user.

The use of positive tones and user-centered phrasing eliminating the hostile, vague and irritating messages, will enhance the user's enjoyment of the system. The system should never place blame upon the user (i.e. pointing out failure), but offer constructive, comprehensive messages which assist in guiding the user to a solution. [SHNE, 86] One goal of a natural language interface should be to offer a two way, human-like dialog and not just the typical, abrupt "** ERROR **" messages.

Dates and Times

If the user is to be able to set the date and time in the system they must be particularized. Are they to be assumed integers separated by hyphens, slashes and colons? For example:

"2/18/88"
"02-18-88"
"7:00:16"
"19:00:16".

Or are alpha strings allowed too? Like:

"feb 18 88"
"feb 18 1988"
"7 o'clock"
"7 pm"

In regard to dates, what formats are available? For example:

month-day-year
day-month-year
year-month-day

All of these ideas need to be evaluated, and of course any limitations must be documented in a manual to the user.

Perhaps as a full blown system, a system clock could be graphically displayed in a designated location of the screen. The user then only needs the date and time function when adjustments are necessary.

Users Manual

Many user manuals today seem to be almost disappointing and incomprehensible to the user. These manuals are written at the technical level and are "over the heads" of those who really need assistance.

User manuals should be a supplement which effectively

aid the user in the development of their understanding of the system. The manual should be a concise, easy to read, set of instructions for use and describe the features of the system (i.e. command structures) as well as the current knowledge and abilities expected of the user. It may perhaps offer a direction or guidance to further literature. For this program, assumptions about file and directory naming should be explained.

On-line help is a new medium form for the manual. However, several studies conclude that on-line assistance is not necessarily a valued asset over the traditional hardcopy manuals. They require the user to have to memorize additional system commands for access and disrupt the work that is in process. In these studies, hardcopy manuals were preferred mainly due to their familiarity. [SHNE, 86]

6.2 System Design

A major drawback of PE-DOS was its lack of flexibility in the overall programming design. Though a system administrator may seem a logical solution to the customization of PE-DOS for individual groups of users, its adherence to a rigid structure of patterns hardwired into a series of networks would make it difficult. The network structures would become increasingly complex in order to extend the functionality of the system (i.e. increase acceptable sentence types, addition of available commands). To implement changes or additions to the network structures,

the system administrator would need to understand the Prolog language along with the concept of networks.

The use of noise words in PE-DOS may allow too much room for error in the analysis process. As a result, the following types of problems are inevitable.

First of all, it is possible that a word defined as a noise word in the lexicon could also be a valid file or directory name in MS-DOS. If a string of noise words occurred in a command and a file or directory name was expected, the last noise word would be assumed to be the name by default; no further analysis is performed on those words. It may be important to drop the concept of noise word elimination all together and preform a full syntactic analysis of the command string.

Secondly, most prepositions have been categorized as noise words. The exception are "to" and "from" when used as keywords in the copy and change networks. PE-DOS searches for particular keywords only when deemed necessary; however, this now seems an inadequate assumption. The use of prepositional phrases may be advantageous to the analysis process. For instance, the PE-DOS command

"copy anyfile.txt on drive a"

would generate the MS-DOS command

"copy anyfile.txt a:"

In other words, anyfile.txt on the current drive would be copied to the drive a. The preposition "on" has been

eliminated as a noise word. However, if a better syntactic analysis is preformed utilizing prepositions, the true intention of

"copy a:anyfile.txt"

would be realized, indicating that the file anyfile.txt which is found on drive a, should be copied to the current drive.

An alternative approach to the design of PE-DOS is to perform a full syntactical analysis instead of the pattern matching employed, which only minimally relies on the syntax of the command issued. The command syntax can indicate a lot of information in order to diagnose the user's intention more efficiently. A possibility would be to use Definite Clause Grammars (DCGs) to syntactically diagram the sentence and then build a semantic structure based upon the English constructs.

The DCGs would allow the generalized analysis for the structural breakdown of all commands. Therefore, all commands could be syntactically diagrammed based upon a single set of DCGs and not a series of individual patterns of networks for each type of command. After this syntactic process, the semantic structure should be created to represent the command.

The creation of the semantic structure would include the necessary components of command verb, argument information, and command specifications. An analysis of the

semantic structure could then be performed using a network configuration to determine the desired MS-DOS command.

CHAPTER 7

Conclusions

In the Future, conversing with computers in much the same manner as we converse with another human may not be as far fetched as it seems today. Developments in this area certainly have progressed from research projects such as ELIZA, to practical systems such as DOSTALK. However, to allow computers to efficiently carry on productive, logical, human-like conversations would be a dramatic achievement in the artificial intelligence arena.

In some respects it may seem ridiculous to spend the time and effort creating programs such as DOSTALK and PE-DOS. The results do not seem overly satisfying in light of the amount of programming effort required. Too many ambiguities exist that must be resolved to perform even the most simplistic operations of a computer operating system. Though the syntax of MS-DOS may seem rigid at times, it is a more direct, efficient and flexible mode of communication with the operating system than any of the so-called friendly interfaces available. Other interface options, such as menus and icons do not permit the room for errors and misunderstandings that a natural language interface does. Yet, they are strictly limited to a set of available commands; they offer a "what you see is what you get" and "what you can't see, you can't have" environment. Natural

language interfaces need constant supervision during the interpretation process; however, they theoretically could allow the same flexibility as the command language.

Too many limitations were imposed upon PE-DOS during its development. As it stands, the natural flow of "normal" English does not result in an acceptable command. The program is too restricted to allow the necessary flexibility required in a full natural language translation program.

The number one limiting factor was the assumption for the structure of the command. A particular pattern must be found: command-verb [arguments] [specifications]. The command is expected to begin with a command verb, followed in order by optional arguments (i.e. names of files, directories, and drives) and specifications (i.e. print the directory in wide format). This pattern arrangement was chosen because it is the logical ordering expected when using MS-DOS and most other operating systems directly. However, this constraint must be eliminated if the user is going to truly converse with the computer via their own natural language. As observed in Chapter 6, several modifications to the current system are necessary in order to create the envisioned interface. Those enhancements could serve as future project topics.

Natural language interfaces, at the present, are too crude to offer an acceptable interface/tutorial system to an operating system environment, and it is probably doubtful whether such a system would ever be useful. As discussed in

Chapter 5: User Feedback, operating systems become almost second nature to those who interact with them frequently. For these users, the currently available technology may prove a loss of productivity. The novice level user would be at a loss of what to ask the system for. Someday, natural language interfaces may prove to become an interesting mode of communication with computers, but they have a long way to go and perhaps may await the super computers of the future.

8. BIBLIOGRAPHY

[BATE, 87] Bates, Madeline & Meltzer, David & Shea, Sandra (1987). "Designing a Practical Interface". AI Expert. vol. 2, no. 5. May, 1987.

[BEEC, 87] Beechhold, Henry (1987). "Getting More From PC-DOS and MS-DOS, Part 1". Family & Home-Office Computing. vol. 5, no. 10. October, 1987.

[BEEC, 87] Beechhold, Henry (1987). "Getting More From PC-DOS and MS-DOS, Part 2". Family & Home-Office Computing. vol. 5, no. 11. November, 1987.

[BERL, 86] Berliner, Don (1986). Managing your Hard Disk. (Indianapolis, IN: Que Corporation, 1986).

[BORL, 86] Borland International, Inc. (1986). Turbo Prolog Users Manual. (Scotts Valley, CA: Borland International, Inc., 1986).

[BORL, 87] Borland International, Inc. (1987). Turbo Prolog Toolbox Users Manual. (Scotts Valley, CA: Borland International, Inc., 1987)

[BROW, 85] Brown, Mikel (1985). "Using Natural Language for Data Base Queries", MS Thesis. Department of Computer Science, Rochester Institute of Technology, 1985.

[DEVO, 83] DeVoney, Chris (1983). IBM's Personal Computer. (Indianapolis, IN: Que Corporation, 1983).

[DUCA, 86] Duncan, Ray (1986). Advanced MS-DOS. (Redmond, WA: Microsoft Press, 1986).

[FISH, 87] Fisher, Sharon (1987). "Many Users Perfer Keyboards to 'Pesty' Devices". Info World. vol. 9, no. 37. September 14, 1987.

[GILR, 86] Gilroy, Faith & Desai, Harsha (1986). "Computer anxiety: sex, race and age". International Journal of Man-Machine Studies. vol. 25, no. 6. December 1986.

[GITT, 86] Gittins, David (1986). "Icon-based human-computer interaction". International Journal of Man-Machine Studies. vol. 24, no. 6. June 1986.

[GRAL, 88] Gralla, Preston (1988). "Program Attempts to Translate DOStalk into Plain English". PC Week. vol. 5, no. 22. May 31, 1988.

[HARR, 85] Harris, Mary Dee (1985). Introduction to Natural Language Processing. (Reston, VA: Reston Publishing Company, Inc., 1985).

[HAYW, 83] Hayward, Mary (1983). "A Menu Driven, User Friendly, Interface to Unix", MS Thesis. Department of Computer Science, Rochester Institute of Technology, 1983.

[HEIN, 87] Heiny, Loren (1987). "Natural Language Programming". PC AI. Fall, 1987.

[JU, 84] Ju, Charlie C. (1984). "The Natural Language Front End Processor for Mistress DataBase". MS Thesis. Department of Computer Science, Rochester Institute of Technology, 1984.

[LANE, 87] Lane, Alex (1987). "DOS in English". Byte. vol. 12, no. 14. December, 1987.

[MARS, 89] Marshall, Martin (1989). "Unix Shell Helps DOS Users with DOS-Like Commands". Infoworld. March 27, 1989.

[MISH, 85] Mishkoff, Henry (1985). Understanding Artificial Intelligence. (Indianapolis, NJ: Howard W. Sams & Co., 1985).

[NICK, 81] Nickerson, Raymond (1981). "Why interactive computer systems are sometimes not used by people who might benefit from them". International Journal of Man-Machine Studies. vol. 15, no. 4. November 1981.

[NORT, 86] Norton, Peter (1986). Inside the IBM PC. (New York, NY: Prentice Hall Press, 1986).

[OBER, 87] Obermeier, Klaus K. (1987), "Natural Language Processing". Byte. vol. 12, no. 14. December, 1987.

[OMAL, 85] O'Malley, Christopher (1985). "What you should know about MS-DOS". Personal Computing. vol. 9, no. 8. August, 1985.

[PETZ, 86 Petzold, Charles (1986). "Operating in a New Environment". PC Magazine. vol. 5, no. 4. February 25, 1986.

[RICH, 88] Richman, Sheldon (1988). "Help for Those Who Can't - or Won't - Learn DOS". Goverment Computer News. vol. 7, no. 9. April 29, 1988.

[RUBI, 84] Rubinstein, Richard & Hersh, Harry (1984). The Human Factor - Designing computer Systems for People. (Burlington, MA: Digital Press, 1984).

[SCHI, 87] Schildt, Herbert (1987). "Natural Language Processing in C". Byte. vol. 12, no. 14. December, 1987.

[SCHN, 84] Schneider, Mike (1984). "Ergonomic Considerations in the Design of Command Languages". in Vassiliou, Yannis, ed. Human Factors and Interactive Computer Systems. (Northwood, NJ: Ablex Publishing Corp., 1984).

[SHAF, 87] Shafer, Dan (1987). Turbo Prolog Primer. (Indianapolis, IN:Howard W. Sams & Company, 1987).

[SHNE, 80] Shneiderman, Ben (1980). Software Psychology. (Boston, MA: Little, Brown and Company, Inc., 1980).

[SHNE, 87] Shneiderman, Ben (1987). Designing the User Interface. (Reading, MA: Addison-Wesley Publishing Company, 1987).

[STEV, 85] Stevens, Lawrence (1985). Artificial Intelligence - The Search for the Perfect Machine. (Hasbrouck Heights, NJ: Hayden Book Company, 1985).

[WILL, 86] Williamson, Mickey (1986). Artificial Intelligence for Microcomputers. (New York, NY: Brady Communications Company, Inc., 1986).

[WINO, 83] Winograd, Terry (1983). Language as a Cognitive Process, Volumne I: Syntax. (Reading, MA: Addison-Wesley Publishing Company, 1983).

[WINS, 84] Winston, Patrick & Prendergrast, Karen, eds. (1984). The AI Business: Commercial uses of Artificial Intelligence. (Cambridge, MA: M.I.T. Press, 1984).

[WILC, 83] Wilcox, Leonard (1983). "Parsing Natural Language", MS Thesis. Department of Computer Science, Rochester Institute of Technology, 1983.

[WOLV, 84] Wolverton, Van (1984). Running MS-DOS. (Bellevue, WA: Microsoft Press, 1984).

[WONG, 86] Wong, Williams, G (1986). "Prolog: a Language for Artificial Intelligence". PC Magazine. vol. 5, no. 17. October 14, 1986.

9. APPENDIXES

APPENDIX A: RECURSIVE TRANSITION NETWORKS

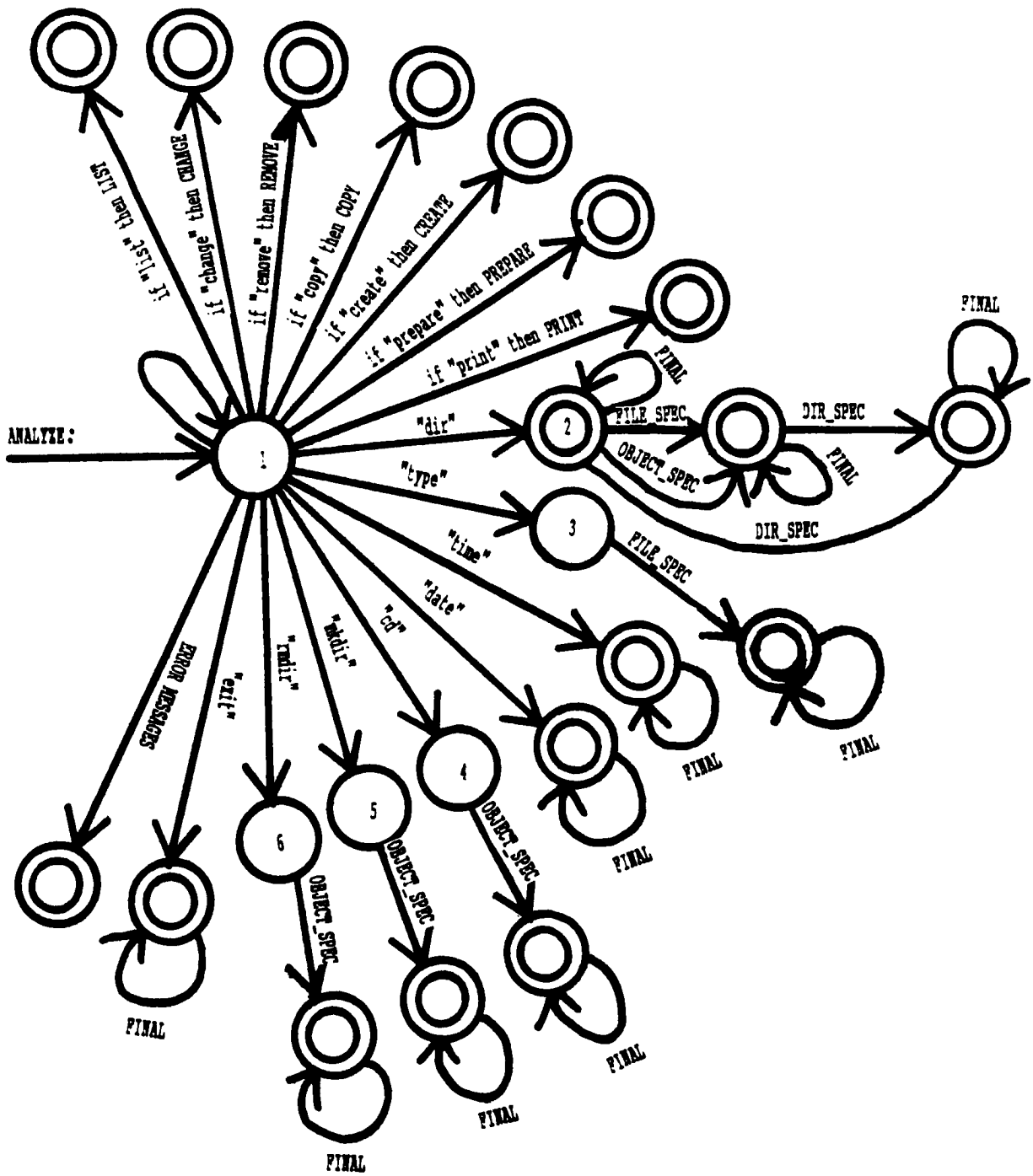


FIGURE 1: ANALYZE

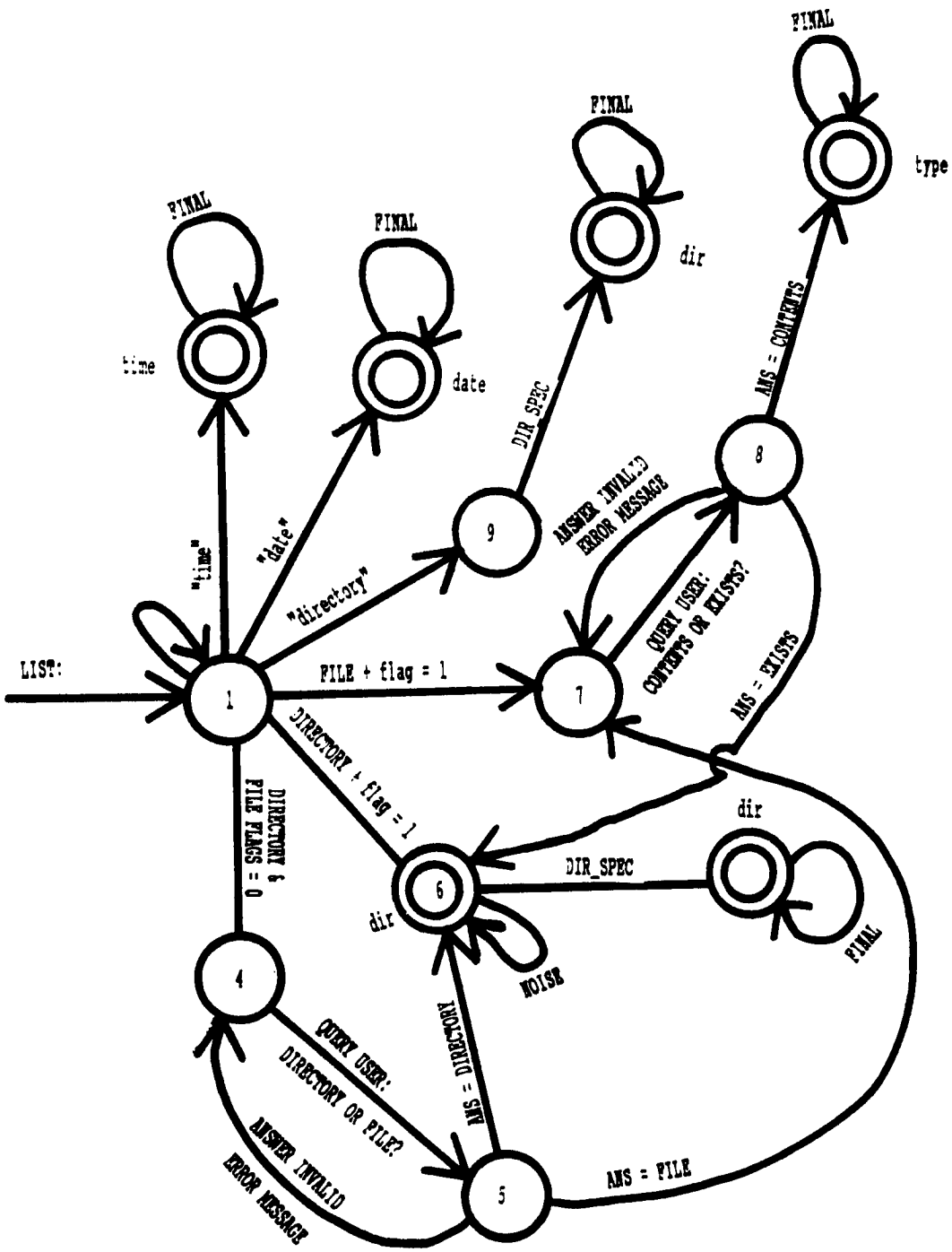


FIGURE 2: LIST

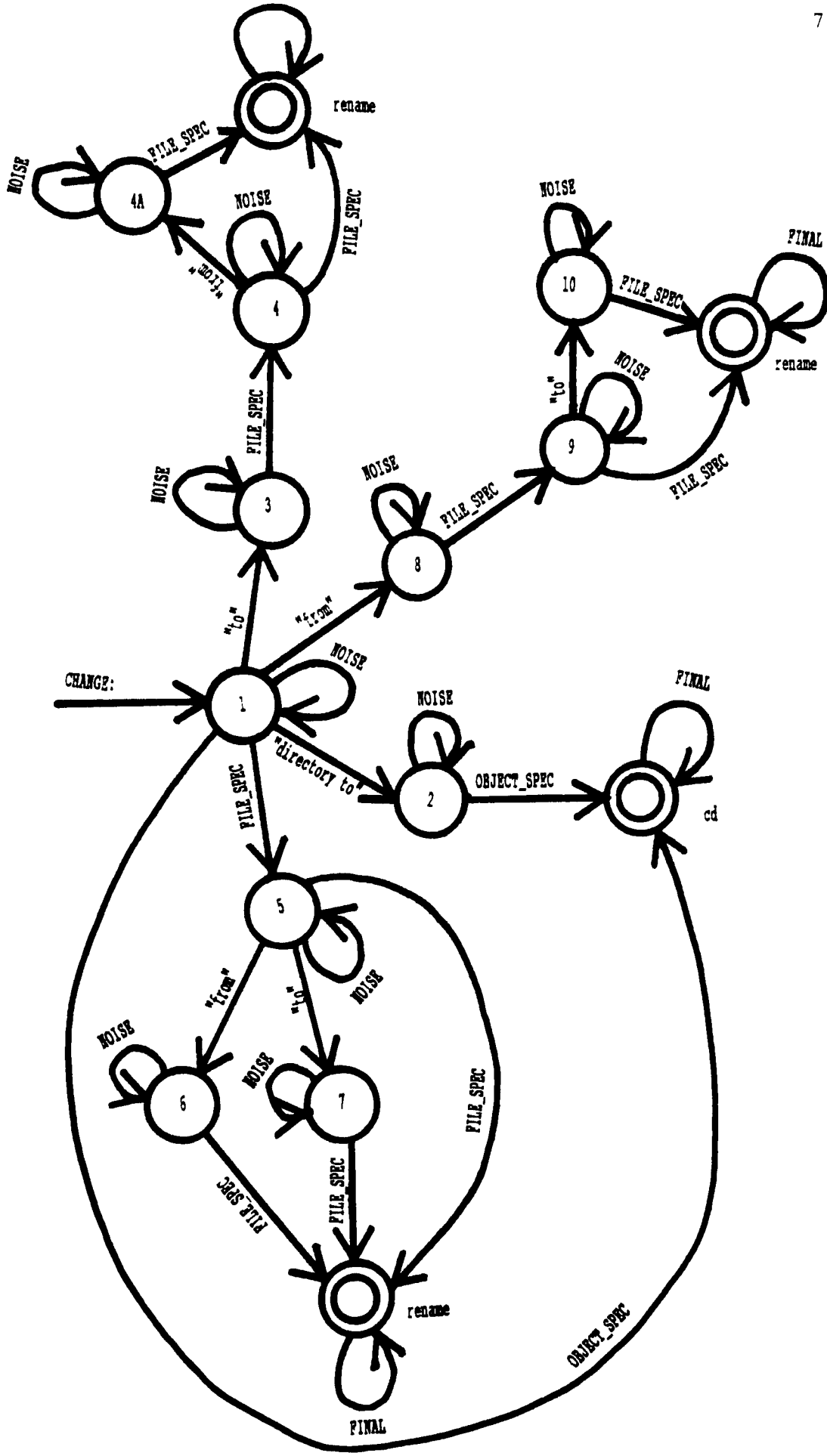


FIGURE 3: CHANGE

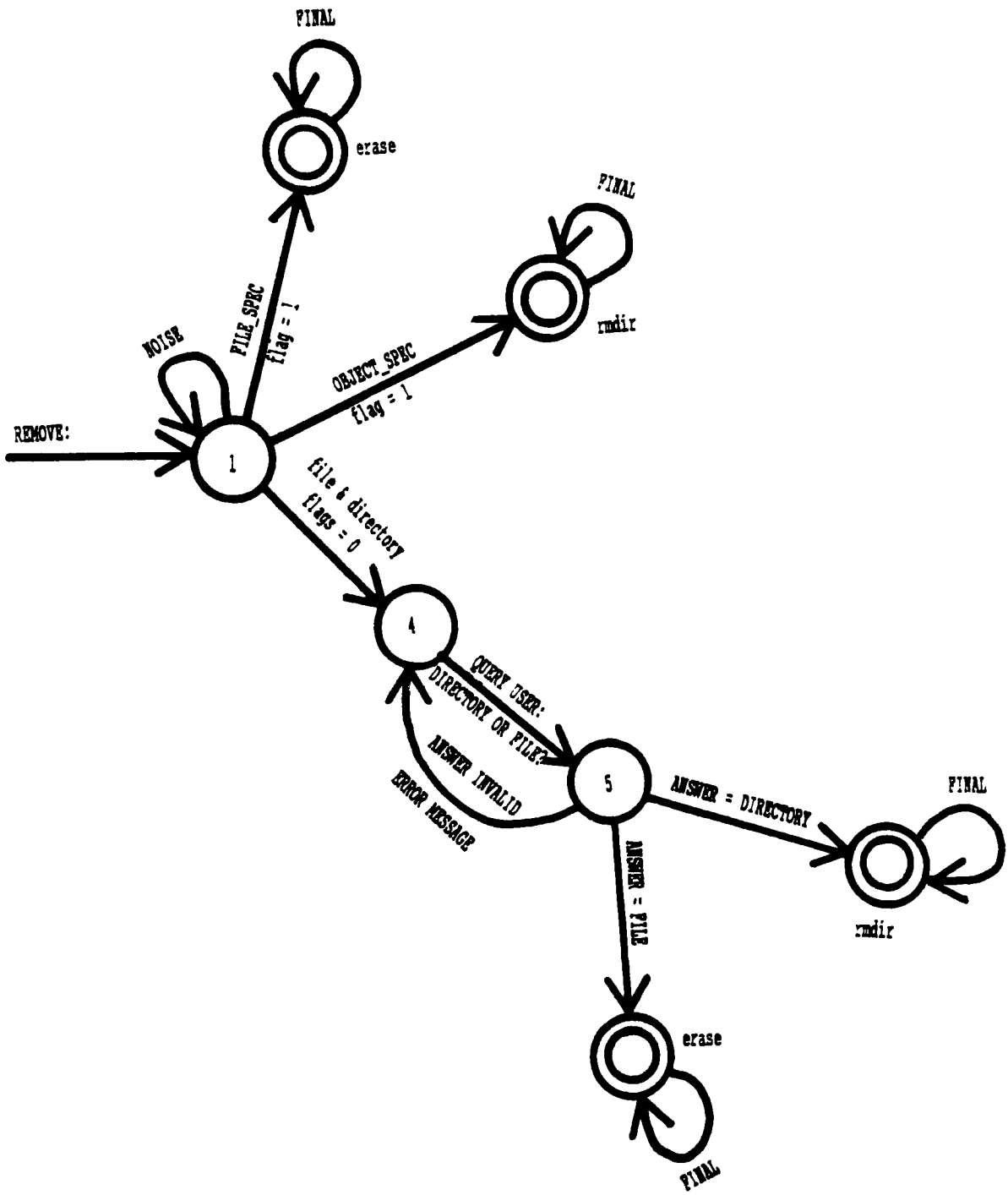


FIGURE 4: REMOVE

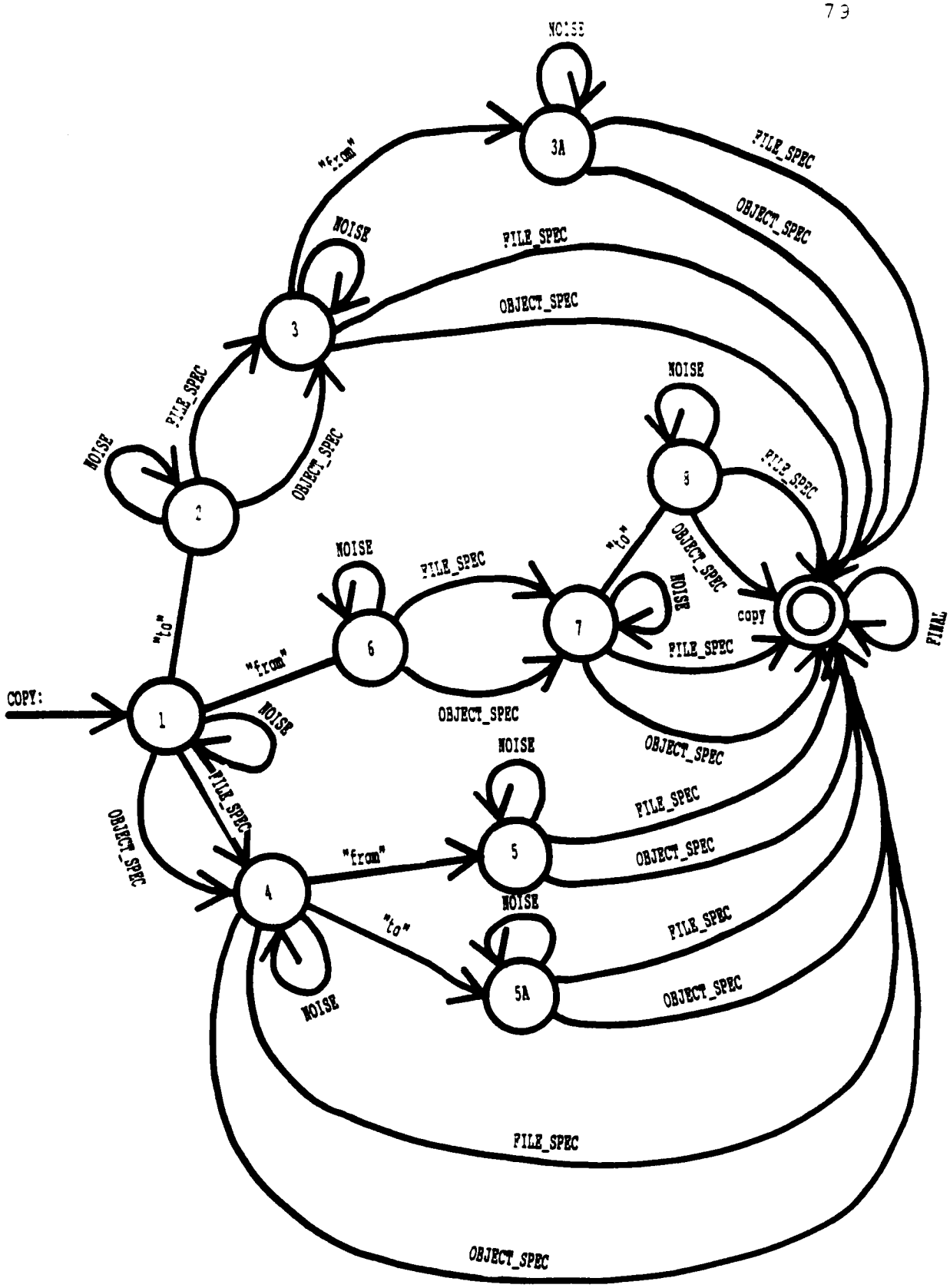


FIGURE 5: COPY

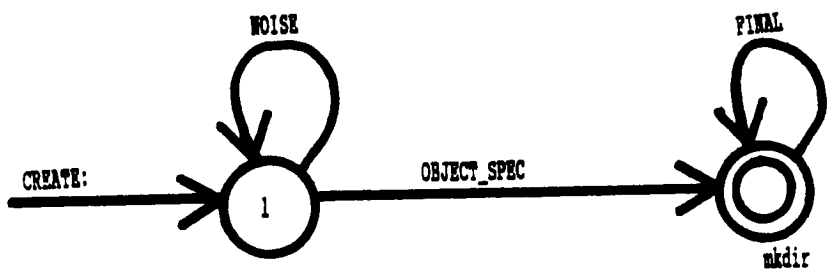


FIGURE 6: CREATE

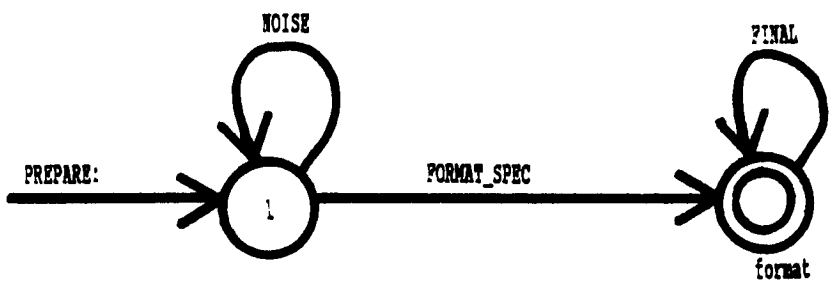


FIGURE 7: PREPARE

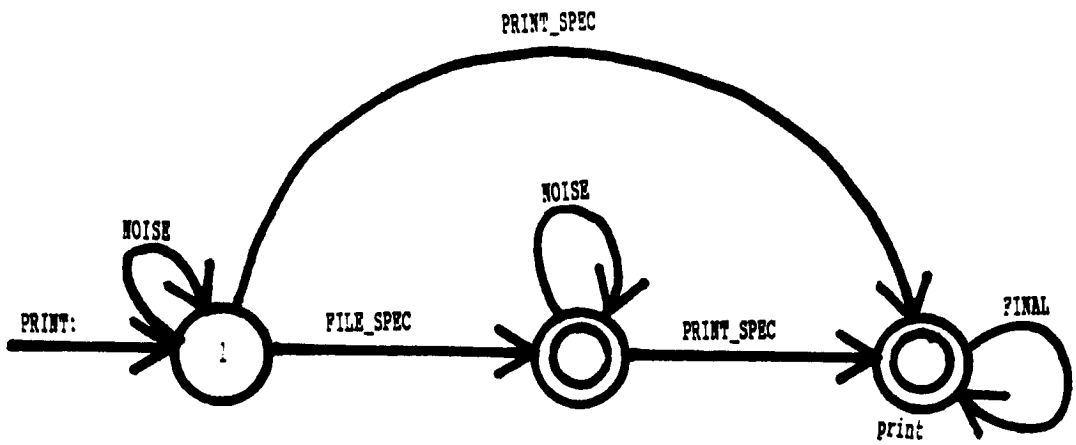


FIGURE 8: PRINT

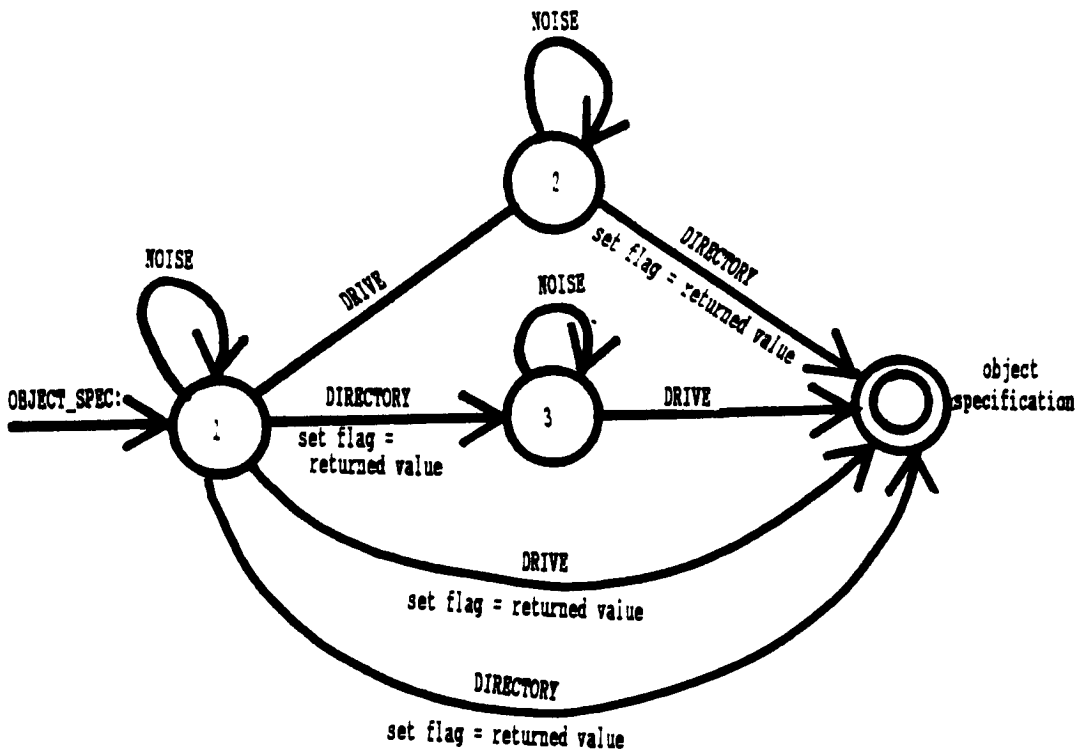


FIGURE 9: OBJECT SPECIFICATION

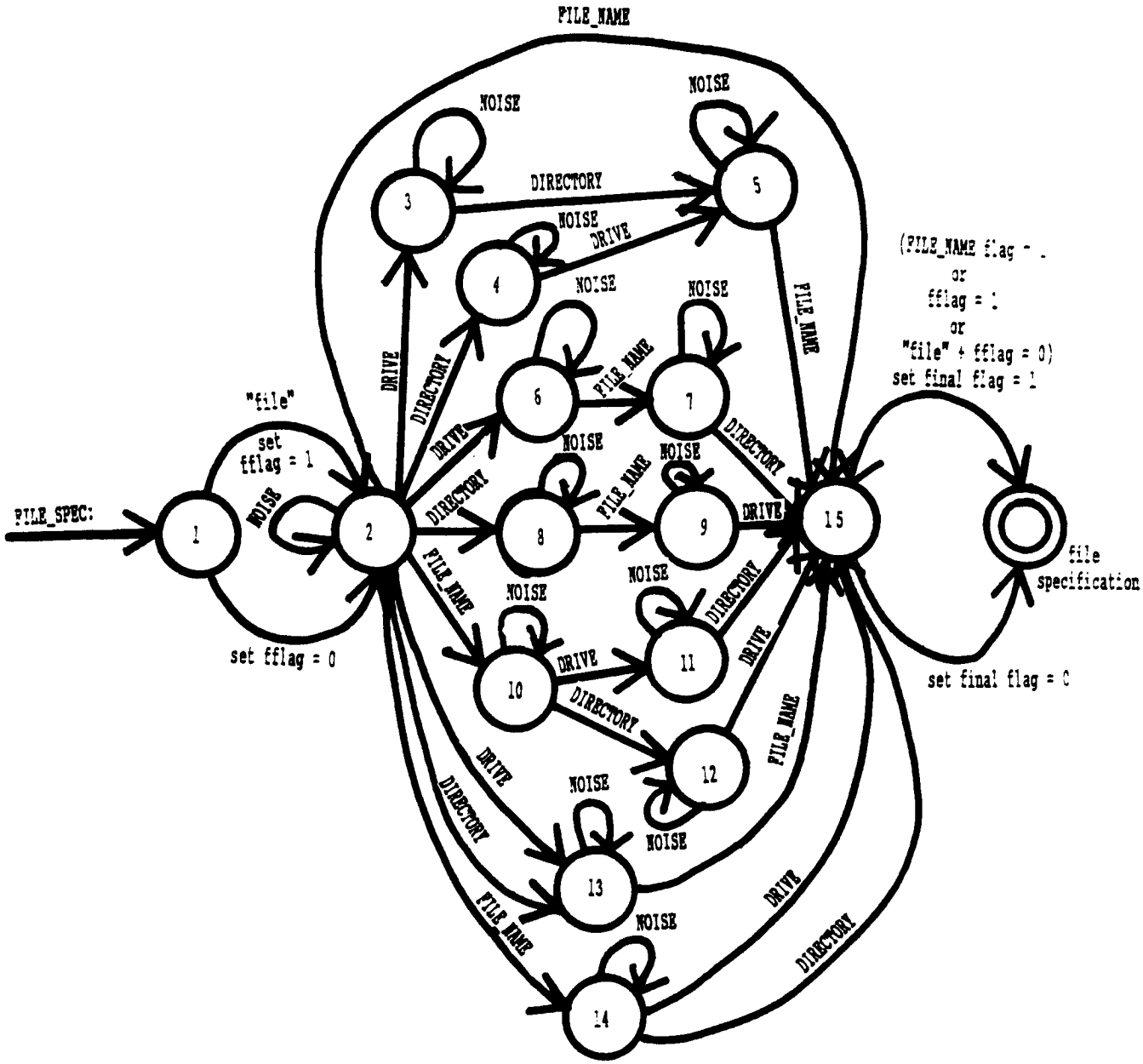


FIGURE 10: FILE SPECIFICATION

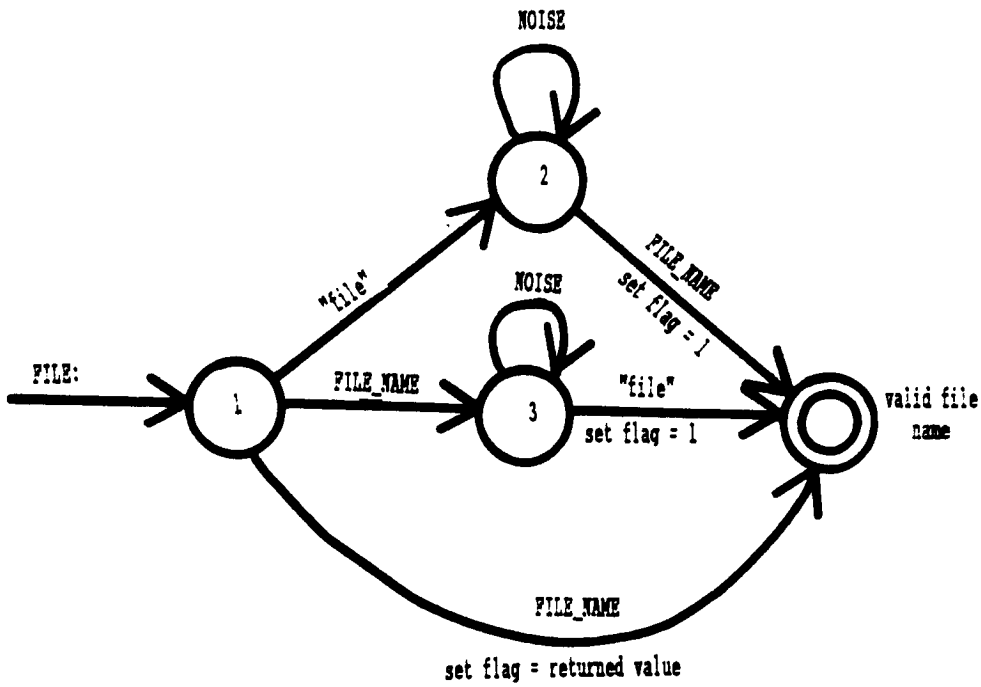


FIGURE 11: FILE

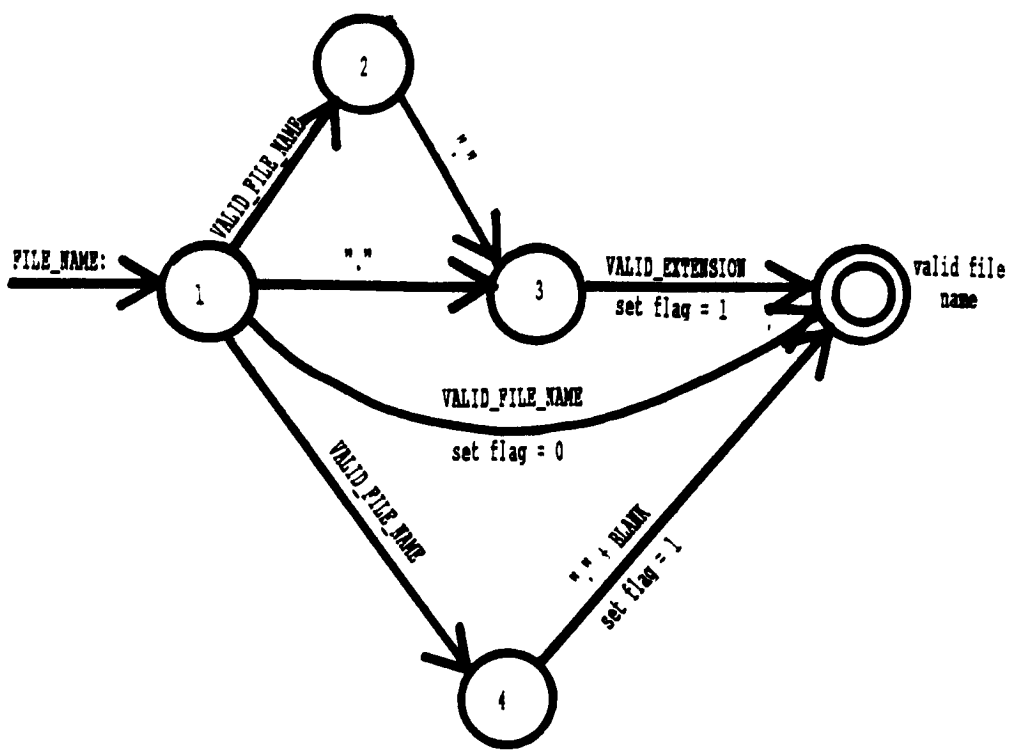


FIGURE 12: FILE NAME

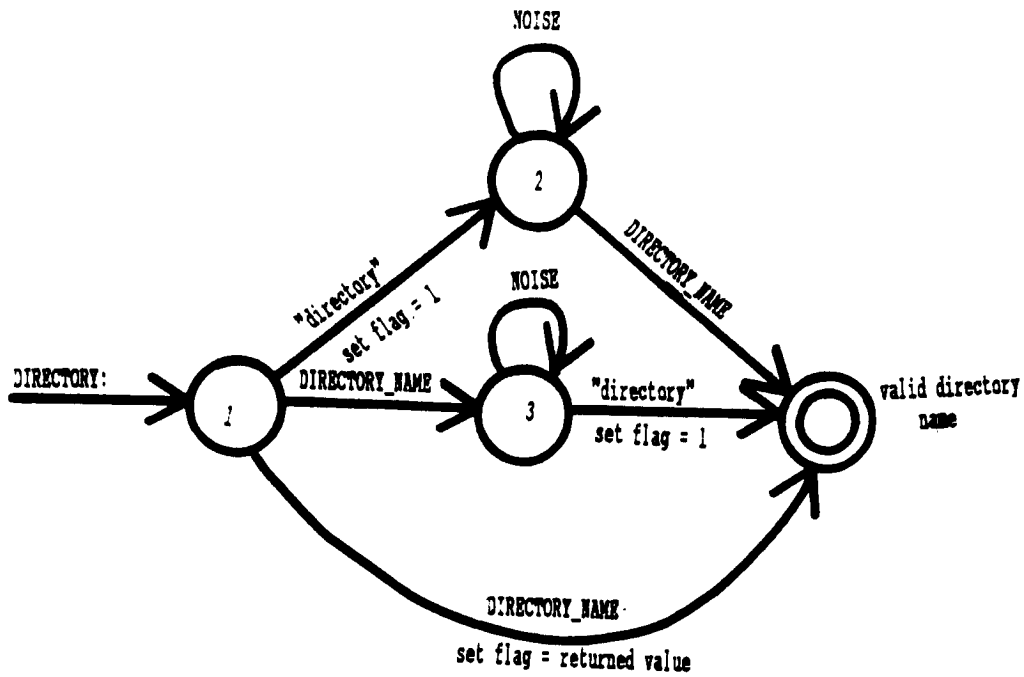


FIGURE 13: DIRECTORY

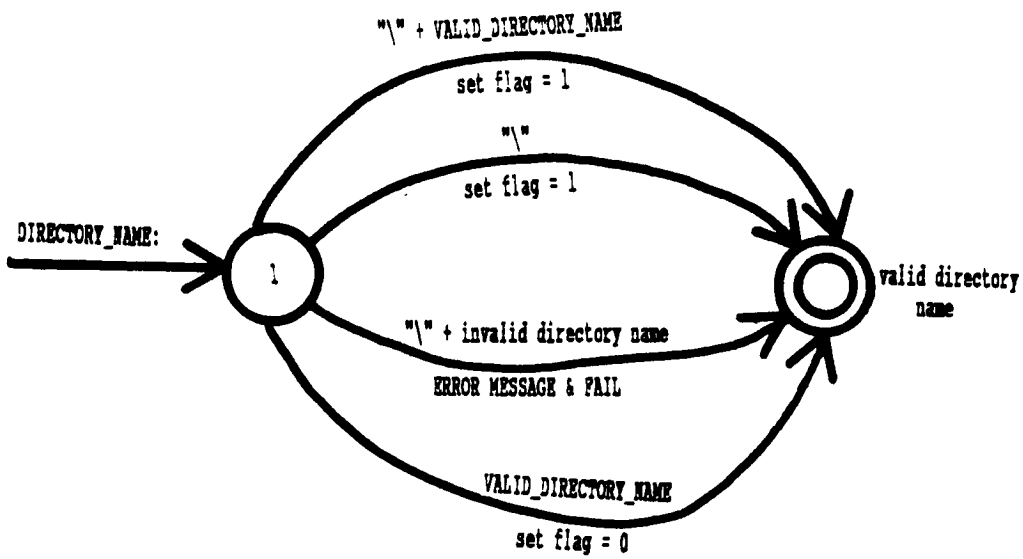


FIGURE 14: DIRECTORY NAME

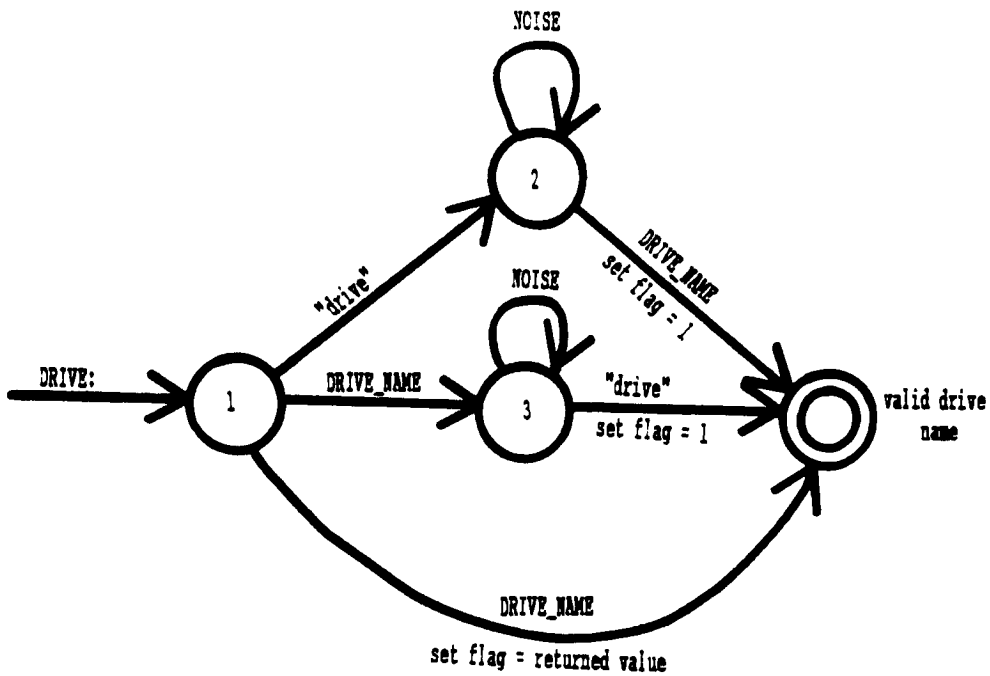
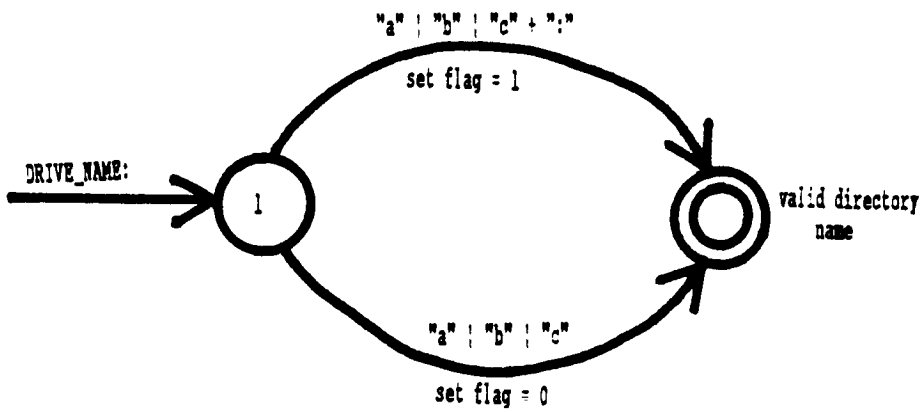


FIGURE 15: DRIVE



NOTE: valid drive names are the letters a, b and c

FIGURE 16: DRIVE NAME

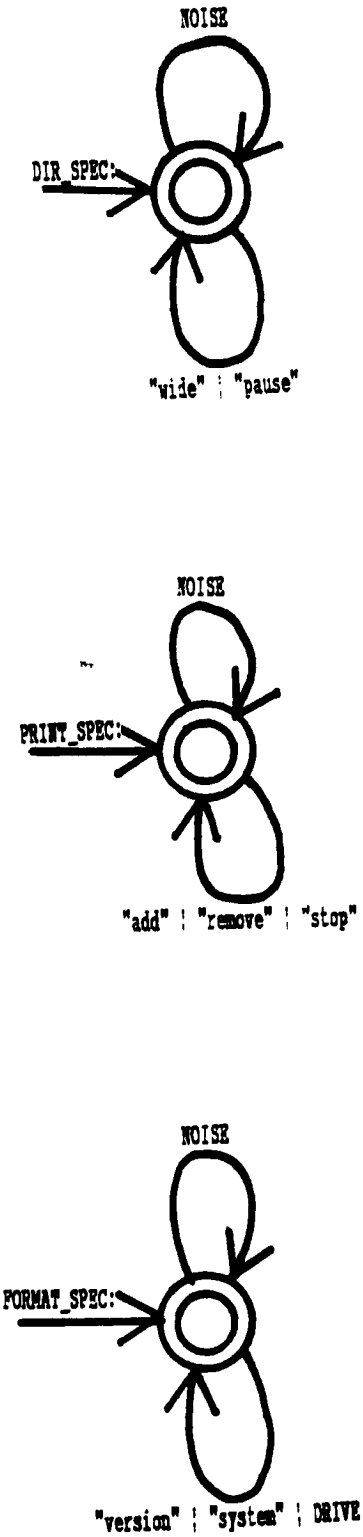


FIGURE 17: SPECIFICATIONS TO VARIOUS COMMANDS

APPENDIX B: PE_DOS CODE

```
/* file pe.ari */

/*************************************************************/
/*
Donna Blodgett Indovina                          078-52-7887
Natural Language Processing Seminar              ICSS-890
May 15, 1989

            A Natural Language Interface to MS-DOS
                      "Plain English"

    This program performs a translation between English and
the MS-DOS command language; it takes an English sentence
and by using the concepts of recursive transition networks
and pattern matching, it performs a translation into the
appropriate MS-DOS command to be executed.
                                                          */
/*************************************************************/

:- consult(lexicon).

:- consult(name_nets).

:- consult(main_nets).

:- consult(input).

:- consult(utilities).
```

```
/*************************************************************/
/*===========================================================*/
/*                          FINAL
     This clause is used to sift out any remaining noise
words once the command has been found (i.e. the final state
is found).                                                   */
/*===========================================================*/

final([]).

final(['.']).

final(['?']).

final(['!']).

final(Tokens) :-
   blank(Tokens,Rest1),
   final(Rest1).

final(Tokens) :-
   blank(Tokens,Rest1),
   noise_word(Rest1,Rest2),
   final(Rest2).

/*===========================================================*/
/*************************************************************/
```

```
/************************************************************/
/*==========================================================*/
/*                    GET FULL TOKEN                        */
/*==========================================================*/

get_full_token([' '|_],'').

get_full_token([X|Tokens],Name) :-
   get_full_token(Tokens,Rest_name),
   concat(X,Rest_name,Name).

/*==========================================================*/
/************************************************************/
```

```
/*************************************************************/
/*===========================================================*/
/*                    WELCOME CLAUSE                         */
/*===========================================================*/

welcome(Perform_flag) :-
   write('           PE_DOS'),nl,
   write('     (Plain English DOS)'),nl,nl,
   welcome_menu,
   get0(Reply),nl,nl,
   set_flag(Reply,Perform_flag),nl.

/*===========================================================*/

welcome_menu :-
   write('Use the system:'),nl,
   write(' (1) to perform commands'),nl,
   write(' (2) as a tutorial'),nl,
   write('Enter Choice: ').

/*===========================================================*/

set_flag(49,'perform').

set_flag(50,'tutorial').

set_flag(_,Perform_flag) :-
   write('*** ERROR: INVALID REPLY. ***'),nl,nl,
   welcome_menu,
   get0(Reply),nl,nl,
   set_flag(Reply,Perform_flag).

/*===========================================================*/
/*************************************************************/
```

```
/*************************************************************/
/*===========================================================*/
/*                    PERFORM COMMAND                        */
/*       This clause will execute the MS-DOS command.        */
/*===========================================================*/

perform_command(_,_,Command) :-
   var(Command),!.

perform_command('tutorial',Paraphrase,Command) :-
   write('English paraphrase: '),
   write(Paraphrase),nl,nl,
   write('MS-DOS command: '),
   write(Command),nl,nl.

perform_command('perform',Paraphrase,Command) :-
   write('English paraphrase: '),
   write(Paraphrase),nl,nl,
   write('MS-DOS command: '),
   write(Command),nl,nl,
   write('Execute command? '),
   get0(Character),
   read_character_list(Character,Answer),
   perform(Answer,Command).

/*===========================================================*/

perform(Answer,Command) :-
   word(Answer,'yes',_),nl,
write('DO NOT ACTUALLY PERFORM COMMAND'),!,nl,nl.
/*   shell(Command),!.  */

perform(Answer,_) :-
   word(Answer,'no',_),nl,
   write('Command not executed....per users request.'),!,
   nl,nl.

perform(_,Command) :-
   write('*** ERROR: INVALID RESPONSE. ***'),nl,nl,
   write('Execute command? '),
   get0(Character),
   read_character_list(Character,Answer),
   perform(Answer,Command).

/*===========================================================*/
/*************************************************************/
```

```
/*************************************************************/
/*===========================================================*/
/*
                              PE

     This is the "main" clause of the program.  It calls on
the clause prompt to query the user for the English like
command, then clause tokenize to break the inputted sentence
up into a string of tokens, then the clause analyze to find
the underlaying MS-DOS command, and finally, the clause to
perform_command to execute the command.  The fail forces
control back to the repeat to form another alteration of
the code.
                                                            */
/*===========================================================*/

pe :-
   welcome(Perform_flag),
   repeat,
      read_command(Sentence_list),nl,
      analyzel(Sentence_list,Paraphrase,Command),
      perform_command(Perform_flag,Paraphrase,Command),nl,nl,
   fail.

/*===========================================================*/
/*************************************************************/
```

```
/* file: lexicon */

/*************************************************************/
/*===========================================================*/
/*
                    LEXICON

     This is the data base of words in which the program
is able to recognize.  Each recognizeable word or token is
associated with a synonym, possibly itself (i.e. the token
display has the synonym display).  The clauses are set up
as differences lists using the following structure:

           word([Token|Rest],Synonym,Rest).

     word = the clause name.
     Token = the word or phrase being looked up.
     Rest = the rest of the list of words.
     Synonym = the synonym for Token.

     During the analysis process the network structure
calls upon the word clause frequently to determine the
meaning of the current token.  It passes the entire
remaining list of tokens.  The clause word strips off the
identified token and returns the synonym and the rest of
the list back to the calling clause.
                                                          */
/*===========================================================*/

/* verbs */

word([display|Rest],list,Rest).
word([show|Rest],list,Rest).
word([list|Rest],list,Rest).
word([reveal|Rest],list,Rest).
word([give|Rest],list,Rest).
word([present|Rest],list,Rest).
word([exhibit|Rest],list,Rest).
word([view|Rest],list,Rest).
word([catalog|Rest],list,Rest).
word([indicate|Rest],list,Rest).
word([tell|Rest],list,Rest).
word([scroll|Rest],list,Rest).
word([type|Rest],list,Rest).
```

```
word([go,' ',to|Rest],change,Rest).
word([change|Rest],change,Rest).
word([rename|Rest],change,Rest).
word([move|Rest],change,Rest).
word([shift|Rest],change,Rest).
word([alter|Rest],change,Rest).
word([convert|Rest],change,Rest).
word([set|Rest],change,Rest).
word([amend|Rest],change,Rest).
word([modify|Rest],change,Rest).
word([translate|Rest],change,Rest).
word([transfer|Rest],change,Rest).

word([zap,' ',out|Rest],remove,Rest).
word([cross,' ',out|Rest],remove,Rest).
word([take,' ',away|Rest],remove,Rest).
word([cut,' ',out|Rest],remove,Rest).
word([scratch,' ',out|Rest],remove,Rest).
word([rub,' ',out|Rest],remove,Rest).
word([do,' ',away,' ',with|Rest],remove,Rest).
word([remove|Rest],remove,Rest).
word([eliminate|Rest],remove,Rest).
word([dislodge|Rest],remove,Rest).
word([abolish|Rest],remove,Rest).
word([erase|Rest],remove,Rest).
word([cut|Rest],remove,Rest).
word([rub|Rest],remove,Rest).
word([zap|Rest],remove,Rest).
word([destroy|Rest],remove,Rest).
word([kill|Rest],remove,Rest).
word([scratch|Rest],remove,Rest).

word([copy|Rest],copy,Rest).
word([cp|Rest],copy,Rest).
word([reproduce|Rest],copy,Rest).
word([dup|Rest],copy,Rest).
word([duplicate|Rest],copy,Rest).
word([transcribe|Rest],copy,Rest).
word([rewrite|Rest],copy,Rest).
word([model|Rest],copy,Rest).
word([double|Rest],copy,Rest).
word([match|Rest],copy,Rest).
word([duplify|Rest],copy,Rest).

word([compare|Rest],compare,Rest).
word([comp|Rest],compare,Rest).
word([commensurate|Rest],compare,Rest).
word([collate|Rest],compare,Rest).
word([contrast|Rest],compare,Rest).
word([match|Rest],compare,Rest).
word([equal|Rest],compare,Rest).
word([tie|Rest],compare,Rest).
word([balance|Rest],compare,Rest).
```

```
word([make|Rest],create,Rest).
word([build|Rest],create,Rest).
word([construct|Rest],create,Rest).
word([synthesize|Rest],create,Rest).
word([produce|Rest],create,Rest).
word([erect|Rest],create,Rest).
word([create|Rest],create,Rest).
word([fabricate|Rest],create,Rest).
word([constitute|Rest].create,Rest).
word([compose|Rest],create,Rest).
word([yield|Rest],create,Rest).

word([prepare|Rest],prepare,Rest).
word([format|Rest],prepare,Rest).
word([arrange|Rest],prepare,Rest).
word([ready|Rest],prepare,Rest).
word([groom|Rest],prepare,Rest).

word([print|Rest],print,Rest).
word([impress|Rest],print,Rest).
word([impression|Rest],print,Rest).
word([imprint|Rest],print,Rest).

word([equal|Rest],equal,Rest).
word([match|Rest],equal,Rest).
word([contrast|Rest],equal,Rest).
word([tie|Rest],equal,Rest).
word([balance|Rest],equal,Rest).
word([compare|Rest],equal,Rest).
word(['='|Rest],equal,Rest).

word([e|Rest],exit,Rest).
word([exit|Rest],exit,Rest).
word([logout|Rest],exit,Rest).
word([bye|Rest],exit,Rest).
word([finish|Rest],exit,Rest).
word([finished|Rest],exit,Rest).

word([cd|Rest],cd,Rest).
word([chdir|Rest],cd,Rest).

/* nouns */

word([date|Rest],date,Rest).
word([day|Rest],date,Rest).

word([time|Rest],time,Rest).
word([hour|Rest],time,Rest).

word([files|Rest],files,Rest).

word([file,' ',name|Rest],file,Rest).
word([f|Rest],file,Rest).
word([file|Rest],file,Rest).
```

```
word([directory,' ',files|Rest],directory,Rest).
word([directory,' ',name|Rest],directory,Rest).
word([d|Rest],directory,Rest).
word([dir|Rest],directory,Rest).
word([directory|Rest],directory,Rest).
word([subdirectory|Rest],directory,Rest).
word([subdir|Rest],directory,Rest).
word([all,' ',files|Rest],directory,Rest).
word([files|Rest],directory,Rest).
word([all,' ',directory|Rest],directory,Rest).

word([drive, ' ',name|Rest],drive,Rest).
word([drive|Rest],drive,Rest).

/* preposition */

word([to|Rest],to,Rest).
word([into|Rest],to,Rest).
word([with|Rest],to,Rest).
word([on|Rest],to,Rest).
word([onto|Rest],to,Rest).

word([from|Rest],from,Rest).

/* auxiliary */

word([what|Rest],what,Rest).
word([whats|Rest],what,Rest).
word([X|Rest],what,Rest) :-
   name(X,[119,104,97,116,39,115]).
word([what,is|Rest],what,Rest).

word([do|Rest],do,Rest).
word([are|Rest],do,Rest).
word([does|Rest],do,Rest).
word([is|Rest],do,Rest).

/* miscellaneous */

word([yes|Rest],yes,Rest).
word([y|Rest],yes,Rest).

word([no|Rest],no,Rest).
word([n|Rest],no,Rest).
```

```
/* noise words */

noise_word([the|Rest],Rest).
noise_word([a|Rest],Rest).
noise_word([me|Rest],Rest).
noise_word([you|Rest],Rest).
noise_word([it|Rest],Rest).
noise_word([is|Rest],Rest).
noise_word([contents|Rest],Rest).
noise_word([of|Rest],Rest).
noise_word([please|Rest],Rest).
noise_word([from|Rest],Rest).
noise_word([name|Rest],Rest).
noise_word([and|Rest],Rest).
noise_word([all|Rest],Rest).
noise_word([to|Rest],Rest).
noise_word([into|Rest],Rest).
noise_word([in|Rest],Rest).
noise_word([with|Rest],Rest).
noise_word([on|Rest],Rest).
noise_word([for|Rest],Rest).
noise_word([use|Rest],Rest).
noise_word([as|Rest],Rest).
noise_word([disk|Rest],Rest).
noise_word([onto|Rest],Rest).
noise_word([from|Rest],Rest).
noise_word([all|Rest],Rest).
noise_word([every|Rest],Rest).
noise_word([with|Rest],Rest).
noise_word([content|Rest],Rest).
noise_word([contents|Rest],Rest).
noise_word([style|Rest],Rest).
noise_word([format|Rest],Rest).
noise_word([',|Rest],Rest).
noise_word([';|Rest],Rest).
noise_word([' '|Rest],Rest).

/* commands */

command([list|Rest],list,Rest).
command([change|Rest],change,Rest).
command([remove|Rest],remove,Rest).
command([copy|Rest],copy,Rest).
command([create|Rest],create,Rest).
command([prepare|Rest],prepare,Rest).
command([print|Rest],print,Rest).
command([dir|Rest],dir,Rest).
command([type|Rest],type,Rest).
command([cd|Rest],cd,Rest).
command([cdir|Rest],cdir,Rest).
command([mkdir|Rest],mkdir,Rest).
command([mk|Rest],mkdir,Rest).
command([rmdir|Rest],rmdir,Rest).
command([rd|Rest],rmdir,Rest).
command([exit|Rest],exit,Rest).
```

```
/* specifications */

dirspec([wide|Rest],' in wide format','/w',Rest).
dirspec([/,wide|Rest],' in wide format','/w',Rest).
dirspec([/,w|Rest],' in wide format','/w',Rest).

dirspec([pause|Rest],' and pause between pages','/p',Rest).
dirspec([/,pause|Rest],' and pause between pages', '/p',Rest).
dirspec([/,p|Rest],' and pause between pages', '/p',Rest).

printspec([add|Rest],' add to print que','/p',Rest).
printspec([add,' ',file|Rest],' add to print que', '/p',Rest).
printspec([/,add|Rest],' add to print que', '/p',Rest).
printspec([/,p|Rest],' add to print que', '/p',Rest).

printspec([X|Rest],' remove from print que','/c',Rest) :-
   word([X],remove,[]).
printspec([X,' ',file|Rest],' remove from print que', '/c',Rest)

   word([X],remove,[]).
printspec([/,X|Rest],' remove from print que', '/c',Rest):-
   word([X],remove,[]).
printspec([/,c|Rest],' remove from print que', '/c',Rest).

printspec([stop|Rest],' stop print que','/t',Rest).
printspec([stop,' ',printing|Rest],' stop print que', '/t',Rest)

printspec([stop,' ',print|Rest],' stop print que', '/t',Rest).
printspec([/,stop|Rest],' stop print que', '/t',Rest).
printspec([/,t|Rest],' stop print que', '/t',Rest).

formatspec([version|Rest],' as system version','/v',Rest).
formatspec([/,version|Rest],' as system version','/v',Rest).
formatspec([v|Rest],' as system version','/v',Rest).
formatspec([/,v|Rest],' as system version','/v',Rest).

formatspec([system|Rest],' as system disk','/s',Rest).
formatspec([/,system|Rest],' as system disk','/s',Rest).
formatspec([s|Rest],' as system disk','/s',Rest).
formatspec([/,s|Rest],' as system disk','/s',Rest).

/* blank space */

blank([' '|Rest],Rest).

/*=========================================================*/
/*********************************************************/
```

```
/* file: name_nets.ari */

/*********************************************************/
/*=======================================================*/
/*          OBJECT SPECIFICATIONS NETWORK                */
/*=======================================================*/

object_spec1(Tokens,Para_object,Object,Rest3,Flag) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   object_spec1(Rest2,Para_object,Object,Rest3,Flag).

object_spec1(Tokens,Para_object,Object,Rest3,Flag) :-
   drive1(Tokens,Drive,Rest1,_),
   blank(Rest1,Rest2),
   object_spec2(Rest2,Para_object,Drive,Object,Rest3,Flag).

object_spec1(Tokens,Para_object,Object,Rest3,Flag) :-
   directory1(Tokens,Directory,Rest1,Flag),
   blank(Rest1,Rest2),
   object_spec3(Rest2,Para_object,Directory,Object,Rest3,_).

object_spec1(Tokens,Para_object,Object,Rest,Flag) :-
   drive1(Tokens,Object,Rest,Flag),
   concat('drive ',Object,Para_object).

object_spec1(Tokens,Para_object,Object,Rest,Flag) :-
   directory1(Tokens,Object,Rest,Flag),
   concat('directory ',Object,Para_object).

object_spec2(Tokens,Para_object,Drive,Object,Rest3,Flag) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   object_spec2(Rest2,Para_object,Drive,Object,Rest3,Flag).

object_spec2(Tokens,Para_object,Drive,Object,Rest,Flag) :-
   directory1(Tokens,Directory,Rest,Flag),
   concat(Drive,Directory,Object),
   concat('drive ',Drive,Temp1),
   concat(Temp1,' directory ',Temp2),
   concat(Temp2,Directory,Para_object).

object_spec3(Tokens,Para_object,Directory,Object,Rest3,_) :-
   noise_word(Tokens,Rest1),
   object_spec3(Rest2,Para_object,Directory,Object,Rest3,_).

object_spec3(Tokens,Para_object,Directory,Object,Rest,
             Flag) :-
   drive1(Tokens,Drive,Rest,_),
   concat(Drive,Directory,Object),
   concat('drive ',Drive,Temp1),
   concat(Temp1,' directory ',Temp2),
   concat(Temp2,Directory,Para_object).
```

```
/*=========================================================*/
/*              FILE SPECIFICATIONS NETWORK                */
/*=========================================================*/

file_spec1(Tokens,Para_file,File_spec,Rest3,Flag) :-
   word(Tokens,'file',Rest1),
   blank(Rest1,Rest2),
   file_spec2(Rest2,Para_file,File_spec,Rest3,Flag,1).

file_spec1(Tokens,Para_file,File_spec,Rest,Flag) :-
   file_spec2(Tokens,Para_file,File_spec,Rest,Flag,0).

file_spec2(Tokens,Para_file,File_spec,Rest3,Flag,
           Temp_flag1) :-
   ((noise_word(Tokens,Rest1),
     blank(Rest1,Rest2));
    (blank(Tokens,Rest2))),
   file_spec2(Rest2,Para_file,File_spec,Rest3,Flag,
              Temp_flag1).

file_spec2(Tokens,Para_file,File_spec,Rest2,Flag,
           Temp_flag1) :-
   file1(Tokens,File_spec,Rest1,Temp_flag2),
   concat('file ',File_spec,Para_file),
   file_spec15(Rest1,Rest2,Flag,Temp_flag1,Temp_flag2).

file_spec2(Tokens,Para_file,File_spec,Rest3,Flag,
           Temp_flag1) :-
   drive1(Tokens,Drive,Rest1,_),
   blank(Rest1,Rest2),
   file_spec13(Rest2,Para_file,Drive,File_spec,Rest3,Flag,
               Temp_flag1).

file_spec2(Tokens,Para_file,File_spec,Rest3,Flag,
           Temp_flag1) :-
   directory1(Tokens,Directory,Rest1,_),
   blank(Rest1,Rest2),
   file_spec13(Rest2,Para_file,Directory,File_spec,Rest3,
               Flag,Temp_flag1).

file_spec2(Tokens,Para_file,File_spec,Rest3,Flag,
           Temp_flag1) :-
   file1(Tokens,File,Rest1,Temp_flag2),
   blank(Rest1,Rest2),
   file_spec14(Rest2,Para_file,File,File_spec,Rest3,Flag,
               Temp_flag1,Temp_flag2).

file_spec2(Tokens,Para_file,File_spec,Rest3,Flag,
           Temp_flag1) :-
   drive1(Tokens,Drive,Rest1,_),
   blank(Rest1,Rest2),
   file_spec3(Rest2,Para_file,Drive,File_spec,Rest3,Flag,
              Temp_flag1).
```

```
file_spec2(Tokens,Para_file,File_spec,Rest3,Flag,
           Temp_flag1) :-
   directory1(Tokens,Directory,Rest1,_),
   blank(Rest1,Rest2),
   file_spec4(Rest2,Para_file,Directory,File_spec,Rest3,Flag,
              Temp_flag1).

file_spec2(Tokens,Para_file,File_spec,Rest3,Flag,
           Temp_flag1) :-
   drive1(Tokens,Drive,Rest1,_),
   blank(Rest1,Rest2),
   file_spec6(Rest2,Para_file,Drive,File_spec,Rest3,Flag,
              Temp_flag1).

file_spec2(Tokens,Para_file,File_spec,Rest3,Flag,
           Temp_flag1) :-
   directory1(Tokens,Directory,Rest1,_),
   blank(Rest1,Rest2),
   file_spec8(Rest2,Para_file,Directory,File_spec,Rest3,Flag,
              Temp_flag1).

file_spec2(Tokens,Para_file,File_spec,Rest3,Flag,
           Temp_flag1) :-
   file1(Tokens,File,Rest1,Temp_flag2),
   blank(Rest1,Rest2),
   file_spec10(Rest2,Para_file,File,File_spec,Rest3,Flag,
               Temp_flag1, Temp_flag2).

file_spec3(Tokens,Para_file,Drive,File_spec,Rest3,Flag,
           Temp_flag1) :-
   ((noise_word(Tokens,Rest1),
     blank(Rest1,Rest2));
    (blank(Tokens,Rest2))),
   file_spec3(Rest2,Para_file,Drive,File_spec,Rest3,Flag,
              Temp_flag1).

file_spec3(Tokens,Para_file,Drive,File_spec,Rest3,Flag,
           Temp_flag1) :-
   directory1(Tokens,Directory,Rest1,_),
   blank(Rest1,Rest2),
   file_spec5(Rest2,Para_file,Drive,Directory,File_spec,
              Rest3,Flag,Temp_flag1).

file_spec4(Tokens,Para_file,Drive,File_spec,Rest3,Flag,
           Temp_flag1) :-
   ((noise_word(Tokens,Rest1),
     blank(Rest1,Rest2));
    (blank(Tokens,Rest2))),
   file_spec4(Rest2,Para_file,Drive,File_spec,Rest3,Flag,
              Temp_flag1).
```

```
file_spec4(Tokens,Para_file,Directory,File_spec,Rest3,Flag,
           Temp_flag1) :-
   drive1(Tokens,Drive,Rest1,_),
   blank(Rest1,Rest2),
   file_spec5(Rest2,Para_file,Drive,Directory,File_spec,
              Rest3,Flag,Temp_flag1).

file_spec5(Tokens,Para_file,Drive,Directory,File_spec,Rest3,
           Flag,Temp_flag1) :-
   ((noise_word(Tokens,Rest1),
     blank(Rest1,Rest2));
    (blank(Tokens,Rest2))),
   file_spec5(Rest2,Para_file,Drive,Directory,File_spec,
              Rest3,Flag,Temp_flag1).

file_spec5(Tokens,Para_file,Drive,Directory,File_spec,Rest2,
           Flag,Temp_flag1) :-
   file1(Tokens,File,Rest1,Temp_flag2),
   file_spec15(Rest1,Rest2,Flag,1,Temp_flag2),
   concat(Drive,Directory,Temp11),
   concat(Temp11,'\',Temp12),
   concat(Temp12,File,File_spec),
   concat('drive ',Drive,Temp1),
   concat(Temp1,' directory ',Temp2),
   concat(Temp2,Directory,Temp3),
   concat(Temp3,' file ',Temp4),
   concat(Temp4,File,Para_file).

file_spec6(Tokens,Para_file,Drive,File_spec,Rest3,Flag,
           Temp_flag1) :-
   ((noise_word(Tokens,Rest1),
     blank(Rest1,Rest2));
    (blank(Tokens,Rest2))),
   file_spec6(Rest2,Para_file,Drive,File_spec,Rest3,Flag,
              Temp_flag1).

file_spec6(Tokens,Para_file,Drive,File_spec,Rest3,Flag,
           Temp_flag1) :-
   file1(Tokens,File,Rest1,Temp_flag2),
   blank(Rest1,Rest2),
   file_spec7(Rest2,Para_file,Drive,File,File_spec,
              Rest3,Flag, Temp_flag1,Temp_flag2).

file_spec7(Tokens,Para_file,Drive,File,File_spec,Rest3,Flag,
           Temp_flag1,Temp_flag2) :-
   ((noise_word(Tokens,Rest1),
     blank(Rest1,Rest2));
    (blank(Tokens,Rest2))),
   file_spec7(Rest2,Para_file,Drive,File,File_spec,
              Rest3,Flag, Temp_flag1,Temp_flag2).
```

```
file_spec7(Tokens,Para_file,Drive,File,File_spec,Rest2,Flag,
           Temp_flag1,Temp_flag2) :-
   directory1(Tokens,Directory,Rest1,_),
   file_spec15(Rest1,Rest2,Flag,1,Temp_flag2),
   concat(Drive,Directory,Temp11),
   concat(Temp11,'\',Temp12),
   concat(Temp12,File,File_spec),
   concat('drive ',Drive,Temp1),
   concat(Temp1,' directory ',Temp2),
   concat(Temp2,Directory,Temp3),
   concat(Temp3,' file ',Temp4),
   concat(Temp4,File,Para_file).

file_spec8(Tokens,Para_file,Directory,File_spec,Rest2,Flag,
           Temp_flag1) :-
   ((noise_word(Tokens,Rest1),
     blank(Rest1,Rest2));
    (blank(Tokens,Rest2))),
   file_spec8(Rest1,Para_file,Directory,File_spec,Rest2,Flag,
              Temp_flag1).

file_spec8(Tokens,Para_file,Directory,File_spec,Rest3,Flag,
           Temp_flag1) :-
   file1(Tokens,File,Rest1,Temp_flag2),
   blank(Rest1,Rest2),
   file_spec9(Rest2,Para_file,Directory,File,File_spec,
              Rest3,Flag, Temp_flag1,Temp_flag2).

file_spec9(Tokens,Para_file,Directory,File,File_spec,
           Rest3,Flag, Temp_flag1,Temp_flag2) :-
   ((noise_word(Tokens,Rest1),
     blank(Rest1,Rest2));
    (blank(Tokens,Rest2))),
   file_spec9(Rest2,Para_file,Directory,File,File_spec,
              Rest3,Flag, Temp_flag1,Temp_flag2).

file_spec9(Tokens,Para_file,Directory,File,File_spec,
           Rest2,Flag, Temp_flag1,Temp_flag2) :-
   drive1(Tokens,Drive,Rest1,_),
   file_spec15(Rest1,Rest2,Flag,1,Temp_flag2),
   concat(Drive,Directory,Temp11),
   concat(Temp11,'\',Temp12),
   concat(Temp12,File,File_spec),
   concat('drive ',Drive,Temp1),
   concat(Temp1,' directory ',Temp2),
   concat(Temp2,Directory,Temp3),
   concat(Temp3,' file ',Temp4),
   concat(Temp4,File,Para_file).
```

```
file_spec10(Tokens,Para_file,File,File_spec,Rest3,Flag,
            Temp_flag1,Temp_flag2) :-
   ((noise_word(Tokens,Rest1),
     blank(Rest1,Rest2));
    (blank(Tokens,Rest2))),
   file_spec10(Rest2,Para_file,File,File_spec,Rest3,Flag,
               Temp_flag1,Temp_flag2).

file_spec10(Tokens,Para_file,File,File_spec,Rest3,Flag,
            Temp_flag1,Temp_flag2) :-
   drive1(Tokens,Drive,Rest1,_),
   blank(Rest1,Rest2),
   file_spec11(Rest2,Para_file,Drive,File,File_spec,Rest3,
               Flag,Temp_flag1,Temp_flag2).

file_spec10(Tokens,Para_file,File,File_spec,Rest3,Flag,
            Temp_flag1,Temp_flag2) :-
   directory1(Tokens,Directory,Rest1,_),
   blank(Rest1,Rest2),
   file_spec12(Rest2,Para_file,Directory,File,File_spec,
               Rest3,Flag,Temp_flag1,Temp_flag2).

file_spec11(Tokens,Para_file,Drive,File,File_spec,Rest3,
            Flag,Temp_flag1,Temp_flag2) :-
   ((noise_word(Tokens,Rest1),
     blank(Rest1,Rest2));
    (blank(Tokens,Rest2))),
   file_spec11(Rest2,Para_file,Drive,File,File_spec,Rest3,
               Flag,Temp_flag1,Temp_flag2).

file_spec11(Tokens,Para_file,Drive,File,File_spec,Rest2,
            Flag,Temp_flag1,Temp_flag2) :-
   directory1(Tokens,Directory,Rest1,_),
   file_spec15(Rest1,Rest2,Flag,1,Temp_flag2),
   concat(Drive,Directory,Temp11),
   concat(Temp11,'\',Temp12),
   concat(Temp12,File,File_spec),
   concat('drive ',Drive,Temp1),
   concat(Temp1,' directory ',Temp2),
   concat(Temp2,Directory,Temp3),
   concat(Temp3,' file ',Temp4),
   concat(Temp4,File,Para_file).

file_spec12(Tokens,Para_file,Directory,File,File_spec,Rest3
            Flag,Temp_flag1,Temp_flag2) :-
   ((noise_word(Tokens,Rest1),
     blank(Rest1,Rest2));
    (blank(Tokens,Rest2))),
   file_spec12(Rest2,Para_file,Directory,File,File_spec,
               Rest3,Flag,Temp_flag1,Temp_flag2).
```

```
file_spec12(Tokens,Para_file,Directory,File,File_spec,Rest2
            Flag,Temp_flag1,Temp_flag2) :-
   drive1(Tokens,Drive,Rest1,_),
   file_spec15(Rest1,Rest2,Flag,1,Temp_flag2),
   concat(Drive,Directory,Temp11),
   concat(Temp11,'\',Temp12),
   concat(Temp12,File,File_spec),
   concat('drive ',Drive,Temp1),
   concat(Temp1,' directory ',Temp2),
   concat(Temp2,Directory,Temp3),
   concat(Temp3,' file ',Temp4),
   concat(Temp4,File,Para_file).

file_spec13(Tokens,Para_file,Object,File_spec,
            Rest3,Flag,Temp_flag1) :-
((noise_word(Tokens,Rest1),
     blank(Rest1,Rest2));
    (blank(Tokens,Rest2))),
   file_spec13(Rest2,Para_file,Object,File_spec,
               Rest3,Flag,Temp_flag1).

file_spec13(Tokens,Para_file,Object,File_spec,
            Rest2,Flag,Temp_flag1) :-
   file1(Tokens,File,Rest1,Temp_flag2),
   file_spec15(Rest1,Rest2,Flag,1,Temp_flag2),
   concat(Object,'\',Temp),
   concat(Temp,File,File_spec),
   concat('object ',Object,Temp1),
   concat(Temp1,' file ',Temp2),
   concat(Temp2,File,Para_file).

file_spec14(Tokens,Para_file,File,File_spec,Rest3,Flag,
            Temp_flag1,Temp_flag2) :-
   ((noise_word(Tokens,Rest1),
     blank(Rest1,Rest2));
    (blank(Tokens,Rest2))),
   file_spec14(Rest2,Para_file,File,File_spec,Rest3,
               Flag,Temp_flag1,Temp_flag2).

file_spec14(Tokens,Para_file,File,File_spec,Rest2,Flag,
            Temp_flag1,Temp_flag2) :-
   drive1(Tokens,Drive,Rest1,_),
   file_spec15(Rest1,Rest2,Flag,1,Temp_flag2),
   concat(Drive,File,File_spec),
   concat('drive ',Drive,Temp1),
   concat(Temp1,' file ',Temp2),
   concat(Temp2,File,Para_file).
```

```
file_spec14(Tokens,Para_file,File,File_spec,Rest2,Flag,
            Temp_flag1,Temp_flag2) :-
   directory1(Tokens,Directory,Rest1,_),
   file_spec15(Rest1,Rest2,Flag,Temp_flag1,Temp_flag2),
   concat(Directory,'\',Temp),
   concat(Temp,File,File_spec),
   concat('directory ',Directory,Temp1),
   concat(Temp1,' file ',Temp2),
   concat(Temp2,File,Para_file).

file_spec15(Tokens,Rest2,1,0,1) :-
   blank(Tokens,Rest1),
   word(Rest1,'file',Rest2).

file_spec15(Tokens,Rest2,1,0,_) :-
   blank(Tokens,Rest1),
   word(Rest1,'file',Rest2).

file_spec15(Tokens,Tokens,1,_,1).

file_spec15(Tokens,Tokens,1,1,_).

file_spec15(Tokens,Tokens,0,_,_).
```

```
/*=============================================================*/
/*                        FILE NETWORK                         */
/*=============================================================*/

file1(Tokens,File,Rest3,1) :-
   word(Tokens,'file',Rest1),
   blank(Rest1,Rest2),
   file2(Rest2,File,Rest3).

file1(Tokens,File,Rest3,1) :-
   file_name1(Tokens,File,Rest1,_),
   blank(Rest1,Rest2),
   file3(Rest2,File,Rest3).

file1(Tokens,File,Rest,Flag) :-
   file_name1(Tokens,File,Rest,Flag).

file2(Tokens,File,Rest2) :-
   blank(Tokens,Rest1),
   file2(Rest1,File,Rest2).

file2(Tokens,File,Rest) :-
   file_name1(Tokens,File,Rest,_).

file3(Tokens,File,Rest2) :-
   blank(Tokens,Rest1),
   file3(Rest1,File,Rest2).

file3(Tokens,File,Rest) :-
   word(Tokens,'file',Rest).
```

```
/*============================================================*/
/*                    FILE NAME NETWORK                       */
/*============================================================*/

file_name1(Tokens,File,Rest2,1) :-
   valid_file_name(Tokens,Name_string,Rest1),
   file_name2(Rest1,Rest_name_string,Rest2,_),
   append2(Name_string,Rest_name_string,Temp1),
   name(File,Temp1).

file_name1(['.'|Tokens],File,Rest,1) :-
   file_name3(Tokens,Temp1,Rest,_),
   append3(46,Temp1,Temp2),
   name(File,Temp2).

file_name1(Tokens,File,Rest,0) :-
   valid_file_name(Tokens,Temp1,Rest),
   append2(Temp1,[46],Temp2),
   name(File,Temp2).

file_name1(Tokens,File,Rest2,1) :-
   valid_file_name(Tokens,Name_string,Rest1),
   file_name4(Rest1,Rest_name_string,Rest2),
   append2(Name_string,Rest_name_string,Temp),
   name(File,Temp).

file_name2(['.'|Tokens],Extension_string,Rest,_) :-
   file_name3(Tokens,Extension,Rest,_),
   append3(46,Extension,Extension_string).

file_name3(Tokens,Extension_string,Rest,1) :-
   valid_extension(Tokens,Extension_string,Rest).

file_name4(['.',' '|Tokens],[46],[' '|Tokens]).
```

```
/*=============================================================*/
/*                    DIRECTORY NETWORK                        */
/*=============================================================*/

directory1(Tokens,Directory,Rest3,1) :-
   word(Tokens,'directory',Rest1),
   blank(Rest1,Rest2),
   directory2(Rest2,Directory,Rest3).

directory1(Tokens,Directory,Rest3,1) :-
   directory_name1(Tokens,Directory,Rest1,_),
   blank(Rest1,Rest2),
   directory3(Rest2,Directory,Rest3).

directory1(Tokens,Directory,Rest,Flag) :-
   directory_name1(Tokens,Directory,Rest,Flag).

directory2(Tokens,Directory,Rest2) :-
   blank(Tokens,Rest1),
   directory2(Rest1,Directory,Rest2).

directory2(Tokens,Directory,Rest) :-
   directory_name1(Tokens,Directory,Rest,_).

directory3(Tokens,Directory,Rest2) :-
   blank(Tokens,Rest1),
   directory3(Rest1,Directory,Rest2).

directory3(Tokens,Directory,Rest) :-
   word(Tokens,'directory',Rest).
```

```
/*=========================================================*/
/*                  DIRECTORY NAME NETWORK                 */
/*=========================================================*/

directory_name1(['\'|Tokens],Directory,Rest,1) :-
   valid_directory_name(Tokens,Directory_name,Rest),
   append3(92,Directory_name,Temp),
   name(Directory,Temp).

directory_name1(['\'|Rest],'\',Rest,1).

directory_name1(['\',X|_],'\',Rest,_) :-
   ((X = ' '); (X = '.'); (X = '!'); (X = '?')),
   write('** ERROR: * '),
   write(X),
   write(' * IS AN INVALID DIRECTORY NAME. **'),nl,nl,
   !,fail.

directory_name1(Tokens,Directory,Rest,0) :-
   valid_directory_name(Tokens,Directory_name,Rest),
   append3(92,Directory_name,Temp),
   name(Directory,Temp).
```

```
/*============================================================*/
/*                      DRIVE NETWORK                         */
/*============================================================*/

drivel(Tokens,Drive,Rest3,1) :-
   word(Tokens,'drive',Rest1),
   blank(Rest1,Rest2),
   drive2(Rest2,Drive,Rest3).

drivel(Tokens,Drive,Rest3,1) :-
   drive_name(Tokens,Drive,Rest1,_),
   blank(Rest1,Rest2),
   drive3(Rest2,Drive,Rest3).

drivel(Tokens,Drive,Rest,Flag) :-
   drive_name(Tokens,Drive,Rest,Flag).

drive2(Tokens,Drive,Rest2) :-
   blank(Tokens,Rest1),
   drive2(Rest1,Drive,Rest2).

drive2(Tokens,Drive,Rest) :-
   drive_name(Tokens,Drive,Rest,_).

drive3(Tokens,Drive,Rest2) :-
   blank(Tokens,Rest1),
   drive3(Rest1,Drive,Rest2).

drive3(Tokens,Drive,Rest) :-
   word(Tokens,'drive',Rest).
```

```
/*=============================================================*/
/*                  DRIVE NAME NETWORK                         */
/*=============================================================*/

drive_name([D,':'|Rest],Drive,Rest,1) :-
   (D = 'a'; D = 'b'; D = 'c'),
   concat(D,':',Drive).

drive_name([D|Rest],Drive,Rest,0) :-
   (D = 'a'; D = 'b'; D = 'c'),
   concat(D,':',Drive).
```

```
/*=============================================================*/
/*                  VALID NAME CHARACTERS                      */
/*=============================================================*/

valid_name_characters([]).

valid_name_characters([Character|Rest_name]) :-
   (((Character >= 97, Character =< 122);     /* characters */
    (Character >= 48, Character =< 57);       /* numbers    */
    (Character = 33);                         /* symbols    */
    (Character >= 35, Character =< 42);       /*    "       */
    (Character = 45);                         /*    "       */
    (Character = 49);                         /*    "       */
    (Character = 64);                         /*    "       */
    (Character = 95);                         /*    "       */
    (Character = 96);                         /*    "       */
    (Character = 123);                        /*    "       */
    (Character = 125))),                      /*    "       */
    valid_name_characters(Rest_name).
```

```
/*=============================================================*/
/*                    VALID FILE NAME                          */
/*=============================================================*/

valid_file_name([Name|Rest],Name_string,Rest) :-
    name(Name,Name_string),
   (not(word([Name|Rest],'file',_)),
    not(word([Name|Rest],'directory',_)),
    not(word([Name|Rest],'drive',_))),
    valid_name_characters(Name_string).



/*=============================================================*/
/*                 VALID EXTENSION NAME                        */
/*=============================================================*/

valid_extension([Extension|Rest],[X,Y,Z],Rest) :-
   name(Extension,[X,Y,Z]),
   valid_name_characters([X]),
   valid_name_characters([Y]),
   valid_name_characters([Z]),!.

valid_extension([Extension|Rest],[X,Y],Rest) :-
   name(Extension,[X,Y]),
   valid_name_characters([X]),
   valid_name_characters([Y]),!.

valid_extension([Extension|Rest],[X],Rest) :-
   name(Extension,[X]),
   valid_name_characters([X]),!.

valid_extension([Extension|_],_,_) :-
   write('** ERROR: * '),
   write(Extension),
   write(' * IS A BAD EXTENSION **'),nl,nl,!.



/*=============================================================*/
/*                 VALID DIRECTORY NAME                        */
/*=============================================================*/

valid_directory_name([Name|Rest],Name_string,Rest) :-
   name(Name,Name_string),
   (not(word([Name|Rest],'file',_)),
    not(word([Name|Rest],'directory',_)),
    not(word([Name|Rest],'drive',_))),
   valid_name_characters(Name_string).
```

```
/* file: main_nets.ari */

/************************************************************/
/*==========================================================*/
/*
                          ANALYZE

     This clause begins the analysis process of the
sentence.  It accepts as input, a list of strings, which
represents the tokenized input from the user.  It returns
the MS-DOS command.
     This is also the start, or first state of the recursive
transition network.
                                                           */
/*==========================================================*/

analyzel(Tokens,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   analyzel(Rest2,Paraphrase,Command),!.

analyzel(Tokens,Paraphrase,Command) :-
   word(Tokens,'list',Rest1),
   blank(Rest1,Rest2),
   list1(Rest2,Paraphrase,Command),!.

analyzel(Tokens,Paraphrase,Command) :-
   word(Tokens,'change',Rest1),
   blank(Rest1,Rest2),
   changel(Rest2,Paraphrase, Command),!.

analyzel(Tokens,Paraphrase,Command) :-
   word(Tokens,'remove',Rest1),
   blank(Rest1,Rest2),
   removel(Rest2,Paraphrase,Command),!.

analyzel(Tokens,Paraphrase,Command) :-
   word(Tokens,'copy',Rest1),
   blank(Rest1,Rest2),
   copy1(Rest2,Paraphrase,Command),!.

analyzel(Tokens,Paraphrase,Command) :-
   word(Tokens,'create',Rest1),
   blank(Rest1,Rest2),
   createl(Rest2,Paraphrase,Command),!.

analyzel(Tokens,Paraphrase,Command) :-
   word(Tokens,'prepare',Rest1),
   blank(Rest1,Rest2),
   preparel(Rest2,Paraphrase,Command),!.

analyzel(Tokens,'prepare current disk for use','format') :-
   word(Tokens,'prepare',Rest),
   final(Rest),!.
```

```
analyze1(Tokens,Paraphrase,Command) :-
   word(Tokens,'print',Rest1),
   blank(Rest1,Rest2),
   print1(Rest2,Paraphrase,Command),!.

analyze1(Tokens,Paraphrase,Command) :-
   command(Tokens,'dir',Rest),
   analyze2(Rest,Paraphrase,Command),!.

analyze1(Tokens,'the MS-DOS command dir','dir') :-
   command(Tokens,'dir',Rest),
   final(Rest),!.

analyze1(Tokens,Paraphrase,Command) :-
   command(Tokens,'type',Rest1),
   blank(Rest1,Rest2),
   analyze3(Rest2,Paraphrase,Command),!.

analyze1(Tokens,'the MS-DOS command time','time') :-
   word(Tokens,'time',Rest),
   final(Rest),!.

analyze1(Tokens,'the MS-DOS command date','date') :-
   word(Tokens,'date',Rest),
   final(Rest),!.

analyze1(Tokens,Paraphrase,Command) :-
   command(Tokens,'cd',Rest1),
   blank(Rest1,Rest2),
   analyze4(Rest2,Paraphrase,Command),!.

analyze1(Tokens,Paraphrase,Command):-
   command(Tokens,'mkdir',Rest1),
   blank(Rest1,Rest2),
   analyze5(Rest2,Paraphrase,Command),!.

analyze1(Tokens,Paraphrase,Command):-
   command(Tokens,'rmdir ',Rest1),
   blank(Rest1,Rest2),
   analyze6(Rest2,Paraphrase,Command),!.

analyze1(Tokens,_,_) :-
   word(Tokens,'exit',Rest),
   final(Rest),
   halt.

analyze1(Tokens,_,_) :-
   command(Tokens,Command,_),
   write('** ERROR: ARGUMENTS TO '),
   write(Command),
   write(' ARE INVALID. **'),nl,nl,!.
```

```
analyze1([Token|_],_,_) :-
   not(word([Token],_,_)),
   write('** ERROR: UNIDENTIFIED WORD. **'),nl,
   write('* '),
   write(Token),
   write(' *  NOT FOUND IN DICTIONARY'),nl,nl,!.

analyze1(Tokens,_,_) :-
   not(command(Tokens,_,_)),
   word(Tokens,_,_),
   write('** ERROR: CAN NOT ANALYZE THIS REQUEST. **'),nl,nl,!.

analyze2(Tokens,Paraphrase,Command) :-
   blank(Tokens,Rest1),
   file_spec1(Rest1,Para_file,File,Rest2,_),
   analyze7(Rest2,Para_file,File,Paraphrase,Command).

analyze2(Tokens,Paraphrase,Command) :-
   blank(Tokens,Rest1),
   object_spec1(Rest1,Para_object,Object,Rest2,_),
   analyze7(Rest2,Para_object,Object,Paraphrase,Command).

analyze2(Tokens,Paraphrase,Command) :-
   blank(Tokens,Rest1),
   file_spec1(Rest1,Para_file,File,Rest2,_),
   final(Rest2),
   concat('dir ',File,Command),
   concat('list the directory contents of ',Para_file,
          Paraphrase).

analyze2(Tokens,Paraphrase,Command) :-
   blank(Tokens,Rest1),
   object_spec1(Rest1,Para_object,Object,Rest2,_),
   final(Rest2),
   concat('dir ',Object,Command),
   concat('list the directory contents of object ',Para_object,
          Paraphrase).

analyze2(Tokens,Paraphrase,Command) :-
   directory_spec(Tokens,Para_specs,Specs,Rest),
   final(Rest),
   concat('dir ',Specs,Command),
   concat('list the directory contents ',Para_specs,
          Paraphrase).

analyze2(Tokens,'list the directory contents','dir') :-
   final(Tokens).

analyze3(Tokens,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file,File,Rest,_),
   final(Rest),
   concat('type ',File,Command),
   concat('list the directory contents of ',Para_file,
          Paraphrase).
```

```
analyze4(Tokens,Paraphrase,Command) :-
   object_spec1(Tokens,Para_object,Object,Rest,_),
   final(Rest),
   concat('cd ',Object,Command),
   concat('the MS-DOS command cd and ',Para_object,Paraphrase)

analyze5(Tokens,Paraphrase,Command) :-
   object_spec1(Tokens,Para_object,Object,Rest,_),
   final(Rest),
   concat('mkdir ',Object,Command),
   concat('the MS-DOS command mkdir and ',Para_object,
          Paraphrase).

analyze6(Tokens,Paraphrase,Command) :-
   object_spec1(Tokens,Para_object,Object,Rest,_),
   final(Rest),
   concat('rmdir ',Object,Command),
   concat('the MS-DOS command rmdir and ',Para_object,
         Paraphrase).

analyze7(Tokens,Para_object,Object,Paraphrase,Command) :-
   final(Rest),
   concat('dir ',Object,Command),
   concat('list the directory contents of ',Para_object,
        ' Paraphrase).

analyze7(Tokens,Para_object,Object,Paraphrase,Command) :-
   directory_spec(Tokens,Specs,Rest),
   final(Rest),
   concat('dir ',Object,Temp1),
   concat(Temp1,' ',Temp2),
   concat(Temp2,Specs,Command),
   concat('list the directory contents of ',Para_object,
         Temp11),
   concat(Temp11,' WITH SPECIFICATIONS',Paraphrase).
```

```
/*=========================================================*/
/*                      LIST NETWORK                       */
/*=========================================================*/

list1(Tokens,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   list1(Rest2,Paraphrase,Command).

list1(Tokens,'list time','time') :-
   word(Tokens,'time',Rest),
   final(Rest).

list1(Tokens,'list date','date') :-
   word(Tokens,'date',Rest),
   final(Rest).

list1(Tokens,Paraphrase,Command) :-
   word(Tokens,'directory',Rest),
   list9(Rest,Paraphrase,Command).

list1(Tokens,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file,File,Rest,1),
   list7(Rest,Para_file,File,Paraphrase,Command).

list1(Tokens,Paraphrase,Command) :-
   object_spec1(Tokens,Para_object,Object,Rest,1),
   list6(Rest,Para_object,Object,Paraphrase,Command).

list1([Name|Tokens],Paraphrase, Command) :-
   file_spec1([Name|Tokens],_,Name1,Rest1,0),
   object_spec1([Name|Tokens],_,Name2,Rest2,0),
   concat('\',Name1,N1),
   concat(Name2,'.',N2),
   eq(N1,N2),
   eq(Rest1,Rest2),
   list4(Name,Rest1,Paraphrase,Command).

list4(Name,Tokens,Paraphrase,Command) :-
   write('Is '),
   write(Name),
   write(' a directory or a file? '),
   get0(Character),
   read_character_list(Character,Answer),
   list5(Answer,Name,Tokens,Paraphrase,Command).

list5(Answer,File,Tokens,Paraphrase,Command) :-
   word(Answer,'file',[]),
   concat('file ',File,Para_file),
   list7(Tokens,Para_file,File,Paraphrase,Command).
```

```
list5(Answer,Directory,Tokens,Paraphrase,Command) :-
   word(Answer,'directory',[]),
   concat('\',Directory,Object),
   concat('directory ',Directory,Para_object),
   list6(Tokens,Para_object,Object,Paraphrase,Command).

list5(_,Name,Tokens,Paraphrase,Command) :-
   write('** ERROR: NOT A VALID REPLY. **'),nl,nl,
   list4(Name,Tokens,Paraphrase,Command).

list6(Tokens,Para_object,Object,Paraphrase,Command) :-
   directory_spec(Tokens,Para_specs,Specs,Rest),
   final(Rest),
   concat('dir ',Object,Temp),
   concat(Temp,Specs,Command),
   concat('list object ',Para_object,Templ),
   concat(Templ,Para_specs,Paraphrase).

list6(Tokens,Para_object,Object,Paraphrase,Command) :-
   final(Tokens),
   concat('dir ',Object,Command),
   concat('list object ',Para_object,Paraphrase).

list7(Tokens,Para_file,File,Paraphrase,Command) :-
   write('    Do you wish to: '),nl,nl,
   write('1. view the file contents'),nl,
   write('2. see if file exist'),nl,nl,
   write('choice: '),
   get0(Answer),nl,
   list8(Answer,Tokens,Para_file,File,Paraphrase,Command).

list8(49,Tokens,Para_file,File,Paraphrase,Command) :-
   final(Tokens),
   concat('type ',File,Command),
   concat('list ',Para_file,Paraphrase).

list8(50,Tokens,Para_file,File,Paraphrase,Command) :-
   list6(Tokens,Para_file,File,Paraphrase,Command).

list8(_,Tokens,Para_file,File,Paraphrase,Command) :-
   write('** ERROR: NOT A VALID REPLY. **'),nl,nl,
   list7(Tokens,Para_file,File,Paraphrase,Command).

list9(Tokens,Paraphrase,Command) :-
   directory_spec(Tokens,Para_specs,Specs,Rest),
   final(Rest),
   concat('dir ',Specs,Command),
   concat('list directory contents',Para_specs,Paraphrase)
```

```
/*=========================================================*/
/*                    CHANGE NETWORK                       */
/*=========================================================*/

change1(Tokens,Paraphrase,Command) :-
   word(Tokens,'to',Rest1),
   blank(Rest1,Rest2),
   change3(Rest2,Paraphrase,Command).

change1(Tokens,Paraphrase,Command) :-
   word(Tokens,'from',Rest1),
   blank(Rest1,Rest2),
   change8(Rest2,Paraphrase,Command).

change1(Tokens,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   change1(Rest2,Paraphrase,Command).

change1(Tokens,Paraphrase,Command) :-
   word(Tokens,'directory',Rest1),
   blank(Rest1,Rest2),
   word(Rest2,'to',Rest3),
   blank(Rest3,Rest4),
   change2(Rest4,Paraphrase,Command).

change1(Tokens,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file,File,Rest1,_),
   blank(Rest1,Rest2),
   change5(Rest2,Para_file,File,Paraphrase,Command).

change1(Tokens,Paraphrase,Command) :-
   object_spec1(Tokens,Para_object,Object,Rest,_),
   final(Rest),
   concat('cd ',Object,Command),
   concat('change to ',Para_object,Paraphrase).

change2(Tokens,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   change2(Rest2,Paraphrase,Command).

change2(Tokens,Paraphrase,Command) :-
   object_spec1(Tokens,Para_object,Object,Rest,_),
   final(Rest),
   concat('cd ',Object,Command),
   concat('change to ',Para_object,Paraphrase).

change3(Tokens,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   change3(Rest2,File,Paraphrase,Command).
```

```
change3(Tokens,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file,File,Rest1,_),
   blank(Rest1,Rest2),
   change4(Rest2,Para_file,File,Paraphrase,Command).

change4(Tokens,Para_file,File,Paraphrase,Command) :-
   word(Tokens,'from',Rest1),
   blank(Rest1,Rest2),
   change4A(Rest2,Para_file,File,Paraphrase,Command).

change4(Tokens,Para_file,File,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   change4(Rest2,Para_file,File,Paraphrase,Command).

change4(Tokens,Para_file1,File1,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file2,File2,Rest,_),
   final(Rest),
   concat('rename ',File2,X),
   concat(X,' ',Y),
   concat(Y,File1,Command),
   concat('change the name of ',File2,Temp1),
   concat(Temp1,' to ',Temp2),
   concat(Temp2,File1,Paraphrase).

change4A(Tokens,Para_file1,File1,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   change4A(Rest2,Para_file1,File1,Paraphrase,Command).

change4A(Tokens,Para_file1,File1,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file2,File2,Rest,_),
   final(Rest),
   concat('rename ',File2,X),
   concat(X,' ',Y),
   concat(Y,File1,Command),
   concat('change the name of ',File2,Temp1),
   concat(Temp1,' to ',Temp2),
   concat(Temp2,File1,Paraphrase).

change5(Tokens,Para_file1,File1,Paraphrase,Command) :-
   word(Tokens,'from',Rest1),
   blank(Rest1,Rest2),
   change6(Rest2,Para_file1,File1,Paraphrase,Command).

change5(Tokens,Para_file1,File1,Paraphrase,Command) :-
   word(Tokens,'to',Rest1),
   blank(Rest1,Rest2),
   change7(Rest2,Para_file1,File1,Paraphrase,Command).

change5(Tokens,Para_file,File,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   change5(Rest2,Para_file,File,Paraphrase,Command).
```

```
change5(Tokens,Para_file1,File1,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file2,File2,Rest,_),
   final(Rest),
   concat('rename ',File1,X),
   concat(X,' ',Y),
   concat(Y,File2,Command),
   concat('change the name of ',Para_file1,Temp1),
   concat(Temp1,' to ',Temp2),
   concat(Temp2,Para_file2,Paraphrase).

change6(Tokens,Para_file,File,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   change6(Rest2,Para_file,File,Paraphrase,Command).

change6(Tokens,Para_file1,File1,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file2,File2,Rest,_),
   final(Rest),
   concat('rename ',File2,X),
   concat(X,' ',Y),
   concat(Y,File1,Command),
   concat('change the name of ',Para_file2,Temp1),
   concat(Temp1,' to ',Temp2),
   concat(Temp2,Para_file1,Paraphrase).

change7(Tokens,Para_file,File,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   change7(Rest2,Para_file,File,Paraphrase,Command).

change7(Tokens,Para_file1,File1,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file2,File2,Rest,_),
   final(Rest),
   concat('rename ',File1,X),
   concat(X,' ',Y),
   concat(Y,File2,Command),
   concat('change the name of ',Para_file1,Temp1),
   concat(Temp1,' to ',Temp2),
   concat(Temp2,Para_file2,Paraphrase).

change8(Tokens,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   change8(Rest2,Para_file,File,Paraphrase,Command).

change8(Tokens,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file,File,Rest1,_),
   blank(Rest1,Rest2),
   change9(Rest2,Para_file,File,Paraphrase,Command).

change9(Tokens,Para_file1,File1,Paraphrase,Command) :-
   word(Tokens,'to',Rest1),
   blank(Rest1,Rest2),
   change10(Rest2,Para_file1,File1,Paraphrase,Command).
```

```
change9(Tokens,Para_file,File,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   change9(Rest2,Para_file,File,Paraphrase,Command).

change9(Tokens,Para_file1,File1,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file2,File2,Rest,_),
   final(Rest),
   concat('rename ',File1,X),
   concat(X,' ',Y),
   concat(Y,File2,Command),
   concat('change the name of ',Para_file1,Temp1),
   concat(Temp1,' to ',Temp2),
   concat(Temp2,Para_file2,Paraphrase).

change10(Tokens,Para_file1,File1,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   change4A(Rest2,Para_file,File1,Paraphrase,Command).

change10(Tokens,Para_file1,File1,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file2,File2,Rest,_),
   final(Rest),
   concat('rename ',File1,X),
   concat(X,' ',Y),
   concat(Y,File2,Command),
   concat('change the name of ',Para_file1,Temp1),
   concat(Temp1,' to ',Temp2),
   concat(Temp2,Para_file2,Paraphrase).
```

```
/*=========================================================*/
/*                    REMOVE NETWORK                       */
/*=========================================================*/

remove1(Tokens,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   remove1(Rest2,Paraphrase,Command).

remove1(Tokens,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file,File,Rest,1),
   final(Rest),
   concat('erase ',File,Command),
   concat('remove ',Para_file,Paraphrase).

remove1(Tokens,Paraphrase,Command) :-
   object_spec1(Tokens,Para_object,Directory,Rest,1),
   final(Rest),
   concat('rmdir ',Directory,Command),
   concat('remove ',Para_object,Paraphrase).

remove1([Name|Tokens],Paraphrase,Command) :-
   file_spec1([Name|Tokens],_,Name1,Rest1,0),
   object_spec1([Name|Tokens],_,Name2,Rest2,0),
   concat('\',Name1,N1),
   concat(Name2,'.',N2),
   eq(N1,N2),
   eq(Rest1,Rest2),
   remove4(Name,Rest1,Paraphrase,Command).

remove4(Name,Tokens,Paraphrase,Command) :-
   write('Is '),
   write(Name),
   write(' a directory or a file? '),
   get0(Character),
   read_character_list(Character,Answer),
   remove5(Answer,Name,Tokens,Paraphrase,Command).

remove5(Answer,Name,Tokens,Paraphrase,Command) :-
   word(Answer,'directory',[]),
   final(Rest),
   concat('\',Name,Directory),
   concat('rmdir ',Directory,Command),
   concat('remove directory ',Directory,Paraphrase).

remove5(Answer,File,Tokens,Paraphrase,Command) :-
   word(Answer,'file',[]),
   final(Rest),
   concat('erase ',File,Command),
   concat('remove file ',File,Paraphrase).

remove5(_,Name,Tokens,Paraphrase,Command) :-
   write('** ERROR: NOT A VALID REPLY. **'),nl,nl,
   remove4(Name,Tokens,Paraphrase,Command).
```

```
/*=========================================================*/
/*                      COPY NETWORK                       */
/*=========================================================*/

copy1(Tokens,Paraphrase,Command) :-
   word(Tokens,'to',Rest1),
   blank(Rest1,Rest2),
   copy2(Rest2,Paraphrase,Command).

copy1(Tokens,Paraphrase,Command) :-
   word(Tokens,'from',Rest1),
   blank(Rest1,Rest2),
   copy6(Rest2,Paraphrase,Command).

copy1(Tokens,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   copy1(Rest2,Paraphrase,Command).

copy1(Tokens,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file,File,Rest1,_),
   blank(Rest1,Rest2),
   copy4(Rest2,Para_file,File,Paraphrase,Command).

copy1(Tokens,Paraphrase,Command) :-
   object_spec1(Tokens,Para_object,Object,Rest1,_),
   blank(Rest1,Rest2),
   copy4(Rest2,Para_object,Object,Paraphrase,Command).

copy2(Tokens,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   copy2(Rest2,Paraphrase,Command).

copy2(Tokens,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file,File,Rest1,_),
   blank(Rest1,Rest2),
   copy3(Rest2,Para_file,File,Paraphrase,Command).

copy2(Tokens,Paraphrase,Command) :-
   object_spec1(Tokens,Para_object,Object,Rest1,_),
   blank(Rest1,Rest2),
   copy3(Rest2,Para_object,Object,Paraphrase,Command).

copy3(Tokens,Para_file,File,Paraphrase,Command) :-
   word(Tokens,'from',Rest1),
   blank(Rest1,Rest2),
   copy3A(Rest2,Para_file,File,Paraphrase,Command).

copy3(Tokens,Para_file,File,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   copy3(Rest2,Para_file,File,Paraphrase,Command).
```

```
copy3(Tokens,Para_file1,File1,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file2,File2,Rest1,_),
   final(Rest1),
   concat('copy ',File2,Temp1),
   concat(Temp1,' ',Temp2),
   concat(Temp2,File1,Command),
   concat('copy from ',Para_file2,Temp11),
   concat(Temp11,' to ',Temp12),
   concat(Temp12,Para_file1,Paraphrase).

copy3(Tokens,Para_file1,File1,Paraphrase,Command) :-
   object_spec1(Tokens,Para_object2,Object2,Rest1,_),
   final(Rest1),
   concat('copy ',Object2,Temp1),
   concat(Temp1,' ',Temp2),
   concat(Temp2,File1,Command),
   concat('copy from ',Para_object1,Temp11),
   concat(Temp11,' to ',Temp12),
   concat(Temp12,Para_file2,Paraphrase).

copy3A(Tokens,Para_file,File,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   copy3(Rest2,Para_file,File,Paraphrase,Command).

copy3A(Tokens,Para_file1,File1,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file2,File2,Rest1,_),
   final(Rest1),
   concat('copy ',File2,Temp1),
   concat(Temp1,' ',Temp2),
   concat(Temp2,File1,Command),
   concat('copy from ',Para_file2,Temp11),
   concat(Temp11,' to ',Temp12),
   concat(Temp12,Para_file1,Paraphrase).

copy3A(Tokens,Para_file1,File1,Paraphrase,Command) :-
   object_spec1(Tokens,Para_object2,Object2,Rest1,_),
   final(Rest1),
   concat('copy ',Object2,Temp1),
   concat(Temp1,' ',Temp2),
   concat(Temp2,File1,Command),
   concat('copy from ',Para_object2,Temp11),
   concat(Temp11,' to ',Temp12),
   concat(Temp12,Para_file1,Paraphrase).

copy4(Tokens,Para_file,File,Paraphrase,Command) :-
   word(Tokens,'from',Rest1),
   blank(Rest1,Rest2),
   copy5(Rest2,Para_file,File,Paraphrase,Command).

copy4(Tokens,Para_file,File,Paraphrase,Command) :-
   word(Tokens,'to',Rest1),
   blank(Rest1,Rest2),
   copy5A(Rest2,Para_file,File,Paraphrase,Command).
```

```
copy4(Tokens,Para_file,File,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   copy4(Rest2,Para_file,File,Paraphrase,Command).

copy4(Tokens,Para_file1,File1,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file2,File2,Rest1,_),
   final(Rest1),
   concat('copy ',File1,Temp1),
   concat(Temp1,' ',Temp2),
   concat(Temp2,File2,Command),
   concat('copy from ',Para_file1,Temp11),
   concat(Temp11,' to ',Temp12),
   concat(Temp12,Para_file2,Paraphrase).

copy4(Tokens,Para_file1,File1,Paraphrase,Command) :-
   object_spec1(Tokens,Para_object2,Object2,Rest1,_),
   final(Rest1),
   concat('copy ',File1,Temp1),
   concat(Temp1,' ',Temp2),
   concat(Temp2,Object2,Command),
   concat('copy from ',Para_file1,Temp11),
   concat(Temp11,' to ',Temp12),
   concat(Temp12,Para_object2,Paraphrase).

copy5(Tokens,Para_file,File,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   copy5(Rest2,Para_file,File,Paraphrase,Command).

copy5(Tokens,Para_file1,File1,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file2,File2,Rest1,_),
   final(Rest1),
   concat('copy ',File2,Temp1),
   concat(Temp1,' ',Temp2),
   concat(Temp2,File1,Command),
   concat('copy from ',Para_file2,Temp11),
   concat(Temp11,' to ',Temp12),
   concat(Temp12,Para_file1,Paraphrase).

copy5(Tokens,Para_file1,File1,Paraphrase,Command) :-
   object_spec1(Tokens,Para_object2,Object2,Rest1,_),
   final(Rest1),
   concat('copy ',Object2,Temp1),
   concat(Temp1,' ',Temp2),
   concat(Temp2,File1,Command),
   concat('copy from ',Para_object1,Temp11),
   concat(Temp11,' to ',Temp12),
   concat(Temp12,Para_file1,Paraphrase).

copy5A(Tokens,Para_file,File,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   copy5A(Rest2,Para_file,File,Paraphrase,Command).
```

```
copy5A(Tokens,Para_file1,File1,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file2,File2,Rest1,_),
   final(Rest1),
   concat('copy ',File1,Temp1),
   concat(Temp1,' ',Temp2),
   concat(Temp2,File2,Command),
   concat('copy from ',Para_file1,Temp11),
   concat(Temp11,' to ',Temp12),
   concat(Temp12,Para_file2,Paraphrase).

copy5A(Tokens,Para_file1,File1,Paraphrase,Command) :-
   object_spec1(Tokens,Para_object2,Object2,Rest1,_),
   final(Rest1),
   concat('copy ',File1,Temp1),
   concat(Temp1,' ',Temp2),
   concat(Temp2,Object2,Command),
   concat('copy from ',Para_file1,Temp11),
   concat(Temp11,' to ',Temp12),
   concat(Temp12,Para_object2,Paraphrase).

copy6(Tokens,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   copy6(Rest2,Paraphrase,Command).

copy6(Tokens,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file,File,Rest1,_),
   blank(Rest1,Rest2),
   copy7(Rest2,Para_file,File,Paraphrase,Command).

copy6(Tokens,Paraphrase,Command) :-
   object_spec1(Tokens,Para_object,Object,Rest1,_),
   blank(Rest1,Rest2),
   copy7(Rest2,Para_object,Object,Paraphrase,Command).

copy7(Tokens,Para_file,File,Paraphrase,Command) :-
   word(Tokens,'to',Rest1),
   blank(Rest1,Rest2),
   copy8(Rest2,Para_file,File,Paraphrase,Command).

copy7(Tokens,Para_file,File,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   copy7(Rest2,Para_file,File,Paraphrase,Command).

copy7(Tokens,Para_file1,File1,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file2,File2,Rest1,_),
   final(Rest1),
   concat('copy ',File1,Temp1),
   concat(Temp1,' ',Temp2),
   concat(Temp2,File2,Command),
   concat('copy from ',Para_file1,Temp11),
   concat(Temp11,' to ',Temp12),
   concat(Temp12,Para_file2,Paraphrase).
```

```
copy7(Tokens,Para_file1,File1,Paraphrase,Command) :-
   object_spec1(Tokens,Para_object2,Object2,Rest1,_),
   final(Rest1),
   concat('copy ',File1,Temp1),
   concat(Temp1,' ',Temp2),
   concat(Temp2,Object2,Command),
   concat('copy from ',Para_file1,Temp11),
   concat(Temp11,' to ',Temp12),
   concat(Temp12,Para_object2,Paraphrase).

copy8(Tokens,Para_file,File,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   copy8(Rest2,Para_file,File,Paraphrase,Command).

copy8(Tokens,Para_file1,File1,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file2,File2,Rest1,_),
   final(Rest1),
   concat('copy ',File1,Temp1),
   concat(Temp1,' ',Temp2),
   concat(Temp2,File2,Command),
   concat('copy from ',Para_file1,Temp11),
   concat(Temp11,' to ',Temp12),
   concat(Temp12,Para_file2,Paraphrase).

copy8(Tokens,Para_file1,File1,Paraphrase,Command) :-
   object_spec1(Tokens,Para_object2,Object2,Rest1,_),
   final(Rest1),
   concat('copy ',File1,Temp1),
   concat(Temp1,' ',Temp2),
   concat(Temp2,Object2,Command),
   concat('copy from ',Para_file1,Temp11),
   concat(Temp11,' to ',Temp12),
   concat(Temp12,Para_object2,Paraphrase).
```

```
/*=============================================================*/
/*                      CREATE NETWORK                         */
/*=============================================================*/

createl(Tokens,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   createl(Rest2,Paraphrase,Command).

createl(Tokens,Paraphrase,Command) :-
   object_spec1(Tokens,Para_object,Object,Rest,_),
   final(Rest),
   concat('mkdir ',Object,Command),
   concat('create ',Para_object,Paraphrase).
```

```
/*============================================================*/
/*                    PREPARE NETWORK                         */
/*============================================================*/

preparel(Tokens,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   preparel(Rest2,Paraphrase,Command).

preparel(Tokens,Paraphrase,Command) :-
   format_spec([' '|Tokens],Para_specs,Specs,Rest),
   final(Rest),
   concat('format ',Specs,Command),
   concat('prepare for use',Para_specs,Paraphrase).
```

```
/*=========================================================*/
/*                    PRINT NETWORK                        */
/*=========================================================*/

print1(Tokens,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   print1(Rest2,Paraphrase,Command).

print1(Tokens,Paraphrase,Command) :-
   print_spec([' '|Tokens],Para_specs,Specs,Rest),
   final(Rest),
   concat('print ',Specs,Command),
   concat('print',Para_specs,Paraphrase).

print1(Tokens,Paraphrase,Command) :-
   file_spec1(Tokens,Para_file,File,Rest,_),
   print2(Rest,Para_file,File,Paraphrase,Command).

print2(Tokens,Para_file,File,Paraphrase,Command) :-
   noise_word(Tokens,Rest1),
   blank(Rest1,Rest2),
   print2(Rest2,Para_file,File,Paraphrase,Command).

print2(Tokens,Para_file,File,Paraphrase,Command) :-
   print_spec(Tokens,Para_specs,Specs,Rest),
   final(Rest),
   concat('print ',File,Temp1),
   concat(Temp1,' ',Temp2),
   concat(Temp2,Specs,Command),
   concat('print ',Para_file,Temp11),
   concat(Temp11,Para_specs,Paraphrase).
```

```
/*============================================================*/
/*              DIRECTORY SPECIFICATIONS                      */
/*============================================================*/

directory_spec(Tokens,Para_specs,Specs,Rest3) :-
   blank(Tokens,Rest1),
   noise_word(Rest1,Rest2),
   directory_spec(Rest2,Para_specs,Specs,Rest3).

directory_spec(Tokens,Para_specs,Specs,Rest3) :-
   blank(Tokens,Rest1),
   dirspec(Rest1,Para_spec1,Spec1,Rest2),
   directory_spec(Rest2,Para_spec2,Spec2,Rest3),
   concat(Spec1,Spec2,Specs),
   concat(Para_spec1,Para_spec2,Para_specs).

directory_spec(Tokens,Para_specs,Spec,Rest2) :-
   blank(Tokens,Rest1),
   dirspec(Rest1,Para_specs,Spec,Rest2).

directory_spec(Tokens,'','',Tokens).
```

```
/*=========================================================*/
/*                  PRINT SPECIFICATIONS                   */
/*=========================================================*/

print_spec(Tokens,Para_specs,Specs,Rest3) :-
   blank(Tokens,Rest1),
   noise_word(Rest1,Rest2),
   print_spec(Rest2,Para_specs,Specs,Rest3).

print_spec(Tokens,Para_specs,Specs,Rest3) :-
   blank(Tokens,Rest1),
   printspec(Rest1,Para_spec1,Spec1,Rest2),
   print_spec(Rest2,Para_spec2,Spec2,Rest3),
   concat(Spec1,Spec2,Specs),
   concat(Para_spec1,Para_spec2,Para_specs).

print_spec(Tokens,Para_spec,Spec,Rest2) :-
   blank(Tokens,Rest1),
   printspec(Rest1,Para_spec,Spec,Rest2).

print_spec(Tokens,'','',Tokens).
```

```
/*============================================================*/
/*                  FORMAT SPECIFICATIONS                     */
/*============================================================*/

format_spec(Tokens,Para_specs,Specs,Rest3) :-
   blank(Tokens,Rest1),
   noise_word(Rest1,Rest2),
   format_spec(Rest2,Para_specs,Specs,Rest3).

format_spec(Tokens,Para_specs,Specs,Rest3) :-
   blank(Tokens,Rest1),
   drive1(Rest1,Drive,Rest2,_),
   format_spec(Rest2,Rest_para_specs,Rest_specs,Rest3),
   concat(' ',Drive,Temp1),
   concat(Temp1,' ',Temp2),
   concat(Temp2,Rest_specs,Specs),
   concat(' drive ',Drive,Temp),
   concat(Temp,Rest_para_specs,Para_specs).

format_spec(Tokens,Para_specs,Specs,Rest3) :-
   blank(Tokens,Rest1),
   formatspec(Rest1,Para_spec1,Spec1,Rest2),
   format_spec(Rest2,Para_spec2,Spec2,Rest3),
   concat(Spec1,Spec2,Specs),
   concat(Para_spec1,Para_spec2,Para_specs).

format_spec(Tokens,'','',Tokens).

/*============================================================*/
```

```
/* file: input.ari */

/*************************************************************/
/*===========================================================*/
/*
                    INPUT PREDICATES

      The purpose of these predicates are to query the user
for a command and then return the inputted request to the
calling predicate.  Each ASCII character entered at the
keyboard is analyzed.  Acceptable characters are upper or
lower case letters, underscore, end of sentence punctuation,
backspace and carriage return and other miscellaneous
characters.
     The following denotes the basic flow of these predicates
(see predicate read_character list).

step 1: The input is read into a list of ASCII characters.

step 2: The funtion of the backspace character is perform upon
        the inputted list.

step 3: The list (of characters) is grouped into words.
                                                            */
/*===========================================================*/

read_character_list(Character,Word_list) :-
   word_chars(Character,List),!,nl,
   elim_backspaces(List,[],Character_list),!,
   word_break(Character_list,[],Word_list),!.
```

```
/*=========================================================*/
/* word_chars: step 1 of read_character_list               */
/*=========================================================*/

/* finish when carriage return is found:                   */

word_chars(13,[]).

/* add a good character onto the list:                     */

word_chars(Character,[Character1|Character_list]) :-
   character(Character,Character1),
   !,
   get0(Character2),
   word_chars(Character2,Character_list).

/* message if an invalid character is typed:               */

word_chars(Character,[]) :-
   not(character(Character,Character)),
   not(end_of_words_char(Character)),
   name(Bad_character,[Character]),
   nl,
   write('ERROR: - '),
   write(Bad_character),
   write(' - IS NOT A VALID CHARACTER!'),fail.
```

```
/*=========================================================*/
/* characters: valid input characters                      */
/*=========================================================*/

character(Character,Character) :-     /* lower-case letter */
   Character >= 97,
   Character =< 122.

character(Character,Character1) :-    /* upper-case letter */
   Character >= 65,
   Character =< 90,
   Character1 is Character + 32.        /* make lower-case */

character(Character,Character) :-              /* numbers */
   Character >= 48,
   Character =< 57.

character(32,32).                               /* blank */

character(8,8).                               /* backspace */

character(95,95).                            /* underscore */

character(27,27).                               /* escape */

character(13,13).                         /* carriage return */

character(46,46).                                    /* . */

character(63,46).                                    /* ? */

character(33,33).                                    /* ! */

character(92,92).                                    /* \ */

character(47,47).                                    /* / */

character(58,58).                                    /* : */

character(42,42).                                    /* * */

character(64,64).                                    /* @ */

character(35,35).                                    /* # */

character(36,36).                                    /* $ */

character(37,37).                                    /* % */

character(38,38).                                    /* & */

character(40,40).                                    /* ( */

character(41,41).                                    /* ) */
```

```
character(45,45).                                    /* - */
character(95,95).                                    /* _ */
character(123,123).                                  /* { */
character(125,125).                                  /* } */
character(39,39).                                    /* ' */
character(96,96).                                    /* ` */
character(59,59).                                    /* ; */
character(44,44).                                    /* , */
```

```
/*=========================================================*/
/* elim_backspaces: step 2 of read_character_list          */
/*=========================================================*/

elim_backspaces([],ListA,ListA).

/* legal backspaces:                                       */

elim_backspaces([8|Rest_list],ListA,Character_list) :-
   elim_last(ListA,ListB),
   elim_backspaces(Rest_list,ListB,Character_list).

elim_backspaces([Head|Rest_list],ListA,Character_list) :-
   append(Head,ListA,ListB),
   elim_backspaces(Rest_list,ListB,Character_list).

/* if backspace is 1st character:                          */

elim_backspaces([8|Rest_list],ListA,Character_list) :-
   elim_backspaces(Rest_list,ListA,Character_list).

/*=========================================================*/
/* elim_last: performs elimination of last backspace
              character                                    */
/*=========================================================*/

elim_last([],[]).              /* try to backspace nothing */

elim_last([S],[]).             /* eliminate last character */

elim_last([Head|Tail],[Head|Rest]) :-
   elim_last(Tail,Rest).
```

```
/*=========================================================*/
/* word_break: step 3 of read_character_list               */
/*=========================================================*/

word_break([],[],[]).

/* will take care of possible blank at end of sentence:    */

word_break([32],[],[]).

/* last word:                                              */

word_break([],Char_list,[Word]) :-
   name(Word,Char_list).

/* handle multiple special characters in sussession:       */

word_break([Character|Rest],[],
      [Special_character|Word_list]) :-
   special_character(Character,Character),
   name(Special_character,[Character]),
   word_break(Rest,[],Word_list).

/* handle special characters:                              */

word_break([Character|Rest],Character_list,
           [Word,Special_character|Word_list]) :-
   special_character(Character,Character),
   name(Word,Character_list),
   name(Special_character,[Character]),
   word_break(Rest,[],Word_list).

word_break([32|Rest],[],[' '|Word_list]) :-
   word_break(Rest,[],Word_list).

/* blank character designates end of word:                 */

word_break([32|Rest],Character_list,[Word,' '|Word_list]) :-
   name(Word,Character_list),
   word_break(Rest,[],Word_list).

/* add character to Character_list for word:               */

word_break([Character|Rest],Character_list,Word_list) :-
   not(Character = 32),
   append(Character,Character_list,New_char_list),
   word_break(Rest,New_char_list,Word_list).
```

```
/*============================================================*/
/* special_characters: punctuationand MS-DOS symbolism     */
/*============================================================*/


special_character(46,46).                                  /* . */

special_character(92,92).                                  /* \ */

special_character(47,47).                                  /* / */

special_character(58,58).                                  /* : */

special_character(63,63).                                  /* ? */

special_character(33,33).                                  /* ! */

special_character(59,59).                                  /* ; */

special_character(44,44).                                  /* , */




/*============================================================*/
/* read_command: called from the main program clause "pe"  */
/*============================================================*/

read_command(Sentence_list) :-
   write('Enter Command: '),
   get0(Character),
   read_character_list(Character,Sentence_list).

/*============================================================*/
/************************************************************/
```

```
/* file: utilities.ari */

/*************************************************************/
/*===========================================================*/
/*            MISCELLANEOUS UTILITIES                        */
/*===========================================================*/

append(X,[],[X]).

append(X,[Head|Tail],[Head|Rest]) :-
   append(X,Tail,Rest).

/*===========================================================*/

append2([X],Y,[X|Y]).

append2([X|Rest],Y,[X|Temp]) :-
   append2(Rest,Y,Temp).

/*===========================================================*/

append3(X,Y,[X|Y]).

/*===========================================================*/
/*************************************************************/
```

APPENDIX C: SCRIPT OF PE_DOS

```
          PE_DOS
     (Plain English DOS)

Use the system:
 (1) to perform commands
 (2) as a tutorial
Enter Choice: 2



Enter command: please show me the files.

English paraphrase: list directory contents
MS-DOS command: dir



Enter command: display the contents of file temp.

   Do you wish to:
1. view the file contents
2. see if file exist
choice: 1

English paraphrase: list file temp.
The MS-DOS command to use is: type temp.

(note: in the request to display file contents, contents is
       eliminated as a noise word.)



Enter command: view time.

English paraphrase: list time
The MS-DOS command to use is: time



Enter command: indicate the date.

English paraphrase: list date
The MS-DOS command to use is: date
```

```
Enter command: modify file x to y.

English paraphrase: change the name of file x. to file y.
The MS-DOS command to use is: rename x. y.



Enter command: rename x y

English paraphrase: change the name of file x. to file y.
The MS-DOS command to use is: rename x. y.



Enter command: change the directory to prolog.

English paraphrase: change to directory prolog
The MS-DOS command to use is: cd prolog



Enter command: zap file x

English paraphrase: remove file x.
The MS-DOS command to use is: erase x.



Enter command: scratch out directory temporary.

English paraphrase: remove directory temporary
The MS-DOS command to use is: rmdir temporary
```

```
Enter command: kill pfs.

Is pfs a directory or file? file

English paraphrase: remove file pfs.
The MS-DOS command to use is: erase pfs.


Enter command: duplicate file x to y.

English paraphrase: copy from file x. to file y.
The MS-DOS command to use is: copy x. y.


Enter command: copy x y.

English paraphrase: copy from file x. to file y.
The MS-DOS command to use is: copy x. y.


Enter command: double x to y.

English paraphrase: copy from file x. to file y.
The MS-DOS command to use is: copy x. y.


Enter command: build answers.

English paraphrase: create directory answers
The MS-DOS command to use is: mkdir answers
```

```
Enter command: create the directory answers.

English paraphrase: create directory answers
The MS-DOS command to use is: mkdir answers



Enter Command: print the file anyfile.txt

English paraphrase: print file anyfile.txt
MS-DOS command: print anyfile.txt



Enter Command: print anything

English paraphrase: print file anyfile.
MS-DOS command: print anyfile.



Enter Command: display directory contents in wide format

English paraphrase: list directory contents in wide format
MS-DOS command: dir /w



Enter Command: ready for use

English paraphrase: prepare for use
MS-DOS command: format
```

```
Enter Command: prepare the a drive for use

English paraphrase: prepare for use drive a:
MS-DOS command: format a:



Enter Command: copy from file anyfile.txt to otherfile.txt

English paraphrase: copy from file anyfile.txt to file
                    otherfile.txt
MS-DOS command: copy anyfile.txt otherfile.txt



Enter Command: copy to a.ari the file b.ari

English paraphrase: copy from file b.ari to file a.ari
MS-DOS command: copy b.ari a.ari



Enter Command: copy from a.ari the file b.ari

English paraphrase: copy from file b.ari to file a.ari
MS-DOS command: copy a.ari b.ari



Enter Command: copy a.ari b.ari

English paraphrase: copy from file a.ari to file b.ari
MS-DOS command: copy a.ari b.ari
```

```
Enter Command: zap anything

Is anything a file or directory? file

English paraphrase: remove file anything.
MS-DOS command: erase anything.



Enter Command: remove the file anyfile.txt

English paraphrase: remove file anyfile.txt
MS-DOS command: erase anyfile.txt



Enter Command: zap anyfile.txt

English paraphrase: remove file anyfile.txt
MS-DOS command: erase anyfile.txt



Enter Command: can you remove the directory stuff

** ERROR: UNIDENTIFIED WORD. **
* can * NOT FOUND IN DICTIONARY.



Enter Command: remove \stuff

** ERROR: ARGUMENTS TO remove ARE INVALID. **
```

```
Enter Command: remove /stuff

English paraphrase: remove
MS-DOS command: rmdir stuff



Enter Command: list stuff.junk

* ERROR: * junk * IS A BAD EXTENSION. **



Enter Command: rename the file x to the file y

English paraphrase: change the name of file x. to y.
MS-DOS command: rename x. y.



Enter Command: change to stuff

English paraphrase: change to directory stuff
MS-DOS command: cd stuff



Enter Command: rename y.txt to b.txt

English paraphrase: change the name of file y.txt to file b.txt
MS-DOS command: rename y.txt b.txt



Enter Command: rename y.txt from b.txt

English paraphrase: change the name of file b.txt to file y.txt
MS-DOS command: rename b.txt y.txt
```

```
Enter Command: rename y.txt b.txt

English paraphrase: change the name of file y.txt to file b.txt
MS-DOS command: rename y.txt b.txt



Enter Command: change from y.txt the file b.txt

English paraphrase: change the name of file y.txt to file b.txt
MS-DOS command: rename y.txt b.txt



Enter Command: show the file anyfile.txt

   Do you wish to:
1. view the file contents
2. see if file exist
choice: 1

English paraphrase: list file temp.
MS-DOS command: type temp.



Enter Command: show the directory in wide format

English paraphrase: list directory contents in wide format
MS-DOS command: dir /w



Enter Command: show anything

Is anything a directory or file? dir

English paraphrase: list object directory anything
MS-DOS command: dir anything
```

```
Enter Command: show file anything

   Do you wish to:
1. view the file contents
2. see if file exist
choice: 2

English paraphrase: list object file anything.
MS-DOS command: dir anything.



Enter Command: show anything

Is anything a directory or file? file

   Do you wish to:
1. view the file contents
2. see if file exist
choice: 2

English paraphrase: list object file anything.
MS-DOS command: dir anything.



Enter Command: show the anything directory

English paraphrase: list object directory anything
MS-DOS command: dir anything



Enter Command: create the directory stuff

English paraphrase: create directory stuff
MS-DOS command: mkdir stuff



Enter Command: create stuff

English paraphrase: create directory stuff
MS-DOS command: mkdir stuff
Enter command: exit
```

```
          PE_DOS
     (Plain English DOS)

Use the system:
 (1) to perform commands
 (2) as a tutorial
Enter Choice: 1



Enter Command: show me the contents of directory answers

English paraphrase: list object directory answers
MS-DOS command: dir answers

Execute command? no

Command not executed...per users request.



Enter Command: time

English paraphrase: list time
MS-DOS command: time

Execute command? yes



Enter Command: copy to a b

English paraphrase: copy from file b to file a
MS-DOS command: copy b a

Execute command? yes



Enter Command: halt
```